HEXWORTH PRIME · OPERATOR QUICK REFERENCE

CTS4321C — Advanced Linux Administration

Week 2 Operator Field Manual

Firewalls • Authentication Hardening • Antivirus & Threat Scanning • Package Management

TOPIC 1	TOPIC 2	TOPIC 3	TOPIC 4
Linux Firewalls iptables · nftables · UFW · Default Deny	Authentication Hardening PAM · SSH · fail2ban · TOTP	Antivirus & Threat Scanning ClamAV · rkhunter · Scheduled Scans	Package Management apt · dpkg · GPG · checkinstall

Quick reference for lab work and incident response. Commands sourced from Week 2 slide decks, lab solutions, and ClamAV 0.103+ / Ubuntu 22.04 documentation.

Course: CTS4321C Advanced Linux Administration
Week: 2
Subtitle: Firewalls | Authentication Hardening | Antivirus & Threat Scanning | Package Management

WEEK 2 OPERATOR QUICK REFERENCE

Firewalls | Authentication | Antivirus | Packages — YOUR CELL IS YOUR RESPONSIBILITY

The rule: one control answers one threat.
Use these workflows to keep the cell hardened, authenticated, clean, and traceable during labs and incidents.

BREACH RESPONSE FLOW

Stage	Question	Tool
Auth	Who got in?	`grep 'Accepted' /var/log/auth.log`
Access	What did they touch?	`grep 'sudo' /var/log/auth.log`
Persistence	Did they leave a backdoor?	`cat ~/.ssh/authorized_keys`
Malware	Did they drop anything?	`clamscan -r /tmp /home`
Network	Is a rule still open?	`iptables -L -n -v`
Packages	Did they install anything?	`dpkg -l | sort -k3 | tail -20`

MATRIX HOUSE OPERATOR RULES

- Lock the perimeter before anything else reaches the network
- Disable password auth — keys and TOTP only
- AV is detection, not prevention — layer it with the firewall
- Never install packages outside the verified repository chain
- Diagnose before fixing; log before deleting

First Commands at Lab Start

Command	Use
`iptables -L -n -v`	Confirm firewall state
`grep 'Accepted|Failed' /var/log/auth.log`	Review recent auth events
`systemctl status clamav-daemon clamav-freshclam`	AV service state
`apt list --upgradable`	Pending security updates
`faillock --user <name>`	Check lockout state

Operator Quick-Reference: Problem to Command

Problem	First Question	Command
Locked out of server	Did you set DROP before ESTABLISHED rule?	`iptables -I INPUT 1 -m state --state ESTABLISHED,RELATED -j ACCEPT`
SSH brute force active	Is fail2ban running?	`fail2ban-client status sshd`
Malware suspected	When was the last scan?	`clamscan -r -i /`
Package install fails	Is the GPG key trusted?	`apt-key list` / `gpg --show-keys`
Locked account	How many failures?	`faillock --user ubuntu`
Firewall rule not working	Is rule order correct?	`iptables -L INPUT --line-numbers -n`

> Week 2 Goal: reduce attack surface to only what is explicitly permitted, authenticated, and verified.

FIREWALLS

iptables / nftables / UFW — YOUR CELL IS YOUR RESPONSIBILITY

Lock down before you connect. Every open port is an attack surface.

The Linux Firewall Stack

```
UFW  -->  iptables / nftables  -->  netfilter (kernel)
Frontend     Userspace tools         Kernel engine
```

All rules ultimately live in netfilter hooks inside the kernel. UFW is a frontend to iptables. iptables on Ubuntu 22.04 is the `iptables-nft` compatibility layer — it translates commands to nftables rules.

Ubuntu 22.04 default state: UFW is installed but **inactive**. No rules are in effect until you explicitly enable the firewall.

iptables: Core Commands

Command	Effect
`iptables -L -v -n`	List all rules, verbose, numeric
`iptables -L INPUT --line-numbers -n`	Show INPUT chain with rule numbers
`iptables -A INPUT -p tcp --dport 22 -j ACCEPT`	Append: allow SSH
`iptables -I INPUT 1 -s 203.0.113.0/24 -j DROP`	Insert at position 1: block subnet
`iptables -D INPUT 3`	Delete rule number 3
`iptables -F INPUT`	Flush (clear) all INPUT rules
`iptables -P INPUT DROP`	Set default policy to DROP
`iptables-save > /etc/iptables/rules.v4`	Save rules to file
`iptables-restore < /etc/iptables/rules.v4`	Restore rules from file

Default-Deny Ruleset (Apply to Every Server)

```
# 1. Allow loopback
iptables -A INPUT -i lo -j ACCEPT

# 2. Allow established and related sessions (DO THIS BEFORE DROP RULES)
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# 3. Drop invalid packets
iptables -A INPUT -m state --state INVALID -j DROP

# 4. Allow specific services
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
iptables -A INPUT -p tcp --dport 53  -j ACCEPT
iptables -A INPUT -p udp --dport 53  -j ACCEPT

# 5. Rate-limit SSH (block brute force)
iptables -A INPUT -p tcp --dport 22 -m state --state NEW \
    -m limit --limit 3/min --limit-burst 3 -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j DROP

# 6. LOG before final DROP (required for audit trail)
iptables -A INPUT -j LOG --log-prefix "CELL-PERIMETER: " --log-level 4

# 7. Final catch-all DROP
iptables -A INPUT -j DROP
iptables -P INPUT DROP
```

Critical ordering rule: The ESTABLISHED,RELATED rule must come before any DROP rules. Setting `-P INPUT DROP` before adding it kills your own SSH session immediately.

DROP vs REJECT vs ACCEPT

Target	Behaviour	Use Where
`ACCEPT`	Packet passes through	All explicitly permitted traffic
`DROP`	Silent discard — sender times out	Public-facing interfaces (default)
`REJECT`	Discard + ICMP error sent back	Internal networks only (fast feedback)

On public interfaces: always DROP. REJECT tells an attacker the firewall is active.

UFW: Quick Reference

```
# Check status
ufw status verbose

# Safe enable sequence (always allow SSH first)
ufw allow ssh
ufw enable

# Common rules
ufw allow 22/tcp
ufw allow 443/tcp
ufw allow 53/udp
ufw allow from 10.0.0.50 to any port 3306
ufw deny 23/tcp
ufw limit ssh          # rate-limit: block after 6 attempts in 30 sec

# Delete rules
ufw status numbered
ufw delete 3

# Application profiles
ufw app list
ufw allow 'Nginx Full'

# Disable / reset
ufw disable
ufw reset
```

Persistence

```
# Install persistence service
apt install netfilter-persistent

# Save current rules
iptables-save > /etc/iptables/rules.v4
netfilter-persistent save

# Enable auto-restore on reboot
systemctl enable netfilter-persistent
```

nftables: Modern Syntax Reference

```
# List all rules
nft list ruleset

# Minimal stateful firewall (/etc/nftables.conf)
table inet filter {
    chain input {
        type filter hook input priority 0; policy drop;
        iif lo accept
        ct state established,related accept
        ct state invalid drop
        tcp dport 22 ct state new limit rate 3/minute accept
        tcp dport 22 ct state new drop
        tcp dport { 443, 8443 } accept
        log prefix "CELL-DROP: " drop
    }
}

# Apply and enable
nft -f /etc/nftables.conf
systemctl enable nftables
```

FIREWALL OPERATOR RULES

- ESTABLISHED,RELATED rule is mandatory before any DROP rules
- Always verify with `iptables -L -n -v` after each rule addition
- `iptables-save` does not persist across reboots — use netfilter-persistent
- UFW is a frontend only — inspect generated rules with `iptables -L ufw-user-input -n -v`
- DROP = stealth on public interfaces; REJECT only on internal segments

AUTHENTICATION HARDENING

PAM | SSH | sudo | fail2ban | TOTP — YOUR CELL IS YOUR RESPONSIBILITY

A password is a single lock. The dual-lock requires: key + second factor + rate limiting.

sshd_config: Hardening Checklist

```
# /etc/ssh/sshd_config — hardened settings

PermitRootLogin              no
PasswordAuthentication       no
PermitEmptyPasswords         no
ChallengeResponseAuthentication yes   # set yes when TOTP is in use

Protocol                     2
Ciphers aes256-gcm@openssh.com,chacha20-poly1305@openssh.com,aes256-ctr
MACs    hmac-sha2-512,hmac-sha2-256
KexAlgorithms curve25519-sha256,diffie-hellman-group-exchange-sha256

AllowUsers deploy ops-user          # whitelist by username
AllowGroups sshusers                # or by group

LoginGraceTime  30
MaxAuthTries     3
MaxSessions      5
```

```
# Apply the new config (test first — never sshd restart over an untested config)
sshd -t                          # syntax check
systemctl reload sshd            # graceful reload
```

SSH Key Authentication

```
# Generate an ed25519 key pair (recommended algorithm)
ssh-keygen -t ed25519 -C "ops-station-2026"
# Enter file: /home/user/.ssh/id_ed25519
# Enter passphrase: (always use one)

# Copy public key to server
ssh-copy-id -i ~/.ssh/id_ed25519.pub user@server

# Manual: append to authorized_keys
cat ~/.ssh/id_ed25519.pub | ssh user@server \
    "mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys"

# Verify
ls -la ~/.ssh/authorized_keys
```

Key types ranked by preference: **ed25519** (recommended), RSA 4096 (compatible), ecdsa-sha2-nistp521. Never use RSA 1024 or DSA — both are broken.

PAM: Module Types and Control Flags

Module Type	Function
`auth`	Verify identity (passwords, 2FA)
`account`	Check access permissions (time, expiry)
`password`	Enforce password change requirements
`session`	Set up and tear down the session (logging, env)

Control Flag	Behaviour
`required`	Must succeed; failure does not stop processing but login fails
`requisite`	Must succeed; failure immediately halts processing
`sufficient`	If this succeeds and nothing prior failed, accept
`optional`	Result is ignored for overall pass/fail

Key files: `/etc/pam.d/sshd`, `/etc/pam.d/common-auth`, `/etc/pam.d/common-password`

Password Policy: pwquality and login.defs

```
# Install pam_pwquality
apt install libpam-pwquality

# /etc/security/pwquality.conf
minlen = 14
dcredit = -1       # require at least 1 digit
ucredit = -1       # require at least 1 uppercase
lcredit = -1       # require at least 1 lowercase
ocredit = -1       # require at least 1 special character
maxrepeat = 3
gecoscheck = 1
dictcheck = 1
enforce_for_root = 1

# /etc/pam.d/common-password (add above pam_unix line)
password required pam_pwquality.so retry=3

# /etc/login.defs — aging policy
PASS_MAX_DAYS   90
PASS_MIN_DAYS    1
PASS_WARN_AGE   14
ENCRYPT_METHOD  yescrypt

# Apply aging to an existing user
chage -M 90 -m 1 -W 14 username
chage -l username        # view current aging settings
```

Account Lockout: pam_faillock

```
# /etc/security/faillock.conf
deny         = 5      # lock after 5 failures
unlock_time  = 900    # auto-unlock after 15 min
fail_interval = 900   # count failures within this window
even_deny_root
audit

# /etc/pam.d/common-auth (add these lines)
auth required                    pam_faillock.so preauth silent
auth [success=1 default=ignore] pam_unix.so nullok
auth [default=die]               pam_faillock.so authfail
auth sufficient                  pam_faillock.so authsucc

# View failures and unlock
faillock --user ubuntu
faillock --user ubuntu --reset
```

TOTP / MFA with pam_google_authenticator

```
# Install
apt install libpam-google-authenticator

# Generate per-user TOTP config (run as the user, not root)
google-authenticator -t -d -f -r 3 -R 30 -w 17
# -t = time-based   -d = disallow reuse   -f = force write
# -r 3 -R 30 = rate-limit 3 logins per 30 sec
# -w 17 = wider window for clock skew

# Add to /etc/pam.d/sshd (enforcement line)
auth required pam_google_authenticator.so

# or with nullok (grace period — users without ~/.google_authenticator skip TOTP)
auth required pam_google_authenticator.so nullok

# /etc/ssh/sshd_config — required settings for TOTP via PAM
KbdInteractiveAuthentication yes
PasswordAuthentication no

# Reload
systemctl reload sshd
```

sudo Configuration

```
# Edit sudoers ONLY with visudo (syntax checks before saving)
visudo

# /etc/sudoers examples
ubuntu  ALL=(ALL:ALL) ALL                    # full sudo
deploy  ALL=(ALL) NOPASSWD: /usr/bin/systemctl reload nginx   # single command no password

# Add a group to sudoers
%sysadmins ALL=(ALL) ALL

# Require TOTP for sudo (add to /etc/pam.d/sudo)
auth required pam_google_authenticator.so
```

fail2ban: Installation and Core Config

```
# Install
apt install fail2ban

# Copy the local config (never edit jail.conf directly)
cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local

# /etc/fail2ban/jail.local — [DEFAULT] section
bantime   = 3600      # ban for 1 hour
findtime  = 600       # count failures within 10 min window
maxretry  = 5         # ban after 5 failures
ignoreip  = 127.0.0.1 10.0.0.0/24    # never ban these IPs (ALWAYS include admin IP)

# Enable sshd jail
[sshd]
enabled = true
port    = ssh
logpath = %(sshd_log)s

# Apply
systemctl reload fail2ban
```

fail2ban: Day-to-Day Operations

Need	Command
Overall status	`fail2ban-client status`
SSH jail detail	`fail2ban-client status sshd`
Manually ban an IP	`fail2ban-client set sshd banip 203.0.113.99`
Unban an IP	`fail2ban-client set sshd unbanip 10.0.0.50`
Watch live bans	`tail -f /var/log/fail2ban.log`
Test a filter	`fail2ban-regex /var/log/auth.log /etc/fail2ban/filter.d/sshd.conf`

AUTH OPERATOR RULES

- `PasswordAuthentication no` + `KbdInteractiveAuthentication yes` enables TOTP via PAM
- Always whitelist your admin IP in fail2ban `ignoreip` before enabling
- `sshd -t` before every `systemctl reload sshd`
- Removing the rogue key from `authorized_keys` is the first containment step after an intrusion
- `even_deny_root` in faillock.conf is a DoS risk on single-admin systems — weigh it

ANTIVIRUS AND THREAT SCANNING

ClamAV | rkhunter | Scheduled Scans — YOUR CELL IS YOUR RESPONSIBILITY

ClamAV is detection, not prevention. Layer it with your firewall and file integrity monitoring.

ClamAV Architecture

Component	Role
`clamscan`	CLI scanner — loads DB each run; use for one-off and scheduled scans
`clamd`	Resident daemon — loads DB once; use for repeated scans and on-access
`clamdscan`	Client for clamd — near-instant scans by reusing the running daemon
`freshclam`	Signature update daemon — keeps DB current from `database.clamav.net`
`clamav-milter`	Mail filter integration (Postfix/Sendmail via milter protocol)

Installation

```
# Install
apt update
apt install clamav clamav-daemon

# Initial signature download (stop freshclam service first)
systemctl stop clamav-freshclam
freshclam
systemctl start clamav-freshclam
systemctl enable clamav-freshclam

# Verify
clamscan --version
# ClamAV 0.103.8/27234/Thu Apr  9 08:00:00 2026

# Start and enable the daemon
systemctl start clamav-daemon
systemctl enable clamav-daemon
systemctl status clamav-daemon
```

Signature Databases

File	Size	Update Frequency	Content
`main.cvd`	~162 MB	Infrequent	6.6M historical signatures
`daily.cld`	~67 MB	Multiple times per day	Emerging threats
`bytecode.cld`	~243 KB	Frequent	Heuristic/behavioral detection

```
# Database file locations
ls -lh /var/lib/clamav/

# Manually trigger update and watch output
freshclam --verbose

# Check freshclam log
tail -20 /var/log/clamav/freshclam.log
```

Scanning: clamscan and clamdscan

```
# Scan a single file
clamscan /tmp/suspicious.zip

# Scan a directory recursively, show only infected files
clamscan -r -i /home/ubuntu/uploads

# Move infected files to quarantine
clamscan -r -i --move=/var/quarantine /var/www/uploads

# Write results to log file
clamscan -r -i --log=/var/log/clamav/scan.log /

# Scan using the daemon (faster — daemon already loaded the DB)
clamdscan /home/ubuntu/uploads

# EICAR test file: verify detection without real malware
printf 'X5O!P%%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*' \
    > /tmp/eicar.com
clamscan /tmp/eicar.com
# /tmp/eicar.com: Eicar-Signature FOUND

# Exit codes: 0 = clean, 1 = infected, 2 = error
```

Quarantine Setup

```
# Create a secure quarantine directory
mkdir -p /var/quarantine
chown root:root /var/quarantine
chmod 700 /var/quarantine    # only root can access

# Scan and quarantine in one step
clamscan -r -i --move=/var/quarantine /var/www/uploads

# List quarantined files
ls -la /var/quarantine/
```

Do not delete infected files — quarantine preserves evidence for forensic analysis. Never point the quarantine directory at a web-accessible path.

Scheduled Scans

```
# Cron method (root's crontab)
crontab -e
# Daily at 02:30 — infected files quarantined, results emailed if threats found
30 2 * * * clamscan -r -i --move=/var/quarantine \
    --exclude-dir=/proc --exclude-dir=/sys \
    --log=/var/log/clamav/nightly-$(date +\%Y\%m\%d).log / 2>&1 \
    | mail -s "ClamAV $(hostname)" secops@example.com

# systemd timer method (recommended — logs to journald)
# /etc/systemd/system/clamav-scan.service
[Unit]
Description=ClamAV nightly filesystem scan
After=clamav-freshclam.service

[Service]
Type=oneshot
User=root
ExecStart=/usr/local/bin/clamav-nightly-scan.sh
StandardOutput=journal
StandardError=journal

# /etc/systemd/system/clamav-scan.timer
[Unit]
Description=ClamAV nightly scan timer

[Timer]
OnCalendar=*-*-* 02:30:00
Persistent=true

[Install]
WantedBy=timers.target

# Enable the timer
systemctl daemon-reload
systemctl enable --now clamav-scan.timer
systemctl list-timers | grep clamav
```

rkhunter: Rootkit Detection

ClamAV does not detect kernel rootkits. rkhunter fills that gap.

```
# Install
apt install rkhunter

# Update signature database
rkhunter --update

# Establish a baseline hash of all system files (run right after a clean install)
rkhunter --propupd

# Run a full check
rkhunter --check

# Cronjob mode — only output warnings, suitable for nightly email
rkhunter --cronjob --report-warnings-only

# Schedule: daily at 03:00
0 3 * * * rkhunter --cronjob --report-warnings-only \
    | mail -s "rkhunter report $(hostname)" admin@example.com

# View the last rkhunter log
cat /var/log/rkhunter.log
```

rkhunter checks: MD5 hashes of system binaries, hidden files, suspicious network ports, SUID root files, known rootkit signatures, and kernel module names.

AV OPERATOR RULES

- Run `rkhunter --propupd` immediately after a clean install — baseline before any threats land
- Always use `--move` instead of `--remove` — quarantine, do not delete
- `clamscan` exit code 1 means infected files found (not an error)
- clamd must be running for `clamdscan` to work — check `systemctl status clamav-daemon`
- freshclam must be running for signatures to stay current — without it, your AV is blind

PACKAGE MANAGEMENT

apt | dpkg | GPG | checkinstall — YOUR CELL IS YOUR RESPONSIBILITY

Every package is a trust decision. Verify the source. Verify the key. Track what you install.

APT Core Commands

Command	Effect
`apt update`	Refresh local package index (no packages changed)
`apt upgrade`	Install newer versions of installed packages
`apt full-upgrade`	Upgrade + handle dependency adds/removes
`apt install nginx`	Install a package
`apt install nginx=1.18.0-6ubuntu14`	Install a specific version
`apt remove nginx`	Remove binary, keep config
`apt purge nginx`	Remove binary and config
`apt autoremove`	Remove orphaned dependency packages
`apt clean`	Delete cached .deb files from `/var/cache/apt/`
`apt list --upgradable`	Show what can be upgraded
`apt search term`	Search by name or description
`apt show nginx`	Show package details

update vs upgrade

```
# The correct sequence for applying security updates
apt update && apt upgrade -y

# Simulate an upgrade without making changes
apt upgrade --simulate

# Upgrade a single package only
apt install --only-upgrade nginx

# Purge all packages in "rc" state (removed but config kept)
dpkg -l | grep "^rc" | awk '{print $2}' | xargs apt purge -y
```

`apt update` changes only `/var/lib/apt/lists/` — no packages are touched. `apt upgrade` uses the already-refreshed index. Running upgrade without update first means stale data, missed security patches.

dpkg: Low-Level Operations

```
# List all installed packages
dpkg -l

# List files installed by a package
dpkg -L nginx

# Find which package owns a file
dpkg -S /usr/bin/curl

# Install a local .deb file
dpkg -i package.deb

# Install multiple .deb files (avoids dependency ordering issues)
dpkg -i deps/libpcap-dev_1.10.1-4_amd64.deb \
         deps/libssl-dev_3.0.2-0ubuntu1.13_amd64.deb

# Verify package integrity (checksums against installed files)
dpkg -V nginx

# Check package status
dpkg -s nginx
```

Repository Configuration

```
# /etc/apt/sources.list — Ubuntu 22.04 Jammy default structure
deb http://archive.ubuntu.com/ubuntu/ jammy main restricted
deb http://archive.ubuntu.com/ubuntu/ jammy-updates main restricted
deb http://security.ubuntu.com/ubuntu  jammy-security main restricted
# Components: main (official), restricted (non-free drivers), universe (community), multiverse (non-
free)

# Drop-in repo files
ls /etc/apt/sources.list.d/

# View all active repository priorities
apt-cache policy | head -40
```

Adding Third-Party Repositories (Modern Pattern)

```
# Example: Docker on Ubuntu 22.04
apt install ca-certificates curl gnupg
install -m 0755 -d /etc/apt/keyrings

# Download and store the vendor's GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
    | gpg --dearmor -o /etc/apt/keyrings/docker.gpg
chmod a+r /etc/apt/keyrings/docker.gpg

# Add the repository (signed-by points to the key file)
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
    https://download.docker.com/linux/ubuntu \
    $(. /etc/os-release && echo $VERSION_CODENAME) stable" \
    | tee /etc/apt/sources.list.d/docker.list

apt update
apt install docker-ce docker-ce-cli containerd.io
```

GPG Key Verification

```
# Inspect a key before trusting it
curl -fsSL https://example.com/repo.gpg | gpg --show-keys
# Compare the fingerprint against the vendor's official documentation

# List keys in a per-repo keyring
gpg --no-default-keyring \
    --keyring /etc/apt/keyrings/docker.gpg \
    --list-keys

# Remove a repository and its key cleanly
rm /etc/apt/sources.list.d/docker.list
rm /etc/apt/keyrings/docker.gpg
apt update
```

Package Pinning

```
# /etc/apt/preferences.d/nginx-hold
# Pin nginx to the current version — do not auto-upgrade
Package: nginx
Pin: version 1.18.0-6ubuntu14.4
Pin-Priority: 1001

# Hold a package from being upgraded
apt-mark hold nginx
apt-mark unhold nginx

# Show held packages
apt-mark showhold
```

Priority values: 1001+ = force install/downgrade; 990 = target release; 500 = default repo; 100 = auto-installed; negative = never install.

Building from Source with checkinstall

```
# Verify checksum BEFORE extracting the source
sha256sum -c gridmon-2.1.0.tar.gz.sha256

# Install build dependencies from local .deb files (offline environments)
dpkg -i deps/libpcap-dev_1.10.1-4_amd64.deb \
         deps/libpcap0.8-dev_1.10.1-4_amd64.deb \
         deps/libssl-dev_3.0.2-0ubuntu1.13_amd64.deb

# Standard build sequence
tar -xzf package-2.1.0.tar.gz
cd package-2.1.0
./configure --prefix=/usr/local
make -j$(nproc)

# Install with checkinstall (creates a .deb and registers with dpkg)
checkinstall --pkgname=gridmon --pkgversion=2.1.0 --backup=no --fstrans=no make install

# Verify the package is tracked
dpkg -l gridmon
dpkg -L gridmon   # list installed files
```

Never use `make install` without `checkinstall` on managed systems. Compiled-from-source binaries that bypass dpkg get no security updates and cannot be cleanly removed.

PACKAGE OPERATOR RULES

- `apt update` before every `apt upgrade` — stale index = missed security patches
- Verify GPG key fingerprints against official vendor docs before trusting
- Always use `checkinstall` instead of bare `make install`
- Verify checksums before extracting source tarballs
- `dpkg -V <package>` checks file integrity against the package's stored checksums

INCIDENT PLAYBOOKS

Common failure chains and the command order that finds them — YOUR CELL IS YOUR RESPONSIBILITY

Playbooks keep operators from guessing under pressure.

Lockdown Protocol — Firewall Incident

Scenario: Deploy firewall rules to block active attacks without losing your own SSH session.

```
1   iptables -L -n -v
    (confirm current state — all ACCEPT or partial rules)

2   iptables -A INPUT -i lo -j ACCEPT

3   iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
    (CRITICAL — this protects your current SSH session)

4   iptables -A INPUT -m state --state INVALID -j DROP

5   (add ACCEPT rules for permitted services)
    iptables -A INPUT -p tcp --dport 22 -m state --state NEW \
        -m limit --limit 3/min --limit-burst 3 -j ACCEPT
    iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j DROP

6   (add DROP rules for attack patterns)
    iptables -A INPUT -p udp --dport 161:162 -s 10.0.3.0/24 -j DROP

7   iptables -A INPUT -j LOG --log-prefix "CELL-PERIMETER: " --log-level 4

8   iptables -A INPUT -j DROP
    iptables -P INPUT DROP

9   iptables -L -n -v
    (verify final rule order)

10  iptables-save > /etc/iptables/rules.v4
    apt install netfilter-persistent
    systemctl enable netfilter-persistent
```

The Insider — Auth Intrusion Response

Scenario: Unauthorized login confirmed. Identify entry vector, catalog activity, harden the cell.

```
cat ~/notes.txt
(read the mission brief — what happened and when)

grep 'Accepted\|Failed' /var/log/auth.log
(identify source IP, auth method, and number of attempts)

grep 'TTY=' /var/log/auth.log
(list all sudo commands executed during the session)

cat /root/.bash_history
(confirm attacker commands — sudo log is preserved even if operator
history was cleared)

cat /home/operator/.ssh/authorized_keys
(check for rogue key installed for persistence — remove any unknown keys)

(harden: install TOTP)
apt install libpam-google-authenticator
google-authenticator -t -d -f -r 3 -R 30 -w 17

(add TOTP to PAM)
echo "auth required pam_google_authenticator.so" >> /etc/pam.d/sshd

(disable password auth in sshd_config)
PasswordAuthentication no
KbdInteractiveAuthentication yes

sshd -t
systemctl reload sshd

/opt/verify/test-2fa.sh
(verify 2FA is active before closing session)
```

AV Threat Scan Response

Scenario: Malware suspected after a suspicious file upload or behavioral alert.

```
systemctl status clamav-daemon clamav-freshclam
(confirm AV services are running and signatures are current)

freshclam --verbose
(force a signature update before scanning)

mkdir -p /var/quarantine
chmod 700 /var/quarantine

clamscan -r -i --move=/var/quarantine /var/www/uploads /tmp /home
(scan high-risk directories; infected files moved to quarantine)

cat /var/log/clamav/nightly-$(date +%Y%m%d).log
(review scan results for infection names and file paths)

rkhunter --check
(check for rootkits — ClamAV does not detect these)

cat /var/log/rkhunter.log
(review rootkit scan results for warnings)

ls -la /var/quarantine/
(review quarantined files — do not delete; preserve for forensics)
```

Field Assembly — Offline Package Install

Scenario: No internet access. Compile and install from source using a local archive.

```
cat ~/MISSION.txt && cat ~/notes.txt
(read configure flags and dependency requirements)

cd /opt/archive
sha256sum -c gridmon-2.1.0.tar.gz.sha256
(verify checksum BEFORE extracting)

dpkg -i deps/libpcap-dev_1.10.1-4_amd64.deb \
         deps/libpcap0.8-dev_1.10.1-4_amd64.deb \
         deps/libssl-dev_3.0.2-0ubuntu1.13_amd64.deb
(install all deps in a single command — avoids ordering issues)

dpkg -i packages/checkinstall_1.6.2-4_amd64.deb

tar -xzf gridmon-2.1.0.tar.gz
cd gridmon-2.1.0

./configure --prefix=/usr/local --enable-grid-capture
(read configure output — missing deps will be reported here)

make -j$(nproc)

checkinstall --pkgname=gridmon --pkgversion=2.1.0 \
    --backup=no --fstrans=no make install

dpkg -l gridmon
/usr/local/bin/gridmon --version
(verify the package is tracked and the binary is functional)

/opt/verify/test-install.sh
```

> Professional administrators do not guess. They gather evidence until the failure becomes obvious.
> Your cell is your responsibility. The grid depends on you.

Matrix House | CTS4321C Advanced Linux Administration | Week 2 Operator Field Manual