

SSH Command Tracking Guide

Detail of command execution auditing and tracking across human SSH sessions and AI agent sessions (`amux`).

Part 1: Arch Linux (`ul`, `bmo95`, `studio-kvm-ssh`)

The Linux Audit Framework (`auditd`) captures `execve` system calls triggered by human user sessions (`audit >= 1000`), recording all interactive and SSH payload commands while strictly filtering out background system and rootless Podman container noise.

1. Installation & Service Setup

Install the `audit` package and enable the `auditd` daemon:

```
$ sudo pacman -S --noconfirm audit
$ sudo systemctl enable --now auditd
```

2. Configuration Rules & Interactive User Filtering

Create `/etc/audit/rules.d/execve.rules` with Audit User ID filtering (`audit >= 1000`):

“ CRITICAL: Do NOT include catch-all `execve` rules without `-F audit>=1000`. Background system daemons and rootless Podman containers started by `user@1100.service` (e.g. MinIO healthchecks, Authentik workers) have `audit =`

4294967295 (unset). An unfiltered rule generates ~8 MB of log noise every 7 minutes, rotating out entire log histories within 30 minutes.

```
$ sudo tee /etc/audit/rules.d/execve.rules > /dev/null << 'EOF'
# Delete pre-existing rules
-D

# Set kernel buffer size
-b 8192

# Log all command executions strictly for human/login users (audit >= 1000)
-a always,exit -F arch=b64 -S execve -F audit>=1000 -k remote_commands
-a always,exit -F arch=b32 -S execve -F audit>=1000 -k remote_commands
EOF
```

Load rules into the active kernel:

```
$ sudo auditctl -D && sudo augenrules --load
$ sudo auditctl -e 1
```

Verify active rules:

```
$ sudo auditctl -l
```

Process UID (uid) vs Audit User ID (audit)

- **Process UID (uid=1100 docker-data & subordinate 624288 - 689823):** Controls unprivileged host kernel UIDs for rootless Podman containers.
- **Audit User ID (audit / loginuid):** Assigned exclusively by PAM when a human authenticates via SSH or TTY (audit = 1000). Background containers started via systemd lingering (user@1100.service) have audit = 4294967295 (unset).
- **Result:** Filter by audit >= 1000.

Part 2: Agent Execution Auditing (amux / systemd journal)

When AI agents or automated scripts run multi-line code via `write-pane -f <file>` or `write-pane -b <base64>` in **amux**, those commands can bypass standard `auditd` `execve` logging (because they are streamed directly into an already-running shell's PTY or an interactive Python/Node REPL).

To ensure complete, tamper-resistant auditing and forensic recovery:

1. Structured Journal Logging

Every call to `write-pane` in `amux` automatically logs directly into the **systemd journal** with structured fields:

- `SYSLOG_IDENTIFIER=amux`
- `AMUX_PANE_ID`: Target pane (`%0`, `%1`, etc.)
- `AMUX_WINDOW_NAME`: Hierarchical task window (e.g. `mc-pokelovers-bluemap`)
- `AMUX_SOURCE`: Original script file path or `base64`
- `AMUX_ARCHIVE_PATH`: Permanent snapshot location in `/data/amux/scripts/`
- `AMUX_PAYLOAD_SIZE`: Injected byte count

Querying agent execution logs:

```
# Stream live agent injections
$ journalctl -t amux -f

# Inspect detailed metadata for recent agent commands
$ journalctl -t amux -n 20 -o verbose
```

2. Permanent Script Archival (`/data/amux/scripts/`)

Before any byte is written to the terminal PTY, `amux` automatically saves a bit-for-bit snapshot to:

```
/data/amux/scripts/YYYY-MM-DD_HHMM.SS_<window_name>.sh
```

- **Durability:** Stored on `/data` with ownership `luke:luke`.
 - **Zero Loss:** Even if an agent immediately removes a temporary file from `/dev/shm` after execution, the exact script executed is preserved permanently.
 - **Disaster Recovery:** `/data/amux/scripts/` is automatically backed up during the daily backup run via `/home/luke/git/backups/ul/scripts/backup/amux`.
-

Part 3: Python Command Inspection & Summarization (`audit-log`)

The `/usr/local/bin/audit-log` Python utility parses, decodes, and summarizes `EXECVE` records from `/var/log/audit/audit.log.*`.

CLI Command Options & Usage

```
# 1. Default view: last 20 executed commands
$ audit-log

# 2. View specific number of entries (e.g. last 50 commands)
$ audit-log 50
$ audit-log -n 50

# 3. Stream all recorded command history
$ audit-log all
$ audit-log --all

# 4. Filter by specific day (YYYY-MM-DD)
$ audit-log --day 2026-08-29

# 5. Filter date range (since date / until date)
$ audit-log --since 2026-08-25 --clean

# 6. Filter repetitive background noise (tmux status bars, pollers)
$ audit-log --clean -n 30

# 7. Regex / substring search across command history
$ audit-log --search "funkwhale|podman" --clean

# 8. Deduplicate identical commands in output
$ audit-log --unique --since 2026-08-28
```

```
# 9. Generate automated daily change log summary outline
$ audit-log --daily --since 2026-08-25 --clean
```

Part 4: OpenBSD (Router `danknasty`)

OpenBSD uses native kernel process accounting (`acct`) to track binary executions, alongside syslog-based `sshd` verbose logging and `doas` auditing.

1. Kernel Process Accounting (`acct`)

Enable Accounting:

```
# Create log file
$ doas touch /var/account/acct

# Enable accounting at boot
$ echo "accounting=YES" | doas tee -a /etc/rc.conf.local

# Start accounting daemon immediately
$ doas /usr/sbin/accton /var/account/acct
```

2. Real-Time Command Inspection (`audit-log`)

Query process execution history using the matching `audit-log` helper script:

```
# View the last 20 executed commands (Default)
$ doas audit-log

# View a specific number of recent commands (e.g. 50 entries)
$ doas audit-log 50
```

```
# View ALL recorded command history
$ doas audit-log all
```

3. SSH Verbose Payload & doas Logging

To log non-interactive SSH connections and payload commands to `/var/log/authlog`:

1. Edit `/etc/ssh/sshd_config`:

```
LogLevel VERBOSE
```

2. Reload `sshd`:

```
$ doas rcctl reload sshd
```

3. Monitor logs:

```
$ doas tail -f /var/log/authlog
```

Revision #12

Created 2026-08-09 16:36:04 UTC by admin

Updated 2026-10-20 03:46:20 UTC by luKe