Finding Why a Backup Fails: Correlating Logs Across Sleep, Locks, and Networks

Ronald G. Thomas

2026-11-16



Each failure was written down somewhere; the work was putting the records side by side until the cause lined up with the effect.

> **Note**
>
> Part 9 of the security series; start with Security Foundations for a Multi-Machine Research Fleet for the overview.

Introduction

Between mid-October and mid-November 2026 the laptop in our fleet, athena, made Time Machine backups to a network share on the home server and, for most of that time, did not complete one. The backups were scheduled every hour, the destination was reachable, and the disk had room. Nothing reported a problem. When the failures were finally examined, they turned out to have four separate causes, each of which hid the next.

This post is about how those causes were found. The architecture they broke is described in Research Backup Architecture and the server in A Home Server for a Five-Machine Research Fleet; here the subject is the method. Almost everything needed was already recorded on the two machines: a power-management log on the laptop, the backup daemon's own log and result codes, and the server's firewall and Samba logs. The skill was in choosing which records to compare, aligning them in time, and then asking whether a plausible mechanism connected them.

Motivations

- An intermittent failure rarely announces its cause; the failing process reports what broke, not why.
- macOS records far more about a backup than System Settings shows, but in places most users never look.
- A coincidence in time is evidence, not proof, and it is easy to over-read on a machine that changes state constantly.
- Fixing one cause can expose another that was masked all along.
- A method written down once can be reused on the next failure.

Objectives

1. List the evidence sources on a Mac client and a Linux server, and the commands that read them.
2. Show the time-correlation method on a real set of 14 failures, including its limitation.
3. Work through four causes in the order they were found: sleep, a corrupted image, the lock screen, and the network.
4. Identify the single control that would have caught all of them.



What Is Backup Failure Forensics?

The term is grander than the practice. It means treating a failing backup the way one would treat any intermittent fault: collect the records that both ends keep, put them on one timeline, and look for events that consistently occur together. Three rules kept the work honest:

1. **Start from the failures, not from a theory.** Extract every failure with its exact time and code before deciding what to compare it with.
2. **Test the alternatives the same way.** If sleep is the suspect, the server's network faults, disk errors and session logs should be checked against the same failure times, and recorded as not matching when they do not.
3. **Require a mechanism.** A timing match on a machine that changes state every few minutes can arise by chance. It becomes persuasive only when the configuration explains *how* one event causes the other.

Prerequisites

- Administrator access to the client Mac and the server, ideally over SSH so that logs can be collected from one place.
- On the Mac: the unified log (`/usr/bin/log`), `pmset`, `tmutil`, and `defaults`.
- On the Linux server: `journalctl` with access to the kernel and service logs (membership in the `adm` group suffices for reading), and the firewall's log.
- A shell that can do date arithmetic. The script below uses the BSD `date -j` found on macOS.

The Evidence Sources

Question	Where it is recorded	Command
When did each backup fail, and how?	the unified log, subsystem `com.apple.TimeMachine`	`/usr/bin/log show --info --predicate ...`
What was the last result?	Time Machine's preferences file	`defaults read /Library/Preferences/com.apple.TimeMachine`
When did the Mac sleep and wake?	the power-management log	`pmset -g log`
How is sleep configured?	the active power profile	`pmset -g`
Did the server see the session drop?	the Samba service log	`journalctl -u smbd`
Did the server block anything?	the kernel log (firewall entries)	`journalctl -k --grep 'UFW BLOCK'`
Did the server's hardware complain?	the kernel log	`journalctl -k`

Two details of the Mac side cost time and are worth knowing in advance.

zsh has a built-in command named `log`. On a Mac whose login shell is zsh, `log show ...` runs the built-in, which either prints nothing or complains about its arguments. Call the program by its path, `/usr/bin/log`, every time, including inside scripts run over SSH.

Successes and failures are logged at different levels. backupd writes failures at the error level and successes at the *info* level, which `log show` omits unless asked. An early count in this investigation reported zero successful backups for a night that in fact had fourteen; the query lacked `--info`. Every query below includes it.

Case 1: Fourteen Failures in a Day

The first examination covered 24 hours. Time Machine had made 24 attempts: 10 succeeded and 14 failed. Twelve failures carried the code `BACKUP_FAILED_DISCONNECTED_NETWORK` (the SMB session to the server was lost mid-backup), one `DISCONNECTED_DISK_IMAGE` (the backup image detached), and one `TARGETVOL_NOT_MOUNTED` (the share was not mounted at the start). The two disconnect codes are logged together when a session drops, so they describe one kind of event, not two.

Collecting the two records

```
# Every Time Machine event of the last day, successes included.
/usr/bin/log show --last 24h --style compact --info \
  --predicate 'subsystem == "com.apple.TimeMachine"' > tm24.log

# One line per sleep or wake transition.
pmset -g log | grep 'Delays to Sleep notifications' > pmsleep.txt
```

The `Delays to Sleep notifications` record is written by the kernel on every power transition, so it serves as a clean list of moments when the machine went to sleep or woke.

Pairing each failure with the nearest transition

The script pairs every failure with the closest power transition and prints the offset in seconds:

```
#!/bin/bash
# corr.sh tm24.log pmsleep.txt
tm="$1"; ps="$2"
to_epoch() { date -j -f '%Y-%m-%d %H:%M:%S' "$1" '+%s'; }
grep 'BackupDispatching\] Backup failed' "$tm" |
while read -r d t rest; do
  f="$d ${t%.*}"; fe=$(to_epoch "$f")
  best=''; bestd=999999
  while read -r sd st _; do
    se=$(to_epoch "$sd $st"); diff=$(( se - fe )); ad=${diff#-}
    if [ "$ad" -lt "$bestd" ]; then
```

```
      bestd=$ad; best="$sd $st (${diff}s)"
    fi
  done < "$ps"
  echo "$f $(echo "$rest" | grep -oE 'BACKUP_[A-Z_]+') $best"
done
```

It is quadratic and slow, and for a day's records that does not matter. The result:

Failure	Code	Nearest transition	Offset
08:09:14	NETWORK	08:09:07	-7 s
11:47:39	NETWORK	11:47:27	-12 s
11:58:47	DISK_IMAGE	11:58:06	-41 s
12:25:17	NETWORK	12:25:22	+5 s
14:57:44	NETWORK	14:57:33	-11 s
17:08:30	NETWORK	17:08:15	-15 s
17:27:19	TARGETVOL_NOT_MOUNTED	17:27:32	+13 s
17:36:31	NETWORK	17:36:37	+6 s
18:05:54	NETWORK	18:05:59	+5 s
18:14:50	NETWORK	18:14:49	-1 s
19:57:06	NETWORK	19:57:05	-1 s
20:16:58	NETWORK	20:17:04	+6 s
22:12:19	NETWORK	22:12:20	+1 s
22:28:43	NETWORK	22:28:49	+6 s

All 14 failures fell within 41 seconds of a sleep or wake transition, and 13 within 15 seconds.

Why the match alone was not enough

The same power log showed why this result needed care. The laptop was changing state constantly: the log for the period consisted almost entirely of short dark wakes (the background wakes that Power Nap schedules) and the sleeps that ended them, each lasting from under a minute to about a quarter of an hour. On a machine that changes power state that often, *any* list of events will sit fairly close to some transition. The correlation was verified, but by itself it could not separate cause from coincidence.

What made it persuasive was the mechanism, read from the configuration:

- On mains power the system was set to sleep after **one minute** of user idle (`pmset -c sleep 1`). Backups therefore ran mostly inside dark wakes.
- backupd holds an assertion that prevents *idle* sleep, but that assertion does not stop the *maintenance* sleep that ends a dark wake.
- The SMB session to the share does not survive sleep (`networkoversleep 0`). On the next wake, backupd found its mount gone and failed.

The alternatives, tested the same way

Three other candidates were checked against the same failure times, on the server:

- **The server's Wi-Fi.** Its kernel logged one fault event from the internal wireless adapter in the window, and it fell between two failures, not at one.
- **The server's backup disk.** No USB reset, I/O error or file-system error was logged for the drive.
- **The server's Samba sessions.** smbd opened and closed sessions for the backup account in step with the laptop's wake cycles: the server side of the same disconnects, not an independent cause.

The fix, and what it did not cover

The fix was one line on the laptop:

```
sudo pmset -c sleep 0   # on mains power, never sleep on idle
```

The next night, backups completed every hour without a gap while the lid was open. Then the lid was closed in the morning, and three more failures followed. Each coincided *to the second* with a dark wake out of the resulting clamshell sleep: `sleep 0` disables idle sleep only, and a MacBook with its lid shut and no external display sleeps regardless.

`sudo pmset -a disablesleep 1` would have closed that gap, and it was applied and then reversed five minutes later. It keeps the machine awake on every power source and with the lid closed, and athena is also carried in a bag, where it must be able to sleep. The settled arrangement is an operating rule, the lid stays open while the laptop sits at its desk, plus one more setting: Power Nap off (`sudo pmset -a powernap 0`), so that a closed-lid laptop sleeps fully instead of waking to start backups it cannot finish.

Case 2: The Corrupted Image

Power Nap was turned off because of what it had already done. With the lid closed overnight, it kept waking the laptop for short windows; Time Machine started a network backup in each, and each was cut off when the window ended. Those interrupted writes accumulated in the hours before the backup image on the share was found corrupt, and they are the likely cause.

The recovery took three steps, in an order that kept the old history available:

1. The damaged image was detached by restarting the laptop, because a read-only consistency check (`fsck_apfs`) was holding it open.
2. On the server, the image was renamed with a `.corrupt-<date>` suffix and made read-only (`chmod -R a-w`), so that files from the old history could still be recovered from it if needed.
3. Time Machine started a fresh full backup into a new image.

The plan for the old image, at 454 GB, is to keep it until the new one has completed and been verified. Deleting it sooner would discard the only history from before the fault.



Case 3: The Lock Screen

The fresh full backup did not complete. For two days every hourly attempt ran for a while, copied data, and then failed: 19 attempts in a row. The disconnects of Case 1 were gone. Something else was failing, and the unified log's usual failure line did not say what.

Reading the result code

Time Machine keeps the outcome of the last run in its preferences file, per destination:

```
defaults read /Library/Preferences/com.apple.TimeMachine |
  grep -E 'RESULT|SnapshotDates' -A2
```

The field holds only the last run's result, and it read `RESULT = 80`; backupd's log showed the same outcome for all 19 attempts. The code is not explained in System Settings, but backupd's log names it when a run ends: `BACKUP_FAILED_DEVICE_LOCKED`. The same log lists the files it could not read, one line each, which made the cause concrete:

```
# The files a run could not read. Use the path; zsh has a builtin log.
/usr/bin/log show --last 2h --style compact --info \
  --predicate 'eventMessage BEGINSWITH "Failed to backup"'
```

There were 255 of them, in two places: the cached attachments of an Exchange calendar (`~/Library/Group Containers/group.com.apple.calendar/Attachments`) and four containers belonging to the Find My widgets. macOS protects these with data protection, so they cannot be read while the user session is locked, and backupd treats one unreadable file as a failed backup. The data had been copied each time; the run was then marked failed in its last step.

Why the earlier fix could not help

The operating rule from Case 1 keeps the lid open, and an open lid keeps the Mac *awake*. It does not keep the session *unlocked*: the screen still locks when the machine is idle, as it should. The sleep fix had removed the failure that ended every run early; with runs now lasting long enough to reach their final step, the lock-screen failure became visible for the first time.

The fix, after a check

Excluding the five folders from Time Machine was the obvious remedy, but an exclusion is only safe if nothing in the folders is unique. Two checks came first:

- Calendar's database listed 976 attachment records for the Exchange account, and every one belonged to an event that still existed, so each attachment's original lived on the server.
- A sample attachment was opened from the server, through the calendar, on two different Macs.

The four Find My widget containers were excluded on the same reasoning but were not examined separately. Then:

```
tmutil addexclusion \
  ~/Library/Group\ Containers/group.com.apple.calendar/Attachments
# and the same for each of the four Find My widget containers
tmutil isexcluded \
  ~/Library/Group\ Containers/group.com.apple.calendar/Attachments
```

Without `-p`, `tmutil addexclusion` marks the folder itself, and the mark travels with it; `tmutil isexcluded` is the right check. The next run, with the screen unlocked, completed: the first completed backup in three days. From then on backups completed every hour, nineteen in a row by the next morning, including through the night with the display off and the session locked, and the log recorded no failures. That the session stayed locked all night is inferred from the display log and from a query in the morning, which found it locked; but before the exclusions, every run under a locked session had failed.

Case 4: The Network Underneath

Two server-side conditions were found along the way. Neither caused the failures above, but both made the backups more fragile.

The firewall hid the share. Time Machine found its destination by Bonjour name, and macOS sends some Bonjour queries to the server's unicast address rather than to the multicast group. The server's default-deny firewall dropped them: its log showed blocked packets on UDP port 5353 from the laptop, and over the same day the laptop logged 105 failures to resolve the share's Bonjour name. One firewall rule allowing port 5353 from the LAN fixed it. Over the next day the server blocked no such packets, and the laptop's resolution failures fell from 105 to 10. The symptom was on the client and the cause on the server, which is why both logs had to be read.

Two Wi-Fi adapters on one network. The server, a laptop without an Ethernet port, had two wireless adapters joined to the same network. Each held its own address and default route, the preferred route switched between them every 10 to 20 minutes, and with Linux's default ARP settings either adapter answered for either address. Pings from the laptop ranged from 7 to 117 ms. This was never isolated as the cause of a particular failure; the fix, a single wired interface, is described in part 6 and is still pending.

Case 5: The Cloud Copies

The same review found the nightly cloud copies failing for reasons unrelated to Time Machine but identical in one respect: nobody was told. Over one week the Google Drive jobs on the two main Macs failed on a per-minute query quota, a lapsed login, a night when neither Mac could resolve Google's hosts at 02:00, and finally authorizations that stopped working on both Macs on the same night. Each failure was recorded in the job's log and in launchd's exit status. The details and fixes are in Research Backup Architecture; the cause of the simultaneous authorization loss is still unknown.

The Control That Covers All of Them

Before any of the cases above, the backup history already held the plainest evidence. `tmutil listbackups` showed no completed backup between 15 October and 5 November, a gap of 21 days, and a second gap of nearly two days later in November. Time Machine keeps daily backups for a month, so a backup made in those periods would still have been listed; the gaps were real. Nothing reported either one.

The four cases share a property: each produced an error in some log, but no single error check would have covered all of them, because each failed in a different way and backupd reported only one cause at a time. What they had in common was their effect, a growing age of the newest completed backup. That is the quantity to alert on.

A small script now runs daily at noon on the laptop. It reads the completion dates that backupd records in its preferences file (which remain available while the network share is unmounted, unlike `tmutil latestbackup`), and sends a push notification when the newest is more than 26 hours old. Its log line is the whole story:

```
2026-11-15 12:00:04 last='2026-11-15 19:23:26 +0000' age=0h max=26h
```

Had it been installed in October, it would have reported the 21-day gap on its second day.

Things to Watch Out For

1. **`log` in zsh is not `/usr/bin/log`.** Use the full path, especially over SSH.
2. **Omitting `--info` hides every success.** A failure count without the matching success count can mislead badly.
3. **A timing match on a busy machine is weak evidence alone.** Look for the mechanism in the configuration, and test the alternatives against the same times.
4. **`pmset sleep 0` does not stop clamshell sleep.** A closed lid sleeps the machine whatever the idle setting says.
5. **Awake is not unlocked.** Files under data protection fail a backup while the session is locked; read `RESULT` and the 'Failed to backup' lines.
6. **Check before excluding.** An exclusion is safe only for data whose original lives somewhere else; verify that it does.
7. **Preserve a damaged image before starting over.** Rename it and make it read-only; it holds the only earlier history.



What Did We Learn?

Lessons Learned

- **The records existed; the alerting did not.** Every cause in this post was written down by the machines at the time it happened.
- **A backup reports one failure at a time.** Fixing sleep exposed the lock screen, because runs finally lasted long enough to reach it.
- **Correlation narrows the search; mechanism settles it.** The timing table pointed at sleep, and the power configuration explained why.

- **Read both ends.** The Bonjour failures were visible on the client and caused on the server.
- **Alert on the outcome, not on each cause.** A staleness check catches causes that have not been discovered yet.

Limitations

- The correlation used a single day of data from one laptop on one network, and the base-rate question was answered by mechanism, not by a statistical test.
- That the laptop stayed locked all night during the confirming run is inferred, not observed.
- The staleness alert has run on schedule, but no backup has been stale since it was installed, so a delivered notification has not yet been observed.
- The server's two-adapter problem remains, and the cause of the cloud authorization loss is unknown.
- No restore from the new backup image has been tested yet.

Opportunities for Improvement

1. Force one staleness alert (by setting the age limit to zero) to confirm that the notification arrives.
2. Extend staleness alerting to the cloud copies, which still rely on a manual check.
3. Turn the correlation script into a reusable tool that takes any two timestamped logs.
4. Run Time Machine's backup verification on the new image, and restore one file from it.
5. Give the server a single wired interface.

Wrapping Up

The laptop's backups failed for four reasons in turn: an idle-sleep timer that cut sessions short, interrupted writes that corrupted the image, protected files that could not be read while the screen was locked, and a firewall that hid the share from Bonjour. Each was found by reading records the machines already kept, aligned in time and checked against a mechanism. None was reported when it happened.

Main takeaways:

- Extract failures with exact times and codes first; then decide what to compare them with.
- On a Mac, `/usr/bin/log show --info`, `pmset -g log` and the `RESULT` field in Time Machine's preferences answer most questions.
- Treat a timing match as a lead, and confirm it in the configuration.
- Keep a damaged backup read-only until its replacement is verified.
- Alert on the age of the newest good backup.

See Also

Security series:

1. Security Foundations for a Multi-Machine Research Fleet
2. Setting Up pass: a Unix Password Manager
3. Secrets Management for the Workflow Construct
4. Migrating Off Dropbox: Beyond Dotfiles
5. Research Backup Architecture: Borg, Cloud Copies, and Time Machine
6. A Home Server for a Five-Machine Research Fleet
7. Object Storage for Research Data: The Layer Git Cannot Hold
8. Running the Quarterly Security Audit for the First Time: What Actually Broke
9. Finding Why a Backup Fails: Correlating Logs Across Sleep, Locks, and Networks (this post)

External references:

- `pmset` manual page: power settings, including `sleep`, `powernap` and `disablesleep`.
- `tmutil` manual page: exclusions and backup listings.
- ntfy: the push-notification service the alerts use.

Reproducibility

What was checked. The failure counts, codes and the correlation table come from the laptop's unified log and power log for 2026-11-10 to 2026-11-11, collected and analyzed on 2026-11-11 with the script above; the server-side checks (kernel, Samba and firewall logs) were read on the same day. The lid-closed failures and the clamshell sleep event are from the follow-up on 2026-11-12. The lock-screen result code, the 'Failed to backup' lines, the calendar-attachment check, the exclusions and the first completed backup are from 2026-11-15; the overnight confirmation is from 2026-11-16. Times are Pacific. Addresses and account names are omitted.

Not done. No restore from the new backup image has been tested, and the staleness alert's notification has not been observed.

AI involvement in this post: the investigation's commands were run with AI assistance, and drafting was AI-assisted from the investigation's notes. The choice of fixes, the operating rule and the editorial judgment are the author's.

Session information:

```
R version 4.6.0 (2026-04-24)
Platform: aarch64-apple-darwin23
Running under: macOS Tahoe 26.6.2

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.6/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.6/Resources/lib/libRlapack.dylib;  LAPACK version 3.12.

locale:
```

```
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: America/Los_Angeles
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

loaded via a namespace (and not attached):
 [1] compiler_4.6.0  fastmap_1.2.0   cli_3.6.6       tools_4.6.0
 [5] htmltools_0.5.9 parallel_4.6.0  otel_0.2.0      yaml_2.3.12
 [9] rmarkdown_2.31  knitr_1.51      jsonlite_2.0.0  xfun_0.62
[13] digest_0.6.39   rlang_1.2.0     evaluate_1.0.5
```

Rendered on 2026-11-16 at 09:39 PST. Source: ~/prj/web/ryyblog/posts/sec-backup-failure-forensics/analysis/report/index.qmd

Let's Connect

Questions, suggestions, or corrections are welcome.

- **GitHub:** rgt47
- **Twitter/X:** @rgt47
- **LinkedIn:** Ronald Glenn Thomas
- **Email:** Contact form

I would like to hear from you if:

- You find an error, or a better approach to anything in this series.
- You have chased an intermittent backup failure and found a better way to line up the evidence.
- You want to compare notes on backups, secrets, or reproducible research infrastructure.

Related posts in this cluster

This post is part of the *Security* series. Recommended reading order:

1. Security Foundations for a Multi-Machine Research Fleet
2. Setting Up pass: a Unix Password Manager
3. Secrets Management for the Workflow Construct
4. Migrating Off Dropbox: Beyond Dotfiles
5. Research Backup Architecture: Borg, Cloud Copies, and Time Machine
6. A Home Server for a Five-Machine Research Fleet
7. Object Storage for Research Data: The Layer Git Cannot Hold
8. Running the Quarterly Security Audit for the First Time: What Actually Broke

9. **Finding Why a Backup Fails: Correlating Logs Across Sleep, Locks, and Networks** (this post)