International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Beyond Deletion Protection: A Recoverability-Tiered Defense Against Accidental and Adversarial Data Loss in Public Cloud

Narayana Rajasekhar Vemburaj

Submitted:02/09/2021 **Revised**:17/10/2021 **Accepted**:29/10/2021

***Abstract*—**All public clouds offer deletion protection, yet data still vanishes: a cleanup job removes a bucket, a schema is dropped in a migration, a shared file system is deleted along with the training corpus it held. The mechanics vary, but the vendor checkbox does not stop them. This paper argues that deletion protection is not a property but a setting, and that recoverability is the property that actually survives contact with real operations. A tiered defense is proposed in which every persistent resource is classified by its demonstrated ability to be brought back — versioned, soft-deleted within a window, restorable from backup only, or unrecoverable — with prevention, detection, and recovery controls assigned according to tier and blast radius. The threat model spans three actor classes: accidental loss from human/automation error, autonomous-agent error at machine speed, and adversarial destruction by ransomware or an insider with stolen credentials. Because an adversary with primary-plane admin credentials can unwind ordinary configuration protections, every resource is graded twice — once for accident, once for compromise — and the gap between the two grades is precisely what retention locks, backup-plane isolation, and verified restores are built to close. The discipline reduces to one sentence: a backup that has never been test-restored is an assumption, not a control.

Keywords*—data protection, recoverability, ransomware, immutable backup, object lock, soft delete, deletion protection, cloud storage, disaster recovery.*

I. INTRODUCTION

On a Tuesday, a teardown ticket was approved by a change window and executed on a Thursday by an engineer following the ticket to remove the environment ml-sandbox. There were 41 resources and no errors. The plan output scrolled past a managed file system carrying 4.2 terabytes, and no one on the review had reason to stop, because everything in ml-sandbox was a sandbox. It said so in the name.

Two years earlier it was a sandbox. Then a feature-engineering pipeline began writing handcrafted training data to the file share, since the share was mounted and there was space, and by the time of the teardown the only copy of the training corpus for a demand-forecasting model lived on it — labeled, joined, and quality-checked data representing 11 months of work, rebuilt nowhere. Everyone had agreed: back the file share up, nightly.

The retraining pipeline failed the following Monday, and the restore process is where the trouble surfaced. The backup job had been green every night for two years. It had been configured to back up the paths that made sense for a sandbox, and a year in, the corpus had migrated to a directory that the job's include pattern no longer matched. Two years of successful, nightly backups of a directory tree that held nothing worth backing up.

It took six weeks of pipeline time to rebuild the corpus from source. Two upstream feeds carried only 30-day contractual retention, so roughly four months of the training window could not be recreated at any cost
— a documented gap the business owner had to sign off on.

This scene blends environments I have worked in and supported as a cloud technical account manager; every mechanic in it has been observed first-hand. What turns it into an article rather than a war story is this: deletion protection was available at every layer of that stack. Backup policies existed. There was a control on paper for exactly this kind of loss. The checkbox was real. The property it was meant to purchase had never been proven, and on the Thursday, it was required, it was not present.

Technical Account Manager
Amazon Web Services, Inc

II. A SETTING, NOT A PROPERTY

Deletion protection is an option. Recoverability is a property. This paper lives in the space between those two sentences.

It is a genuinely useful environment: a managed database service that refuses a delete API call when a protection flag is set; object versioning that retains a noncurrent version behind a delete marker; soft delete on blobs; a Terraform lifecycle block with prevent destroy. Each protects a particular resource along a particular path, and all of them are worth using.

Even so, there are three mundane ways they fail, and each has been observed directly. They go unset — the flag protects the database someone remembered to flag, and does nothing for the one a script provisioned last quarter. They get undone — a cost review shortens a retention window, or a lifecycle rule silently expires the noncurrent versions that were the whole point of versioning. And they sit unproven — the backup job that ran green for two years while backing up the wrong directory. A setting is a description of what you asked the platform to do. It says nothing about what would happen if the data were needed back tonight.

Recoverability, as used here, is measured: for a given resource, the demonstrated ability to return it to a known-good state, within a bounded time, along a path that has been exercised recently enough to be trusted. The load-bearing word is demonstrated, and the rest of this paper is machinery built around that word.

A companion piece in this series established an adjacent discipline: hard deletion of persistent state is ineligible for autonomous execution, every destroy is staged behind a cooling-off window, and the gate that allows deletion to start requires a fresh, proven reverse-restore path before it opens. That gate is not re-derived here. This paper is the second half of the same contract — the engineering that makes "a fresh, proven restore path" a true sentence for everything that matters, including on the day the actor destroying the data is hostile.

III. THREE WAYS THE DATA DIES

There are three actor classes in the threat and three in the defense, and the defense must hold for all of them, because the storage substrate cannot tell them apart.

Class one is accident: human and automation error such as a cleanup job driven by a stale last-access heuristic, a destructive schema migration, or an environment teardown that no one flagged as no longer a sandbox. These occur in daylight, inside approved change windows, and in practice they account for the largest share of real losses — which is why they are listed first.

Class two is agent error. A delete-capable autonomous actor calls the same APIs a human would, but at machine speed and for a seemingly good reason. Two hard lines from this series apply directly: agent identities cannot be deleted from datastores or backups at the substrate level regardless of the model's confidence, and hard deletion of persistent state does not qualify for autonomous execution under any circumstance. Neither rule is re-derived here. What class two adds is quieter: the agent need not call delete at all to cause deletion. If it marks a resource as ephemeral in its own inventory, or a teardown plan is approved by a tired human rubber-stamping a draft, the loss has simply moved into class one at machine speed. The defense cannot assume it knows which class it is facing.

Class three is adversarial and different in kind, because the attacker goes after the recovery paths first: obtain credentials, enumerate backup infrastructure, shorten retention policies, delete snapshots, disable versioning — so that preexisting noncurrent versions expire or vanish — and only then encrypt the primary data. By the time encryption shows up on a dashboard, the reverse gears are already gone. A malicious insider follows the same shape, holding both the credentials and the map. The pattern recurs in public post-mortems: an attacker who gained control of a cloud console deleted a company's primary data and its backups together, and the company reported it could not continue operating [1]. The backups existed; they simply lived where the attacker's credentials could also reach, and any recovery path that a compromised credential can switch off will be switched off first.

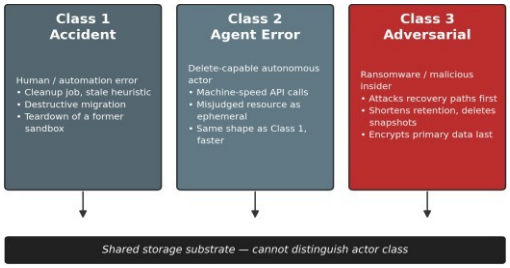


Fig. 1. The three threat-actor classes converge on a storage substrate that cannot distinguish between them, so the defense must hold uniformly across all three.

Countermeasures for class three — offline or otherwise isolated backups, tested restores — are well established in guidance from industry and government sources [2], [3]. What has been missing in practice is the connective rule for which resources need those countermeasures, and how much is enough. That connective rule, the recoverability-tiered defense, is the subject of the rest of this paper: it sets the number of controls a resource is owed from its measured tier and blast radius, and it assigns both a normal and an adversarial grade.

TABLE I. Threat Actor Classes and Primary Defenses

Class	Actor	Typical Trigger	Primary Defense
1	Human / automation	Cleanup job, destructive migration, stale-name teardown	Change review, tier floor, blast-radius cap
2	Autonomous agent	Machine-speed delete; resource misjudged as ephemeral	Deletion gate; no-verified-inverse discipline
3	Adversary / insider	Stolen credentials; retention shortened before encryption	Immutability lock + backup-plane isolation

IV. THE TIER LADDER

Classify every persistent resource by how it can actually be brought back. Four tiers, ordered from cheapest recovery to none, are shown in Fig. 2 and summarized in Table II.

Tier 1 — versioned. The inverse of loss is a single API call: noncurrent object versions, database point-in-time restore within its window, or a file system with continuous snapshots. Recovery is near-complete and takes minutes for object versions and snapshots, or hours for a large database point-in-time restore, without negotiating with a separate backup plane.

Tier 4 — unrecoverable. No path exists. When it is gone, it is gone in the sense that matters — as with

Tier 2 — soft-deleted within a window. The resource can be recovered for a limited time after deletion: delete markers on live versions, blob soft delete, a staged deletion window, or the quarantine phase of a staged destroy. The window is a running clock, and recovery is cheap as long as someone acts before it closes.

Tier 3 — backup-only. Recovery means restoring from a separate copy, bounded by that backup's own recovery-point and recovery-time limits, and only if a restore has actually been exercised.

the four months of feed history in the opening scene.



Fig. 2. The four-tier recoverability ladder, ordered from cheapest recovery (Tier 1) to none (Tier 4), with representative mechanisms and typical recovery time.

TABLE II. Recoverability Tier Definitions

Tier	Name	Recovery Mechanism	Representative Example
1	Versioned	Object versions, DB point-in-time restore, continuous snapshots	S3/Blob versioning, RDS PITR
2	Soft-deleted	Delete markers, blob soft delete, staged deletion window	Azure Blob soft delete
3	Backup-only	Restore from a separate, distinct copy	AWS Backup, GCP disk snapshots
4	Unrecoverable	No recovery path exists	Untracked / unbacked file share

Tier is measured from live configuration, not intent. A bucket with versioning enabled is Tier 1 regardless of the architecture diagram; add a lifecycle rule that purges noncurrent versions after seven days and it becomes Tier 2 with a seven-day window. A database is Tier 1 inside its point-in-time window and Tier 3 outside it. Grading is per loss path, not per resource: the point-in-time window that makes a database Tier 1 against a bad migration disappears with the instance if deletion is allowed to discard the automated backups behind it, so a scanner has to grade "resource deleted" separately from "data corrupted." The opening scene's file share was Tier 4 in practice while everyone believed it was Tier 3, because an include pattern had drifted and no one had audited it since. Tier assignment reflects the scanner's reading of actual retention rules, actual backup scope, and actual lock state, and it is only as current as the last scan.

Controls are then owed in proportion to tier and

blast radius, where blast radius is the scope a loss would touch — dependent services, users, downstream obligations — as defined by a companion guard. At Tier 4, a scratch bucket is fine; write the acceptance down and move on. A Tier 4 resource holding six weeks of training corpus is a finding with an owner and a deadline. The proportionality rule, in one sentence: if a resource's blast radius clears the cap, its accidental-grade tier cannot be below Tier 2, and if its only tier is Tier 3, it must carry a scheduled, proven restore on a cadence matched to how fast its data changes.

Machine learning assets make the failure mode concrete. A training corpus, a feature store, a model registry, and the experiment metadata that makes a result reproducible are all expensive to rebuild, are fed by upstream sources with their own retention clocks,

and are habitually stored wherever inherited a sandbox's protections. The asset with the longest rebuild time and the lowest visibility is, more often than not, sitting nowhere near a dashboard.

V. THE ADVERSARY GRADES THE LADDER TWICE

Place class three against the ladder, and the ladder moves. Every mechanism in Tiers 1–3 is, by default, a configuration item an administrator can change: versioning can be suspended, a soft-delete retention period can be shortened, a lifecycle rule can be added, and backups can be deleted using the same credentials that write the primary data. Every resource therefore carries two grades — accidental and adversarial — where the adversarial grade is the tier the resource still holds once the attacker controls the primary plane's admin account. Unprotected recovery paths can collapse toward Tier 4 under administrator compromise. Fig. 3 illustrates how scoping administrative access can reduce the impact of control-plane compromise.

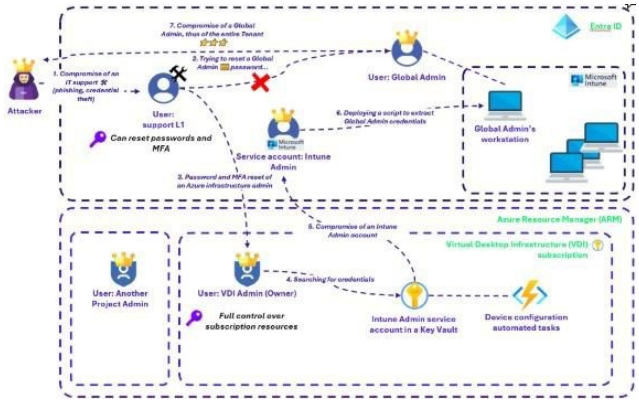


Fig. 3. Enterprise Access Model (1/2): How to scope your Control Plane to secure your Cloud Administration and prevent a global Cloud compromise (RiskInsight)

A. Immutability locks

The first family makes a copy that cannot be deleted, and whose retention cannot be shortened, by anyone. Azure's immutable blob storage provides locked, time-based retention policies [7]; AWS Backup Vault Lock applies the same idea to recovery points [8]; in compliance mode, S3 Object Lock refuses to shorten or remove retention [4]. A lighter cousin, MFA delete, requires a second factor before a version is permanently deleted or versioning is suspended [9], but it is lighter because only the bucket owner's root account can enable it, it is CLI/API-only, and it cannot coexist with a bucket that has lifecycle configuration. The question that separates a real lock from a setting is simple: can the credentials that write

Immutability alone inside the primary account is not sufficient — which motivates the second control

the data also shorten the clock? If yes, the lock is a setting once more.

Every one of these locks has a documented path around it that targets the container rather than the data. Google Bucket Lock automatically applies a project lien when retention is locked; project deletion first requires removing that lien [5]. Closing the AWS account can eventually destroy protected data despite object-level retention, so independent account controls remain necessary [4]. The lock therefore has to travel with a container-level protection
— independent approval for GCP lien removal and separation of AWS account-root closure authority.

family.

B. Backup-plane isolation

The second family keeps at least one recovery copy in a different account, under a different credential domain, connected to the primary plane only by the one-way copy operation itself, as shown in Fig. 4. This is a logical air gap: if the primary plane is compromised in any degree, no credential that lives there can reach the copies. The classic 3-2-1 rule — three copies, two media, one offsite [10] — takes on a new meaning in the cloud, where "offsite" means a different credential domain rather than a different building; cross-region replication inside the same account is geography, and ransomware does not care about geography.

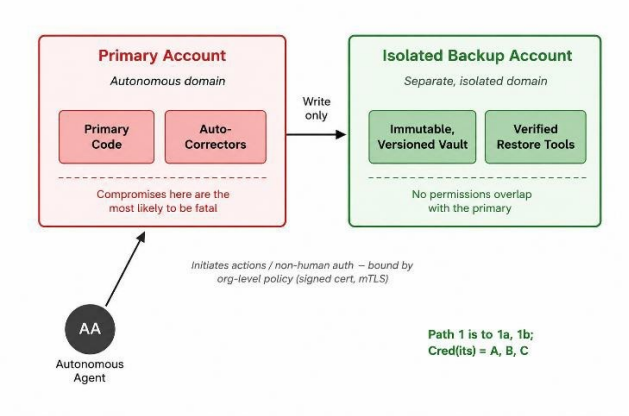


Fig. 4. Backup-plane isolation as a logical air gap. Copying links separate credential domains; closure and project-lien removal require independent authorization.

A resource's adversarial grade is Tier 3 wherever an immutable, isolated, and verified copy exists, because the surviving path is a restore from that copy rather than anything reachable from the primary plane; without such a copy the adversarial grade collapses toward Tier 4. The same mandate applies above the blast-radius cap: require an adversarial grade of Tier 3 or better, and track it as a distinct grade from the accidental one, because the two fail on different days.

TABLE III. Representative Immutability and Isolation Mechanisms by Provider

Control	AWS	Azure	GCP
Object lock / hold	S3 Object Lock [4]	Immutable Blob Storage [7]	Bucket Lock [5]; Object holds [6]
Backup vault lock	AWS Backup Vault Lock [8]	—	—
MFA-gated deletion	MFA Delete [9]	—	—
Deny-by-policy	Service Control Policies [11]	—	IAM Conditions [12]

Control	AWS	Azure	GCP
Container-level protection	Separate root-account approval	—	Project lien

VI. PREVENT, DETECT, RECOVER

The tier model sets how much control a resource is owed; the lifecycle in Fig. 5 sets where those controls live. Three stages, each with concrete mechanisms.

A. *Prevent*

Retention and immutability locks as described above. Deny-by-policy on destructive calls to protected resources: on AWS, a service control policy denying Delete* on tagged datastores and backup vaults, alongside less aggressive destroyers; on GCP, IAM Conditions restrict grants, while Organization Policy governs supported configuration constraints [11]–[13]. Consistent with a companion operations trilogy in this series, each control belongs at a different layer — locks at the storage substrate, deny-policies at the org control plane, approvals at the process layer — so that compromise of one layer does not take out the others.

B. *Detect*

Real-time alerts on destructive API calls recorded in the audit log: deleting a bucket, modifying a backup vault, shortening a retention period, suspending versioning, or deleting many snapshots at once. These events already land in CloudTrail and Cloud Audit Logs [14], [15], but in most shops nobody has wired the specific destructive verbs to a pager. Volume anomalies matter as much as single calls — the ransomware signature in storage telemetry is a burst of high-volume reads followed by overwrites under a new key, or delete rates orders of magnitude above baseline. Managed threat detection now covers some of this for object storage [16], and detection is what makes Tier 2 worth anything: if nobody notices and acts inside a soft-delete window, nothing was actually saved.

C. *Recover*

Recovery is the stage everyone budgets for and the one nobody rehearses. Its mechanisms are version restore, soft-delete undelete, point-in-time restore, and the isolated immutable copy that forms the floor under all of them.

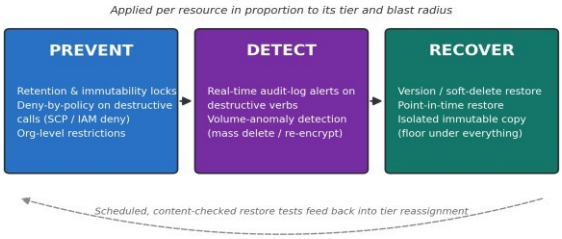


Fig. 5. The prevent-detect-recover control lifecycle. Controls at each stage are applied per resource in proportion to its tier and blast radius, and scheduled restore tests feed back into tier reassignment.

VII. A BACKUP THAT RESTORED CIPHERTEXT

If a backup has not been test-restored, it is an assumption, not a control. The reason this sentence gets repeated to teams is GitLab's public post-mortem from January 2017, which belongs in the canon: after a team accidentally deleted a production data set, they worked through their backup and replication procedures and found that none of the scheduled mechanisms could be relied on during the actual disaster [17]. This is a well-run engineering organization openly publishing the gap between backups that exist and restores that have been tested — the same hole in the wall wearing different clothes as the opening scene of this paper.

A verified-restore discipline is planned, time-boxed, and reviewed for content. It is scheduled — a restore test for every above-cap resource, on a cadence matched to how fast its data changes, with the result recorded somewhere the tier scanner can read, so that

a failed or stale proof automatically demotes the tier. It is scoped — restore the real object to a real target and use it; an exit code alone is not sufficient evidence that the job succeeded. And it is content-checked — row or object counts against an independent ledger, checksums against digests stored outside the backup plane, a sample of records actually decoded. For an ML corpus, the strongest check available is a data-validation suite, or training a small model on the restored sample; a corpus can be correct, complete, and still wrong.

Class three adds one lesson accidents do not teach. A patient ransomware operator dwells long enough to encrypt or corrupt data slowly, aging past the backup rotation before anything alarms. A restore test in the adversarial model therefore has to confirm that the material being restored predates the suspected compromise — that it is plaintext, not encrypted — and that older recovery points are retained long enough to cover a dwell time measured in weeks rather than hours. A green restore of attacker-encrypted data is one of the more painful dashboards an operator can face, and it has happened to real victims.

VIII.THE LADDER, WRITTEN DOWN

Everything above is prose, and prose is where hand-waves hide. The claims are short enough to state exactly. Order the grades so that higher means more recoverable: $r(x) \in \{0,1,2,3\}$ is the measured grade of resource x, with 0 for unrecoverable, 1 for backup-only, 2 for soft-deleted within a window, and 3 for versioned. Let $B(x)$ be the blast radius of x, as defined by a companion guard, and $C_rebuild(x)$ its reconstruction cost from source. The proportionality rule is then

$$r(x) \;\geq\; g(\, B(x),\, C_rebuild(x)\,) \quad (1)$$

where g is non-decreasing in both arguments and r(x) is computed live from the scanner's reading of configuration, never declared from intent. The boundary condition is that g is at least 2 against accident; the adversarial mandate holds the same floor of at least 1 above the blast-radius cap, via a proven restore path the attacker's credentials cannot reach.

A companion piece already sizes the clock at Grade 2, so it is reused rather than re-derived. Let T_detect be the time from deletion to full recovery, including the detection delay, modeled as a random variable, and let R be the retention window. The loss becomes permanent when that combined clock exceeds the purge:

$$Pr(unrecoverable) \;=\; Pr(\, T_detect \;>\; R\,) \quad (2)$$

TABLE IV. Notation Used in the Formal Model (Section VIII)

Symbol	Meaning
r(x)	Measured recoverability grade of resource x, in {0,1,2,3}
B(x)	Blast radius of resource x
C_rebuild(x)	Reconstruction cost of x from source
g(·,·)	Required-grade function, non-decreasing in both arguments
T_detect	Time from deletion to full recovery, incl. detection delay
R	Retention window before purge
P_i	The i-th recovery path available for resource x
Cred(P_i)	Set of credentials able to destroy path P_i
A	Credential / escalation set reachable by the attacker
Symbol	**Meaning**

V_i	Time of the last verified end-to-end restore of path P_i
Δ	Data change rate / staleness bound for a verified copy
τ_i	Collection time of the verified copy on path P_i
t_c	Earliest suspected time of compromise

R is sized from a high quantile of T_detect under a standard right-censored failure-time model [18]; the formula is unchanged, only who is doing the deleting has changed.

The adversarial grade is a statement about credentials, not about copies. Let P_1,…,P_k — noncurrent versions, snapshots, the isolated copy — be the k recovery paths for x. Let Cred(P_i) be the credentials able to destroy path i, and let A be the set of credentials or escalations reachable from the attacker's initial foothold, closed under every escalation the environment allows. Recovery resists the attacker exactly when at least one path's destroying credentials fall outside that reachable set:

$$\exists\, i \in \{1, \dots, k\} : \mathit{Cred}(P_i) \cap A = \varnothing (3)$$

— one reverse gear the attacker cannot reach. The correlated-failure argument behind this — that independence of paths only holds when the individual failure probabilities are themselves uncorrelated — is derived in a companion Separation of Duties piece rather than repeated here. Same-account copies and cross-region replicas add path count without adding independence; an extra copy in a genuinely separate credential domain is what empties the intersection in the survival condition above and restores independence.

A retention lock is a predicate the substrate enforces, which is what distinguishes it from a setting. Let u(t) be the time until which an object locked at time t is guaranteed to be retained. For every sequence of data-plane and retention API calls, by any principal including the root account [4], [5], [7], [8]:

$$t_1 \le t_2 < u(t_1) \Rightarrow u(t_2) \ge u(t_1) (4)$$

Retention can be extended before it expires; it cannot be shortened, no matter who the caller is. Whether that implication holds in an actual configuration, rather than only on the architecture diagram, is exactly the differentiating question raised earlier. The implication is a property of the object itself
— it does not delete the project or close the account containing the locked copy; the lock, like a lien or a deny policy, travels with the object.

Finally, the freshness precondition. A recovery path counts toward k only if it has been recently and verifiably proven, and the verified copy itself predates the trouble. Let V_i be the last verified end-to-end restore time of path i, Δ the allowed interval between restore checks, τ_i the collection time of the verified copy, and t_c the earliest suspected compromise time. At decision time t:

$$\mathit{counts}(P_i, t) \Leftrightarrow t - V_i \le \Delta \text{ and } \tau_i < t_c (5)$$

The second clause is the one a patient attacker works to defeat, by letting the corruption age long enough that every τ_i rotates past t_c — the formal definition of green restores of attacker-encrypted data. Δ and retention depth remain policy decisions per resource class rather than fixed constants here; the real values depend on an operator's own detection latency and its feeds' clocks, and turning T_detect quantiles and restore-verification cadences from a production estate into published numbers is left as future work.

IX. WHERE IT BREAKS

The edges are sharp. A defense dressed in a suit and tie is the most vulnerable kind.

Locks are immutable by design. A mis-sized retention period means living with the storage cost for years, and a locked WORM copy can land on the wrong side of a right-to-erasure obligation. A companion deletion-gate piece flagged the same interaction and deferred it to counsel; this paper does the same rather than pretend the tension resolves cleanly. Scope the smallest region that covers the

recovery need, and have counsel review any scope

that contains personal data.

Tier assignments rot. Every lifecycle rule, retention change, or backup-scope edit can silently demote a resource, so classification has to be recomputed continuously from configuration rather than trusted from memory — and the scanner's own output is an attack target, since an adversary who can edit the inventory can hide a demotion. The scanner needs its own credentials and its own plane, or it is just another setting.

Isolation carries running friction. Cross-plane restores take longer, a break-glass path has to exist into the backup plane for genuine emergencies, and that break-glass path — along with any insider holding legitimate backup-plane credentials — is itself an attack surface. Separation of duties applies at that layer exactly as much as it applies at the primary layer.

Detection lives on base rates. A legitimate lifecycle cleanup looks like a mass delete; a routine re-encryption migration looks like ransomware. Destructive-verb alerts only earn their keep once someone tunes them against the rhythm of a specific account, which is unglamorous work with no demo to show for it.

X. THE CORPUS QUESTION

Every team that asks for a starting point gets the same one, doable in a week.

Pick the one dataset that would take longest to rebuild if lost. In an honest ML shop that estimate is usually measured in weeks, and the training set is typically the answer. Grade its tier from current configuration — actual retention rules, actual backup scope, actual lock state. Then regrade it without the defender's assumptions: which of those protections survive if today's platform admin credentials are compromised? Finally, run one restore end to end and compare the result against something that never touched the backup plane.

In practice the corpus tends to land about a tier below where the team believed it sat, and the adversarial grade tends to land at the bottom of the stack. That gap is the finding. The teardown ticket in the opening scene was reviewed, approved, and executed correctly — exactly like the next ticket in the queue. When the belief is wrong, the approved property is the wrong one. The fix is to purchase the actual property, and prove it on a schedule, in writing.

REFERENCES

[1] *Ars Technica*, "AWS console breach leads to demise of service with 'proven' backup plan," Jun. 18, 2014. [Online]. Available: https://arstechnica.com/information-technology/2014/06/aws-console-breach-leads-to-demise-of-service-with-proven-backup-plan/

[2] Cybersecurity and Infrastructure Security Agency (CISA) and Multi-State Information Sharing and Analysis Center (MS-ISAC), "Ransomware Guide," Sept. 2020. [Online]. Available: https://its.ny.gov/cisa-ransomware-guide

[3] R. Chandramouli and D. Pinhas, *Security Guidelines for Storage Infrastructure*, NIST Special Publication 800-209. Gaithersburg, MD, USA: National Institute of Standards and Technology, Oct. 2020. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-209

[4] Amazon Web Services, "Locking objects with Object Lock," *Amazon Simple Storage Service User Guide*. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/object-lock.html

[5] Google Cloud, "Use and lock retention policies," *Cloud Storage Documentation*. [Online]. Available: https://cloud.google.com/storage/docs/using-bucket-lock

[6] Google Cloud, "Object holds," Cloud Storage Documentation. [Online]. Available: https://cloud.google.com/storage/docs/object-holds

[7] Microsoft, "Store business-critical blob data with immutable storage," *Azure Storage Documentation*. [Online]. Available: https://learn.microsoft.com/en-us/azure/storage/blobs/immutable-storage-overview

[8] Amazon Web Services, "Backup Vault Lock," *AWS Backup Developer Guide*. [Online]. Available: https://docs.aws.amazon.com/aws-backup/latest/devguide/vault-lock.html

[9] Amazon Web Services, "Configuring MFA delete," *Amazon Simple Storage Service User Guide*. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/MultiFactorAuthenticationDelete.html

[10] P. Krogh, *The DAM Book: Digital Asset Management for Photographers*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2009.

[11] Amazon Web Services, "Service control policies (SCPs)," *AWS Organizations User Guide*. [Online]. Available: https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_policies_scps.html

[12] *Google Cloud, "IAM Conditions overview," Identity and Access Management Documentation. [Online]. Available: https://cloud.google.com/iam/docs/conditions-overview*

[13] Google Cloud, "Organization Policy overview," Google Cloud Documentation. [Online]. Available: https://docs.cloud.google.com/organization-policy/overview

[14] Amazon Web Services, "Logging Amazon S3 API calls using AWS CloudTrail," *Amazon Simple Storage Service User Guide*. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/cloudtrail-logging.html

[15] Google Cloud, "Cloud Audit Logs overview," *Cloud Logging Documentation*. [Online]. Available: https://cloud.google.com/logging/docs/audit

[16] Amazon Web Services, "GuardDuty S3 Protection finding types," *Amazon GuardDuty User Guide*. [Online]. Available: https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_finding-types-s3.html

[17] GitLab Inc., "Postmortem of database outage of January 31," *GitLab Blog*, Feb. 10, 2017. [Online]. Available: https://about.gitlab.com/blog/postmortem-of-database-outage-of-january-31/

[18] J. D. Kalbfleisch and R. L. Prentice, *The Statistical Analysis of Failure Time Data*, 2nd ed. Hoboken, NJ, USA: Wiley, 2002.