

When Revocation Does Not End Access

Temporal Repair Dependencies in Application Compromise

Zawwar Sami

Independent Researcher, Systems Engineer, and Cybersecurity Specialist

ORCID: 0009-0004-5819-3017

DOI: [10.5281/zenodo.23435843](https://doi.org/10.5281/zenodo.23435843)

17 October 2026 | Version 1 | Research preprint

Abstract

What happens when the old credentials stop working, but the application can still create new ones? I investigate this recovery failure through a small running system: an issuer, a protected resource and a separate observer. Across 48 distinct repair-and-renewal schedules, 20 retain access even though every original credential is rejected. Each schedule is repeated three times. The failure depends on the order in which authority is removed and its source is stopped. I express that dependency as a renewal graph and prove a conditional repair-order rule, checked against 98,697 graph-order pairs. A second failure survives an otherwise effective provisioning barrier: work accepted before containment creates new credentials after revocation. An executed walkthrough compares leaving that work pending, cancelling it and rejecting it at commit using an issuance epoch. The last two prevent reopening in the tested sequence; a positive control confirms that new work can succeed after service resumes. Equifax, Exchange and MOVEit establish the need to examine application, identity and data boundaries separately. The contribution is a reproducible recovery test that follows replacement credentials and pending issuance through to the protected operation, with explicit conditions for what its result establishes.

Keywords: incident recovery; temporal repair dependency; revocation; authority renewal; experimental security; application compromise.

1. The old credential fails. What can still replace it?

Suppose a compromised application can create an account and a session at a protected service. A responder restores the application, disables the account and invalidates the session. Every action succeeds. Is access closed?

The answer depends on what happened between those actions. If the application recreated an account after its removal, the final system can still admit that account even after the application itself is restored. A successful revocation and a successful recovery are different claims. The first concerns the authority removed; the second must cover the authority that could replace it.

This distinction is easy to state and surprisingly easy to test badly. In the experiment reported here, an observer using only the original credentials sees rejection in every completed trial. In 20 of the 48 distinct schedules, another credential issued during the repair still retrieves the protected canary. The original-credential test is correct about its inputs and inadequate as a global closure test.

I start with the operation I want to stop: reading a protected object. Then I ask which credentials permit that read, what can create another credential, and what a proposed repair actually changes. That keeps the investigation attached to something observable. A patch, a restored process and a rejected token each answer a different question.

The experiments below turn those questions into running code. The historical cases explain why these boundaries deserve attention; their published records do not establish that the experimental sequence occurred at the affected organizations.

1.1 A running case: access returns after all three repairs

Consider a transfer service with a provisioning component and a separate data store. The component creates an application principal and a session. Restoring the component stops it issuing more credentials, but the data service retains credentials already issued. This is the architecture of the local experiment, not a reconstruction of a particular victim.

I run the following sequence against two HTTP processes. First, issue the original credentials. Second, accept a request to issue another pair, but leave that request in a queue. Third, block new provisioning requests, restore the issuer, delete the principals and invalidate the sessions. The observer now receives HTTP 401 for both original credentials. Finally, let the already accepted request commit. It creates a replacement principal and session, and the protected read succeeds.

The decisive event is the last one. No new request crosses the barrier. Work already inside it supplies the replacement authority. The queue is therefore part of the recovery boundary, even though it is neither an account nor a session.

```
$ python3 temporal_repair/walkthrough.py
SCENARIO admission-only
Initial issuance: 201; queued issuance: 202 accepted
After repair: original principal=401; original session=401
Drain accepted job: HTTP 201
Readback: open

SCENARIO cancel-pending
Initial issuance: 201; queued issuance: 202 accepted
After repair: original principal=401; original session=401
Drain accepted job: HTTP 409
Readback: closed-for-supplied-credentials

SCENARIO epoch-fence
Initial issuance: 201; queued issuance: 202 accepted
After repair: original principal=401; original session=401
Drain accepted job: HTTP 409
Readback: closed-for-supplied-credentials

POSITIVE CONTROL after reopening
New-epoch queue: 202; drain: 201; protected read: 200
```

Listing 1. Actual standard output from the supplied walkthrough, with blank spacing normalized and the invocation shown. It ran on 16 October 2026. The three treatments and reopening control are additional demonstrations, separate from the 171 systematic service trials reported below. HTTP 409 alone does not establish recovery; the separate protected read supplies the scoped result.

The two successful treatments act at different points. Cancellation removes accepted work before the final revocations. The epoch fence leaves the old job present but rejects its attempt to commit. After reopening, a newly accepted job carries the current epoch and succeeds. This positive control matters: a permanently broken service could also deny the old job.

The failure is small enough to inspect. The stronger question is whether the same reasoning survives other repair orders and renewal opportunities. Sections 2-4 provide the model and systematic tests; Section 5 shows the implementation and how to read its evidence.

2. A model of renewable authority

2.1 State, renewal and repair

An authority state is any represented capability that can enable a protected operation or create another represented capability. Examples include an application identity with provisioning rights, a usable account, an application session, or an accepted provisioning job. An account name alone is not an authority state unless its effective ability is specified.

Let $G = (V, E)$ be a loop-free directed renewal graph; self-edges add no reactivation transition after their sole source is cleared and are omitted. An edge $u \rightarrow v$ means that, while u is active, a permitted renewal transition can activate v . It does not mean that every real system renews immediately. Let $x(v)$ be a Boolean indicating whether authority v is active. A repair $\text{clear}(v)$ sets $x(v)$ to zero. An enabled renewal may subsequently set it to one again. A schedule contains repair actions interleaved with zero or more permitted renewals.

The basic problem has four explicit assumptions: every modeled authority is initially active; each is cleared exactly once; clearing is effective but does not permanently immunize the target against renewal; and no unrepresented source, pending job or external transition creates authority. The objective is removal of every authority in this graph. A narrower objective, such as denying one database operation, requires its relevant dependency subgraph and can admit a cheaper intervention.

A repair order is robust in this model if all modeled authority is absent at completion for every permitted interleaving. This is stronger than observing one quiet execution. An idle adversary can make a poor order look successful.

2.2 Conditional scheduling proposition

Proposition 1. Under the assumptions above, a once-per-node repair order is robust if and only if $\text{clear}(u)$ precedes $\text{clear}(v)$ for every renewal edge $u \rightarrow v$. Equivalently, the repair order must be a topological order of G .

Sufficiency. Proceed through a topological order. When a node is cleared, every predecessor that could recreate it has already been cleared. By induction, those predecessors cannot themselves be recreated: every incoming source of each predecessor occurred still earlier. Thus no subsequent permitted renewal can reactivate any cleared node. After the last repair, all nodes are absent.

Necessity. Suppose an edge $u \rightarrow v$ violates the order, so v is cleared before u . Choose a permitted adversary schedule that performs no renewal until immediately after $\text{clear}(v)$. At that point u is still active because it has not yet been cleared. It can recreate v . Since $\text{clear}(v)$ occurs only once, v remains active at completion. Therefore the order is not robust.

The proposition is an elementary scheduling consequence of the stated transition system. The contribution is its explicit use as a repair-contract test with falsifiable premises, not priority for topological sorting, dependency analysis or sequential incident response. It also says nothing about which business interruption is acceptable.

A directed cycle has no such once-per-node robust order. In this model, it needs an additional intervention: disable enough renewal edges to break the cycle, clear an appropriate group atomically, or use another mechanism that changes the assumptions. Repeated clearing may work

under a favorable schedule, but it does not provide this worst-case guarantee. A containment action is useful because of the transitions it actually blocks, rather than its label.

2.3 Why pending work belongs in the graph

Consider a provisioning request accepted before containment and committed afterward. Once accepted, the job can act as a new source q of authority. Blocking ingress to its original issuer does not clear q . The graph that omits q can recommend an order that is internally correct and operationally incomplete.

This yields two different clocks: the time new issuance becomes impossible and the time already accepted issuance becomes unable to commit. A durable revocation must cover both. In an implementation with a true cancellation or commit-time authorization fence, the second boundary may coincide with the first. In an implementation that authorizes only on queue admission, it need not.

For each authority plane v , let $c(v)$ be the last effective creation or reactivation before the observation horizon, and $r(v)$ the last effective revocation covering it. In a complete, immediate, correctly ordered event trace, $r(v)$ later than $c(v)$ is required for that plane to be absent at the horizon. Completion acknowledgements are not substitutes for those effective events. Concurrent or uncertain timestamps require an unresolved result unless an ordering mechanism, transaction version or conservative interval establishes the relation.

3. Executable design and observation boundary

3.1 Two services and a separate observer

The artifact uses Python 3.12.14 and SQLite on Linux. All network services bind to loopback. The issuer process represents a stipulated compromised provisioning component. It can ask the resource process to create a random principal credential and an opaque bearer session. Either credential can retrieve a random synthetic canary from the resource's `/data` endpoint. The resource persists the two credential classes in separate tables.

The controller invokes three repairs: R restores the issuer by disabling its renewal function; P deletes principal credentials at the resource; S invalidates bearer sessions at the resource. Issuer restoration does not erase resource-side state. Principal deletion does not invalidate independent sessions. Those semantics are explicit fixture choices, not claims about a particular vendor's identity model.

The observer is launched as a separate operating-system process for each observation. It imports neither service nor controller code, reads no database or internal state, and requests the canary over HTTP using the credentials issued during that trial. The runner checks that its process identity differs from the controller and services. A successful read provides a digest of the returned canary. HTTP 401 means rejection for that supplied credential; transport failure or an unexpected response means unresolved.

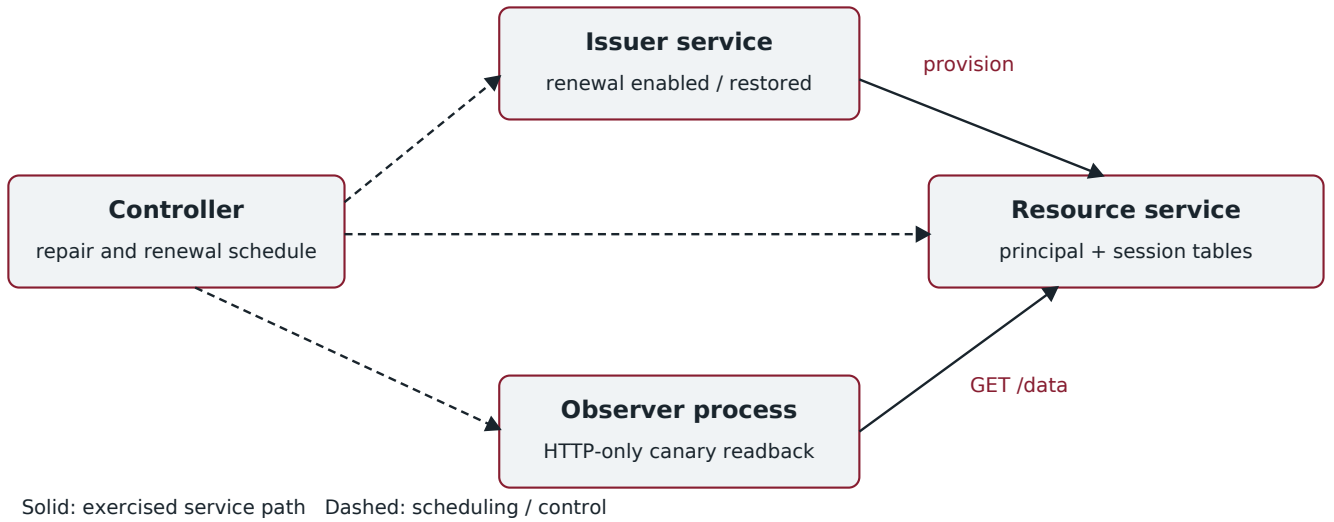


Figure 1. Executed testbed and trust boundary. The observer exercises the protected interface rather than trusting a repair acknowledgement. Process separation removes shared in-memory state and shared repair helpers from readback; it does not create independently administered or adversarially isolated evidence collection.

These processes share a host, administrator and author. A compromised resource could fabricate replies. The test therefore assumes the resource's enforcement and the host are trusted. It measures the stated service semantics, not forensic resistance to a malicious operating system.

3.2 Intervention schedules and controls

There are six permutations of R, P and S. For each order I execute all eight three-bit renewal masks. A mask bit determines whether the issuer is asked to renew immediately after the corresponding repair. A restored issuer rejects renewal. Thus 48 distinct schedules cover quiet execution, renewal in selected gaps and renewal after every step. Each schedule is repeated three times, producing 144 base trials. The repeats check execution consistency; they are not 144 independent incidents.

A synchronous containment variant first blocks provisioning at the resource, then executes each of the six repair orders with all renewal attempts enabled. It is repeated three times, adding 18 trials. Two negative controls test a success acknowledgement that performs no descendant repair and a readback endpoint that is unavailable.

A delayed variant adds a one-item provisioning queue. Queue admission checks the provisioning barrier; draining an accepted item does not check it again. That models a declared admission-time authorization policy. The experiment compares three sequences, each repeated three times: drain after revocation; drain before final revocation; cancel pending work before final revocation. This adds nine trials and tests a different premise from synchronous renewal.

The protocol is deterministic about action boundaries rather than dependent on arbitrary wall-clock sleeps. It tests causally defined interleavings, not production latency, race probability or throughput. Raw JSON retains repair order, attempted renewal, HTTP result, credential fingerprints, canary digest and observer process identity. Raw synthetic credential values are not retained in result files.

3.3 Exhaustive graph check

A separate implementation enumerates every loop-free directed graph on one through four labeled nodes and every node-repair permutation. After each clear it computes the closure of all enabled renewal edges, representing an allowed saturated adversary schedule. It compares the resulting state with the proposition's topological-order prediction. No HTTP code is imported into this model check.

Nodes	Directed graphs	Graph-order pairs	Closed pairs	Prediction mismatches
1	1	1	1	0
2	4	8	4	0
3	64	384	48	0
4	4,096	98,304	1,536	0
Total	4,165	98,697	1,589	0

Table 1. Bounded exhaustive implementation check. The proof covers arbitrary finite graphs under its premises; enumeration tests that the supplied transition implementation agrees on this finite domain. These counts describe constructed graphs, not a distribution of enterprise architectures.

4. Observed results and failure diagnosis

4.1 Repair order changes the effective final state

Across the 48 unique base schedules, 20 end with a credential that still retrieves the canary. The other 28 reject all credentials issued in their trial. Every original credential is rejected in every final state. All three repetitions agree, giving 60 open states among 144 base runs, with zero disagreements between the declared transition prediction and observed access.

Repair order	Open schedules out of eight	What can survive
R -> P -> S	0	No later renewal source
R -> S -> P	0	No later renewal source
P -> R -> S	4	Principal recreated after its deletion
P -> S -> R	6	Principal or session recreated before issuer restoration
S -> R -> P	4	Session recreated after its invalidation
S -> P -> R	6	Session or principal recreated before issuer restoration

Table 2. Unique schedules, before repetition. Renewal masks are experimental choices; the fractions are not operational failure probabilities.

	000	001	010	011	100	101	110	111
R > P > S	CLOSED	CLOSED	CLOSED	CLOSED	CLOSED	CLOSED	CLOSED	CLOSED
R > S > P	CLOSED	CLOSED	CLOSED	CLOSED	CLOSED	CLOSED	CLOSED	CLOSED
P > R > S	CLOSED	CLOSED	CLOSED	CLOSED	OPEN	OPEN	OPEN	OPEN
P > S > R	CLOSED	CLOSED	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
S > R > P	CLOSED	CLOSED	CLOSED	CLOSED	OPEN	OPEN	OPEN	OPEN
S > P > R	CLOSED	CLOSED	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN

R: restore issuer P: delete principals S: invalidate sessions

Mask bits request renewal after repair positions 1, 2 and 3.

Figure 2. Observed final access across all 48 distinct schedules. A mask bit enables one renewal attempt after the corresponding repair position. Crimson means that some issued credential still retrieves the canary; pale means that every supplied issued credential is rejected. Results repeat identically across three executions.

One trace makes the distinction concrete. In order $P \rightarrow R \rightarrow S$ with mask 100, P deletes the original principal, the active issuer creates a new principal and session, R stops further renewal, and S deletes every session. The original principal and both sessions now fail. The replacement principal succeeds. None of the three repairs lied about its local effect. The error would be treating their acknowledgements as proof of a stronger final property.

An original-credential-only observer reports rejection throughout the final checks. If that scoped fact is interpreted as global closure, it produces 20 false closures across the 48 unique schedules. An observer that probes every credential generation issued in this fixture detects all 20 open states. This comparison is between two declared observation policies, not between the proposed method and a professional incident-response product.

4.2 The barrier works until accepted work is omitted

The synchronous provisioning barrier closes all six repair orders in every repetition: 18 runs reject all issued credentials. This is a useful ablation. Once the resource blocks creation, the order constraint imposed by that creation path disappears. It is not necessary to insist on R first when another effective intervention has already disabled the relevant renewal edge.

The delayed-work test shows why the adjective effective matters. A queued job is admitted before the barrier. The barrier then blocks new requests, the issuer is restored, and both descendant classes are revoked. At that point every known credential fails. Draining the accepted job creates a new usable principal and session. Access reopens in all three late-drain runs.

Queue treatment	Final access in three repetitions	Diagnostic meaning
Drain after the barrier and descendant revocations	Open in all three	Admission was blocked; accepted work still committed
Drain before the final descendant revocations	Closed in all three	Final removals covered the last committed generation
Cancel pending work before the final revocations	Closed in all three	The additional renewal source was removed

Table 3. Controlled pending-work comparison. The queue's admission-time policy is stipulated and visible in source. The finding is a counterexample to an incomplete barrier contract, not evidence that every asynchronous product behaves this way.

A wait of an arbitrary length is not the same as evidence that a queue is drained. When completion cannot be bounded or observed, the honest result is unresolved. A commit-time authorization check, a transaction fence, or cancellation can change the model; its actual semantics must be tested.

4.3 Negative controls and what they establish

When the resource acknowledges descendant repairs without deleting their rows, protected reads remain successful and the observer reports open. When the observer cannot reach its target, it reports unresolved. Neither case is silently converted to closure. These controls test whether the artifact is reading effective outcomes rather than merely repeating successful action labels.

The canary has already been retrieved before repair, so every experiment also has a historical disclosure. Rejecting all future reads does not erase that earlier copy. The closure claim concerns future access to a stated operation and horizon; it cannot repair an event that has already occurred.

5. A generation-aware repair contract

I use a repair contract to state what must be true before access can be called closed. Each protected operation needs a declared boundary, current authority inventory, renewal dependencies, repair footprint, pending-work policy and observation horizon. A credential hash or opaque generation identifier can connect issuance and rejection evidence without recording live secrets.

Contract obligation	Evidence needed	Failure that invalidates closure
Bound the objective	Protected endpoint, data class and acceptance rule	A clean file check is substituted for a data-access test
Bound replacement authority	Effective issuers and incoming renewal paths	A secondary issuer or replica remains outside scope
Establish an issuance barrier	Effective denial or restored source, with version/order evidence	An acknowledgement is treated as immediate enforcement
Resolve accepted work	Cancelled jobs, drained queue or commit-time fence	A previously accepted job can still create authority
Remove descendants	Effective principal/session removal after the relevant last creation	An older generation is removed while a newer one remains
Test the protected operation	Readback using affected generations and relevant paths	Only the original credential is tested
State uncertainty	Observer trust, time bounds and uncovered planes	Unavailable evidence is interpreted as absence

Table 4. Repair contract derived from the experiments. Supporting records can come from existing response systems; the contract does not require a new orchestration platform.

5.1 A precise closure claim

A useful closure statement is: within boundary B and horizon H, no authority covered by inventory K can perform protected operation O; each represented renewal path is disabled or its output is covered by a later effective repair; and accepted work that could recreate K has been resolved. Evidence identifies the relevant versions, ordering and observer coverage.

This statement tells the next investigator exactly what was tested and what remains outside the claim. If the inventory is incomplete, the result is conditional. If an observer shows a working route, closure is contradicted regardless of other missing evidence.

For a complete fixed-scope model with sound repair and renewal semantics, a closed verdict is justified when every state compatible with the admitted evidence excludes O throughout H. If compatible states differ on reachability, the verdict is unresolved. This conservative reasoning is established practice in verification; the specific contribution here is the replacement-authority and pending-work contract it is used to check.

5.2 Code and tool walkthrough

The implementation separates enforcement, scheduling and observation. `resource_service.py` enforces protected access and stores credentials; `issuer_service.py` requests new authority; `observer.py` reads the protected interface; `run_experiment.py` controls schedules and preserves results; `check_graphs.py` independently checks the graph model. No private sourcebook or personal repository is required to reproduce them.

```
python3 temporal_repair/check_graphs.py
python3 temporal_repair/run_experiment.py
```

The first command writes `graph_results.json`. The second writes `http_results.json`, `http_summary.json` and `http_trials.csv` beneath `temporal_repair/results`. The README describes the response meanings and the source locations behind every result table. Each run creates temporary local service state and terminates its own services afterward.

```
# Observer: verdict concerns the supplied credential inventory.
if any(r['status'] == 200 for r in rows):
    verdict = 'open'
elif not rows or any(r['status'] != 401 for r in rows):
    verdict = 'unresolved'
else:
    verdict = 'closed-for-supplied-credentials'
```

Listing 2. Readable extraction of the executed observer decision in `observer.py`. A scoped rejection verdict is deliberately not named global closure. A server error or missing inventory cannot satisfy it.

```
# Independent outcome prediction for the synchronous fixture.
last_creation = max(
    [i for i, t in enumerate(trace)
     if t['renewal_status'] == 201], default=-1)
predicted_open = any(
    order.index(a) <= last_creation
    for a in ('disable-principals', 'invalidate-sessions'))
```

Listing 3. The runner compares this generation-order prediction with external HTTP readback; the provisioning-barrier variant separately predicts no further creation. The less-than-or-equal relation matters: the scheduled creation occurs after the repair at the same action index. This predicate assumes the immediate synchronous semantics and does not certify the delayed queue variant.

An investigator can inspect the raw $P \rightarrow R \rightarrow S$, mask 100 trace, verify the generation fingerprints, compare HTTP 401 with HTTP 200, then disable the renewal attempt and rerun. This is the smallest experiment that separates failure of the removal operation from successful removal followed by reacquisition.

5.3 The code that admits the job, and the code that stops it

The resource uses Python's HTTP server and SQLite. There is no exploitation framework behind the result: the issuer's ability to provision is an explicit premise. `urllib.request` exercises the HTTP boundary; `subprocess` launches the separate observer; SQLite stores the two credential classes. These tools serve different purposes. A successful database deletion checks a local effect. Only the protected HTTP read checks whether a supplied credential still works.

The earlier admission-only implementation counted pending jobs. The revised implementation records the issuance epoch at admission. Ordinary isolation still blocks new admissions without changing that epoch, preserving the original failure. The additional `/fence` action advances the epoch and blocks admissions. The worker checks the recorded epoch immediately before inserting replacement credentials.

```
# /queue, after the admission barrier check:
state['pending'].append(state['epoch'])

# /fence:
state['epoch'] += 1
state['isolated'] = True

# /drain, after checking that pending work exists:
accepted_epoch = state['pending'].pop(0)
if accepted_epoch != state['epoch']:
    return self.reply(409, {'error': 'stale issuance epoch'})
```

Listing 4. Excerpts from the executed `resource_service.py`. Credential insertion follows the drain check. Advancing the epoch does not revoke existing credentials; the principal and session repairs remain necessary.

Here the server handles one request at a time, so another request cannot move the fence between the check and insertion. A concurrent implementation needs that guarantee explicitly: compare and commit under an appropriate transaction or serialization mechanism, keep the epoch durable, and prevent an older worker from writing after the fence. Copying this conditional into a distributed worker without those properties would not reproduce the guarantee. Chubby already describes generation-bearing sequencers checked by the receiving server to reject invalid requests [28, Section 2.4]. The local epoch check applies that established principle to the accepted issuance job; this experiment tests its role in the stated recovery sequence.

The reopening control uses `/resume` to admit new work in the current epoch. It then resets the synthetic issuer, accepts a new job and reads the canary using the resulting credentials. This exercises service restoration as well as rejection. It does not establish availability under load, crash recovery or multiple workers.

5.4 Follow one failure through the raw records

Run `python3 temporal_repair/walkthrough.py` for the narrated sequence. Run `python3 temporal_repair/run_experiment.py` for all schedules, then `python3 temporal_repair/check_graphs.py` for the separate finite graph check. The walkthrough writes a transcript and JSON observations under `temporal_repair/results`. The systematic runner writes `http_results.json` and `http_trials.csv`.

In `http_results.json`, select repetition 0, order `disable-principals`, `restore-issuer`, `invalidate-sessions`, and `mask [1, 0, 0]`. The first trace entry acknowledges principal deletion and records renewal status 201. Its observer probes include both credential generations. The final observation rejects the original principal and both sessions but accepts the replacement principal. The `old_handles` field records rejection for the old pair alone. Comparing that field with `final` exposes the missed access route without interpreting a dashboard or trusting a repair label.

A credential fingerprint ties a successful probe to an issued generation without storing its usable secret. A canary digest ties that success to the protected object. Neither proves that an uncontrolled environment has no other credential: the fixture's complete issuance inventory is doing real work in that conclusion. In a field investigation, an unknown issuer or missing issuance log is an evidence gap to resolve.

6. What the three incidents contribute

The historical cases motivate different state planes and diagnostic branches. They do not establish the frequency of renewal races, prove that these particular repair orders occurred, or validate the testbed as an Exchange or MOVEit emulator. The comparison uses primary documentary and investigator reports, with researcher reconstructions identified separately. Extended stage tables and counterfactual worksheets are included in the public artifact. The Accellion advisory supplies a supplementary disclosure comparison [15], without adding a fourth primary case.

6.1 Equifax: the application was not the whole authority boundary

The House investigation describes two web servers and two application servers in the ACIS environment, accessible application credentials and database reach beyond the service's requirement [16, pp. 31-33, 72-79]. GAO distinguishes early command execution from the later theft incident and reports access to 48 unrelated databases beyond the three required for ACIS [1]. Apache's S2-045 advisory identifies the multipart error-handling execution weakness [17].

The diagnostic path is request interpretation, application execution, credential availability, database authorization, data retrieval and transfer. These are different boundaries. A firewall can correctly permit traffic to the service while the permitted application path becomes harmful. Database authorization can correctly accept a credential whose filesystem exposure should never have occurred. Calling the whole sequence a patch failure loses the repair footprints.

A practical investigation works backward from the protected operation. An unusual database query calls for the effective principal, source context and session history. A suspicious server-side file calls for preservation and execution evidence; finding a file does not by itself prove execution occurred. The House report's web-shell and monitoring observations supply historical context, not a complete command transcript [16].

A useful counterfactual is narrowly testable: preserve required dispute processing while denying the application access to unrelated databases. It needs both security and business regression checks. Reducing stored sensitive data changes the possible consequence, while traffic inspection changes observability; neither automatically substitutes for authorization scope. The House and GAO accounts differ on the duration of an expired inspection certificate. Their shared finding is the inspection gap; raw certificate evidence was not acquired here [1,16, p. 89 n. 554].

For the present model, Equifax motivates distinguishing application and downstream data authority. It does not document that a repaired application recreated a revoked database identity. That additional temporal mechanism remains an experimental question rather than a historical attribution.

6.2 Exchange: direct access and persistence are separate branches

Microsoft and Volexity describe on-premises Exchange exploitation involving server-side request forgery, mailbox access, web-shell placement and subsequent operations, with different combinations across investigated victims [2-4]. Orange Tsai's vulnerability research explains frontend/backend trust and a researcher-developed chain [18]. That reconstruction must not be relabeled as the exact path taken at every incident.

The important architectural feature is that an exposed service legitimately possesses routes to protected mailbox operations. One branch can use that trust to read data directly. Another can create durable execution and support later access. A universal sequence requiring a web shell before every disclosure is therefore wrong. The absence of a discovered shell does not exclude a direct-data branch.

The tool names in reporting have a functional role. A server-side web shell supports repeated execution. Procdump supports memory acquisition; Exchange PowerShell supports mailbox-related operations; archive utilities prepare collections [2]. A tool list alone does not establish which account ran it, what it read or whether it succeeded. Diagnosis needs process context, mailbox access records, filesystem evidence and credential scope appropriate to the claim.

Patching closes a vulnerable transition. Hunting examines already produced state. Restoring an affected server addresses one execution plane. Investigating accounts and downstream systems addresses a different footprint [3]. In the repair contract, those must appear as separate obligations. No experiment here deploys Exchange, validates a particular cumulative update or measures those controls against an active adversary.

6.3 MOVEit: rebuilding one tier does not erase another tier's state

Mandiant reports the MOVEit campaign and the LEMURLOOT web shell's interaction with application data and identity state [5]. The joint CISA/FBI advisory explicitly warns that database accounts can persist after rebuilding the web server [6]. That statement provides a direct historical restoration-boundary example: a web-tier rebuild need not remove database-side authority.

Horizon3's patch-based vulnerability reconstruction is useful for understanding input handling, session state and the route to dangerous database operations. Its authors distinguish the reconstructed path from certainty about the adversary's exact route [19]. Progress's June statements document patch response and subsequent review; a later distinct vulnerability is not, by itself, proof that the original fix failed [20,21].

The functional architecture is a public transfer interface attached to valuable files, metadata and application identities. The system has to read and deliver data to perform its legitimate role. The diagnostic issue is which authorization context separates an ordinary transfer from an unauthorized request, and which state an intruder can alter after crossing that boundary.

A minimal restoration test records the relevant database identities, rebuilds the web tier in an authorized isolated environment, and checks those identities at the database or application enforcement boundary. A clean web directory cannot answer that question. The present HTTP fixture reproduces the logical separation between issuer restoration and resource-side credentials; it does not reproduce SQL injection, LEMURLOOT, or MOVEit's authentication semantics.

Across all three cases, the useful question is which state and transition a proposed action actually changes. The temporal experiment adds a second question: which other state can recreate what was just removed? A case can establish the need for that question without proving every answer in the model.

7. Related work and the boundary of the contribution

Saltzer and Schroeder's complete-mediation discussion already warns about remembered authorization and recovery [7]. Attack graphs and MulVAL represent chains involving vulnerabilities, identities, connectivity and privileges [9,10]. NIST incident-response guidance and Microsoft's responder workflow already address persistence, compromised accounts and recovery scope [3,8]. The present work does not introduce those concepts.

Retro and WARP use recorded dependencies and selective re-execution to repair the consequences of intrusion [22,23]. WARP distinguishes integrity recovery from disclosure that cannot be undone. MOBAL explicitly plans sequential incident response under model misspecification [12, Sections

5-6]. These are substantial predecessors. The current artifact neither replaces their engines nor establishes better recovery performance; it isolates a narrower failure contract that can be inspected without deploying a recovery platform.

EviGuard is a particularly close comparison [14]. Its versioned evidence graph, deterministic claim verifiers and response gate distinguish supported, contradicted and unknown claims. Its formal guarantee conditions automatic execution on supported critical preconditions and explicitly depends on the completeness of those preconditions [14, Sections 4.4-5]. This study instead tests a post-repair invariant across renewable credential generations and accepted work. That obligation could be expressed as an added response precondition; no impossibility or deficiency of EviGuard is inferred. Its implementation was not reproduced.

Practical identity documentation further limits broad novelty claims. AWS documents an issuance-time boundary for revoking older role sessions and accounts for policy propagation in its console workflow [25]. Microsoft distinguishes identity-provider sessions from application-issued sessions that require application-side revocation [26]. RFC 7009 specifies token revocation, possible cascading effects, propagation concerns and the different consequences of self-contained versus referenced tokens [27]. The distinction between revoking one token and eliminating all access is therefore established.

Neil and colleagues study account-remediation advice through a five-phase model and a 57-service analysis [24]. Their empirical subject is advice coverage; this paper's subject is executable repair ordering and the evidence needed to justify its final state. Alert-driven graphs and CVSS address still other decisions, respectively forecasting attacker actions and representing severity with contextual metrics [11,13]. They are not patch-only baselines.

The new result offered here is the executable connection between three questions: which repair order tolerates renewal, which credential generations a closure test must cover, and which accepted work can invalidate its answer afterward. The graph proof states the first condition; the 48 schedules expose the second; the queue and epoch-fence walkthrough test the third. None makes topological sorting, fencing or token revocation a new invention. Together they provide a concrete recovery test that can fail, explain why it failed and distinguish two repairs on the same implementation. Its benefit to responders still needs independent evaluation.

8. Limits, practical use and next tests

The controlled experiments establish the behavior of the supplied implementation. They do not estimate real incident prevalence or show that responders using this contract outperform existing guidance. The three repetitions test consistency. The historical cases are purposive and rely on reported evidence rather than complete victim telemetry. A human analyst study and Exchange/MOVEit product tests were not performed.

The observer's credential inventory is complete because issuance in this fixture is controlled and recorded. That is a strong premise. An undiscovered issuer, unlogged credential, replica, cache, active connection or remote access path can defeat a broader inference. Likewise, a resource that checks current account state on every session request has different edges from one that accepts independent sessions. The graph must follow the implementation, not a familiar identity diagram.

Availability is not optimized here. Blocking all provisioning may be unacceptable, and disabling the wrong authority can interrupt legitimate work. Repair selection needs evidence preservation, business dependencies, service restoration and cost. The contract exposes those decisions; it does not authorize an automatic destructive response.

The strongest next experiment is a small independently administered service deployment with two issuers, delayed propagation and a deliberately withheld credential generation. Evaluate whether the contract reports unresolved when its inventory or observer loses coverage. Then compare analysts using ordinary incident guidance with analysts given the explicit contract, measuring missed access routes, inappropriate closure, disruption and time. That study should test an actual benefit rather than assume it from an attractive diagram.

A useful present application is narrower: take one proposed closure statement and identify the observation that could disprove it. If the statement relies on rejection of an old token, attempt authorized access through a replacement generation. If it relies on isolation, check what accepted work can still commit. If it relies on a rebuilt host, inspect the authority stored elsewhere. The pressure point is the smallest missing dependency that changes the final answer.

9. Conclusion

The old credential can be dead while access remains alive. In the running examples, the missing authority is either a credential created between repairs or a job accepted before containment. Both are visible once the investigation follows creation through to the protected read.

My proposed test is straightforward: identify what can issue replacement authority, establish when that ability ends, resolve work already accepted, revoke the resulting generations, and test the operation that mattered in the first place. Where those facts are missing, the result stays unresolved. Where a replacement credential succeeds, the recovery claim has a reproducible counterexample.

The contribution is useful at that level of precision. It identifies a failure, proves a limited ordering condition, and supplies code that demonstrates both the failure and two ways to prevent its delayed form. The next step is to test those obligations against independently operated services and responder decisions.

Data and code availability

The accompanying public reproducibility package contains all six executable modules, exact JSON/CSV results, source register, extended incident walkthroughs, evidence and intervention worksheets, and build instructions. It uses synthetic credentials and canaries, not private incident logs. Results were generated on 16 October 2026. Public sources remain with their publishers. A source register records the sections inspected; cited performance from other systems is not adopted or reproduced.

References

- [1] U.S. Government Accountability Office. Data Protection: Actions Taken by Equifax and Federal Agencies in Response to the 2017 Breach. GAO-18-559, 30 August 2018. <https://www.gao.gov/assets/gao-18-559.pdf>
- [2] Microsoft 365 Security and Microsoft Threat Intelligence. HAFNIUM targeting Exchange Servers with 0-day exploits. 2 March 2021, subsequently updated. <https://www.microsoft.com/en-us/security/blog/2021/03/02/hafnium-targeting-exchange-servers/>
- [3] Microsoft. Guidance for responders: Investigating and remediating on-premises Exchange Server vulnerabilities. March 2021. <https://www.microsoft.com/en-us/msrc/blog/2021/03/guidance-for-responders-investigating-and-remediating-on-premises-exchange-server-vulnerabilities/>
- [4] Grunzweig J, Meltzer M, Koessel S, Adair S, Lancaster T. Operation Exchange Marauder: Active Exploitation of Multiple Zero-Day Microsoft Exchange Vulnerabilities. Volexity, 2 March 2021; update 8 March 2021. <https://www.volexity.com/blog/2021/03/02/active-exploitation-of-microsoft-exchange-zero-day-vulnerabilities/>

- [5] Zaveri N et al. Zero-Day Vulnerability in MOVEit Transfer Exploited for Data Theft. Mandiant, 2 June 2023; attribution update 9 June 2023. <https://cloud.google.com/blog/topics/threat-intelligence/zero-day-moveit-data-theft/>
- [6] CISA and FBI. #StopRansomware: CL0P Ransomware Gang Exploits CVE-2023-34362 MOVEit Vulnerability. AA23-158A, 7 June 2023. <https://www.cisa.gov/news-events/cybersecurity-advisories/aa23-158a>; <https://content.govdelivery.com/accounts/USDHSCISA/bulletins/35ecb08>
- [7] Saltzer JH, Schroeder MD. The Protection of Information in Computer Systems. Proceedings of the IEEE, 63(9), 1278-1308, 1975. <https://web.mit.edu/saltzer/www/publications/protection/Basic.html>
- [8] Nelson A, Rekhi S, Souppaya M, Scarfone K. Incident Response Recommendations and Considerations for Cybersecurity Risk Management: A CSF 2.0 Community Profile. NIST SP 800-61r3, April 2025. <https://doi.org/10.6028/NIST.SP.800-61r3>
- [9] Sheyner O, Haines J, Jha S, Lippmann R, Wing JM. Automated Generation and Analysis of Attack Graphs. IEEE Symposium on Security and Privacy, 2002. Author abstract: <https://www.cs.cmu.edu/afs/cs/project/calder/www/sp02.html>
- [10] Ou X, Govindavajhala S, Appel AW. MulVAL: A Logic-based Network Security Analyzer. 14th USENIX Security Symposium, 2005. https://www.usenix.org/legacy/events/sec05/tech/full_papers/ou/ou.pdf
- [11] Băbălău I, Nadeem A. Forecasting Attacker Actions using Alert-driven Attack Graphs. arXiv:2408.09888v1, 2024. <https://arxiv.org/abs/2408.09888>
- [12] Hammar K, Li T. Online Incident Response Planning under Model Misspecification through Bayesian Learning and Belief Quantization. arXiv:2508.14385, 2025; related AISEC publication DOI 10.1145/3733799.3762965. <https://arxiv.org/abs/2508.14385>
- [13] FIRST. Common Vulnerability Scoring System v4.0: Specification Document. <https://www.first.org/cvss/v4.0/specification-document>
- [14] Zhou H et al. EviGuard: Machine-Verifiable Evidence Grounding for LLM-Based Industrial Incident Reasoning. Applied Sciences 16(18), 8925, 2026. <https://doi.org/10.3390/app16188925>
- [15] Joint cybersecurity authorities. Exploitation of Accellion File Transfer Appliance. AA21-055A, 24 February 2021. <https://www.cisa.gov/news-events/cybersecurity-advisories/aa21-055a>; <https://content.govdelivery.com/accounts/USDHSCISA/bulletins/2c39d96>
- [16] U.S. House of Representatives, Committee on Oversight and Government Reform. The Equifax Data Breach: Majority Staff Report, 115th Congress. December 2018. <https://oversight.house.gov/wp-content/uploads/2018/12/Equifax-Report.pdf>
- [17] Apache Software Foundation. S2-045: Possible Remote Code Execution when performing file upload based on Jakarta Multipart parser. March 2017. <https://cwiki.apache.org/confluence/spaces/WW/pages/68717257/S2-045>
- [18] Tsai O. A New Attack Surface on MS Exchange Part 1 - ProxyLogon! August 2021. <https://blog.orange.tw/posts/2021-08-proxylogon-a-new-attack-surface-on-ms-exchange-part-1/>
- [19] Hanley Z. MOVEit Transfer CVE-2023-34362 Deep Dive and Indicators of Compromise. Horizon3.ai, 9 June 2023. <https://horizon3.ai/attack-research/attack-blogs/moveit-transfer-cve-2023-34362-deep-dive-and-indicators-of-compromise/>
- [20] Barretto R. An Update on the Steps We are Taking to Protect MOVEit Customers. Progress, 5 June 2023. <https://www.progress.com/blogs/update-steps-we-are-taking-protect-moveit-customers>
- [21] Barretto R. Working to Enhance the Security of MOVEit Transfer Products through Partnership and Transparency. Progress, 13 June 2023. <https://www.progress.com/blogs/working-enhance-security-moveit-transfer-products-through-partnership-transparency>
- [22] Kim T, Wang X, Zeldovich N, Kaashoek MF. Intrusion Recovery Using Selective Re-execution. 9th USENIX Symposium on Operating Systems Design and Implementation, 2010. <https://www.usenix.org/conference/osdi10/intrusion-recovery-using-selective-re-execution>

- [23] Chandra R, Kim T, Shah M, Narula N, Zeldovich N. Intrusion Recovery for Database-backed Web Applications. ACM Symposium on Operating Systems Principles, 2011.
<https://people.csail.mit.edu/nickolai/papers/chandra-warp.pdf>
- [24] Neil L, Bouma-Sims E, Lafontaine E, Acar Y, Reaves B. Investigating Web Service Account Remediation Advice. Seventeenth Symposium on Usable Privacy and Security, pp. 359-376, 2021.
<https://www.usenix.org/conference/soups2021/presentation/neil>
- [25] Amazon Web Services. Revoke IAM role temporary security credentials. IAM User Guide.
https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_revoke-sessions.html
- [26] Microsoft. Revoke user access in an emergency in Microsoft Entra ID. Microsoft Learn.
<https://learn.microsoft.com/en-us/entra/identity/users/users-revoke-access>
- [27] Lodderstedt T, Dronia S, Scurtescu M. OAuth 2.0 Token Revocation. RFC 7009, August 2013.
<https://www.rfc-editor.org/rfc/rfc7009.html>
- [28] Burrows M. The Chubby Lock Service for Loosely-Coupled Distributed Systems. 7th USENIX Symposium on Operating Systems Design and Implementation, pp. 335-350, 2006.
https://www.usenix.org/legacy/event/osdi06/tech/full_papers/burrows/burrows_html/
- All online references accessed 16 October 2026. The CISA canonical endpoints returned access errors during retrieval; the linked official distribution copies supplied their advisory text.