Cyber Apocalypse 2026 — Cloud — Wrong Stamp

Category: Cloud **Challenge:** Wrong Stamp **Flag format:** `HTB{<result>}` **Flags:** 8

---

1. Challenge Brief

> Elric Ashspar finds a seizure stamp in a dead clerk's bag near Stonepass. Vaultrune uses copies of that stamp to take supplies from Sythra's border guards, leaving a road Stormbound needs open without them. The stamp looks old, but Elric spots fresh tool marks on it. He must find the flaw, prove the stamp is fake, and give the guards a quick way to reject future copies.

Translated out of the fiction:

Story element	Cloud reality
The seizure stamp	An IAM user's long-lived access key
Copies of the stamp	The same `AKIA…` key replayed from an attacker-controlled host
"The stamp looks old"	The key is legitimate and previously well-used — signature verification passes
"Fresh tool marks"	A different Boto3/Botocore version + a foreign source IP on the same key
Prove the stamp is fake	Correlate CloudTrail metadata, not credential validity
Reject future copies	A detection/response control on key-reuse from new IPs

Two services are exposed:

- `154.57.164.71:31642` — an HTTP briefing page (the "Courier's Packet") that hands out player credentials.
- `154.57.164.71:31897` — the AWS API endpoint (a LocalStack-style emulator).

---

2. Recon

2.1 The briefing port

The briefing port answers plain HTTP with a custom server banner:

```
$ curl -s -i http://154.57.164.71:31642/ | head -3
HTTP/1.0 200 OK
Server: StonepassBriefing/1.0 Python/3.12.13
Content-Type: text/html; charset=utf-8
```

The page renders three "letters"; the second one is a *Sealed Writ* containing starting credentials, but the HTML ships only a `Loading credentials...` placeholder. The credentials are pulled client-side:

```
$ grep -oE 'fetch\([^)]*\)' briefing.html
fetch("/player-creds.json", { cache: "no-store" })
```

Fetching that path directly:

```
$ curl -s http://154.57.164.71:31642/player-creds.json
{
  "user": "stonepass-investigator",
  "access_key_id": "AKIA4WCUSFKCCSCSYQ78",
  "secret_access_key": "ocBbgUid9EtrU7RkrwZdtbb7oOl1iTX/RaKN6AjA",
  "region": "us-east-1",
  "endpoint": "http://127.0.0.1:4566"
}
```

> **Gotcha:** the `endpoint` field says `http://127.0.0.1:4566`. That is the *container-internal* address. From outside you must point `--endpoint-url` at the instance IP and the **AWS API port** from the instance card (`31897` here), not the briefing port. The briefing text calls this out explicitly.

2.2 The AWS port

Hitting the API port without SigV4 confirms what it is:

```
$ curl -s http://154.57.164.71:31897/
{"__type":"AccessDeniedException","message":"User is not authorized to perform:
MissingAuthentication"}
```

2.3 Session setup

```
export AWS_ACCESS_KEY_ID=AKIA4WCUSFKCCSCSYQ78
export AWS_SECRET_ACCESS_KEY='ocBbgUid9EtrU7RkrwZdtbb7oOl1iTX/RaKN6AjA'
export AWS_DEFAULT_REGION=us-east-1
export AWS_ENDPOINT_URL=http://154.57.164.71:31897   # AWS CLI v2 honours this; v1 needs --endpoint-
url
```

```
$ aws --endpoint-url http://154.57.164.71:31897 sts get-caller-identity
{
    "UserId": "AIDAZLS84L3S5K0I8E54",
    "Account": "491827305948",
    "Arn": "arn:aws:iam::491827305948:user/stonepass-investigator"
}
```

---

3. Mapping the investigator's permissions

The investigator role is deliberately narrow. Most control-plane reads are refused:

```
$ aws --endpoint-url http://154.57.164.71:31897 cloudtrail describe-trails
An error occurred (AccessDeniedException) when calling the DescribeTrails operation:
User is not authorized to perform: cloudtrail:DescribeTrails
```

`cloudtrail:LookupEvents` is the one door that opens:

```
$ aws --endpoint-url http://154.57.164.71:31897 cloudtrail lookup-events --max-results 5
{
    "Events": [
        {
            "EventId": "43c62885-c72e-442f-8203-20e54a597f49",
            "EventName": "StopLogging",
            ...
```

This is the whole shape of the challenge: **you cannot query the trail's configuration, only its history.** Every answer has to be reconstructed from event metadata.

---

4. Pulling the full history

`LookupEvents` returns 50 records per page and is `NextToken`-paginated. Rather than clicking through by hand, dump everything once and analyse offline:

```
#!/usr/bin/env python3
"""dump_trail.py — paginate cloudtrail:LookupEvents into events.json"""
import subprocess, json

ENDPOINT = "http://154.57.164.71:31897"
events, token = [], None
```

```
while True:
    cmd = ["aws", "--endpoint-url", ENDPOINT, "cloudtrail", "lookup-events",
           "--max-results", "50", "--output", "json"]
    if token:
        cmd += ["--next-token", token]
    page = json.loads(subprocess.run(cmd, capture_output=True, text=True).stdout)
    events += page["Events"]
    token = page.get("NextToken")
    if not token:
        break

print("total events:", len(events))
json.dump(events, open("events.json", "w"), indent=1)
```

```
$ python3 dump_trail.py
total events: 273
```

Note that the interesting fields live inside `CloudTrailEvent`, which is a **JSON string**, not an object. `sourceIPAddress`, `userAgent`, `errorCode` and `accessKeyId` are *only* available after parsing it — the top-level summary hides all four. That nesting is where most people lose the trail.

---

5. Analysis

5.1 Group by source IP

```
import json, collections
rows = sorted((json.loads(e["CloudTrailEvent"]) for e in json.load(open("events.json"))),
              key=lambda c: c["eventTime"])
d = collections.defaultdict(lambda: [0, set(), set()])
for c in rows:
    ip = c.get("sourceIPAddress")
    d[ip][0] += 1
    d[ip][1].add(c["userIdentity"].get("userName"))
    d[ip][2].add(c["userIdentity"].get("accessKeyId"))
for ip, (n, users, keys) in d.items():
    print(f"{ip:16} {n:4}  {users}  {keys}")
```

```
10.30.41.118      253  {'stonepass-warden'}  {'AKIAC0YTKTUA3NLO9PQX'}
192.0.2.55          6  {'stonepass-warden'}  {'AKIAC0YTKTUA3NLO9PQX'}
127.0.0.1           8  {None}                {'AKIA4Q7N2P8R1W6K9M3L'}
10.244.21.215       6  {None}                {None}
```

Four sources, but only two matter:

- **`10.30.41.118`** — 253 events over three days (`2026-07-23T07:18` → `2026-07-25T20:45`). RFC1918 space, one identity, routine admin work. This is the **legitimate internal operator**.
- **`192.0.2.55`** — 6 events inside a 10-second window. `192.0.2.0/24` is **TEST-NET-1** (RFC 5737), the documentation range used as a stand-in for "external internet host". This is the **attacker**.

The other two are environment noise, and it's worth being explicit about why they're excluded:

- `127.0.0.1` with key `AKIA4Q7N2P8R1W6K9M3L` — the provisioning script. Its events (`CreateTrail`, `StartLogging`, `CreateUser`, `PutUserPolicy`, `CreateAccessKey`) all fire at `20:44:5x`, i.e. *before* the scenario timeline, and they are what mint the `stonepass-investigator` user you were handed.
- `10.244.21.215` with no identity — a Kubernetes pod CIDR issuing an unauthenticated `ListBucket` every ~10 seconds that always fails `AccessDeniedException`. A health check.

5.2 The forgery — the flaw in the stamp

Both the operator and the attacker present **the same access key**, `AKIAC0YTKTUA3NLO9PQX`, and both authenticate cleanly as `stonepass-warden`. Signature validation cannot tell them apart; the stamp is a perfect copy. The difference is in the tooling that wields it:

```
for ua, n in collections.Counter(
        c.get("userAgent", "")[:46] for c in rows
        if c["userIdentity"].get("userName") == "stonepass-warden").most_common():
    print(f"{n:4}  {ua}")
```

```
 253  Boto3/1.34.0 md/Botocore#1.34.0 ua/2.0 os/linux
   6  Boto3/1.29.7 md/Botocore#1.29.7 ua/2.0 os/linux
```

The split is exact — 253 legitimate events on **Boto3 1.34.0**, all six attacker events on **Boto3 1.29.7**, an older release, from a different kernel build (`5.15.0-107-generic` vs the operator's `5.15.167.4-l…`). Those are Elric's *fresh tool marks*: the credential is genuine, the hand holding it is not. Identity alone proves nothing here; **identity + source IP + user-agent** proves the copy.

5.3 The handoff

Printing every event in the minute spanning the transition makes the takeover unambiguous:

```
20:45:32.796  10.30.41.118   GetCallerIdentity
20:45:33.905  10.30.41.118   ListUsers
20:45:34.918  10.30.41.118   GetUser          user=stonepass-warden
20:45:36.127  10.30.41.118   ListAccessKeys   user=stonepass-warden   <-- LAST internal act
20:45:38.080  10.244.21.215  ListBucket       AccessDenied            [health check]
20:45:48.843  10.244.21.215  ListBucket       AccessDenied            [health check]
20:45:57.115  192.0.2.55     GetTrailStatus                           <== ATTACKER BEGINS
20:45:59.427  192.0.2.55     DeleteTrail      AccessDenied
20:45:59.475  10.244.21.215  ListBucket       AccessDenied            [health check]
20:46:01.356  192.0.2.55     ListBucket       stonepass-audit-trail-logs
20:46:03.366  192.0.2.55     ListObjectsV2    stonepass-audit-trail-logs
20:46:05.178  192.0.2.55     GetObject        NoSuchKey
20:46:06.765  192.0.2.55     StopLogging      stonepass-audit-trail   <== AUDIT BLINDED
```

The internal host goes silent at `20:45:36.127` and **never speaks again** — verified, not assumed:

```
after = [c["eventTime"] for c in rows
         if c.get("sourceIPAddress") == "10.30.41.118" and c["eventTime"] > "2026-07-25T20:45:57"]
print(after)   # []
```

There is a clean 21-second seam. The internal operator's last act was `ListAccessKeys` against `stonepass-warden` — enumerating the very key that reappears from the internet moments later.

5.4 The attacker's six moves

The whole intrusion is six calls in 9.65 seconds, and it reads like a textbook anti-forensics playbook:

#	Time (UTC)	Action	Parameters	Result
1	20:45:57.115	`GetTrailStatus`	`name=stonepass-audit-trail`	OK — confirm the trail is live
2	20:45:59.427	`DeleteTrail`	`name=stonepass-audit-trail`	**DENIED** — AccessDeniedException
3	20:46:01.356	`ListBucket`	`stonepass-audit-trail-logs`	OK — find where logs land
4	20:46:03.366	`ListObjectsV2`	`stonepass-audit-trail-logs`, `max-keys=10`	OK
5	20:46:05.178	`GetObject`	`…/2026/06/24/audit.log.gz`	FAILED — NoSuchKey
6	20:46:06.765	`StopLogging`	`name=stonepass-audit-trail`	OK — **audit blinded**

Step 2 is the pivot. `DeleteTrail` was refused, so destroying the evidence outright was off the table; the attacker fell back to `StopLogging`, which merely halts delivery and requires a weaker permission. Note the irony that seals the case: `StopLogging` is itself a management event, so the *last thing the trail ever recorded was its own silencing*, complete with source IP and user agent.

---

6. Flags

#	Question	Answer	Flag
1	Last CloudTrail API action by the compromised user from the internal IP immediately before the attacker session began	`ListAccessKeys`	`HTB{ListAccessKeys}`
2	First CloudTrail API action called from the attacker IP	`GetTrailStatus`	`HTB{GetTrailStatus}`
3	API action the attacker attempted that was explicitly denied before the trail was stopped	`DeleteTrail`	`HTB{DeleteTrail}`
4	S3 bucket the attacker enumerated before stopping the trail	`stonepass-audit-trail-logs`	`HTB{stonepass-audit-trail-logs}`
5	Name of the CloudTrail trail that was stopped	`stonepass-audit-trail`	`HTB{stonepass-audit-trail}`
6	IAM username whose credentials executed the trail disable	`stonepass-warden`	`HTB{stonepass-warden}`
7	IP address from which the trail was disabled	`192.0.2.55`	`HTB{192.0.2.55}`
8	API action used to disable the audit trail	`StopLogging`	`HTB{StopLogging}`

Evidence per flag

1. **`ListAccessKeys`** — `20:45:36.127Z`, `sourceIPAddress=10.30.41.118`, `requestParameters={"userName":"stonepass-warden"}`. Confirmed to be the final `10.30.41.118` event in the entire 273-event dataset. Don't be tripped up by the health-check `ListBucket` calls at `20:45:38` and `20:45:48` — those come from `10.244.21.215`, are unauthenticated, and always fail.
2. **`GetTrailStatus`** — `20:45:57.115Z`, the earliest of exactly six events from `192.0.2.55`.
3. **`DeleteTrail`** — `20:45:59.427Z`, `errorCode=AccessDeniedException`. It is the only denied call in the attacker's chain and the only `DeleteTrail` in the whole dataset.
4. **`stonepass-audit-trail-logs`** — target of the attacker's `ListBucket` / `ListObjectsV2` / `GetObject` at `20:46:01` – `20:46:05`, immediately before `StopLogging`.
5. **`stonepass-audit-trail`** — from `StopLogging`'s `requestParameters.name` and the event's resource ARN `arn:aws:cloudtrail:us-east-1:491827305948:trail/stonepass-audit-trail`.
6. **`stonepass-warden`** — `userIdentity.type=IAMUser`, `userName=stonepass-warden`, `arn:aws:iam::491827305948:user/stonepass-warden`, `principalId=AIDAMTS9SMC3N8PQ1BG1`.
7. **`192.0.2.55`** — `sourceIPAddress` on the `StopLogging` event.
8. **`StopLogging`** — `eventName` at `20:46:06.765Z`, `readOnly=false`, `managementEvent=true`.

Verification one-liner for the decisive event:

```
$ aws --endpoint-url http://154.57.164.71:31897 cloudtrail lookup-events \
    --lookup-attributes AttributeKey=EventName,AttributeValue=StopLogging \
    --query 'Events[0].CloudTrailEvent' --output text | python3 -m json.tool \
  | grep -E 'eventName|sourceIPAddress|userName|"name"|accessKeyId|userAgent'
        "userName": "stonepass-warden",
        "accessKeyId": "AKIAC0YTKTUA3NLO9PQX",
        "eventName": "StopLogging",
        "sourceIPAddress": "192.0.2.55",
        "userAgent": "Boto3/1.29.7 md/Botocore#1.29.7 ...",
            "name": "stonepass-audit-trail"
```

---

7. "A quick way to reject future copies"

The last third of the brief is the defensive ask, and it's the part worth actually answering. The forged stamp passed every check AWS makes, because AWS only checks the signature. The control has to sit one layer up, on the metadata.

7.1 Detect — alert the moment a trail stops

`StopLogging` and `DeleteTrail` are exactly what EventBridge is for. This is the single highest-value rule in the account:

```
{
  "source": ["aws.cloudtrail"],
  "detail-type": ["AWS API Call via CloudTrail"],
  "detail": {
    "eventSource": ["cloudtrail.amazonaws.com"],
    "eventName": ["StopLogging", "DeleteTrail", "UpdateTrail", "PutEventSelectors"]
  }
}
```

Wire it to an SNS topic that pages a human. In this incident that alarm would have fired 9 seconds into the intrusion.

7.2 Detect — the same key from a second source

The generalisable signal is a credential appearing from an IP it has never used. An Athena query over the CloudTrail log bucket, hunting keys that straddle the RFC1918 boundary:

```
SELECT useridentity.accesskeyid,
       useridentity.username,
       count(*)                                  AS calls,
       array_agg(DISTINCT sourceipaddress)       AS source_ips,
```

```
       array_agg(DISTINCT useragent)              AS user_agents
FROM cloudtrail_logs
WHERE eventtime > date_add('day', -7, now())
  AND useridentity.accesskeyid LIKE 'AKIA%'
GROUP BY 1, 2
HAVING count(DISTINCT sourceipaddress) > 1
   AND bool_or(NOT regexp_like(sourceipaddress, '^(10\.|172\.(1[6-9]|2[0-9]|3[01])\.|192\.168\.)'))
ORDER BY calls DESC;
```

The user-agent column is what turns a suspicion into proof — a genuine operator does not downgrade Boto3 mid-session.

7.3 The same check, runnable right now

```
#!/usr/bin/env python3
"""stamp_check.py — flag any AKIA key used from more than one IP or SDK build."""
import json, collections, ipaddress, subprocess, sys

ENDPOINT = sys.argv[1] if len(sys.argv) > 1 else "http://154.57.164.71:31897"

# Corporate space. Deliberately NOT ipaddress.is_private: the stdlib counts
# TEST-NET-1 (192.0.2.0/24) as private, which would hide the attacker.
CORPORATE = [ipaddress.ip_network(n) for n in ("10.0.0.0/8", "172.16.0.0/12", "192.168.0.0/16")]

def fetch():
    events, token = [], None
    while True:
        cmd = ["aws", "--endpoint-url", ENDPOINT, "cloudtrail", "lookup-events",
               "--max-results", "50", "--output", "json"]
        if token:
            cmd += ["--next-token", token]
        page = json.loads(subprocess.run(cmd, capture_output=True, text=True).stdout)
        events += page["Events"]
        token = page.get("NextToken")
        if not token:
            return events

def offnet(ip):
    try:
        addr = ipaddress.ip_address(ip)
    except ValueError:
        return False                       # service principals, "AWS Internal", etc.
    return not any(addr in net for net in CORPORATE)

profile = collections.defaultdict(lambda: {"ips": set(), "uas": set(), "user": None})
```

```
for e in fetch():
    c   = json.loads(e["CloudTrailEvent"])
    key = c["userIdentity"].get("accessKeyId")
    if not key or not key.startswith("AKIA"):
        continue
    p = profile[key]
    p["ips"].add(c.get("sourceIPAddress"))
    p["uas"].add(c.get("userAgent", "").split(" ")[0])
    p["user"] = c["userIdentity"].get("userName")

for key, p in profile.items():
    if len(p["ips"]) > 1 or len(p["uas"]) > 1:
        print(f"[!] FORGED STAMP SUSPECTED  {key}  (user={p['user']})")
        for ip in sorted(p["ips"], key=str):
            print(f"      ip  {ip}{'   <-- OFF-NETWORK' if offnet(ip) else ''}")
        for ua in sorted(p["uas"]):
            print(f"      sdk {ua}")
```

```
$ python3 stamp_check.py
[!] FORGED STAMP SUSPECTED  AKIAC0YTKTUA3NLO9PQX  (user=stonepass-warden)
      ip  10.30.41.118
      ip  192.0.2.55   <-- OFF-NETWORK
      sdk Boto3/1.29.7
      sdk Boto3/1.34.0
```

> **Analyst trap worth knowing:** the obvious implementation is `not ipaddress.ip_address(ip).is_private`, and it *silently fails on this exact case*. Python's stdlib includes TEST-NET-1 (`192.0.2.0/24`) in its private list, so `is_private` returns `True` for the attacker and the alert never fires. Enumerate RFC1918 explicitly — the same reason the Athena query above spells the three ranges out in its regex instead of leaning on a helper.

7.4 Prevent

Detection is the "quick way"; these are the durable fixes:

- **Retire long-lived `AKIA` keys.** Static keys are copyable by definition — that is the entire vulnerability in this challenge. Move humans to identity-center sessions and workloads to IAM roles.
- **Bind credentials to a network.** An IAM policy condition on `aws:SourceIp` (or a VPC endpoint policy with `aws:SourceVpce`) makes a stolen key useless off-network. The forged stamp would have failed at step 1.
- **Log tamper-proofing.** Enable CloudTrail **log file validation**, use an **organization trail** the member account cannot stop, and apply **S3 Object Lock** to the log bucket. `StopLogging`

succeeded here because the trail was account-local and the principal held `cloudtrail:StopLogging`.

- **Least privilege.** `stonepass-warden` had no business holding `StopLogging`. The `DeleteTrail` denial shows someone was *partway* to getting this right — the boundary was drawn one permission too wide.
- **Require MFA for destructive calls.** `sessionContext.attributes.mfaAuthenticated` was `"false"` on the `StopLogging` event. A `aws:MultiFactorAuthPresent` condition would have blocked it.

---

8. Takeaways

- **`LookupEvents` is the fallback when everything else is denied.** `DescribeTrails`, `GetTrailStatus` and `s3api` were all refused; the event history alone carried all eight answers.
- **`CloudTrailEvent` is a JSON string inside JSON.** `sourceIPAddress`, `userAgent`, `errorCode` and `accessKeyId` do not exist in the summary object. Parse the inner blob or you will miss the case.
- **Baseline the noise before reading the story.** Two of four source IPs here were provisioning and health-check traffic. Discard them explicitly, with a reason, rather than eyeballing the tail.
- **A valid signature is not an authentic actor.** Same key, same username, same account — different IP and different SDK build. Correlate identity *with* provenance.
- **The blinding is itself the evidence.** `StopLogging` is a management event, so it logs the hand that pulls the plug. Anti-forensics that runs through the audit plane always leaves one final record.

Case closed: the stamp `AKIAC0YTKTUA3NLO9PQX` was copied from `stonepass-warden`. At `2026-07-25T20:46:06.765Z` an operator at `192.0.2.55`, wielding Boto3 1.29.7 instead of the warden's 1.34.0, failed to delete `stonepass-audit-trail` and settled for `StopLogging` it — 30 seconds after the real warden last enumerated his own keys from `10.30.41.118`.