INSTRUCTIONS - page 1 of 3

Paste each blue prompt to the agent, in order, in the SAME chat. Exact text - nothing to figure out.

Setup (do this once)

1) Open a Terminal. 2) Type cd ~/tollbooth (Enter). 3) Start the agent: type claude (Enter). 4) Paste the blue prompts below in order, SAME chat, so it remembers. Type ONLY the blue text (no quotes).

~20 min per visit. Do the [CORE] phases (1, 2, 5) for the full story. [DEEP] phases (3, 4) are for a quiet moment or to run later - the sheet is yours to keep. No agent? terminal commands are on page 3.

[CORE] Network forensics

Phase 1 - paste this, then Enter:

> Using your cybersecurity skills, investigate lab-tollbooth.pcap: find how the web server's credentials leaked, extract the leaked AccessKeyId (it starts with ASIA), and tell me which external IP sent the malicious request. List the skills you used.

You will see: an SSRF request to 169.254.169.254 and a leaked key starting with ASIA.

[CORE] Cloud forensics - the pivot

Phase 2 - same chat:

> Now use your CloudTrail and S3 skills on the cloudtrail/ folder: with that leaked AccessKeyId, list everything the attacker did and which files they stole from the S3 bucket.

You will see: recon calls, then 6 GetObject downloads of customer data by the leaked key.

[DEEP] Baseline vs. anomaly

Phase 3 - same chat:

> Separate the attacker from normal use: what does legitimate activity on this account look like, and what makes the attacker's calls stand out? Could you have caught them WITHOUT already knowing the key?

You will see: legit calls come from the instance's own IP; the attacker's come from outside - catchable by origin, not just the key.

[DEEP] Map it - and check the agent

Phase 4 - same chat:

> Map the full attack to MITRE ATT&CK technique IDs, then add the NIST CSF 2.0 categories and MITRE D3FEND techniques that are relevant to this incident. Skip MITRE ATLAS and NIST AI RMF - this is not an AI-system attack, so they do not apply; say so rather than forcing a mapping. Check whether MITRE F3 (fraud) applies given the stolen payment data, and say if it does not. For every ID in every framework: is it directly evidenced in THIS data, or are you inferring it?

You will see: a clean ATT&CK chain, plus CSF categories (PR.AA / PR.PS / DE.CM) and D3FEND countermeasures (D3-ITF / D3-OTF / D3-NTA) - and watch it correctly SKIP ATLAS/AI RMF instead of forcing them in, and over-mapping ("exfiltration/T1567" when the data only shows reads = T1530). Correct any of these. This is the whole point.

[CORE] Incident report

Phase 5 - same chat:

> Write a short incident report: the full attack chain, the VERIFIED ATT&CK IDs, and the single change that would have stopped it.

You will see: SSRF -> stolen creds -> discovery -> S3 read, and "enforce IMDSv2" as the one fix. *Full answers: page 3.*

CHECKPOINTS - page 2 of 3

On track when the agent surfaces these. Not the full answer - that is page 3.

Phase 1 - Network

SSRF: GET /proxy?url=http://169.254.169.254/... -> a JSON cred blob with an AccessKeyId starting ASIA, and exactly one external attacker IP.

Phase 2 - Cloud pivot

The same key does recon (GetCallerIdentity, ListBuckets, IAM listing) then a burst of GetObject - arriving from an OUTSIDE IP, unlike the legit calls from the instance itself.

Phase 3 - Anomaly

Legit baseline = the instance's own IP / console. Attacker = same key, different origin. The tell is location + burst, not the key value - so yes, findable without it.

Phase 4 - Mapping + self-check

A per-step chain of ATT&CK IDs. The agent may OVER-map (S3 reads tagged as exfiltration/T1567). Correct read = T1530 (data from cloud storage). The catch is the learning.

Phase 5 - Report

One chain, an ID per step, and a single fix (IMDSv2) that breaks the whole thing.

Explore without spoiling it

```
cat cloudtrail/*.json | jq -r '.Records[].sourceIPAddress' | sort -u          # who is talking?
cat cloudtrail/*.json | jq -r '.Records[].userIdentity.accessKeyId' | sort | uniq -c
```

ANSWERS - page 3 of 3

The safety net + the terminal fallbacks.

1 Network

SSRF on /proxy (10.0.1.50) forced a fetch of 169.254.169.254 (IMDSv1) -> role creds leaked. Key ASIAJ7A6EXAMPLEK3Y99 (role acme-webapp-role). Injector: 203.0.113.66.

2 Cloud pivot

Same key from 203.0.113.66 (external) vs 10.0.1.50 (legit): recon, then 6x GetObject on acme-customer-data (payment_tokens.csv, ssn_index.parquet, db-dump). Attacker IP in pcap AND CloudTrail = proof of pivot.

3 Anomaly

Legit calls originate from the instance / console; the attacker reuses the key from an external IP in a tight burst. Detectable by origin + rate even with the key unknown.

4 Mapping + the correction

ATT&CK: T1552.005 (IMDS) -> T1078.004 (stolen key) -> T1580 / T1087.004 (discovery) -> T1530 (S3 read). NOT T1567 - no exfil-OUT event exists here; that is the agent's classic over-map to catch. CSF 2.0: PR.AA (credential/identity exposure), PR.PS (platform-hardening gap = IMDSv1), DE.CM (nothing alerted on the burst - the real detection gap). D3FEND: D3-ITF / D3-OTF (traffic-filtering fixes), D3-NTA (the detection approach itself). ATLAS and AI RMF do NOT apply - no AI system in this incident; the agent should say so, not force a tag. F3 is a judgment call given the stolen payment tokens - "maybe, but this attack is not a fraud-scheme TTP" is the strong answer.

5 Fix

Enforce IMDSv2 (HttpTokens=required) - breaks the entire chain at step one.

[ADVANCED] Does it scale? (optional, if a queue is thin)

python3 generate-bigdata.py --events 1000000 --days 3 then python3 bigquery.py --build

```
python3 bigquery.py --key ASIAJ7A6EXAMPLEK3Y99     # same 12 events out of ~1,000,000, in seconds
python3 bigquery.py --anomaly                       # finds the needle WITHOUT the key
```

Terminal fallbacks (no agent)

```
tshark -r lab-tollbooth.pcap -Y 'http.request.uri contains "169.254"' -T fields -e ip.src
tcpdump -nA -r lab-tollbooth.pcap | grep -A6 AccessKeyId       # the leaked JSON
cat                              cloudtrail/*.json                                |                              jq
'.Records[]|select(.userIdentity.accessKeyId=="ASIAJ7A6EXAMPLEK3Y99")|{t:.eventTime,e:.eventName}'
```

INSTRUCTIONS - page 1 of 3

Same setup. Paste each blue prompt to the agent, in order, same chat.

Setup (if you closed the terminal)

Open a Terminal. Type cd ~/tollbooth/opendoor (Enter). Type claude (Enter). Paste ONLY the blue text below.

~20 min. [CORE] phases (1, 3, 5) tell the full story. [DEEP] phases (2, 4) are for a quiet queue or to run later. No agent? terminal commands are on page 3.

[CORE] Full configuration audit

Phase 1 - paste this, then Enter:

> Using your cloud-audit skills, audit every AWS config file in this folder: the S3 bucket policy, the IAM role's permissions, and the EC2 instance-metadata setting. Map each misconfiguration to its CIS control AND its NIST CSF 2.0 category. List the skills you used.

You will see: three findings: a public S3 bucket, an over-permissioned IAM role, and IMDSv1 still allowed - each tagged with a CIS control and a CSF category.

[DEEP] Rank by blast radius

Phase 2 - same chat:

> Of those findings, which single one is most dangerous, and why? Rank all three by how much damage each enables on its own.

You will see: reasoning about which matters most: credential theft vs. self-admin IAM vs. public data exposure.

[CORE] Correlate with the breach

Phase 3 - same chat:

> Now look at ../lab-tollbooth.pcap and ../cloudtrail: for each misconfiguration, did the attacker EXPLOIT it, only PROBE it, or IGNORE it? Show me the evidence for each.

You will see: IMDSv1 exploited, the IAM role only probed, the public bucket bypassed - and they are NOT all the same.

[DEEP] Check the agent's call

Phase 4 - same chat:

> Double-check the public bucket: the data WAS stolen, so was the public policy actually the path? Check whether the exfil requests were anonymous or authenticated with the stolen key.

You will see: the theft was AUTHENTICATED with the stolen key - the public policy was NOT the route. A latent exposure, not the breach path. Correct the agent if it said "exploited."

[CORE] Prioritized remediation

Phase 5 - same chat:

> Give me a prioritized fix list: what to change FIRST to break the actual attack, versus what to fix because it is a latent risk even though it was not used here. For each fix, name the MITRE D3FEND countermeasure technique it implements.

You will see: IMDSv2 first (breaks the real chain); tighten IAM; close the public bucket anyway - each tied to a D3FEND ID. *Full answers: page 3.*

CHECKPOINTS - page 2 of 3

On track when the agent surfaces these. Not the full answer - that is page 3.

Phase 1 - Audit

Three findings: bucket policy Principal '*' + Public Access Block all-false (public S3, CIS 2.1.4 / CSF PR.PS); IAM with PutRolePolicy/AttachRolePolicy/PassRole on '*' (self-privesc, CIS 1.15 - NOT 1.16, that is stale v1.5 numbering / CSF PR.AA); MetadataOptions.HttpTokens='optional' (IMDSv1, CIS 5.7 - NOT 5.6, that is stale v3.0 numbering / CSF PR.PS). CIS numbers drift between benchmark versions - always confirm against the CURRENT release (v5.0.0), never quote from memory.

Phase 2 - Ranking

A defended ordering. Reasonable top pick: IMDSv1, because it is the actual entry point for credential theft. The point is the reasoning, not one 'right' order.

Phase 3 - Correlation

Each finding labelled exploited / probed / bypassed with log evidence - and they are deliberately NOT all the same.

Phase 4 - The catch

The exfil GetObjects are AUTHENTICATED (stolen key), not anonymous - so the public policy was not the path. If the agent said the bucket was 'exploited', this is where you correct it.

Phase 5 - Fix plan

Ordered by real risk: IMDSv2 first (breaks the chain), then IAM least-privilege, then close the public bucket (latent).

Explore without spoiling it

```
jq '.Statement[].Principal' s3-bucket-policy.json                        # any wildcard?
jq '.Reservations[].Instances[].MetadataOptions.HttpTokens' ec2-metadata-options.json
```

ANSWERS - page 3 of 3

The safety net + the terminal fallbacks.

1 Audit - three findings + CIS + CSF

PUBLIC BUCKET: 'PublicRead' (Principal '*') + Block Public Access all-false. CIS v5.0.0 2.1.4 (S3 Block Public Access) / CSF PR.PS. IAM PRIVESC: 'TooBroad' = PutRolePolicy/AttachRolePolicy/PassRole on '*' (role can self-admin). CIS v5.0.0 1.15 (full-admin IAM policy attached - 1.16 in the CURRENT benchmark is an unrelated AWS-Support-role control, an easy stale-number mistake) / CSF PR.AA. IMDSv1: HttpTokens='optional'. CIS v5.0.0 5.7 (EC2 Metadata Service IMDSv2-only - 5.6 in the current benchmark is VPC peering routing, also unrelated) / CSF PR.PS.

2 Ranking

Defensible top pick = IMDSv1: it is the live entry point (creds actually stolen through it). IAM privesc = highest POTENTIAL (full admin) but unused. Public bucket = real exposure, not the path here.

3 Correlation

IMDSv1 = EXPLOITED (creds stolen, used from 203.0.113.66). IAM = PROBED ONLY (enumeration, no PutRolePolicy in logs). Public bucket = BYPASSED (see below).

4 The correction

Exfil GetObjects are AUTHENTICATED with the stolen key, not anonymous - the public policy was never used. The bucket is a latent exposure, NOT the breach route. Agents often mis-call this 'exploited'.

5 Fix set (ordered)

1) HttpTokens=required (IMDSv2) - breaks the chain. D3FEND: D3-ITF (inbound traffic filtering on the /proxy SSRF). 2) Remove iam:* on '*'. D3FEND: no direct technique - this is an authorization-scoping fix, not a traffic control. 3) Block Public Access + drop Principal '*'. D3FEND: D3-OTF (outbound/access filtering on the bucket).

Terminal fallbacks (no agent)

```
jq -r '.Statement[]|select(.Principal=="*")|.Sid' s3-bucket-policy.json          # public grant
jq -r '.Reservations[].Instances[].MetadataOptions.HttpTokens' ec2-metadata-options.json
cat ../cloudtrail/*.json | jq -r '.Records[]|select(.sourceIPAddress=="203.0.113.66")|.eventName'
```