AWS Identity and Secrets

The Atarus series

Title	Status
AWS Identity and Secrets	This book
Foundations I, II and III	In preparation
Scope, Findings and the Report	In preparation
Cloud Penetration Testing	In preparation

Every book in the series prints only captured output, and the same company, Wrexby Outfitters, runs through all of them. Foundations, in preparation, will be the way into the series. This book holds six chapters of Cloud Penetration Testing, with three new chapters around them, and covers identity and secrets in one AWS account.

THE ATARUS SERIES

AWS Identity and Secrets

A field manual for testing IAM policies, escalation paths, outside access and stored secrets from one pipeline key

Atarus Research

ATARUS PRESS

First edition, 2026-11-11

AWS Identity and Secrets: A field manual for testing IAM policies, escalation paths, outside access and stored secrets from one pipeline key. Copyright © 2026 Atarus Offensive Security LLC.

All rights reserved. No part of this book may be reproduced, stored in a retrieval system or transmitted in any form or by any means without the prior written permission of the publisher, except for short quotations in reviews and articles.

The PDF edition is licensed to the person who bought it, for their own reading. Please do not post it or pass it on; a colleague who needs a copy can buy one at atarussecurity.com/books/aws-identity-and-secrets.

First edition, 2026-11-11.

This PDF edition has no ISBN. An ISBN identifies the paperback only.

Published by Atarus Press, an imprint of Atarus Offensive Security LLC.
atarussecurity.com/books/aws-identity-and-secrets
security@atarussecurity.com

Product and company names in this book may be trademarks of their owners. We use them editorially, to identify the products we describe, with no intention of infringing any trademark. No owner of a product named here has endorsed this book.

The information in this book is supplied as is, without warranty of any kind. The authors and Atarus Offensive Security LLC have taken care in its preparation, and every command output in it was captured from a real run, but they accept no liability to any person for any loss or damage caused, or said to be caused, directly or indirectly by the instructions, commands or software in it. Software and code that this book shows or links to keep their own licenses; if you reuse them, the license is yours to respect.

Authorization. Use the techniques in this book only on systems you own or have written permission to test. Unauthorized access to a computer is a crime in most jurisdictions, whatever your intent. The lab account in this book is deliberately vulnerable. If you build one like it, expose none of its resources to the internet and never let its access key leave your own machine. In AWS this means accounts you own and opened for the purpose, and never an account that belongs to a client or an employer without a signed letter. The preface repeats this notice because it matters more than anything else on this page.

Captured runs printed in this book are listed with their hashes at atarussecurity.com/books/aws-identity-and-secrets/captures. Corrections are listed at atarussecurity.com/books/aws-identity-and-secrets/errata.

For the platform teams who read the findings and fix them.

Contents in brief

	About this book	xi
PART I	**Before the first call** Chapters 1 to 3	1
PART II	**Mapping one key's reach** Chapters 4 to 6	67
PART III	**Where the secrets are** Chapters 7 and 8	167
PART IV	**Closing the review** Chapter 9	225
	Appendices	245
	Glossary	257
	Index	263

Contents

	About this book	xi
	Acknowledgments	xvii
	About Atarus Research	xviii
PART I	**Before the first call**	
1	What this book is, and the letter behind the test	1
2	The lab: the accounts, the profiles, the window and the log	17
3	IAM policy evaluation	41
PART II	**Mapping one key's reach**	
4	The AWS first hour	67
5	The four AWS IAM path families	95
6	Resource policies that grant access from outside	137
PART III	**Where the secrets are**	
7	Secrets in configuration: functions, user data, parameters	167
8	Who can read the secrets: Secrets Manager, KMS key policies and grants	197
PART IV	**Closing the review**	
9	The field checklist, and where to go next	225
	Appendix A: Tools and profiles for this book	245
	Appendix B: The Wrexby company file, AWS side	248

Appendix C: The lab accounts, their cost and their values 253

Glossary 257

Index 263

Updates and errata 268

Colophon 269

About this book

This book teaches you to test what one AWS credential can reach in one account. It reads the policies that decide each call and the paths that turn a pipeline key into more. Then it reads the resource policies that let outsiders in, and the secrets stored in configuration and in Secrets Manager. It starts from a signed scope letter and one access key and ends with a checklist you can run on a client account in one test window. Every command in it was run, and every output printed is the output that came back.

Where this book comes from

Chapters 3 to 8 are six chapters of Cloud Penetration Testing, the larger volume of this series, which is still in preparation and covers four clouds from the first hour to the report. We took the six that test identity and secrets in AWS, renumbered them and cut every pointer to a chapter this book does not hold. Chapters 1, 2 and 9 are new. They carry what the six chapters assumed from the rest of the volume: the scope letter, the lab and the operator log at the start, and the checklist at the end.

Who should read it

You have done a year of IT work: administering servers, deploying code to a cloud account, or reviewing access for an audit. You can work in a Linux shell, read JSON and say what HTTP and TLS are. You have signed in to the AWS console and seen an IAM user or a role. You have not yet been paid to test an account.

If you came to cloud testing from somewhere else, make sure of three things this book assumes: working in a shell, how a request crosses a network, and keeping notes of every command as you run it. Foundations, the first book of the series, is in preparation and will teach them. If you already test AWS accounts for a living, read chapters 1 and 2 for the conventions the rest uses, then take the chapters in any order.

The road map

The book has four parts and nine chapters, in the order a test runs. Every chapter opens with the time it takes, what you need and when you can skip it.

- **Part I, Before the first call.** What this book is and the letter behind the test; the lab account, its profiles, the test window and the operator log; and how AWS combines allows, denies, boundaries and guardrails into one decision. Chapters 1 to 3.
- **Part II, Mapping one key's reach.** The first hour with one pipeline key, the four families of IAM escalation paths, and the resource policies that grant access from outside the account. Chapters 4 to 6.
- **Part III, Where the secrets are.** Secrets left in configuration, in functions, user data and parameters; and who can read the secrets that are stored properly, through Secrets Manager policies, KMS key policies and grants. Chapters 7 and 8.
- **Part IV, Closing the review.** The six techniques as one checklist in engagement order, what the book leaves out, and the Learn page behind each chapter. Chapter 9.

How to read it

Read chapters 1 and 2 first, whatever your experience. The scope check, the profiles and the operator log are used in every chapter after them, and the later chapters do not repeat them. Chapter 3's policy evaluation rules are the ones every later chapter reasons with.

One engagement runs through the book. Wrexby Outfitters, the fictional outdoor retailer that runs through the series, is rebuilding its web shop this winter in a new AWS account, and has engaged us to review it before the shop takes its first order. The letter starts with one pipeline key, the access key of the IAM user `ci-deploy`, and grows only as far as the client's questions do. Appendix B holds the company file for the AWS side.

Chapters 1 to 8 close with "If this chapter did not work for you", the real failures we hit, each with its fix. Each of them also has a section called

"What the log kept", which names the record each step left in the client's logs, so the defender's view sits beside every action.

A read-along, with our lab

This book is a read-along. The lab is the firm's own: a small AWS Organization with a shop account, a partner account and a logging account, built by the firm from its own build plan and from seeds written for our capture harness, none of them published. Every output printed here is a real run in that lab, and no output was typed by hand.

No kit, download or hosted lab comes with this book. Appendix B describes the account's shape, and Appendix C what it cost us. If you build a copy by hand, you do so at your own risk and cost, and Appendix C says what that risk is.

If you are reading only

Reading only is the way this book expects to be read, and it costs nothing. Every command output in it is a captured run, printed in full with its run id beneath it. Each step shows the command, the real output and one sentence on what it means, so a reader who runs nothing can follow every chapter.

The chapters keep their labs as written, and every lab in this book is Core: it produces the chapter's artifact. Each lab ends with a line that names where the prose resumes, for example "Reading only? Continue at What the log kept."

What you need

- **To read.** Nothing beyond the book.
- **To follow at the keyboard in an account of your own.** One AWS account and a second that plays the partner, ideally under a small Organization; AWS CLI v2, which printed version 2.39.3 in every capture; jq 1.7.1; and a POSIX shell. Appendix A lists the tools and profiles, and Appendix C the accounts.
- **A budget alarm before anything else.** Set a 10 USD monthly budget alarm in every account before you create anything in it. Appendix C says

what each part of the lab cost us and on what date we checked the price.

- **Never a client's account.** Do not run these steps in an account that belongs to an employer or a client, even one you administer, unless a signed letter names it. A lab account is one you own, opened for this book, with nothing in it you would miss.

Conventions

Commands you type are printed in dark blocks labeled command. Outputs are printed on a gray tint with the run id underneath, which you can look up in the capture manifest. A block without a run id is never output; it is a file you write or a line you type. In a draft edition, a block marked draft is a capture still to run; the first edition has none.

Values that differ for every account are written as shell variables, such as `${LAB_ACCOUNT_ID}` and `${LAB_INSTANCE_ID}`. Appendix C lists all nine and what each printed as in our lab. The lab builds in `us-east-1`; where a Learn page shows another region, the chapter says so.

Our captures replace every real identifier with a fixed placeholder, and the AWS ones recur often enough to learn. Account ids print in the family AWS uses in its own documentation. `111122223333` is the shop account, which your lab account plays, and `123456789012` the management account; `444455556666` is the partner's account. The Organization prints as `o-a1b2c3d4e5`, and the guardrail policy as `p-guard01`. The instance `prod-api` prints as `i-0abc1234def567890` and `webhooks-2` as `i-0aaaa000000000001`. A role that IAM Identity Center creates prints as `AWSReservedSSO_<name>_aaaa000000000001`, with its real name in the middle and our suffix in place of the random one AWS adds. Access key ids print as `AKIA...EXAMPLE`, and every address the harness sees is replaced before anything is printed.

Wrexby Outfitters is fiction. Atarus, the firm behind this book, owns the domain wrexby.com, which serves a small brand site and sends no mail, so every wrexby.com address in the book is invented. Every Wrexby account, bucket, role and secret exists only inside our lab, or inside yours if you build one.

Terms are defined in plain words the first time they appear and collected in the glossary with the chapter that teaches each. Prices and vendor facts carry the date they were checked.

The lab rule

Set the profile and variables the chapter names before its first command. When your output differs from ours, compare the identity and the account first, since most differences are a profile pointing somewhere else, and the last section of the chapter second. Each lab ends by removing what it made and showing that the account is as it was.

What this book leaves out

This book tests identity and secrets in one AWS account from one key. It leaves out networks, storage exposure beyond the policies it reads, and writing the report. It also leaves out a review of role trust policies, the build pipeline and its trust, what is read from the management account, the partner account's own policies, the trail's event selectors and delivery, GuardDuty's findings, other services' configuration, the metadata endpoint beyond one token setting, and persistence. Azure, Google Cloud and Oracle Cloud are left out too. Chapter 9 lists each one and the chapter where it came up.

Where the captures and errata live

The capture manifest for this edition, with the hash of every output printed here, is at atarussecurity.com/books/aws-identity-and-secrets/captures. Corrections are at atarussecurity.com/books/aws-identity-and-secrets/errata, each with the chapter, the page, the wrong text, the fix and the date. If a correction changes a captured output, we rerun the capture; we never edit an output by hand. Report an error through the form on that page.

Use these techniques only where you are allowed to

Use the techniques in this book only on systems you own or have written permission to test. Unauthorized access to a computer is a crime in most jurisdictions, and not knowing the law is no defense. A cloud account is a system someone owns, and a key that works is not permission: test an ac-

count only when a signed letter names it, its identities, its window and every change you may make. The account our captures came from is deliberately vulnerable; if you build one like it, expose none of its resources to the internet. Chapter 1 covers the letter in detail.

Atarus Research

Acknowledgments

Six of these chapters were written first for Cloud Penetration Testing, and they carry every correction that volume's reviewing passes made to them before they were cut into this book. We thank the readers who report errors through the errata page; every correction they lead to is listed there with the date of the fix.

About Atarus Research

Atarus Research is the research and publishing name of Atarus Offensive Security LLC, a penetration testing firm whose cloud work is the review this book teaches: written scope, read-only by default, every finding matched to the client's own logs. It publishes the Learn library at atarussecurity.com, whose cloud pages are the starting point for most chapters here, the advisories the firm files, and this series. The firm signs its work as a firm. The authority of a page here rests on the captured run behind it, which you can check, and not on a name on the cover.

About the technical review

Every chapter passes three reviewing passes before it is typeset: one for the prose, one for the technical claims, and one as the reader the book is for. The technical pass checks every printed output, and every claim about what a call returns or records, against its capture in the manifest.

CHAPTER 1

What this book is, and the letter behind the test

WHAT YOU WILL BE ABLE TO DO

- Read any output in this book by its capture header: the run, the step, the time in UTC, the account, the tool version and the hash, and know which values are placeholders
- Check that a starting key reaches the account, the alias and the Organization a scope letter names, and recognize an out of scope call refused by a service control policy
- Show that the one change a letter permits would succeed without making it, with an EC2 dry run and the empty tag read that follows it, and record both in a scope check record

LAB

Read along with our lab account, or repeat in an AWS account of your own: five read or dry-run calls as ci-deploy and one call refused on purpose; nothing is written; cost near zero

HOURS

1

SKIP IF

You have tested AWS accounts under a signed letter and already check the account id, the alias, the Organization and a dry run of the permitted change before the first real call

Scenario

Wrexby's web shop is being rebuilt this winter, and the new shop runs in an AWS account of its own. Wrexby has engaged us to review that account before the new shop takes its first order. An engagement is one piece of paid testing work, with a start, an end and a client. The security lead's first draft of the terms asked us to test "the web shop's AWS environment". We sent it back. A description is not a scope, because the AWS command line

reaches any account a key belongs to, whether or not anyone meant to include it.

The second draft names identifiers. It gives one AWS account by its twelve-digit id and its alias, and two regions. It names one starting credential: the access key of `ci-deploy`, the user Wrexby's build pipeline deploys with. The platform team asked us to start from that key because it is the one long-lived key in the account. The letter permits one change, a tag on the instance `prod-api`. It sets a window with a start and an end in UTC. It sets a stop condition: any identifier that does not match the letter.

The letter is signed, and the window opens at the start of the kickoff call. The platform team sent two questions ahead of the call. How do we know the key points where the letter says? And what will "one tag" look like in their records? This chapter answers both with five calls that read or test, and one call refused on purpose, before anything else in the test runs.

What you will do

This chapter does two jobs. It says what the book is and how its outputs were made, and it runs the scope check that opens every test in it. The scope check is short, and the rest of the book leans on it.

Who this book is for

This book is for people who test AWS accounts, and for the people who defend them and want to know what a test does. It covers one situation in depth: one AWS account, one pipeline key handed over at the start, and the question of what that key can reach. It tests the policies of IAM, Identity and Access Management, the AWS service that decides who may do what. It follows the paths by which a key climbs to more access, the policies that let outsiders in, and the secrets the account stores. It does not cover networks, storage exposure beyond the policies it reads, or writing the report. Chapter 9 lists what it leaves out.

You need to be at ease in a shell: running commands, setting a variable, reading an error. You need to read JSON, the bracketed text format AWS uses for its documents and many of its replies. You do not need to know

IAM already. Every AWS term is defined the first time it is used, in the chapter that uses it or an earlier one.

The book uses the AWS CLI, the command-line tool AWS publishes for calling its services, at version 2. Each call the CLI makes is an API call: one request to one AWS service, which the service answers or refuses. Every command in the book is a CLI command or a short shell line around one.

What the parts cover

Part I, before the first call, holds this chapter, chapter 2 and chapter 3. Chapter 2 sets up the lab: the account, the profiles, the test window and the operator log. Chapter 3 shows how AWS decides one request when several policies apply. Part II maps one key's reach. Chapter 4 spends the first hour on four questions about the key. Chapter 5 tests the four families of paths by which a key becomes an administrator. Chapter 6 reads the resource policies that let principals outside the account in. Part III asks where the secrets are. Chapter 7 finds them in configuration, and chapter 8 reads who can open them. Part IV, chapter 9, puts the techniques in the order a test runs them.

How the outputs in this book were made

Every output in this book is a capture. A capture is the text a command printed in a real run in our lab, copied into the page by the tool that ran it, never typed. Our lab is the firm's own: an AWS account built as a small copy of Wrexby's, inside an Organization we run for testing. We built it, we ran every command in it, and the pages print what came back.

The label beside each output names its run, such as `r-20261111-033755-93b6`, and its step. A run is one sitting of our capture tool, which runs a fixed list of commands in order. The run id resolves in the capture manifest, the list of every run the book prints, with each command and its time. Each output also opens with a header line from the tool. Step 1's output below shows one. The step in a label is the capture tool's own number, which need not match the chapter's numbered steps.

The header names the run and the step, then the time in UTC. UTC, Coordinated Universal Time, is the clock with no time zone and no summer shift, and AWS records use it. Next comes `capture-a us-east-1`, the

name of our lab account and the region the run used. The bench edition names the version of our test workstation. The tools field gives the AWS CLI version, 2.39.3 throughout. The last field is a SHA-256 hash, a fingerprint of the block exactly as printed below the header, so you can check it from the page. The public capture manifest lists the same value, and the firm's private record ties each block to its raw output. A block with nothing below the header carries the hash of no bytes at all, `e3b0c442...`.

Our captures replace real identifiers with placeholders. Our lab account prints as `111122223333`. The Organization's management account prints as `123456789012`, and the Organization itself as `o-a1b2c3d4e5`. Where your own account prints its id, ours prints a placeholder, and a step says so once where it matters.

You can read this book in two ways. A reader who only reads can follow every step from the printed output and the sentence that explains it. A reader with an AWS account of their own can repeat the steps there. Chapter 2 describes how our lab is built, so a practitioner can build a copy by hand. We supply no kit with this book.

The letter behind the test

A scope letter, also called the rules of engagement, is the signed document that says what a test may touch. It names accounts and credentials by identifier, never by description. It sets the window, the span of time in which testing may happen, as a start and an end in UTC. It lists every change we may make, by resource. It names the client contact, the one person at the client we tell about anything urgent. Wrexby's letter names the security lead.

Wrexby's letter names one AWS account and two regions. A region is one of AWS's separate locations, each with its own copy of most services. A resource made in one region exists only there, so the regions in the letter are the places a test may look.

It names one starting credential. An access key is a long-lived pair of values, a key id and a secret, that signs API calls as one IAM user. An IAM user is an identity inside one account, with its own policies and, here, its

own key. `ci-deploy` is the user the build pipeline runs as, and its key is the only one we start from.

It permits one change: a tag on the instance `prod-api`. An EC2 instance is a virtual machine that AWS's EC2 service runs. A tag is a key and value label on a resource, which changes nothing about how the resource runs. The letter names the tag key, `atarus-scope`, so Wrexby can find it.

It sets the stop condition. If any identifier the command line reports does not match the letter, we run nothing else and tell the client contact. A key that lands in the wrong account is the commonest way a test goes out of scope. Nobody can see that mistake without the check.

It holds us to a read-only posture. Anything the letter does not name as a change is a read. A read asks a service what it holds and changes nothing. Every call in this book that changes something is named as a change, before it runs.

It sets status calls: short scheduled calls with the client contact during the window. At each one we say what we have done and anything urgent we found. A finding is a weakness we can show with evidence and write into the report. Wrexby's first status call comes an hour after the first-hour pass begins, and chapter 4 spends that hour.

The letter adds three gates around any change beyond the tag, and chapter 5 relies on them. Each proof must be added to the letter by name before it runs, with the resource it touches and the window it runs in. The client contact hears before each proof and again after it ends. A proof whose response differs from the one expected is reversed first and explained second. Chapter 5 calls these the engagement gates. In your own lab you hold them too, so the habit is in place before a client's account.

The rules we keep, and the authorization notice

Use these techniques only on accounts you own or have written permission to test. Unauthorized access to a computer system is a crime in most jurisdictions, whatever the intent. A signed letter is the permission, and its identifiers are its limits. Calls outside them are outside the permission, however close the account looks.

Our lab account is weak on purpose. Its pipeline key holds a path to administrator, which chapter 5 tests. If you build a copy, treat its keys as live credentials. Never put them in a public repository, a shared folder or a chat. Delete the key when you finish with the lab.

This chapter's lab

The scope check is the first thing a test runs, before the first real call. It asks five questions. Who is the key? Is this the account the letter names, by id and by alias? Does an Organization sit above it? What does a call out of scope look like? Would the one permitted change succeed, and did testing it leave anything behind?

The record is a scope check record in your notes. For each identifier it holds the value the command line reports beside the value in the letter. It holds the refused call, and the dry run with the empty tag read after it. Chapter 2 moves these lines into the operator log.

If you repeat the steps, two settings carry every command. `AWS_PROFILE` names the CLI profile, the stored set of credentials and settings, that holds the `ci-deploy` key. `AWS_DEFAULT_REGION` is `us-east-1`. Steps 5 and 6 also use `${LAB_INSTANCE_ID}`, the id of your copy of `prod-api`. Chapter 2 sets all three; you can read this chapter first and run its commands after chapter 2. The cost is close to nothing.

Steps 1 to 6 are the Core lab and produce the record.

Steps

1. Who is this key?

Every call runs as some principal, the identity AWS checks permissions for. An ARN, Amazon Resource Name, is the full name AWS gives every principal and resource. It holds the service, the account id and the name.

`get-caller-identity` needs no permission and cannot be denied, so it is always the first call. STS, the Security Token Service, is the AWS service that answers it and issues role sessions. `--query` picks the fields to print, and `--output text` prints them on one line, separated by a tab. You will see the account id, then the ARN of the `ci-deploy` user.

```
aws sts get-caller-identity --query '[Account,Arn]' --output text
```

COMMAND

```
# capture r-20261111-033755-93b6 step 01-caller-identity,
  2026-11-11T03:38:11Z, capture-a us-east-1, bench 2026.11-
  cloud-0.3, tools aws=2.39.3, sha256
  5e7d1e9dad59ad8eee274aff9e26271df3a97f5f583854fd7e53d67484b9cd1c
111122223333  arn:aws:iam::111122223333:user/ci-deploy
```

CAPTURE r-20261111-033755-93b6 STEP 01-caller-identity

The account id, `111122223333`, is our lab's placeholder; yours prints your own. The ARN ends in `user/ci-deploy`, so the key belongs to the IAM user the letter names, and not to a role. Write both values in the record, beside the letter's.

If this did not work. An account or user you did not expect means the CLI used another profile. Usually `AWS_PROFILE` is not set in this terminal; export it and run the call again. An error that no credentials were found means the profile holds none. On an engagement, a mismatch that survives a fresh terminal is the stop condition: run nothing else, and tell the client contact.

2. Does the alias match the letter?

An account alias is a name an account carries in place of its id, in sign-in pages and some outputs. The id is the evidence, because the alias is a label anyone with the right IAM permission can change. The alias is still worth reading: a letter that names an alias the account does not carry was written from someone's memory. You will see one word, the alias.

```
aws iam list-account-aliases --query 'AccountAliases' --output
  text
```

COMMAND

```
# capture r-20261111-033755-93b6 step 02-account-alias,
  2026-11-11T03:38:12Z, capture-a us-east-1, bench 2026.11-
  cloud-0.3, tools aws=2.39.3, sha256
  c077b191ab390e1be5d25eb9211b28191d82be1c0cfb37b6b2011df32deedbf9
capture-a
```

CAPTURE r-20261111-033755-93b6 STEP 02-account-alias

Our lab account's alias is `capture-a`, the name every capture header carries. Wrexby's letter names the alias its platform team set for the shop account, and the two must agree. Write the alias beside the letter's.

If this did not work. An empty line means the account has no alias. That is not a stop, since the id already matched, but note it and ask the client contact which name the letter meant. A refusal naming `iam:ListAccountAliases` means the key cannot read it. Record the refusal and continue on the id alone.

3. Does an Organization sit above the account?

An AWS Organization groups accounts under one management account, which sets guardrails for the accounts below it, the member accounts. A guardrail here is a service control policy, an SCP: a policy the management account attaches above member accounts. It caps what any principal in them can do, the account's own administrators included, and it grants nothing.

The management account matters to the letter. It owns the guardrails, and often the record of API calls for every account. Whoever holds it has to agree to the test as well. You will see three fields: the Organization id, the management account id, and the feature set.

```
aws organizations describe-organization --query 'Organization.[Id,
  MasterAccountId,FeatureSet]' --output text
```

COMMAND

```
# capture r-20261111-033755-93b6 step 03-organization-membership,
  2026-11-11T03:38:13Z, capture-a us-east-1, bench 2026.11-
  cloud-0.3, tools aws=2.39.3, sha256
  0c5df215d1fe3248ebe3e14c409b19ac9d399c99203871f6a097b3ba455ee436
o-a1b2c3d4e5  123456789012  ALL
```

CAPTURE r-20261111-033755-93b6 STEP 03-organization-membership

The account is a member of `o-a1b2c3d4e5`, whose management account is `123456789012`; both are placeholders. AWS still calls the management account the master account in this field's name. The feature set `ALL` means the Organization has every feature turned on, and SCPs need that. So guardrails can sit above this account, and step 4 meets one. Write the

three values in the record. On Wrexby's account the management account belongs to its IT team, and the letter names that team's consent.

If this did not work. An error saying the account is not in an Organization means it stands alone. There are then no SCPs above it, and step 4's refusal will not happen. That is not a stop. Write it in the record's coverage notes, since chapters 3 and 4 read guardrails a standalone account does not have. Chapter 2 says how our lab differs from a standalone account.

4. What does out of scope look like?

Wrexby's letter names two regions, and its guardrail is meant to refuse every other one. Our lab's guardrail does the same for `us-east-1` and `us-west-2`. This call asks AWS Lambda, the service that runs code as functions, to list functions in a region outside both. `AWS_REGION=eu-west-1` before the command sets the region for that one command. AWS CLI v2 lets `AWS_REGION` win over `AWS_DEFAULT_REGION`. You will see an `AccessDeniedException` that ends with an explicit deny in a service control policy.

```
AWS_REGION=eu-west-1 aws lambda list-functions --query
  'Functions[].FunctionName' --output text
```

COMMAND

```
# capture r-20261108-185059-ea1e step 02-outside-region-is-
  refused, 2026-11-08T18:51:15Z, capture-a us-east-1, bench
  2026.11-cloud-0.3, tools aws=2.39.3, sha256
  acfa74499dbc2aefb0859dbab85df89f396ec618fa187e7738bd11db37ba91b7

aws: [ERROR]: An error occurred (AccessDeniedException) when
  calling the ListFunctions operation: User: arn:aws:
  iam::111122223333:user/ci-deploy is not authorized to perform:
  lambda:ListFunctions on resource: * with an explicit deny in a
  service control policy: arn:aws:organizations::123456789012:
  policy/o-a1b2c3d4e5/service_control_policy/p-guard01
```

CAPTURE r-20261108-185059-ea1e STEP 02-outside-region-is-refused

The header still says `us-east-1`, because that is the run's default; the command itself went to `eu-west-1`. The blank line is one the CLI prints before an error. The error names the caller, the action and the resource, then the reason: a Deny in an SCP, with the SCP's ARN. That ARN carries

the management account's placeholder and the Organization's id, the same two values step 3 printed. An explicit deny is a Deny statement that matched the request, and chapter 3 shows why one beats every allow.

This is what out of scope looks like from the command line, when a guardrail is doing its job. The refusal says nothing about the key's own rights; the guardrail stopped the call before they counted. Write the refusal in the record, with the SCP's ARN. This run was taken on 8 Nov 2026, three days before the others, and shows the same guardrail.

If this did not work. A list of function names, or an empty line, means no guardrail refused the call. Either the region is allowed above your account, or no SCP reaches it. In your own lab, that is a fact about how you built it. On an engagement it is a finding about the guardrail. The region stays out of scope either way, because the letter, not the guardrail, sets the scope.

5. Would the permitted change succeed?

The letter permits one tag on `prod-api`, and the platform team wants to see what that looks like before it happens. A dry run asks a service to check permission for a call and stop there. EC2 offers one through `--dry-run`, so the call proves the permission exists and writes nothing. The tag key and value, `atarus-scope` and `check`, are the ones the letter names. You will see an error that is the good result: `DryRunOperation`, saying the request would have succeeded.

```
aws ec2 create-tags --dry-run --resources ${LAB_INSTANCE_ID}
  --tags Key=atarus-scope,Value=check
```

COMMAND

```
# capture r-20261111-033755-93b6 step 04-change-tested-by-dry-run,
  2026-11-11T03:38:14Z, capture-a us-east-1, bench 2026.11-
  cloud-0.3, tools aws=2.39.3, sha256
  82cf53a89c7c9404878046ed761e5186b07221f7dde2a05f50463c62110194e0

aws: [ERROR]: An error occurred (DryRunOperation) when calling the
  CreateTags operation: Request would have succeeded, but DryRun
  flag is set.
```

CAPTURE r-20261111-033755-93b6 STEP 04-change-tested-by-dry-run

EC2 reports a dry run's answer as an error either way, and the CLI exits with status 254, as it does for any error a service returns. So read the code in the parentheses, not the exit status. `DryRunOperation` means `ci-deploy` holds the permission to tag this instance and EC2 wrote nothing. Write the dry run in the record, with its time.

The instance id in `${LAB_INSTANCE_ID}` belongs to `prod-api`, the instance this dry run named. Chapters 4, 5 and 7 read the same instance through the same variable.

If this did not work. `UnauthorizedOperation` in place of `DryRunOperation` means the call would be refused. Either the key does not hold the permission, or a guardrail above the account denies it, and the message says which. Neither answer changes anything. An error that the instance id is malformed, or a missing value, means `${LAB_INSTANCE_ID}` is empty in this terminal; export it and run again. Dropping `--dry-run` is the change itself, and it needs the letter's consent for that tag on that resource.

6. Did the dry run leave anything behind?

A proof is not finished until the cleanup is shown. This read asks for any tag with the key `atarus-scope` on the instance. `--filters` narrows the answer to that instance and that key, and the backslash continues the command on a second line. You will see nothing below the header: an empty result.

```
aws ec2 describe-tags --filters Name=resource-id,Values=${LAB_
  INSTANCE_ID} Name=key,Values=atarus-scope \
  --query 'Tags[].[Key,Value]' --output text
```

COMMAND

```
# capture r-20261111-033755-93b6 step 05-cleanup-verify-no-tag,
  2026-11-11T03:38:15Z, capture-a us-east-1, bench 2026.11-
  cloud-0.3, tools aws=2.39.3, sha256
  e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

CAPTURE r-20261111-033755-93b6 STEP 05-cleanup-verify-no-tag

The header is the whole block. No tag with that key is on the instance, so the dry run wrote nothing and the account is as it was. An empty result is

only proof because step 5 named the same instance and the same key. Write the read in the record under the dry run, with its time.

That closes the scope check. The record now answers both of the platform team's questions. The key points at the account the letter names, by id, alias and Organization. One tag will look like a `CreateTags` event from `ci-deploy`, and today's attempt was a dry run that left nothing.

If this did not work. A line reading `atarus-scope` and `check` means the tag exists. Either step 5 ran without `--dry-run`, or an earlier test left the tag. Stop and tell the client contact, because a change was made before the window's consent covered it. Remove the tag only with their agreement, and record both the tag and its removal.

Reading only? Continue at What the log kept.

What the log kept

Every AWS call in this chapter is a management event, a record AWS keeps of a call that reads or changes configuration. CloudTrail is the AWS service that keeps these records. Each event names the caller, the time, the source address and the request. Steps 1, 2, 3 and 6 are reads. They appear under `ci-deploy` as `GetCallerIdentity`, `ListAccountAliases`, `DescribeOrganization` and `DescribeTags`.

Step 5 appears as `CreateTags`, with an error code in place of a result. We looked it up about fourteen minutes later, from the read-only review profile chapter 2 describes. The lookup bounds the search to the minute of the scope check, 03:38:00Z to 03:39:00Z in our run, and prints four fields of the event. You will see the time, the event name, `false` for read-only, and the code `Client.DryRunOperation`.

```
aws cloudtrail lookup-events --start-time ${LAB_WINDOW_START}
  --end-time ${LAB_WINDOW_END} \
  --lookup-attributes AttributeKey=EventName,AttributeValue=
    CreateTags \
  --query "Events[?contains(Username || '', 'ci-deploy')].
    CloudTrailEvent" --output text \
  | jq -r '[.eventTime, .eventName,
    (.readOnly|tostring), .errorCode] | @tsv'
```

COMMAND

```
# capture r-20261111-035238-03cf step 01-dry-run-event,
  2026-11-11T03:52:54Z, capture-a us-east-1, bench 2026.11-
  cloud-0.3, tools aws=2.39.3 jq=1.7.1, sha256
  0c3e2c519b835e103aa26368f82167ab0eb26f6d949b0b88c93bdf82b2974ad5
2026-11-11T03:38:15Z  CreateTags  false  Client.DryRunOperation
```

CAPTURE r-20261111-035238-03cf STEP 01-dry-run-event

CloudTrail classes the dry run as a write, since `readOnly` is `false`, although nothing was written. A reviewer who searches for changes will find it. So the record you wrote matters: it tells Wrexby in advance that one `CreateTags` event from `ci-deploy` is ours and changed nothing. For this lookup the two window variables held that minute; chapter 2 sets them for a whole session, and chapter 6 explains why the lookup waits and why it tests `Username` that way.

Step 4's refused call is recorded too, in `eu-west-1`, the region it was sent to. Lambda's event names carry its API version, so the name to search for is `ListFunctions20150331`. Chapter 7 shows a Lambda event name of this kind in a capture. We did not look this event up. A refusal by a guardrail is still an event under the key that tried. Tell the client contact about it at the kickoff call, so nobody chases a call we made on purpose.

Check your work

You have finished when the scope check record holds six entries, each with its time in UTC:

1. The account id and the ARN from step 1, beside the letter's values.
2. The alias from step 2, beside the letter's.

3. The Organization id, the management account id and the feature set from step 3.
4. The refused call from step 4, with the SCP's ARN.
5. The dry run from step 5, with the code `DryRunOperation`.
6. The empty tag read from step 6.

Every value in the record is an identifier. None is a secret, so the record can go to the client as it is. If you read along and ran nothing, check that you can say what each of the six captures proves. Then name the two that printed an error on purpose.

Nothing was written, so there is nothing to remove. Step 6 was the cleanup check.

If this chapter did not work for you

Three failures the commands make likely. Each rests on a capture.

Step 5 ended with an error status, and a script or a checklist marked it failed. That is the result you want. EC2 reports every dry run as an error, and the CLI exits with status 254 for it. Step 5's capture shows the error text. Read the code in the parentheses. `DryRunOperation` means the permission is there and nothing was written. `UnauthorizedOperation` means the call would be refused, by the key's own policies or by a guardrail. If you wrap these commands in a script, test for the code in the message and not for the exit status.

Step 5 or step 6 was refused with an explicit deny in a service control policy. The terminal has `AWS_REGION` set to a region the guardrail does not allow. Step 4's capture shows the same refusal for a Lambda call. AWS CLI v2 lets `AWS_REGION` win over `AWS_DEFAULT_REGION` and over the profile. Run `echo "$AWS_REGION"`. If it prints a region, run `unset AWS_REGION` and repeat the step. Step 4 sets the variable for one command only, so typing it as printed leaves nothing behind.

Step 1 printed an account id or a user that is not the one in the letter. The profile in this terminal holds a different key, or `AWS_PROFILE` is not set and the CLI used its default profile. Stop before any other call. Nothing in

step 1 touched the wrong account beyond naming it. Run `echo "$AWS_PROFILE"`, set it to the profile that holds the starting key, and run step 1 again. On an engagement, record the wrong identity and the time, because the call is in that account's event history.

We captured the mildest case: the terminal was on the review profile, so the account is right and the principal is not. You will see the same account id, then a role where the letter names a user.

```
aws sts get-caller-identity --query '[Account,Arn]' --output text
```

COMMAND

```
# capture r-20261111-033850-2942 step 01-review-profile-identity,
  2026-11-11T03:38:51Z, capture-a us-east-1, bench 2026.11-
  cloud-0.3, tools aws=2.39.3, sha256
  3583cae0d998968f62dbc686840d23052430cc672a2d9dffdeda7b10177c4b3d
111122223333    arn:aws:sts::111122223333:assumed-role/atarus-
  review/atarus-review
```

CAPTURE `r-20261111-033850-2942` STEP `01-review-profile-identity`

`assumed-role/atarus-review` is the read-only role chapter 2 describes, not `user/ci-deploy`. Chapter 2 also prints what later reads look like when a profile is left on another account.

Where this goes next

Chapter 2 describes the lab these captures came from: the account, its three profiles, the test window and the operator log this record moves into. Chapter 3 then asks what `ci-deploy` is allowed to do, and why step 4's guardrail won over every allow. Chapter 5 is where a proof must change something and put it back, under the engagement gates this chapter set out.

ABOUT THIS CHAPTER

Captured runs printed in this chapter: `r-20261111-033755-93b6`, `r-20261108-185059-ea1e`, `r-20261111-035238-03cf`, `r-20261111-033850-2942`.

The sample ends here

You have read the front matter and chapter 1 of AWS Identity and Secrets, the free sample. The full book has eight more chapters: the lab and its conventions, how AWS evaluates a policy, the first hour with the key, four families of IAM escalation path, resource policies that let another account in, secrets left in configuration, who can read the secrets in Secrets Manager and KMS, and a closing checklist that puts the techniques in the order an engagement uses them. The appendices, glossary and index follow.

The full book, as a PDF, is at

`atarussecurity.com/books/aws-identity-and-secrets`

The contents pages in this sample give the page numbers of the full book.