Automated Analysis of AWS Access Control

Andrew Gacek
Automated Reasoning in Identity, Amazon Web Services
September 17, 2020

Introducing AWS Identity and Access Management (IAM) Access Analyzer

IAM Access Analyzer uses a form of mathematical analysis called *automated reasoning*, which applies logic and mathematical inference to determine all possible access paths allowed by a resource policy.

ers to
nment. With
missions
d AWS Lambda

ent manual
checks ... s are added or updated. Using IAM Access Analyzer, customers can proactively address any
resource p ... curity and governance best practices around resource sharing and protect their resources from
unintended a ... alyzer delivers comprehensive, detailed findings through the AWS IAM, Amazon S3, and AWS Security
Hub consoles and ... ts APIs. Findings can also be exported as a report for auditing purposes. IAM Access Analyzer findings
provide definitive an ... who has public and cross-account access to AWS resources from outside an account.

IAM Access Analyzer uses a form of mathematical analysis called automated reasoning, which applies logic and mathematical inference to determine all possible access paths allowed by a resource policy. This means that IAM Access Analyzer can evaluate hundreds or even thousands of policies across a customer's environment in seconds, and deliver comprehensive findings about resources that are accessible from outside the account. We call this provable security.

With this launch, IAM Access Analyzer is available at no additional cost in the IAM console and through APIs in all commercial AWS Regions. IAM Access Analyzer is also available through APIs in AWS GovCloud (US).

To learn more about IAM Access Analyzer, see the feature page.

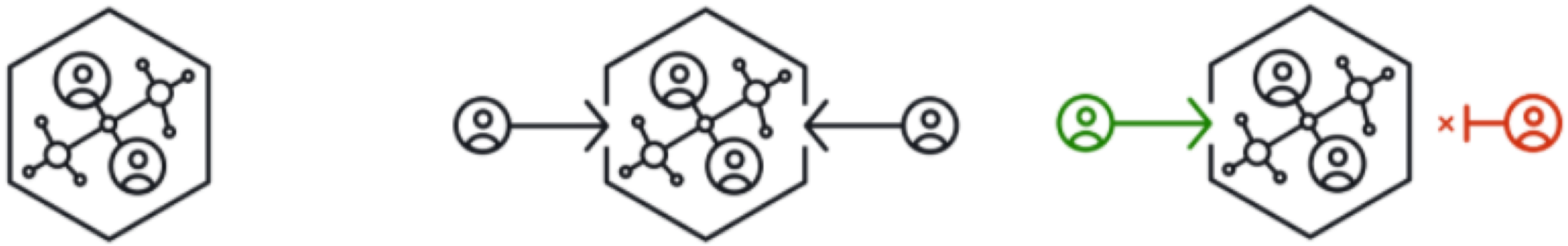
IAM > Access Analyzer

Access Analyzer

Monitor access to resources

Create analyzer

How it works



1 Create an analyzer

You can set the scope for the analyzer to an organization or an AWS account. This is your zone of trust. The analyzer scans all of the supported resources within your zone of trust.

2 Review active findings

When Access Analyzer finds a policy that allows access to a resource from outside of your zone of trust, it generates an active finding. Findings include details about the access so that you can take action.

3 Take action

If the access is intended, you can archive the finding so that you can focus on reviewing active findings. If the access is not intended, you can resolve the finding by modifying the policy to remove access to the resource.

Getting started

- What is Access Analyzer?
- Access Analyzer User Guide

What is cloud computing?

"on-demand delivery of IT resources via the Internet with pay-as-you-go pricing."

$f: \ldots \to \mathit{computer}$

$g: \ldots \to \mathit{database}$

$f^{-1}: \mathit{computer} \to \ldots$

Compute
- EC2
- Lightsail
- Lambda
- Batch
- Elastic Beanstalk
- Serverless Application Repository
- AWS Outposts
- EC2 Image Builder

$$\lambda(code) \equiv f^{-1} \circ code \circ f$$

Developer Tools
- CodeDeploy

Application Integration
- Step Functions
- Amazon AppFlow
- Amazon EventBridge
- Amazon MQ
- Simple Notification Service
- Simple Queue Service
- SWF

sqs: … → *elastic, reliable queue*

Storage
- S3
- EFS
- FSx
- S3 Glacier
- Storage Gateway
- AWS Backup

$s3$: … → *durable data storage*

- Kinesis Video Streams
- MediaConnect
- MediaStore
- MediaTailor
- Elemental Appliances & Software

Database
- RDS
- DynamoDB
- ElastiCache
- Neptune
- Amazon Redshift
- Amazon QLDB
- Amazon DocumentDB
- Amazon Keyspaces

ddb : … → *fast key-value database*

Policy example

```
- Effect: Allow
  Condition:
    StringEquals:
      SourceVpc:
      - “vpc-a”
      - “vpc-b”

- Effect: Allow
  Condition:
    StringEquals:
      PrincipalOrgID: “o-2”

- Effect: Deny
  Condition:
    StringEquals:
      SourceVpc: “vpc-b”
    StringNotEquals:
      PrincipalOrgID : “o-1”
```

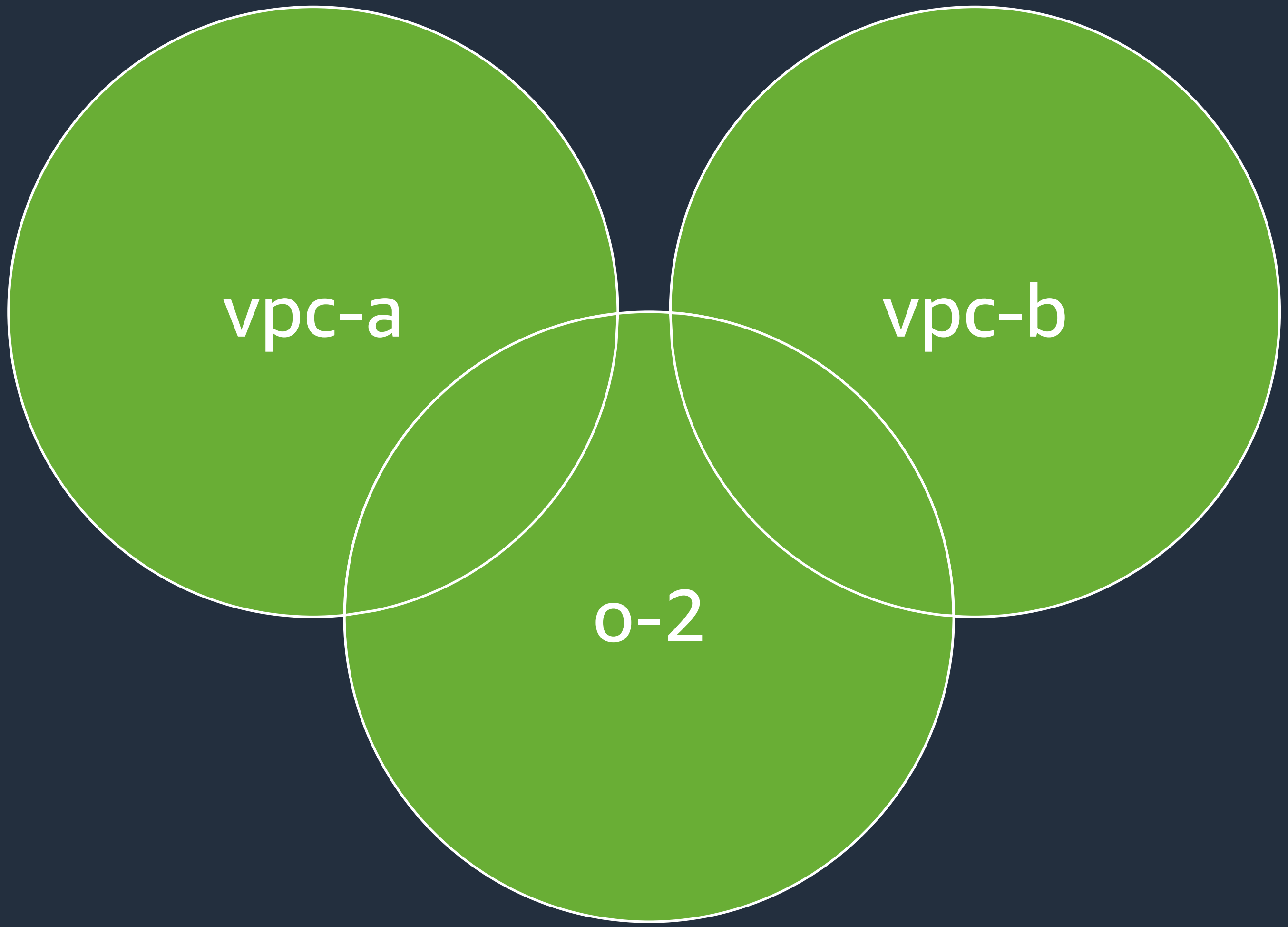
No Access

Policy example

```
- Effect: Allow
  Condition:
    StringEquals:
      SourceVpc:
      - “vpc-a”
      - “vpc-b”

- Effect: Allow
  Condition:
    StringEquals:
      PrincipalOrgID: “o-2”

- Effect: Deny
  Condition:
    StringEquals:
      SourceVpc: “vpc-b”
    StringNotEquals:
      PrincipalOrgID : “o-1”
```

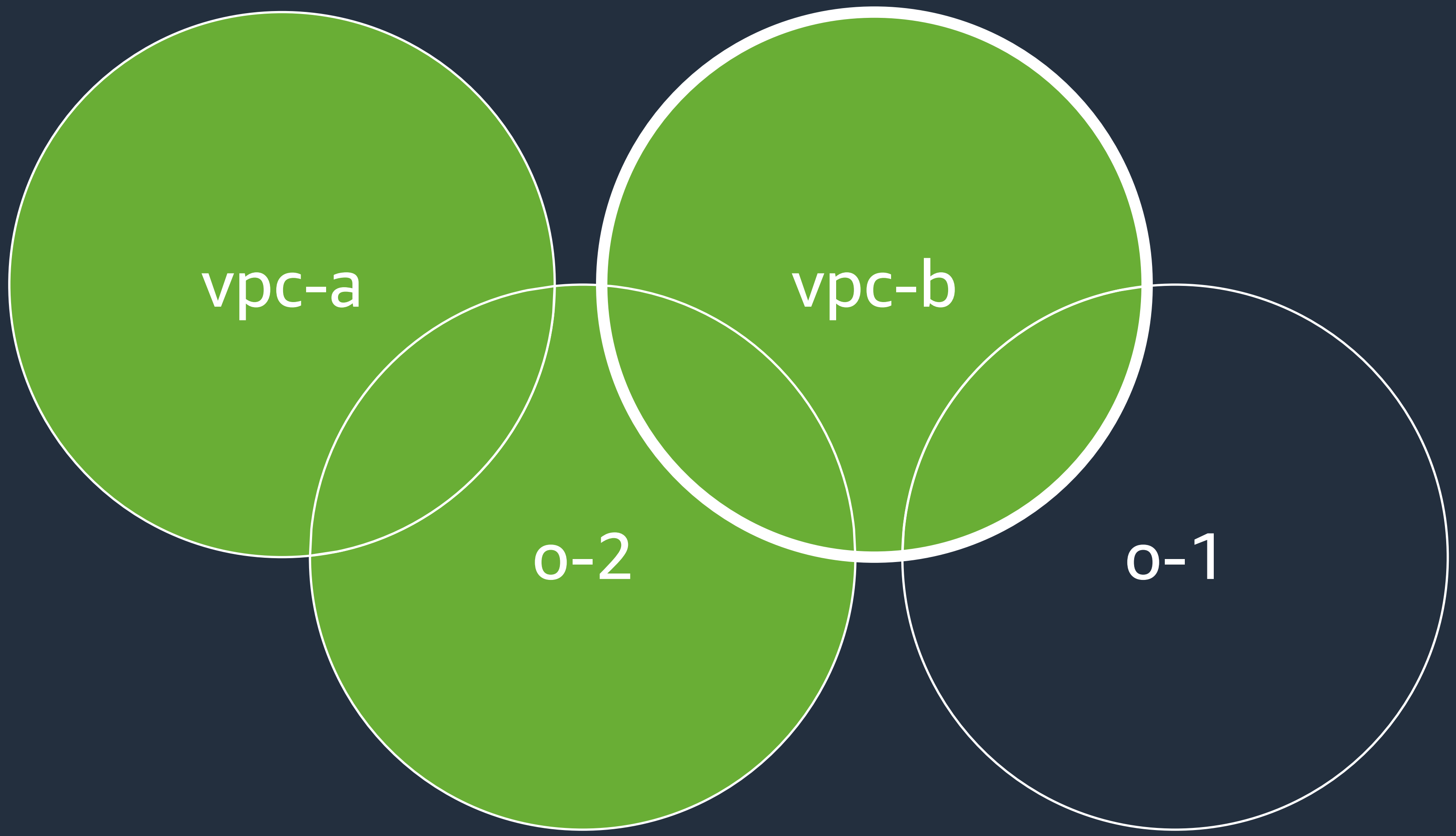
Policy example

```
- Effect: Allow
  Condition:
    StringEquals:
      SourceVpc:
      - “vpc-a”
      - “vpc-b”

- Effect: Allow
  Condition:
    StringEquals:
      PrincipalOrgID: “o-2”

- Effect: Deny
  Condition:
    StringEquals:
      SourceVpc: “vpc-b”
    StringNotEquals:
      PrincipalOrgID : “o-1”
```



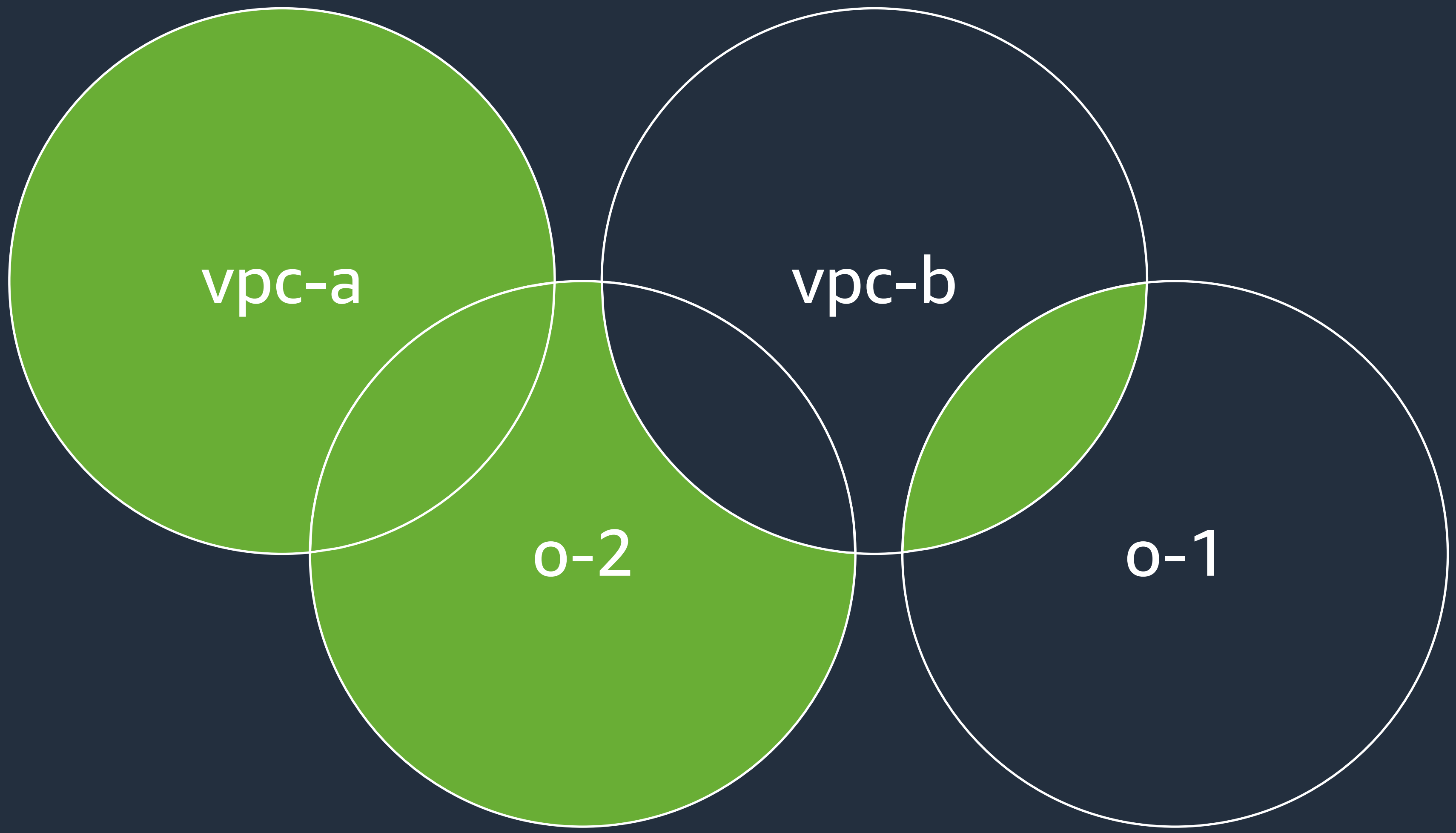
Policy example

```
- Effect: Allow
  Condition:
    StringEquals:
      SourceVpc:
      - “vpc-a”
      - “vpc-b”

- Effect: Allow
  Condition:
    StringEquals:
      PrincipalOrgID: “o-2”

- Effect: Deny
  Condition:
    StringEquals:
      SourceVpc: “vpc-b”
    StringNotEquals:
      PrincipalOrgID : “o-1”
```



Policy example

```
- Effect: Allow
  Condition:
    StringEquals:
      SourceVpc:
      - “vpc-a”
      - “vpc-b”

- Effect: Allow
  Condition:
    StringEquals:
      PrincipalOrgID: “o-2”

- Effect: Deny
  Condition:
    StringEquals:
      SourceVpc: “vpc-b”
    StringNotEquals:
      PrincipalOrgID : “o-1”
```



Semantic-based Automated Reasoning for AWS Access Policies using SMT

John Backes, Pauline Bolignano, Byron Cook, Catherine Dodge, Andrew Gacek,
Kasper Luckow, Neha Rungta, Oksana Tkachuk, Carsten Varming
Amazon Web Services

***Abstract*—Cloud computing provides on-demand access to IT resources via the Internet. Permissions for these resources are defined by expressive access control policies. This paper presents a formalization of the Amazon Web Services (AWS) policy language and a corresponding analysis tool, called ZELKOVA, for verifying policy properties. ZELKOVA encodes the semantics of policies into SMT, compares behaviors, and verifies properties. It provides users a sound mechanism to detect misconfigurations of their policies. ZELKOVA solves a PSPACE-complete problem and is invoked many millions of times daily.**

I. Introduction

Cloud computing provides on-demand access to IT resources via the Internet. The convenience of accessing resources in the cloud is made secure by user-specified *access control policies*. An access control policy is an expressive specification of what resources can be accessed, by whom, and under what conditions. Properly configured policies are an important part of an organization's security posture. The

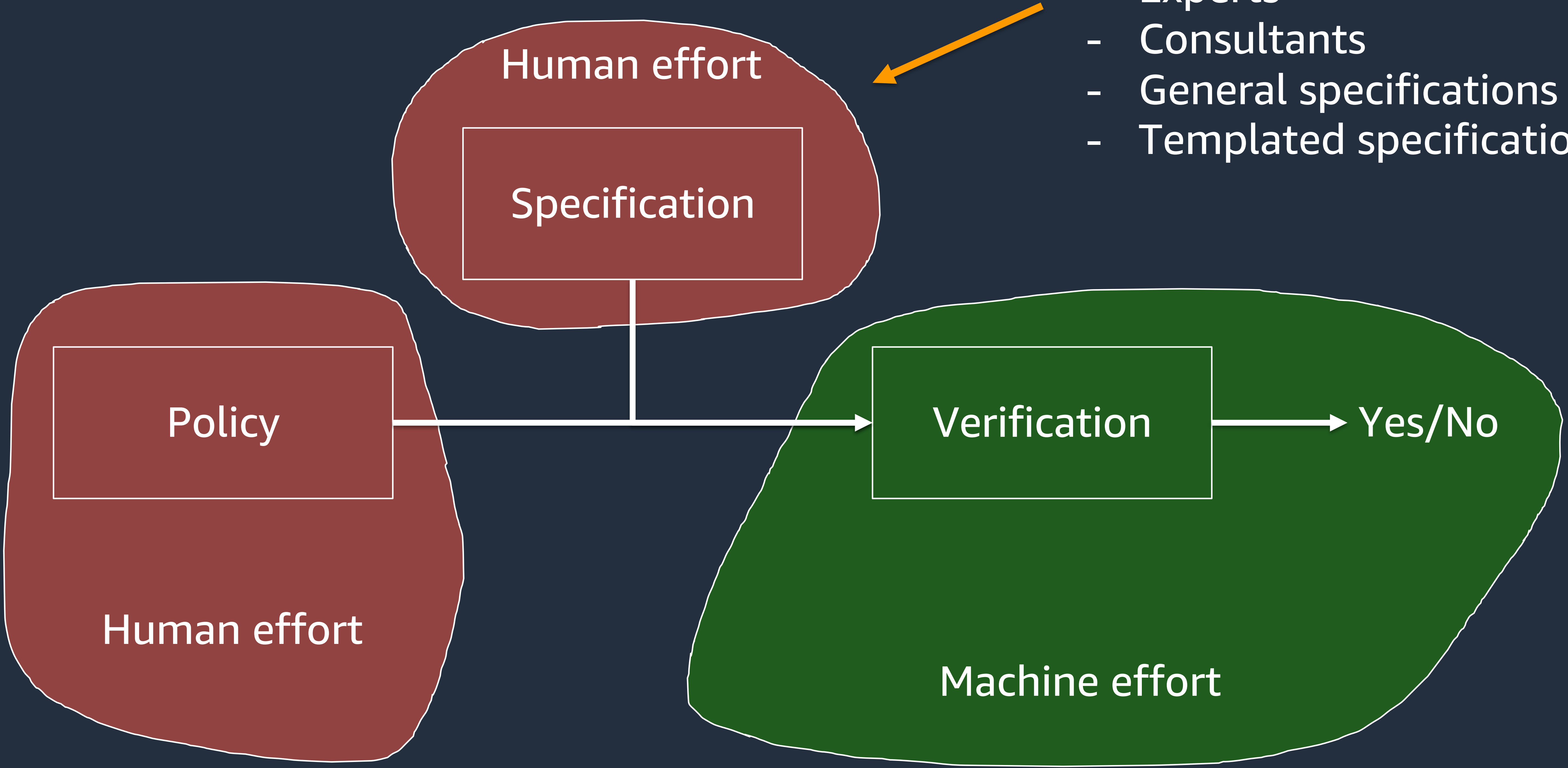
In this paper, we present the development and application of ZELKOVA, a policy analysis tool designed to reason about the semantics of AWS access control policies. ZELKOVA translates policies and properties into Satisfiability Modulo Theories (SMT) formulas and uses SMT solvers to check the validity of the properties. We use off-the-shelf solvers and an in-house extension of Z3 called Z3AUTOMATA.

ZELKOVA reasons about all possible permissions allowed by a policy in order to verify properties. For example, ZELKOVA can answer the questions "Is this resource accessible by a particular user?" and "Can an arbitrary user write to this resource?". The property to be verified is specified in the policy language itself, eliminating the need for a different specification or formalism for properties. In addition, ZELKOVA provides many built-in checks for common properties.

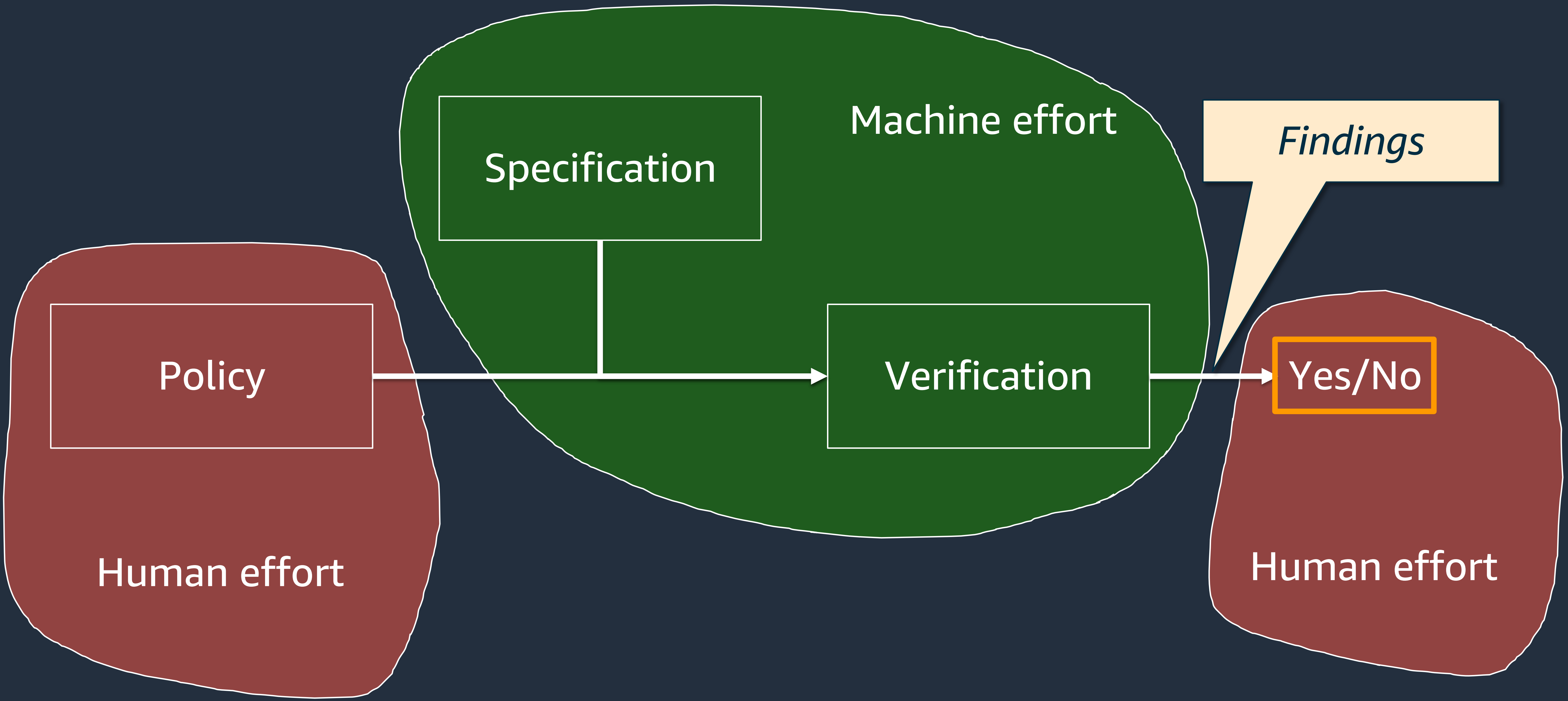
The SMT encoding uses the theory of strings, regular expressions, bit vectors, and integer comparisons. The use of

Traditional verification approach

- Experts
- Consultants
- General specifications
- Templated specifications



Access Analyzer verification approach



Desired properties of findings

Sound – *Every* access is represented by *some* finding

Precise – findings *adhere closely* to the allowed access

Compact – the set of findings is *small*

Stratified predicate abstraction

```
- Effect: Allow
  Condition:
    StringEquals:
      SourceVpc:
      - “vpc-a”
      - “vpc-b”

- Effect: Allow
  Condition:
    StringEquals:
      PrincipalOrgID: “o-2”

- Effect: Deny
  Condition:
    StringEquals:
      SourceVpc: “vpc-b”
    StringNotEquals:
      PrincipalOrgID : “o-1”
```

$$p_{\top} \equiv \top$$

$$p_a \equiv SourceVpc = \text{“vpc-a”}$$

$$p_b \equiv SourceVpc = \text{“vpc-b”}$$

$$q_{\top} \equiv \top$$

$$q_1 \equiv PrincipalOrgID = \text{“o-1”}$$

$$q_2 \equiv PrincipalOrgID = \text{“o-2”}$$

Stratified predicate abstraction

```
- Effect: Allow
  Condition:
    StringEquals:
      SourceVpc:
      - “vpc-a”
      - “vpc-b”

- Effect: Allow
  Condition:
    StringEquals:
      PrincipalOrgID: “o-2”

- Effect: Deny
  Condition:
    StringEquals:
      SourceVpc: “vpc-b”
    StringNotEquals:
      PrincipalOrgID : “o-1”
```

everybody has access

Organization o-1 has access

$p_\top \wedge q_\top$	$p_\top \wedge q_1$	$p_\top \wedge q_2$
$p_a \wedge q_\top$	$p_a \wedge q_1$	$p_a \wedge q_2$
$p_b \wedge q_\top$	$p_b \wedge q_1$	$p_b \wedge q_2$

Organization o-1 coming from vpc-b has access

Stratified predicate abstraction

```
- Effect: Allow
  Condition:
    StringEquals:
      SourceVpc:
      - “vpc-a”
      - “vpc-b”

- Effect: Allow
  Condition:
    StringEquals:
      PrincipalOrgID: “o-2”

- Effect: Deny
  Condition:
    StringEquals:
      SourceVpc: “vpc-b”
    StringNotEquals:
      PrincipalOrgID : “o-1”
```

? $p_\top \wedge q_\top$	$p_\top \wedge q_1$	$p_\top \wedge q_2$
$p_a \wedge q_\top$	$p_a \wedge q_1$	$p_a \wedge q_2$
$p_b \wedge q_\top$	$p_b \wedge q_1$	$p_b \wedge q_2$

Stratified predicate abstraction

```
- Effect: Allow
  Condition:
    StringEquals:
      SourceVpc:
      - “vpc-a”
      - “vpc-b”

- Effect: Allow
  Condition:
    StringEquals:
      PrincipalOrgID: “o-2”

- Effect: Deny
  Condition:
    StringEquals:
      SourceVpc: “vpc-b”
    StringNotEquals:
      PrincipalOrgID : “o-1”
```

$p_\top \wedge q_\top$	$p_\top \wedge q_1$	$p_\top \wedge q_2$
? $p_a \wedge q_\top$	$p_a \wedge q_1$	$p_a \wedge q_2$
$p_b \wedge q_\top$	$p_b \wedge q_1$	$p_b \wedge q_2$

Stratified predicate abstraction

```
- Effect: Allow
  Condition:
    StringEquals:
      SourceVpc:
      - “vpc-a”
      - “vpc-b”

- Effect: Allow
  Condition:
    StringEquals:
      PrincipalOrgID: “o-2”

- Effect: Deny
  Condition:
    StringEquals:
      SourceVpc: “vpc-b”
    StringNotEquals:
      PrincipalOrgID : “o-1”
```

$p_\top \wedge q_\top$	$p_\top \wedge q_1$	$p_\top \wedge q_2$
$p_a \wedge q_\top$	$p_a \wedge q_1$	$p_a \wedge q_2$
$p_b \wedge q_\top$	$p_b \wedge q_1$	$p_b \wedge q_2$

Stratified predicate abstraction

```
- Effect: Allow
  Condition:
    StringEquals:
      SourceVpc:
      - "vpc-a"
      - "vpc-b"

- Effect: Allow
  Condition:
    StringEquals:
      PrincipalOrgID: "o-2"

- Effect: Deny
  Condition:
    StringEquals:
      SourceVpc: "vpc-b"
    StringNotEquals:
      PrincipalOrgID : "o-1"
```

$p_\top \wedge q_\top$	$p_\top \wedge q_1$	$p_\top \wedge q_2$
$p_a \wedge q_\top$	$p_a \wedge q_1$	$p_a \wedge q_2$
$p_b \wedge q_\top$	$p_b \wedge q_1$	$p_b \wedge q_2$

Stratified predicate abstraction

```
- Effect: Allow
  Condition:
    StringEquals:
      SourceVpc:
      - “vpc-a”
      - “vpc-b”

- Effect: Allow
  Condition:
    StringEquals:
      PrincipalOrgID: “o-2”

- Effect: Deny
  Condition:
    StringEquals:
      SourceVpc: “vpc-b”
    StringNotEquals:
      PrincipalOrgID : “o-1”
```

$$\sigma_1 \equiv p_a \wedge q_\top \equiv SourceVpc = \text{“vpc-a”}$$

$$\sigma_2 \equiv p_\top \wedge q_2 \equiv PrincipalOrgID = \text{“o-2”}$$

$$\sigma_3 \equiv p_b \wedge q_1 \equiv SourceVpc = \text{“vpc-b”} \wedge PrincipalOrgID = \text{“o-1”}$$

$$\Sigma \equiv \{\sigma_1, \sigma_2, \sigma_3\}$$

Formal properties of findings

Sound – *Every* access is represented by *some* finding

Coverage – $\gamma(p) \subseteq \gamma(\Sigma)$

Precise – findings *adhere closely* to the allowed access

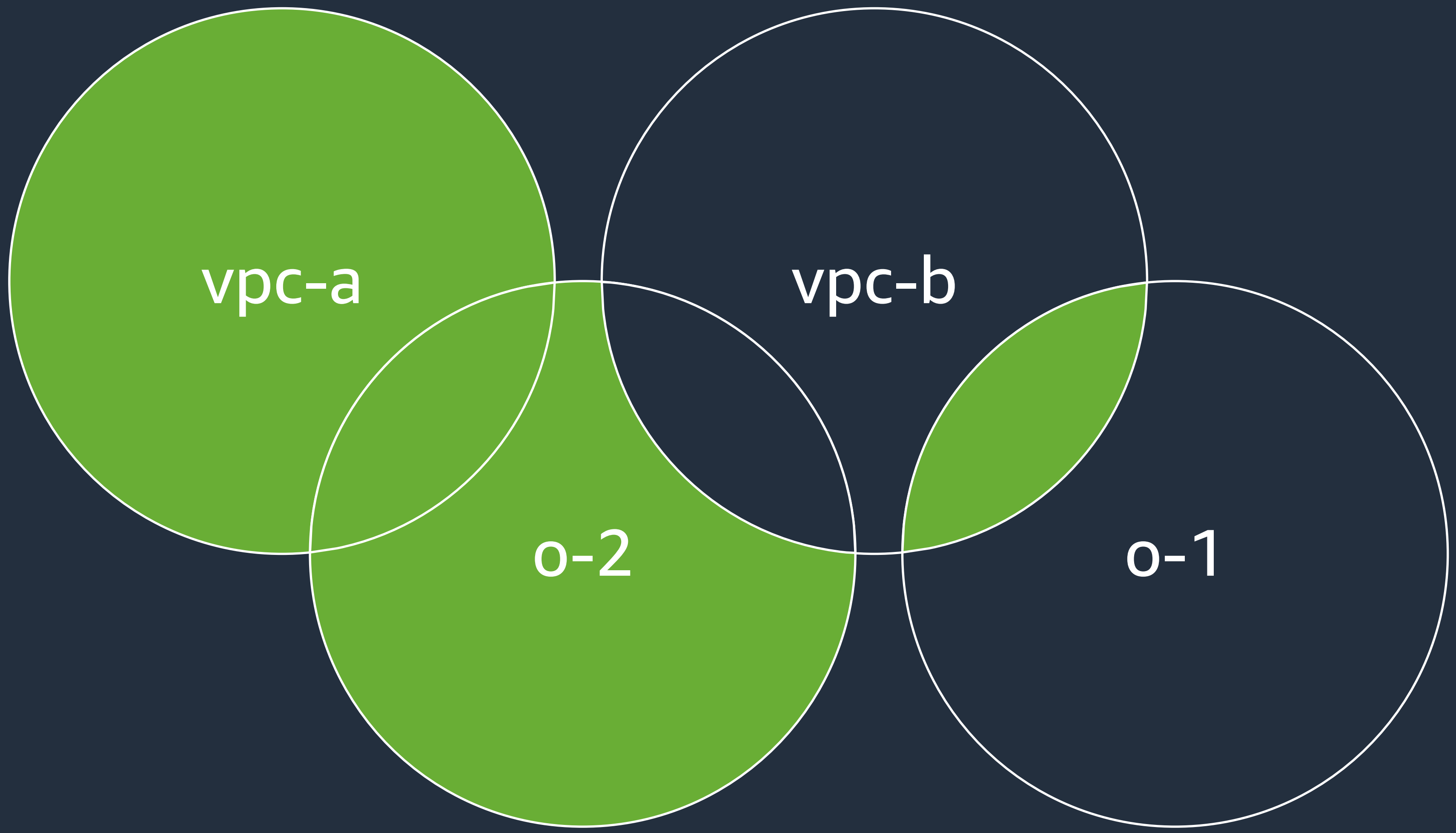
Irreducible – $\exists r \in \gamma(p) \cap \gamma(\sigma). \forall \sigma' \sqsubset \sigma. r \notin \gamma(\sigma')$

Compact – the set of findings is *small*

Minimality – $\forall \Sigma' \subset \Sigma. \gamma(\Sigma') \subset \gamma(\Sigma)$

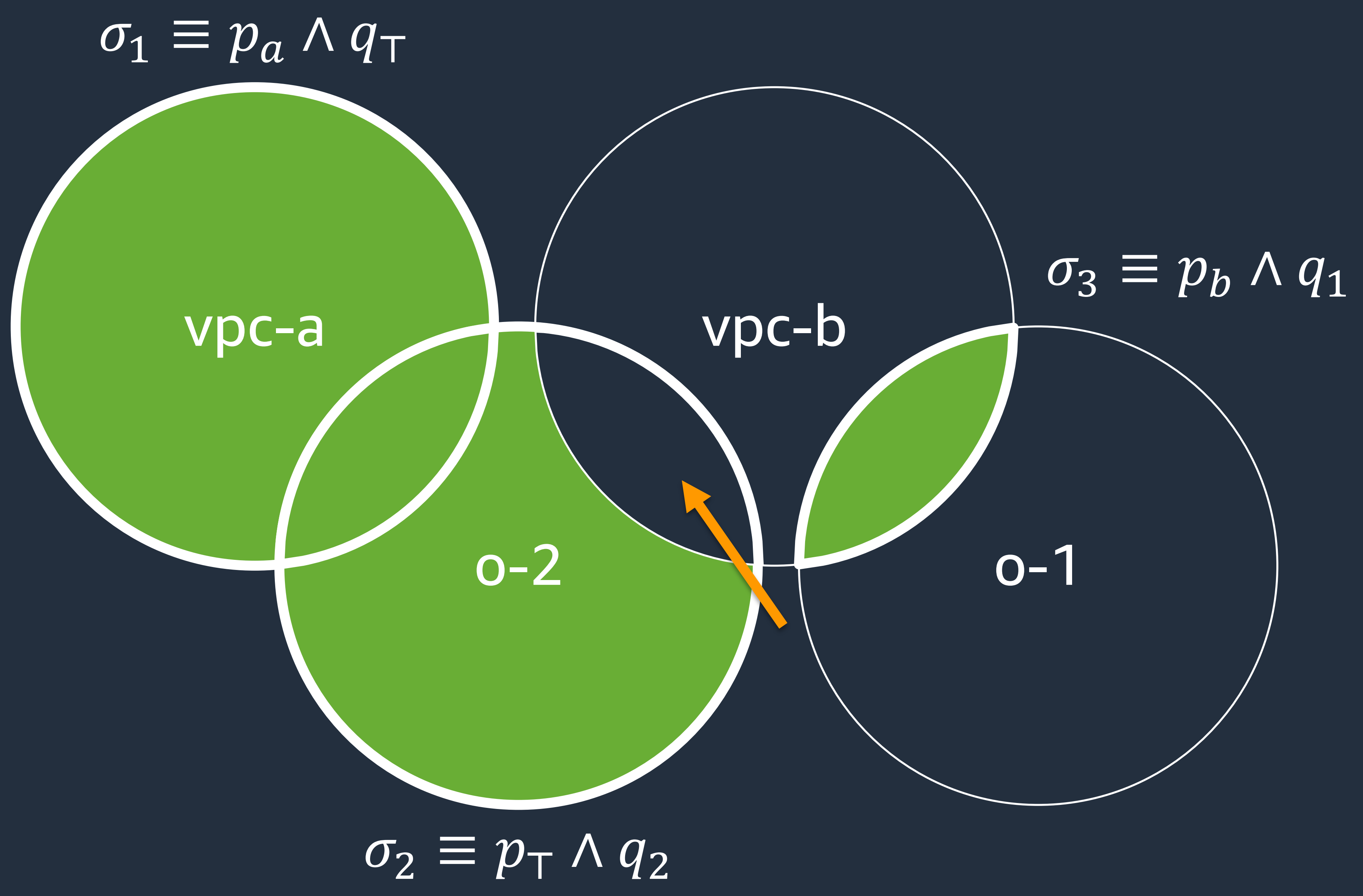
Sound, precise, compact

```
- Effect: Allow
  Condition:
    StringEquals:
      SourceVpc:
      - “vpc-a”
      - “vpc-b”

- Effect: Allow
  Condition:
    StringEquals:
      PrincipalOrgID: “o-2”

- Effect: Deny
  Condition:
    StringEquals:
      SourceVpc: “vpc-b”
    StringNotEquals:
      PrincipalOrgID : “o-1”
```



Sound, precise, compact

```
- Effect: Allow
  Condition:
    StringEquals:
      SourceVpc:
      - "vpc-a"
      - "vpc-b"

- Effect: Allow
  Condition:
    StringEquals:
      PrincipalOrgID: "o-2"

- Effect: Deny
  Condition:
    StringEquals:
      SourceVpc: "vpc-b"
    StringNotEquals:
      PrincipalOrgID : "o-1"
```



Simple, modular, sensible

```
- Effect: Allow
  Condition:
    StringEquals:
      SourceVpc:
      - “vpc-a”
      - “vpc-b”

- Effect: Allow
  Condition:
    StringEquals:
      PrincipalOrgID: “o-2”

- Effect: Deny
  Condition:
    StringEquals:
      SourceVpc: “vpc-b”
    StringNotEquals:
      PrincipalOrgID : “o-1”
```

$$SourceVpc = \text{“vpc-a”}$$ ✓

$$PrincipalOrgID = \text{“o-2”}$$ ✗

$$SourceVpc = \text{“vpc-b”} \wedge PrincipalOrgID = \text{“o-1”}$$?

Simple, modular, sensible

```
- Effect: Allow
  Condition:
    StringLike:
      PrincipalOrgID:
      - "o-123"
      - "o-456"
      - "o-78*"
```

? $q_{\top} \equiv \top$

$$q_{123} \equiv PrincipalOrgID = \text{“o-123”}$$

$$q_{456} \equiv PrincipalOrgID = \text{“o-456”}$$

Simple, modular, sensible

```
- Effect: Allow
  Condition:
    StringLike:
      PrincipalOrgID:
      - "o-123"
      - "o-456"
      - "o-78*"
```

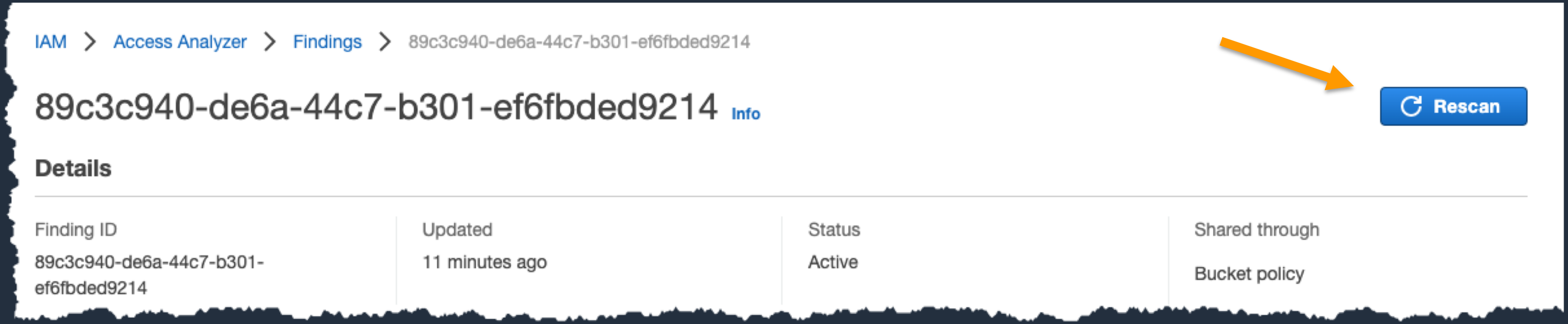
$$q_{\top} \equiv \top$$

$$q_{123} \equiv PrincipalOrgID = \text{“o-123”}$$

$$q_{456} \equiv PrincipalOrgID = \text{“o-456”}$$

Demo

Benefits of Automated Reasoning



Automated Analysis of AWS Access Control

https://aws.amazon.com/security/provable-security/

> **Matt Weagle** @mweagle
>
> "All possible access paths are verified by mathematical proofs" 😍

> **⚡ Jean Yang ⚡** @jeanqasaur
>
> So cool that AWS now uses formal methods to analyze IAM Access!!

> **bjorg** @bjorg
>
> Oh, this is a big deal! Understanding IAM policy consequences is essential. This tool should make it a lot easier!
>
> Identify Unintended Resource Access with AWS Identity and Access Management (IAM) Access Analyzer

> **Robert Syvarth** @rsyvarth
>
> 🔒 The new IAM Access Analyzer is awsome! Glad to see AWS focusing on making it easier to verify workload security

> **Brandon West** @bwest
>
> Today we launched a first-of-its-kind service that uses automated reasoning to identify unintended resources access paths. It's pretty badass.

> **Lars** @bob5ec
>
> "IAM Access Analyzer provides answers of who has public and cross-account access to AWS resources."
>
> Formal methodes for the win:
> "IAM Access Analyzer uses a form of mathematical analysis called automated reasoning, which applies logic and mathematical inference to ..."
>
> **What's New on AWS** @awswhatsnew
> Introducing AWS Identity and Access Management (IAM) Access Analyzer

> **Danielle Ruderman** @rudermires
>
> New launch today—AWS IAM Access Analyzer—exciting provable security work out of the AWS Automated Reasoning Group #reInvent2019

> **stephenschmidt** @StephenSchmidt
>
> You don't need to be a logician to use IAM Access Analyzer. Turn this on now, it's available at no charge! #reInvent #provablesecurity

> **Brigid Johnson** @bjohnso5y
>
> Just launched! IAM Access Analyzer: continuously monitor, comprehensively analyze, and gain certainty for cross account access controls. All backed by fancy math using automated reasoning. Come see it live in SEC316. But really go turn it on, it's quick.