



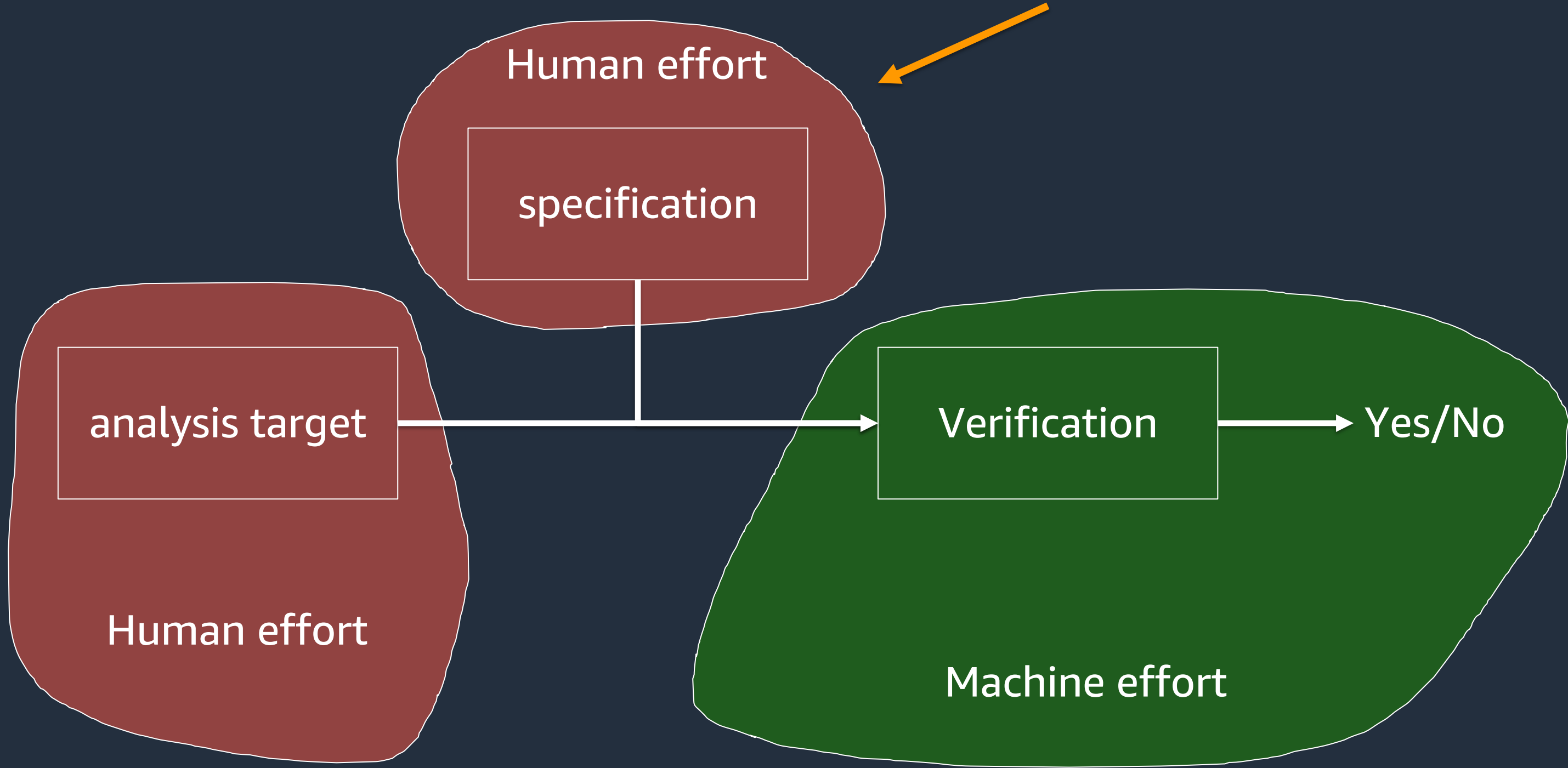
Demystify Your Trust Boundary with Interactive Refinement

Vaibhav Sharma

Automated Reasoning in Consumer Payments, Amazon.com

May 19, 2022

Traditional verification approach



Access Analyzer

Monitor access to resources

Create analyzer

How it works



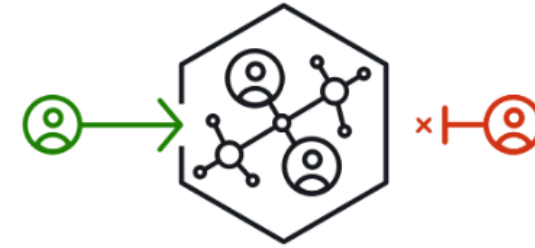
1 Create an analyzer

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

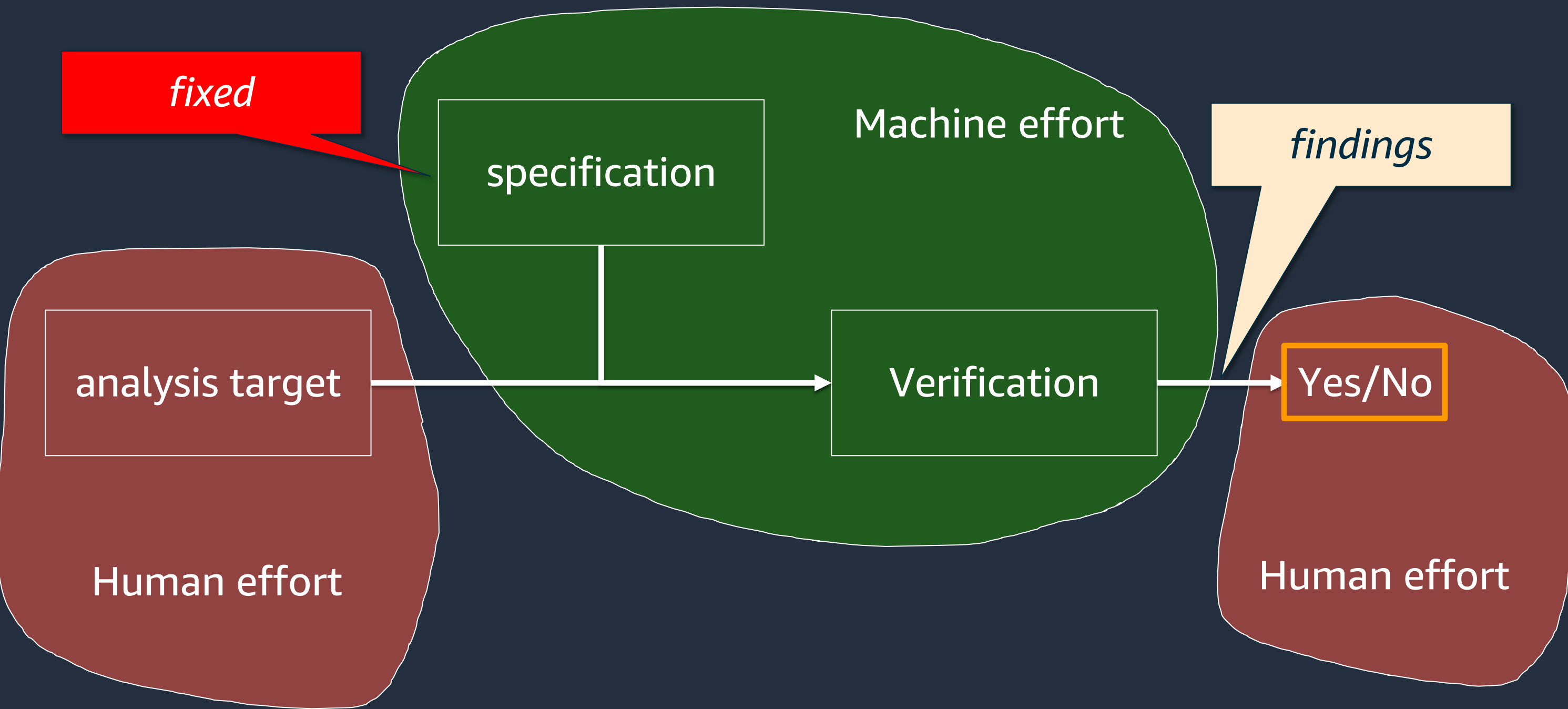
When Access Analyzer finds a policy that allows access to a resource from outside of your zone of trust, it generates an active finding. Findings include details about the access so that you can take action.

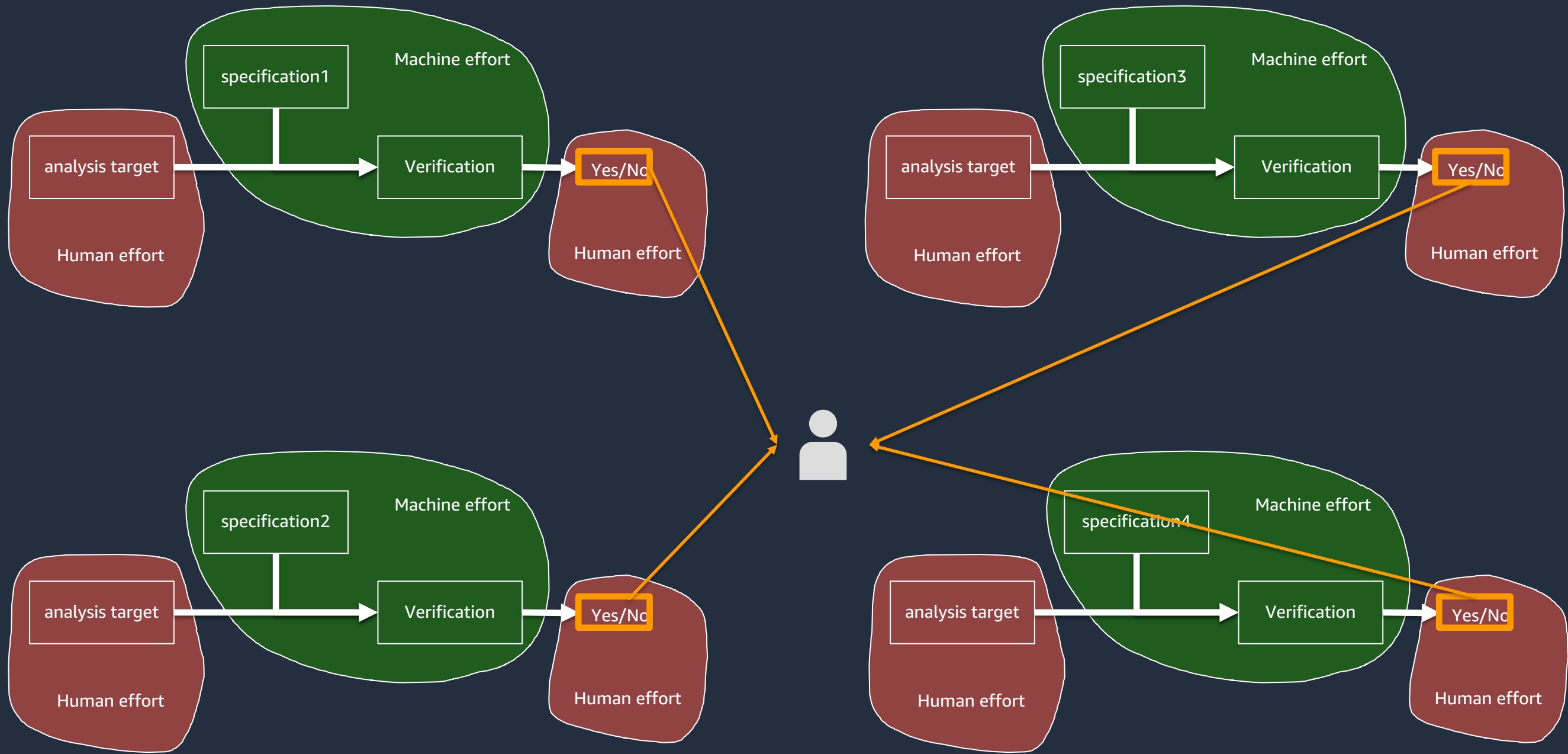


3 Take action

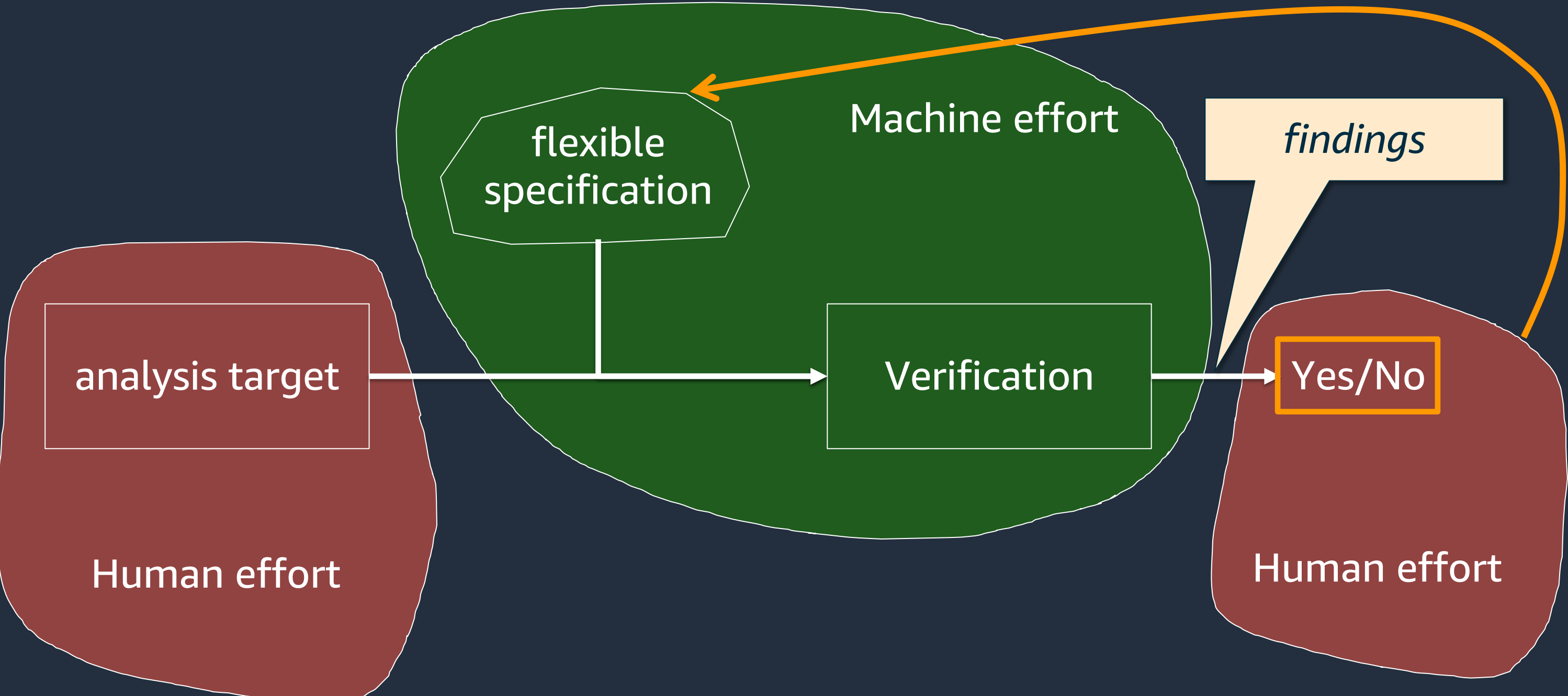
If the access is intended, you can archive the finding so that you can focus on reviewing active findings. If the access is not intended, you can resolve the finding by modifying the policy to remove access to the resource.

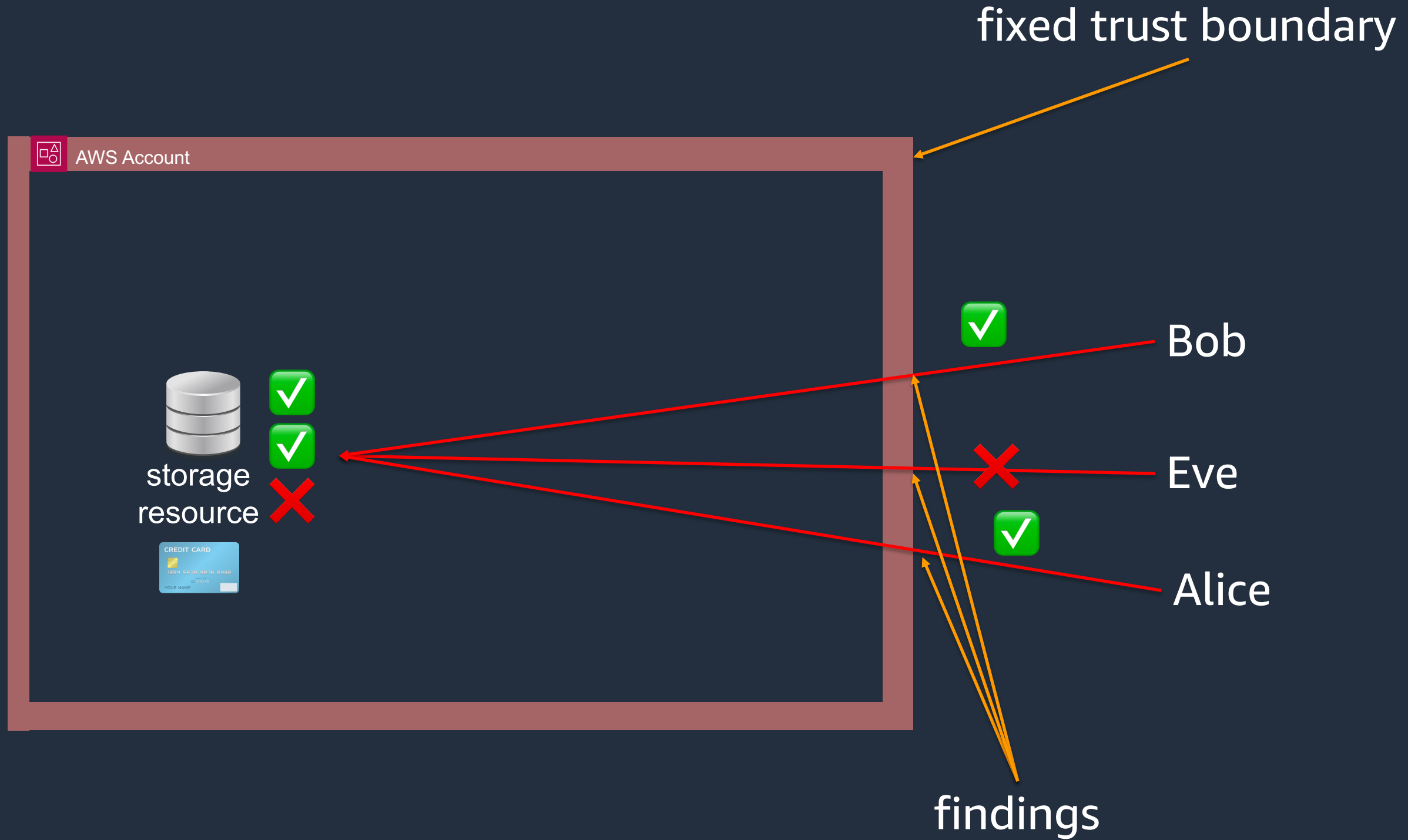
Fixed specification approach





Interactive verification approach



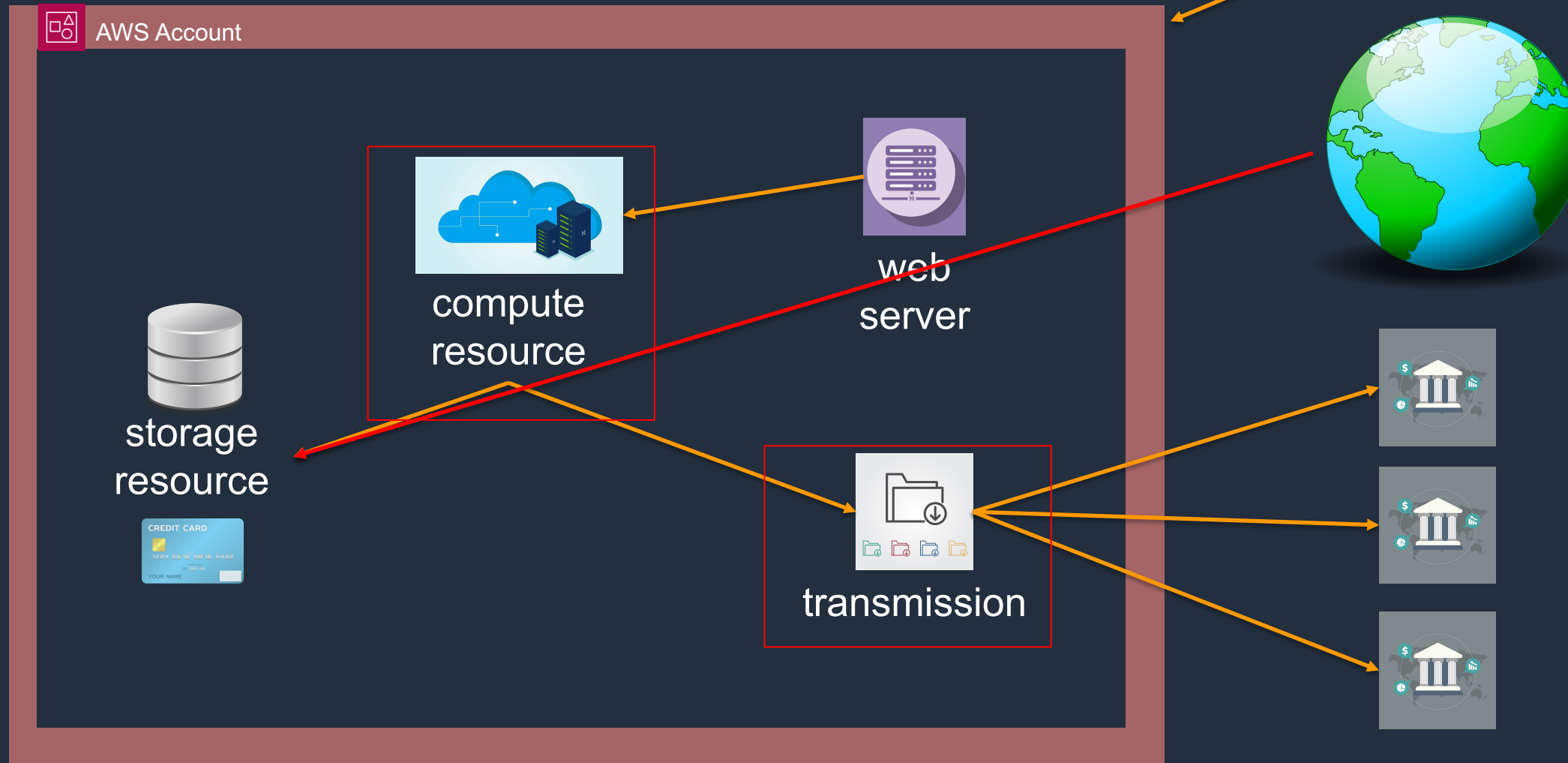




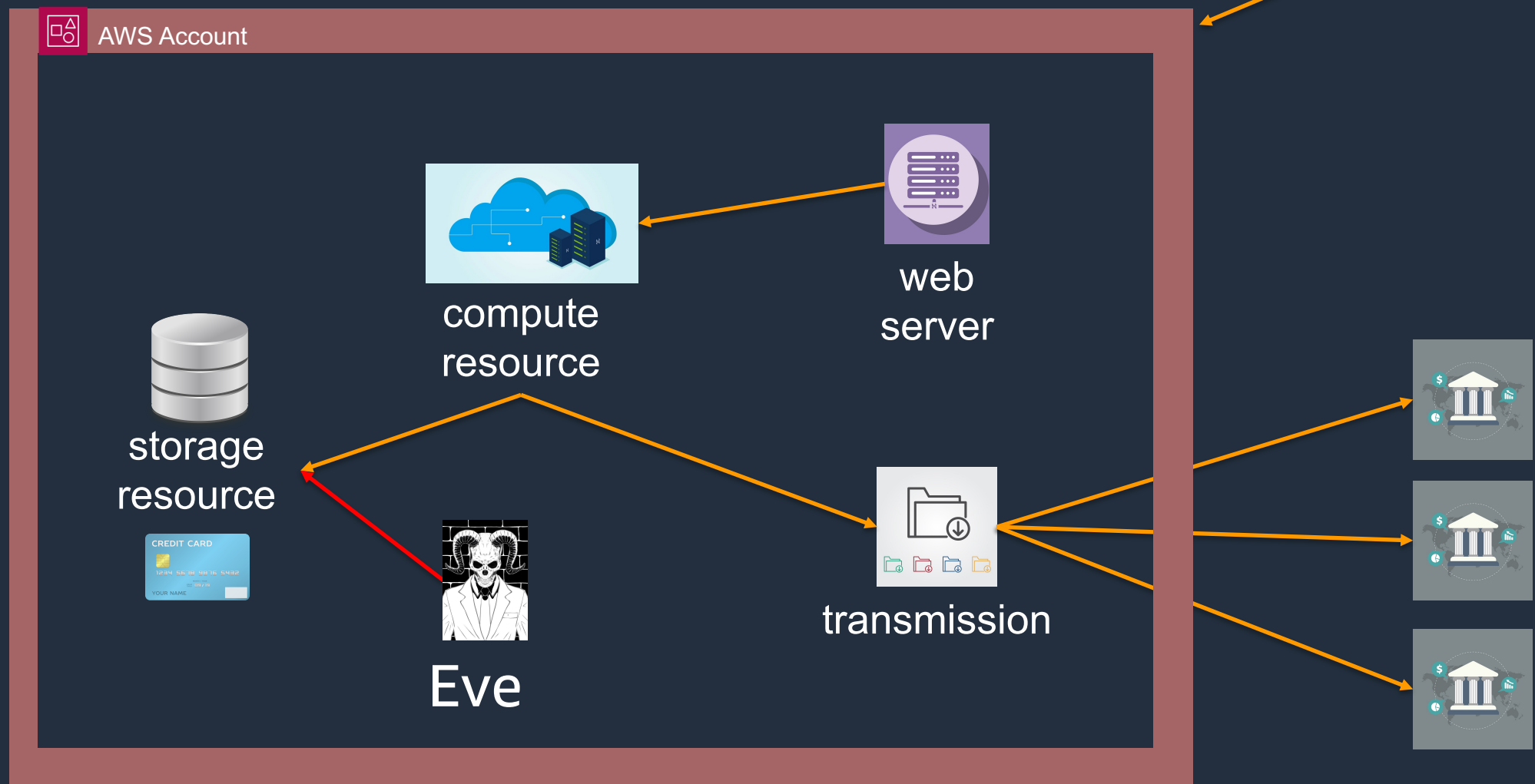
Tenet

Customer trust in Amazon's stewardship of their money is paramount. We work to **secure our customers' funds and personal financial information above all else.**

fixed trust boundary



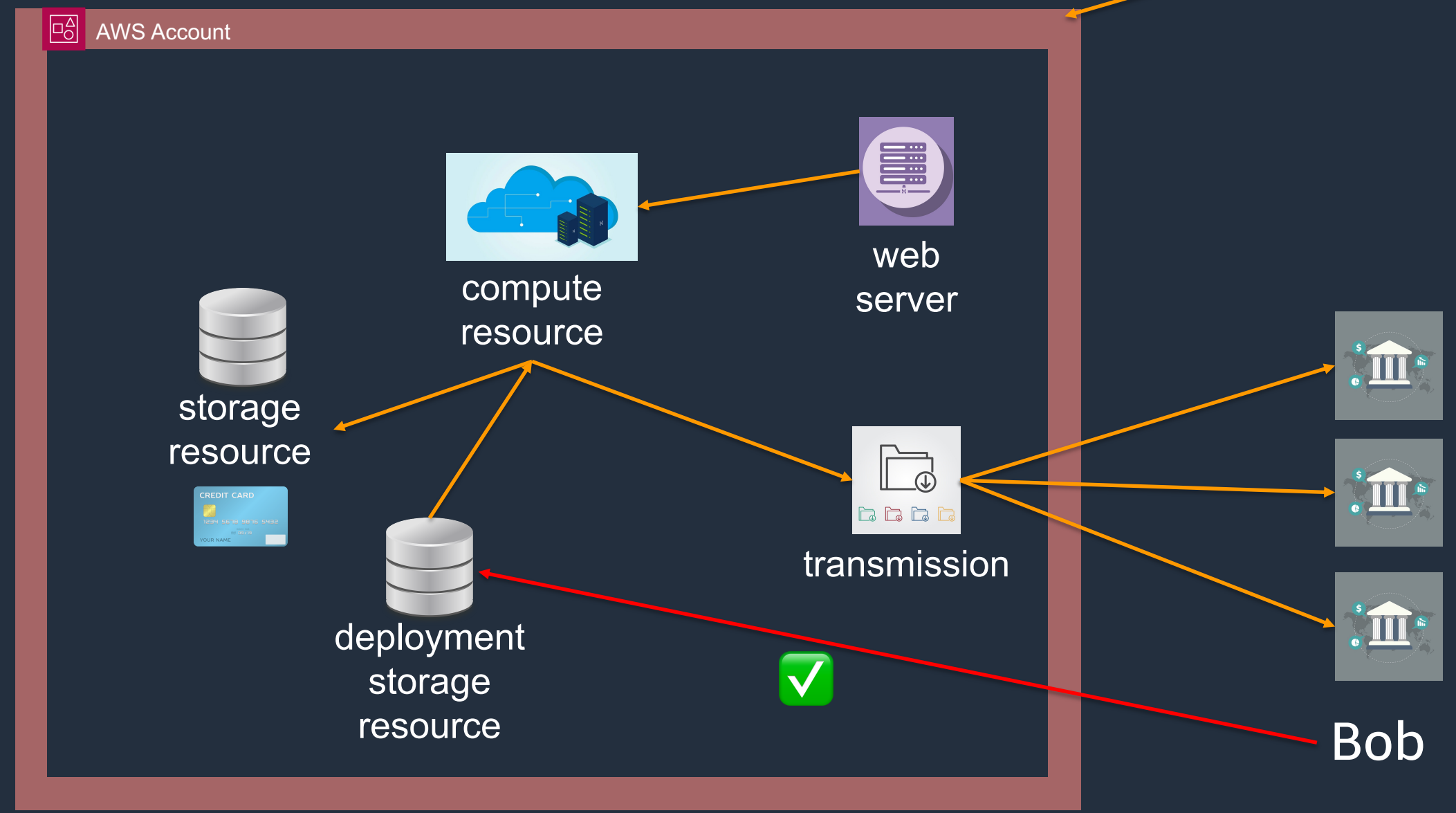
fixed trust boundary



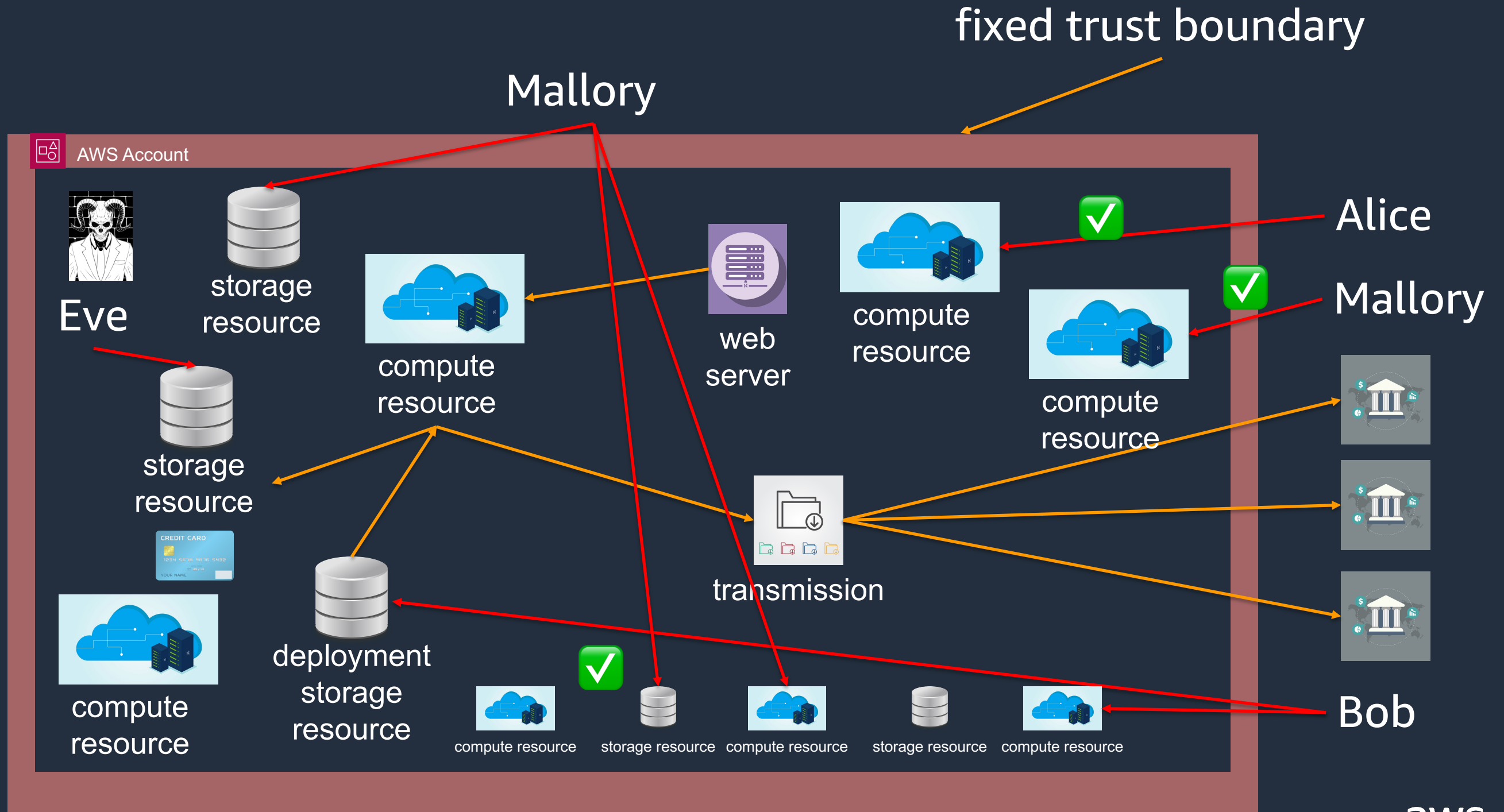
Tenet

We obsess over insider threats as much as external threats. We build our systems in a way that no operator access to production is required. We automate sensitive tasks, and enforce multi-person approval to perform them.

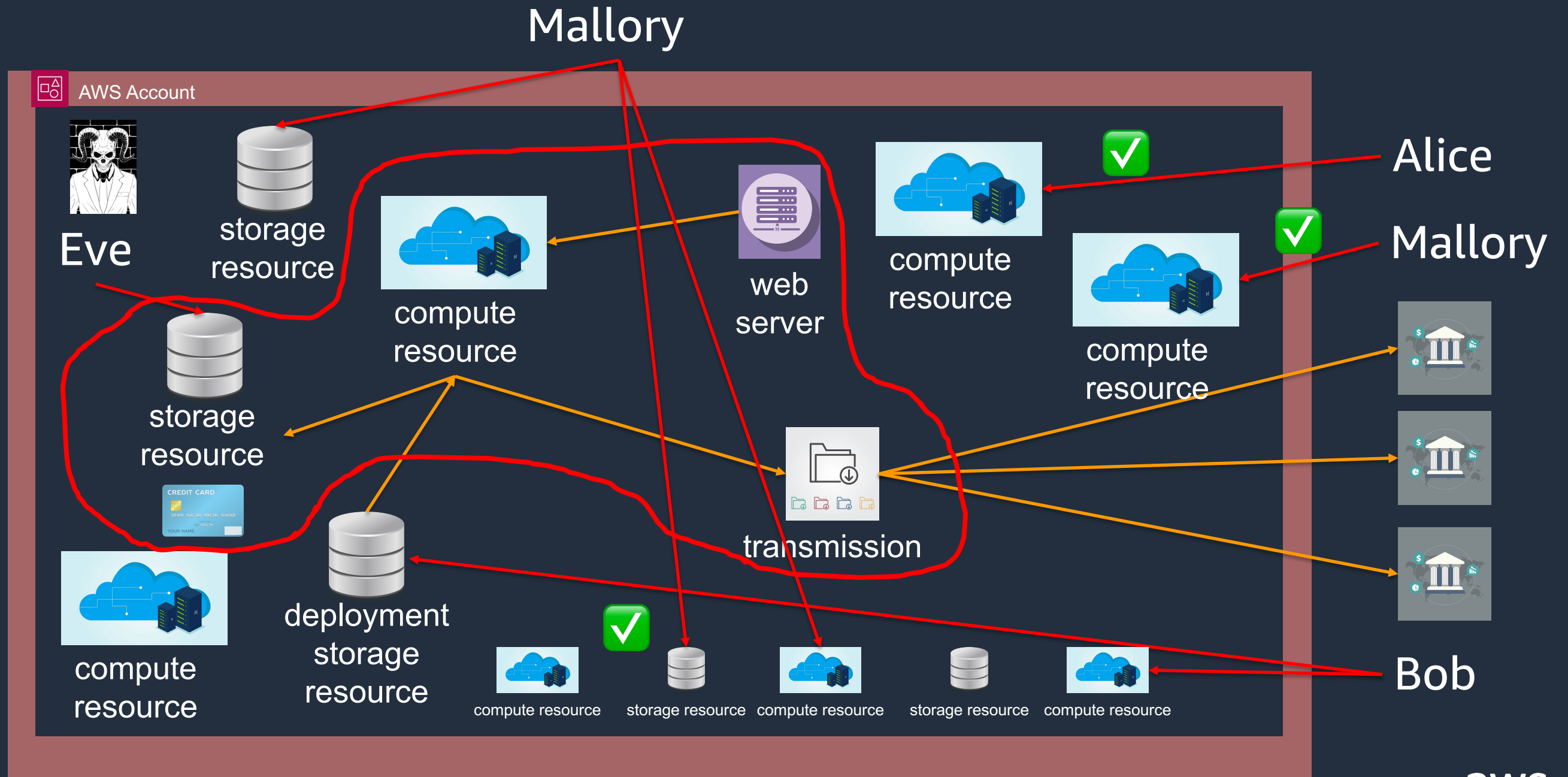
fixed trust boundary



Fixed trust boundary



Flexible trust boundary





empty red zone

1. $RZ = \text{set of resources}$
2. $\text{findings}(RZ) = \{(src, dst) \mid src \in RZ \ \&\& \ dst \notin RZ \ \&\& \ \text{allowed}(src, dst)\}$
3. $\text{allowed}(src, dst) =$
 - $\text{allowed}_{perm}(src, dst) \ ||$
 - $\text{allowed}_{network}(src, dst) \ ||$
 - $\text{allowed}_{code}(src, dst)$

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.

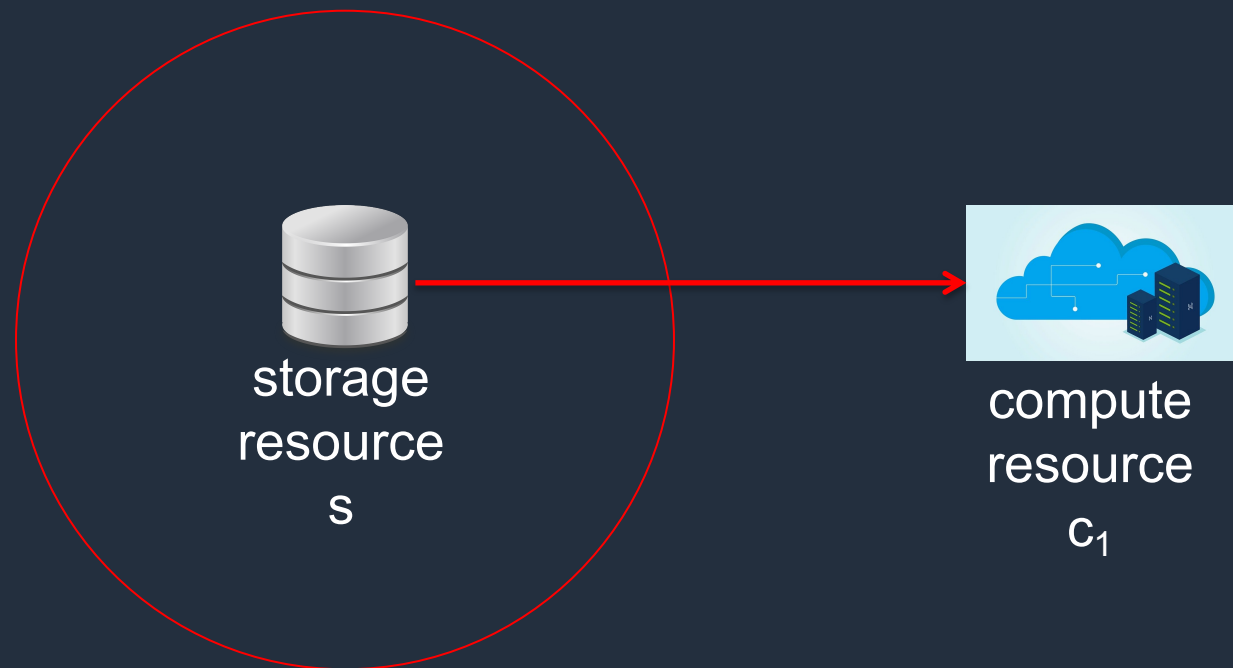
ZELKOVA reasons about all possible permissions allowed by a policy in order to verify properties. For example, ZELKOVA can answer the questions “Is this resource accessible by a particular user?” and “Can an arbitrary user write to this resource?”. The property to be verified is specified in the policy 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 the wildcards `*` (any number of characters) and `?` (exactly one



*A session p





1. $RZ = \{s\}$

2. $\text{findings}(RZ) = \{(s, c_1)\}$

Reachability Analysis for AWS-based Networks

J. Backes¹, S. Bayless¹⁴, B. Cook¹², C. Dodge¹, A. Gacek¹, A.J. Hu⁴, T. Kahsai¹, B. Kocik¹, E. Kotelnikov¹³, J. Kukovec¹⁵, S. McLaughlin¹, J. Reed⁶, N. Rungta¹, J. Sizemore¹, M. Stalzer¹, P. Srinivasan¹, P. Subotić¹², C. Varming¹, B. Whaley¹

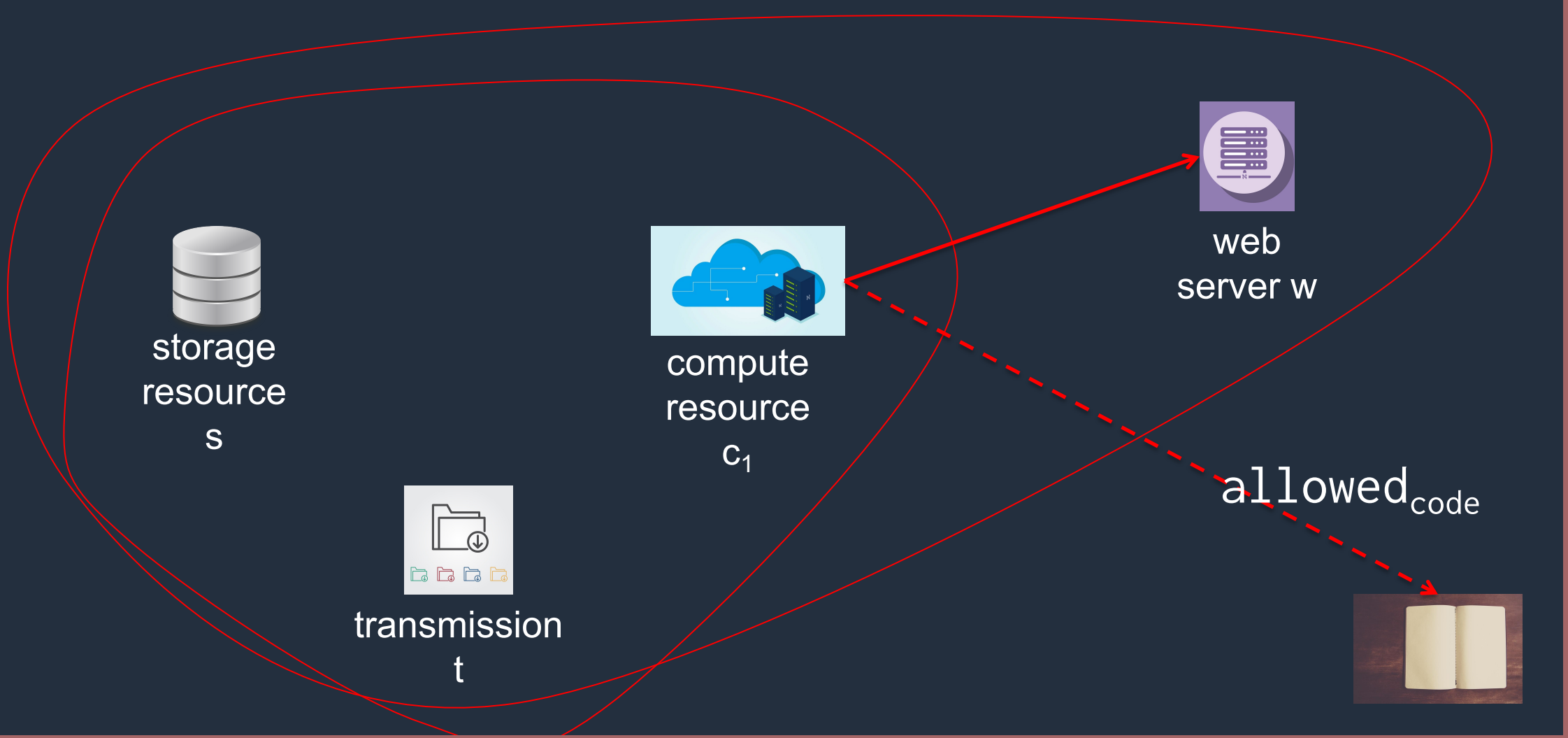
¹Amazon, ²University College London, ³Chalmers University of Technology, ⁴University British Columbia, ⁵TU Wien, ⁶Semmler Inc

Abstract. Cloud services provide the ability to provision virtual networked infrastructure on demand over the Internet. The rapid growth of these virtually provisioned cloud networks has increased the demand for automated reasoning tools capable of identifying misconfigurations or security vulnerabilities. This type of automation gives customers the assurance they need to deploy sensitive workloads. It can also reduce the cost and time-to-market for regulated customers looking to establish compliance certification for cloud-based applications. In this industrial case-study, we describe a new network reachability reasoning tool, called TIROS, that uses off-the-shelf automated theorem proving tools to fill this need. TIROS is the foundation of a recently introduced network security analysis feature in the *Amazon Inspector* service now available to millions of customers building applications in the cloud. TIROS is also used within Amazon Web Services (AWS) to automate the checking of compliance certification and adherence to security invariants for many AWS services that build on existing AWS networking features.

 AWS Account

1. RZ =
2. find

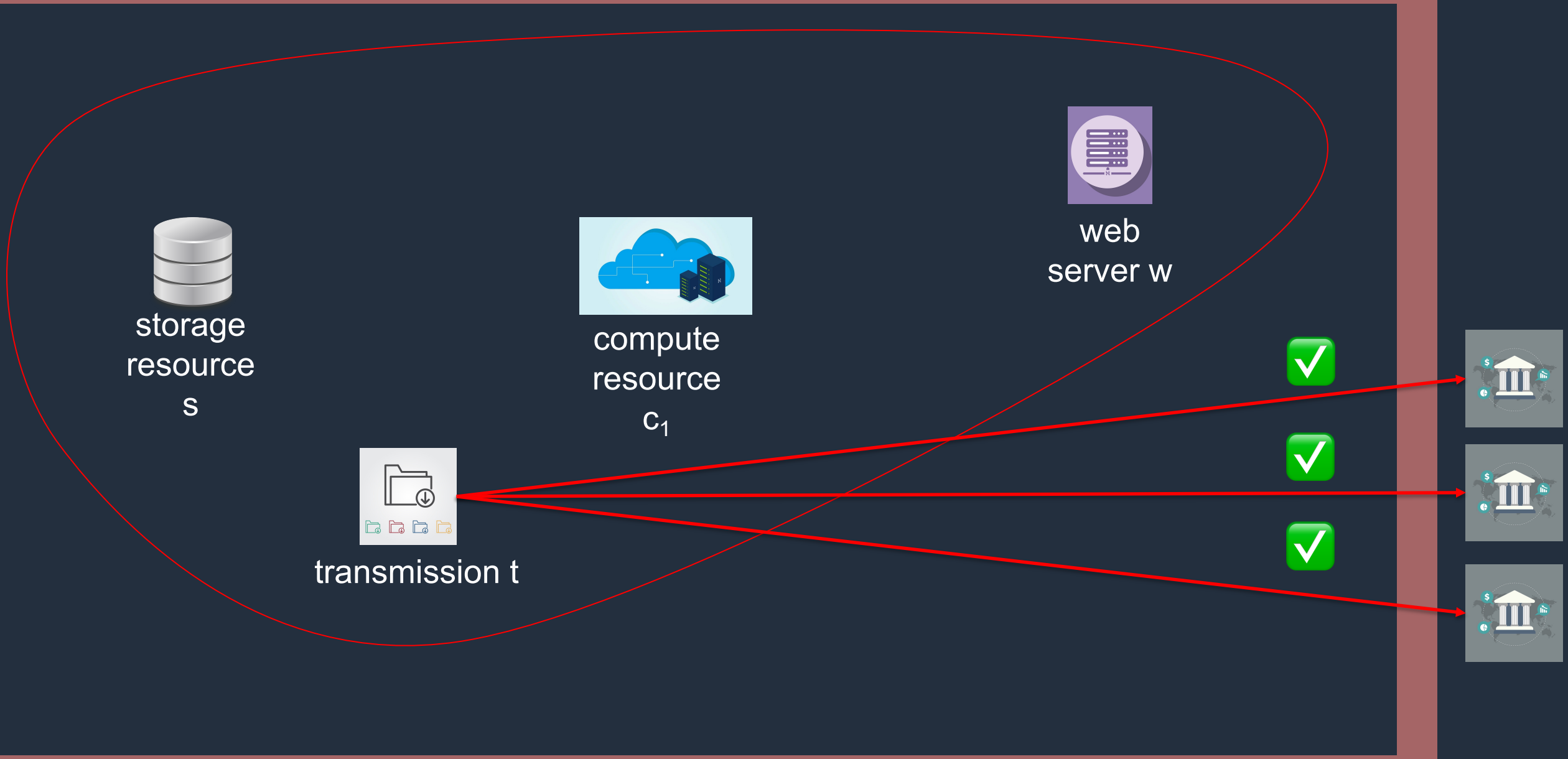
AWS Account



Code analysis for red zones

```
1. try {
2.   String tokenized = tokenizer.tokenize(creditCardNum); // creditCardNum is sensitive
3.   log.info("Tokenized : {}", tokenized); // no issue here
4. } catch (Exception ex) {
5.   log.error("Failed to tokenize credit card"); // no issue here
6. }
```


AWS Account



Automatic Specification

Fixed trust boundary

The only people outside of my **trust boundary** who have access are **X, Y, and Z**



Interactive trust boundary

The only people outside of my **trust boundary** who have access are **X, Y, and Z**



AWS Account



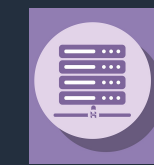
Eve



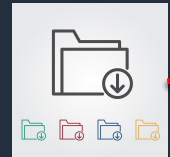
storage resource



compute resource



web server



transmission



Thank you!

Vaibhav Sharma

Contact information

svaib@amazon.com

Learn more:

Blogs

1. <https://aws.amazon.com/blogs/security/protect-sensitive-data-in-the-cloud-with-automated-reasoning-zelkova/>
2. <https://aws.amazon.com/blogs/aws/amazon-s3-block-public-access-another-layer-of-protection-for-your-accounts-and-buckets/>

Papers

1. <https://www.amazon.science/publications/reachability-analysis-for-aws-based-networks>
2. <https://www.amazon.science/publications/stratified-abstraction-of-access-control-policies>

Documentation

1. <https://aws.amazon.com/security/provable-security/>
2. <https://www.amazon.science/research-areas/automated-reasoning>

What is cloud computing?

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

