Cloud Resource Protection via Automated Security Property Reasoning

Zhixing Xu
Amazon Web Services

Shengjian Guo
Amazon Web Services

Oksana Tkachuk
Amazon Web Services

Saeed Nejati
Amazon Web Services

Niloofar Razavi
Amazon Web Services

George Argyros
Amazon Web Services

ABSTRACT

As cloud computing gains widespread adoption across various industries, securing cloud resources has become a top priority for cloud providers. However, ensuring configuration security among highly interconnected cloud resources is challenging due to the complexities of resource modeling, correlation analysis, and large-scale security checks. To tackle those practical challenges, we propose Security Invariants (SI), a precise, effective, and scalable tool that proactively protects cloud resources by automated security reasoning. We have integrated SI into the rigorous Amazon Web Services (AWS) security review process. Partnered with security engineers and other security scanners, SI periodically scans billions of cloud resources in pre-launch services for potential security risks, maximizing the security guarantees of cloud applications. The continuous assessment of evolving resources not only brings a deep understanding of cloud security risks but also introduces a generalized solution from the holistic security analysis perspective.

KEYWORDS

Cloud Security, Cloud Resource, Resource Configuration

1 INTRODUCTION

Over the past years, cloud services have achieved a remarkable expansion in terms of their scale and complexity. This growth, consequently, necessitates the development of holistic methodologies for upholding robust security postures. A cloud resource in Amazon Web Services (AWS) usually refers to any virtual entity customers can create and interact with, like an EC2 computing instance [5], an S3 storage bucket [6], etc. Building cloud applications requires interaction among multiple resources. We refer to these interactions as cross-resource relationships. AWS defines rigorous security properties, e.g., low-privilege resources must not access critical storage, to ensure cloud security. The assessment of various security properties often involves a thorough analysis of per-resource configurations and cross-resource relationships, which is hard to accomplish manually. Thus, automated security property analysis plays a vital role for supporting security engineers to conduct security reviews of AWS resources, maximizing the guarantees of secure cloud applications. However, modeling and analyzing the security properties against numerous cloud resources is non-trivial, owing to multiple challenges.

First, modeling the security properties among resources presents a significant challenge. Given the abundance of cloud services and their resources, a comprehensive model must provide abstractions of the properties pertaining to each resource and describe the cross-resource relationships. Public tools, such as ScoutSuite [17], typically check individual resource configurations, lacking a holistic view that analyzes all the correlated resources in a cloud application. However, complex cloud security requirements, like preventing privilege escalation risks [16], often involve relationship analysis on resources across different services.

The second challenge resides in the reasoning of the modeled security properties and the cross-resource relationships. We need to conduct a precise analysis upon the modeled data for any security assessment from security engineers. AWS uses IAM policy [8] to manage complex resource permissions by combination of policy statements. Existing third-party tools [1, 14, 17] rely on heuristic-based syntactic policy analysis. Their approaches are susceptible to both false positive and false negatives. To guarantee precision, an analysis tool requires either robust reasoning capability of policies or seamless integration with automated reasoning engines, like Zelkova [10], an AWS policy analyzer.

Further, the scalability and real-time accuracy of the security analysis matters. A cloud application may contain millions of resources that are subject to frequent updates after the initial deployment. To facilitate continuous monitoring of deployed resources on a regular cadence, the security analysis must possess the capability to analyze all the modified resources at scale. Infrastructure as Code (IaC) is a paradigm that manages and provisions cloud infrastructure through code, which allows for pre-deployment security checking through code analysis [13, 14]. However, IaC analysis alone is insufficient because dynamic resource properties may manifest in deployment and cannot be analyzed at the IaC level. Further, deployments may deviate from the original IaC due to manual interventions. Consequently, a scalable approach on deployed resources plays a crucial role in enabling automated analysis.

In this paper, we present Security Invariants (SI), an accurate, effective, and scalable security analysis tool that tackles the aforementioned challenges. We make three contributions in this paper:

- We model the security properties and relationships of interconnected AWS cloud resources using ontology models.
- We develop a security analysis technique that performs automated reasoning on millions of cloud resources in hours.
- We evaluate the scalability and accuracy of SI and show that SI can analyze more than 600 million resources in 24 hours, with a Not-Useful rate below 7%.

ASE '24, October 27-November 1, 2024, Sacramento, CA, USA

ACM ISBN 979-8-4007-1248-7/24/10.
https://doi.org/10.1145/3691620.3695279

2 BACKGROUND

2.1 Resource Configuration Security

Resource configuration security in cloud application refers to the practice of ensuring that the various components in a cloud application are configured in a secure and compliant manner. This involves setting up parameters and permissions to minimize security risks.

Automated security analysis techniques ensure proper resource configuration security at scale. This begins with creating models of security configurations and relationships among resources. Then, automated tools specify the security properties and systematically analyze the models to identify potential violations. Further, the cloud provider should regularly run automated analyses to adapt to changes in the cloud environment, ensuring ongoing security and compliance of cross-resource configurations.

2.2 Cross-resource Risk

Existing techniques [1, 7, 17] primarily check configuration risks on a single resource. For example, an S3 bucket has no server-side encryption or an IAM role [8] that allows world accessibility.

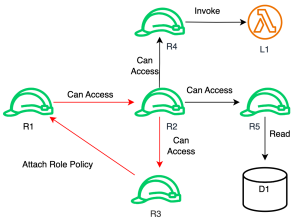

Figure 1: A cross-resource analysis example

However, single-resource security checks are inadequate for cross-resource interactions. Figure 1 shows cloud resource relationships in an application. Green hat-shaped nodes represent IAM roles for AWS identity and access management. The orange circle `L1` and black cylinder `D1` denote AWS Lambda [9] (compute) and DynamoDB [4] (database) services, respectively. The edges represent cross-resource relationships. For instance, `R1` accessing `R2` means `R1` can assume `R2`'s permissions. This typically occurs when IAM policies allow `R1` to perform `STS:AssumeRole` on `R2`, granting `R1` access to `R2`'s temporary credentials.

The red arrows indicate a privilege escalation risk. Role `R3` has the permission to attach policies to `R1`, granting `R1` new permissions. However, `R1` can assume `R3`'s permissions via `R2`, enabling self-editing of permissions. This undesired relationship poses a security risk, as `R1` can escalate its permissions to `Admin` level.

This privilege escalation case demonstrates the risk in cross-resource configuration. We have developed practical cross-resource security posture analysis in AWS with three emphases. First, AWS currently offers thousands of unique types of resources. To effectively perform cross-resource analysis, we propose efficient modeling for abstraction of the resources and their properties. Second, we use a highly scalable technique that handles the analysis of massive resources in the model. Third, cross-resource relationships are typically organized through IAM policies. To ensure correct generation and analysis of the edges, we utilize automated reasoning-based approach to perform a rigorous security analysis.

3 METHODOLOGY

3.1 Workflow

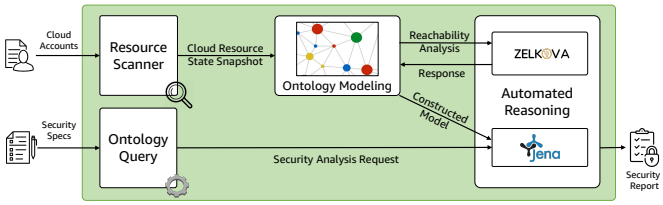

Figure 2: The high-level workflow of `SI`

AWS requires web services to undergo rigorous security reviews before launch. Security engineers drive the security review to cover all aspects of cloud security assessments with a set of security tools. `SI`, integrated into the workflow automation, is crucial for preventing resource misconfiguration risks in the security review pipeline. Before a service launch, `SI` automatically runs in its pre-production stage and generates security reports and risk findings. These findings are triaged by security engineers with the assistance of other tools and presented to service builders. Builders must either make proper remediations for a finding or mark it as Not-Useful by proving it will not lead to insecurity in the building environment.

Figure 2 displays the input and output, the core components, and the flow of `SI`. The first input includes a list of AWS cloud accounts whose resources need to get analyzed. `SI` leverages an AWS SDK-based Resource Scanner to programmatically retrieve the state snapshots of cloud resources under each account.

Once the Resource Scanner completes data acquisition, it forwards a resource snapshot to the Ontology Modeling (OM) component in the pipeline. The OM is responsible for constructing a comprehensive, precise, and expressive graph model to represent a cloud resource, including the resource itself as node and both inter- and intra-resource relationships as edges (ref. Section 3.2).

The second input consists of security property specifications, which state the rules for the secure cloud resource configurations. AWS defines a set of secure rules for each type of cloud resource and we use the specifications to formally depict the rules. Ontology Query component then transforms the specifications into queries and encode them into analysis request for the automated reasoning component. If any properties are violated, `SI` will output corresponding findings in the security report.

The Automated Reasoning (AR) component has two engines: Zelkova [12], an AWS-owned policy logic analyzer; and Apache Jena [2], an open-source reasoner that infers logical consequences based on ontology models. Accordingly, AR plays two roles in the pipeline. First, Zelkova acts as the reachability analyzer to reason about the existence of the access-based edges in the OM model. Second, Jena performs the security analysis by checking the ontology queries against the data model from OM. With the two engines, the AR component can determine if the data model complies with the provided security specifications. After AR assesses the compliance of the scanned resources against the specifications, `SI` generates a detailed security report for subsequent security review steps.

3.2 The Ontology Modeling

As displayed in Figure 2, we construct an ontology model for the snapshots of cloud resource states. Due to the large-scale nature of cloud resources, a graph-based modeling approach becomes natural. The graph nodes represent entities, which can be either AWS resources or the resource attributes; and the edges represent relationships between the entities. Ontologies provide a formal and clear description of the modeled resources.

We use Figure 3 to demonstrate a simplified model containing several S3 bucket and CloudTrail [3] entities. CloudTrail is an event auditing service in AWS. Here each circle is a node in the model, and each arrow represents an edge. A node can represent either an AWS resource or an attribute of that resource. In Fig 3, the red circles are resources and the black circles are attributes. An edge from a resource node to an attribute node indicates the resource owns the attribute, while an edge between two resources shows the cross-resource relationship. The graph in Figure 3 contains three S3 buckets, namely `Bucket1`, `Bucket2`, and `LogBucket`, along with one CloudTrail trail `Trail1`. The two cross-resource edges show that `Bucket2` stores logs to both `LogBucket` and `Trail1`. This model effectively represents resource attributes and cross-resource relationships in evolving cloud environment, abstracting the resources semantics for security auditing such as secure logging enforcement.

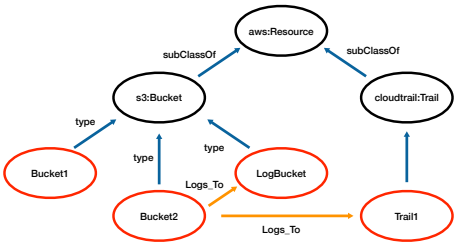

Figure 3: Graph-based Ontology Model Example

Note that Figure 3 visualizes a snippet of resources for demonstration purpose. In practice, we leverage RDF (Resource Description Framework) files to organize and store the large and complicated graph data. Based on the ontology modeling, `SI` performs the security reasoning for the provided security specifications against the modeled resources.

3.3 The Automated Reasoning

To perform security analysis against millions of cloud resources, we need scalable analyzing techniques. As mentioned in Section 3.1, `SI` leverages two reasoners for the large-scale analysis.

Shown in Figure 2, we use Zelkova, a Satisfiability Modulo Theories (SMT) Solver-based policy logic analyzer in resource modeling. AWS uses IAM policies to define resource access permissions. Zelkova automatically translates IAM policies into SMT formulas, encode queries as logical assertions, and pass the resulting problems to SMT solvers. The backend SMT solver determines if any requests satisfy the given query or proves that no such requests exist. With this capability, Zelkova can answer questions such as "Is policy A as permissive as policy B?" or "Does policy A allow access from resource R?". `SI` establishes a set of standard questions to determine the cross-resource relationship in terms of permitted reachability. When encountering IAM policy-defined permissions in a resource snapshot, the OM module forms an analysis request based on the IAM policy, sends it to Zelkova, and uses the response to update the edges between the resource nodes.

The other reasoner is Apache Jena, an open-source framework that infers logical consequences from a set of asserted facts or axioms. We use Jena to perform security analysis against the ontology model. The RDF model format allows Jena to generate security analysis results based on the specifications written as ontology queries. Details about the security specifications and corresponding ontology queries are in Sec 3.4.

We combine the two techniques to handle the challenges of reasoning complex security properties. The reasoners help `SI` perform accurate cross-resource, cross-service, and cross-region analysis.

3.4 The Security Analysis

To analyze the security property configurations of the cloud resources deployed under the input AWS accounts, specifications for the security properties are formally stated in the form of ontology queries. These queries are then applied to the ontology model and reasoned by the Jena reasoner. The reasoning results are presented to the user in the form of a security report. The security report not only gives the compliance to the specification but also provides detailed information on both compliant and non-compliant resources.

For example, AWS has a security property that requires S3 buckets deployed in production to have logging enabled. This mechanism ensures that people can investigate any S3 events upon a security incident. Listing 1 shows a snippet of the ontology query used to formalize the S3 property. The query is written in SPARQL [15], a standardized query language for querying RDF.

Listing 1: An Example of SPARQL query

```
1  ...
2  select distinct ?bucketARN where
3      # condition 1: Exclude logging bucket
4      {
5          ?bucket rdf:type s3:Bucket .
6          ?bucket aws:Name ?bucketARN .
7          ?bucket s3:bucketName ?bucketName .
8          FILTER NOT EXISTS {?bucket s3:destinationBucketName
               ?loggingBucketName}
9          MINUS {
10             ?sourceResource ?loggingBucket ?bucketName .
11             ?loggingBucket rdfs:subPropertyOf aws:logsTo .
12         }
13     }
14     # condition 2: Bucket logging to itself
15    ....
```

The code from Line 5 to Line 8 queries an ontology model for each S3 bucket in the model. This code piece determines if the buckets have a destination bucket for logging and filter the ones that do not have one. This may seem enough to check for the bucket logging property and identify the buckets without logging enabled. However, `SI` considers more subtle scenarios. In this example, lines 9 to 12 exclude the case where the bucket itself is a logging bucket. We can't require a logging bucket to be logged; otherwise, we fall into an infinite logging requirement. Therefore, if the bucket itself is a logging bucket, the analysis will regard it as a compliant resource.

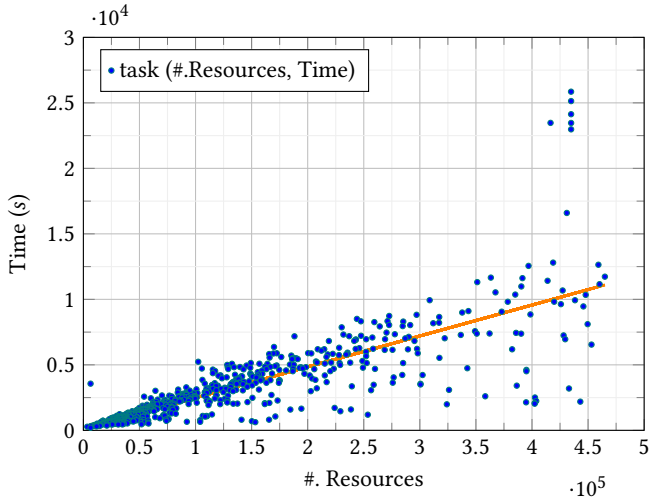

Figure 4: Linear correlation of #.Resources and analysis Time

Aside from this condition, SI also covers other practical cases. For instance, the logging bucket may not exist; the bucket might be logging to itself; and the bucket could be logging to CloudTrail, etc.

Enabling S3 bucket logging is generally viewed as a simple security property. However, this example shows that to analyze even a simple case, checkers need to check several configurations of each resource and the cross-resource relationships. This fact further emphasizes the need for a holistic ontology modeling of resources and a reasoning engine to analyze the model efficiently at scale.

4 EVALUATION

We have integrated SI as a resource configuration scanner into the rigorous AWS security review pipeline. SI periodically scans pre-launch cloud resources for potential risks and drives necessary remediations. We study SI's performance with three questions:

- **RQ1**: Is SI scalable to perform cloud security analysis?
- **RQ2**: Is SI precise for practical uses in security reviews?
- **RQ3**: Is SI able to prevent critical cloud security risks?

4.1 Runtime Scalability

This section studies the scalability of SI. We randomly sampled 690 tasks from a 24-hour window to illustrate the overarching linear correlation between the amount of resources in each task and the time used for that task. A task refers to running the SI analysis against a set of AWS accounts contained in an AWS security review.

Figure 4 provides a comprehensive overview. Each data point in the figure corresponds to an individual task executed by SI. The horizontal axis (*x-axis*) delineates the quantity of scanned resources per task, while the vertical axis (*y-axis*) illustrates the analytical time consumed by each task. Notably, the prominently highlighted orange line denotes the automatically computed linear correlation derived from the underlying data distribution.

Our observation reveals that, despite the presence of 8 data points deviating above the linear trend, approximately 98.8% of the dataset manifests proximity or adherence to the linear regression line. This outcome underscores the high predictability of SI, due to its scalable design. Moreover, for the 8 cases, the analysis duration is approximately 2.5 to 4 times longer compared to the time averaged from comparable tasks. However, this disparity remains within a manageable threshold for SI. Note that, a significant portion, specifically 44.3%, of the tasks have resource quantities exceeding 100,000. These tasks are commonly treated as "large tasks". SI has consistently showcased its scalable analysis capability when confronted with such substantial tasks.

Table 1: Evaluation of Runtime Performance

Task	#.Acct	#.Res	#.Reg	#.Serv	#.Checks	#.Findings	Time(s)
T1	29	200,759	17	12	101	34	4,903
T2	8	21,404	7	20	129	263	8,856
T3	17	457,370	17	20	129	482	44,333
T4	11	92,665	7	13	105	64	2,183
T5	72	646,631	7	17	113	88	15,798
T6	8	60,281	3	12	107	44	1,637
T7	16	475,779	7	18	121	231	27,783
T8	1	1,400	17	13	110	50	312
T9	58	423,999	17	13	104	101	12,206
T10	27	184,720	4	17	113	1,556	24,137
T11	33	107,809	5	16	110	273	30,932
T12	9	8,621	7	16	110	360	5,068

Next, we use Table 1 to depict detailed statistics about 12 sampled tasks from Figure 4. Column #.**Acct** shows the number of AWS accounts contained in each task. Column #.**Res** lists the quantities of all resources, e.g., S3 buckets, EC2 instances, Simple Message Queues and so on, owned by all accounts in each task. In general, more resources demand longer analysis time. Columns #.**Reg** and #.**Serv** indicate how many geographical regions and AWS services the resources distribute, respectively. Column #.**Checks** states the number of security rules SI performed. The last column presents the total analysis time #.**Time** in seconds.

In Table 1, the resources in each task range from 1,400 to 646,631 with an average value of 223,453. Also, the pre-launch resources, geographically distributed in 4-17 AWS regions, belong to frequently used AWS services such as EC2, S3, DynamoDB, SQS, API Gateway, etc. For each task, SI runs more than 100 checks. In total, SI reports 3,546 findings in 3,130,187 resources. Each finding represents further investigation needed by engineers and additional automated tools in subsequent steps of the review process, prior to launch.

The analysis time for individual tasks tends to align proportionally with the task's resource amount, consistent with the trend shown in Figure 4. With this property, we utilized parallel computing to expedite task executions.

RQ1 Answer: SI analysis time scales linearly with the number of resources. SI's detection supports various cloud resources distributed among 17 global regions. Thus, SI scales well to the AWS cloud security review tasks.

4.2 Detection Precision

This section demonstrates the precision of the security findings reported by SI. Specifically, we focus on the Not-Useful (NU) findings manually marked by the AWS service builders. Once an AWS service

gets onboarded to SI, SI would analyze all service resources and notify the service builders of potential security risks. The builders must make proper remediations to any reported risk findings. However, if the builders can demonstrate that a finding does not lead to insecurity for specific reasons, they can mark the finding as NU. We group the observed NU reasons into three categories: **Test Res**: The resources are for testing or temporary use purposes, so they have low security impacts. **Mitigated by other controls (Mitigated)** : The resources' findings are already mitigated by other means. **Incorrect**: False positive cases due to the faulty detection of SI.

Table 2: Overall Not-Useful rate and breakdown

#.Total NU	#.Test Res	#.Mitigated	#.Incorrect
7.0%	1.7%	5.2%	0.1%

Table 2 summarizes the NU rate and its breakdown gathered from all security findings of the dataset in Figure 4. Column #.Total NU shows the total NU rate while the rest three columns displays the rate of each category. In general, the lower NU rate, the better detection precision. SI has a low overall NU rate of 7.0%, and the #.Mitigated configurations occupied 74.3% of the total NU findings. For example, if the AWS DynamoDB table records are encrypted at the customer side, then it is not a security risk despite server-side encryption is not being enabled. Column #.Test Res indicates about 24.3% NU findings are from testing resources. SI keeps monitoring testing resources since any careless configurations may cause risky problems if they could subtly reach non-testing resources. Finally, column #.Incorrect indicates that 0.1% out of all findings are real false positives due to the problematic detection mechanism. Those are low-volume but indeed important cases that SI must address to improve its detection.

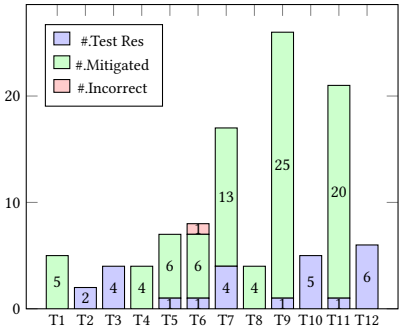

Figure 5: The categorized NU findings in each task

Figure 5 visualizes the categorized NU findings from each sampled task in Table 1. The *x-axis* lists the task aliases and *y-axis* shows the stacked NU findings of the three categories. Among the 109 NU findings of all 12 tasks, the #.Mitigated category owns a majority of 83 findings. 25 findings belonging to #.Test Res are observed in 9 out of all 12 tasks. And there is only 1 finding of #.Incorrect category in task T6. Overall, the proportions of each category is consistent with Table 2.

RQ2 Answer: SI owns a high detection precision measured by the 7% overall NU rate. 98.6% of the NU findings are due to practical reasons that can get alleviated strategically while only 0.1% of all findings are false positives that SI must address.

4.3 Security Risks

This section discusses the cloud resource misconfiguration risks that SI focuses on. We map SI's checks to relevant Common Weakness Enumeration (CWE) IDs to depict their general purposes.

We map 128 checks in SI to 15 CWE categories regarding the CWE definitions. The following table gives examples of the CWE and the corresponding SI check that is related to the CWE.

CWE	Category	SI Check Examples
CWE-269	Improper Privilege Management	IAM Res with Privilege Escalation risk
CWE-312	Cleartext Storage of Sensitive Info	Res without Service Side Encryption
CWE-863	Incorrect Authorization	Resource Policy Allow World Access
CWE-778	Insufficient Logging	Resource without Proper Logging
CWE-820	Missing Synchronization	Infrastructure Drift from IaC
CWE-668	Exposure of Resource to Wrong Sphere	Non-prod Res Accesses Prod Res

Take CWE-312 and CWE-863 as examples. SI's checks for those two are about resources that lack server-side encryption and over-permissive policies that are globally accessible, respectively. The next two categories CWE-269/668 and the corresponding SI checks primarily address insecure configurations that can lead to privilege escalation and unintended cross-resource attacks. These issues are specific to cloud security, and SI covers a comprehensive set of scenarios to ensure AWS resources are protected from such attacks. Further, SI checks under CWE-778 deal with improper logging for resources. Secure logging plays an important role in resource forensic analysis when unusual resource states appear. This category inspects inconsistency between the expected resource status defined by Infrastructure as Code (IaC) configurations and the current status of the same resource.

RQ3 Answer: SI practically detects security risks before resource launching, depicted in 15 CWE categories. SI maximizes the security guarantees by proactively eliminating such risks.

5 RELATED WORK

Cauli et al. [13] formalized CloudFormation (CFn) templates and applied semantic reasoning for security assessment. AWS's public tool CloudFormation Linter [14] parses Cfn templates and makes syntactic reasoning. Additionally, CloudFormation Hook [10] allows customized logics for the resource inspection. However, existing tools primarily focus on assessment of individual resource configurations. ScoutSuite [17] is an open source security auditing tool supporting multiple cloud providers. Similar tools [1, 7] lack the ability to perform holistic security analysis that involves cross-resource relationships. Complex cloud security risks, such as AWS IAM privilege escalation [16, 18], often require multiple escalation steps and involve resources across services. Applying formal reasoning methods to cloud security have been popular in recent years. Zelkova [12] is an AWS policy analysis tool that encodes the semantics of AWS IAM policies into SMT for property verification. Tiros [11] uses off-the-shelf automated theorem proving tools to analyze network reachability. As described in Section 3.3, SI uses Zelkova as one of the backend automated reasoning engines.

6 CONCLUSION

In this paper, we state the key problems in cloud resource configuration security analysis, and present SI, a precise, effective, and scalable tool that address the challenges in Amazon Web Services.

REFERENCES

[1] 2024. prowler. https://github.com/prowler-cloud/prowler
[2] Apache. [n. d.]. Apache Jena. https://jena.apache.org/
[3] AWS. 2024. Amazon CloudTrail. https://aws.amazon.com/cloudtrail/
[4] AWS. 2024. Amazon DynamoDB. https://aws.amazon.com/dynamodb/
[5] AWS. 2024. Amazon EC2. https://aws.amazon.com/ec2/
[6] AWS. 2024. Amazon S3. https://aws.amazon.com/s3/
[7] AWS. 2024. AWS Config. https://aws.amazon.com/config
[8] AWS. 2024. AWS Identity and Access Management. https://aws.amazon.com/iam
[9] AWS. 2024. AWS Lambda. https://aws.amazon.com/lambda
[10] AWS. 2024. Developing Cloudformation hooks. https://docs.aws.amazon.com/cloudformation-cli/latest/userguide/hooks.html
[11] John Backes, Sam Bayless, Byron Cook, Catherine Dodge, Andrew Gacek, Alan J Hu, Temesghen Kahsai, Bill Kocik, Evgenii Kotelnikov, Jure Kukovec, et al. 2019. Reachability analysis for AWS-based networks. In Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II 31. Springer, 231–241.
[12] John Backes, Pauline Bolignano, Byron Cook, Catherine Dodge, Andrew Gacek, Kasper Luckow, Neha Rungta, Oksana Tkachuk, and Carsten Varming. 2018. Semantic-based automated reasoning for AWS access policies using SMT. In 2018 Formal Methods in Computer Aided Design (FMCAD). IEEE, 1–9.
[13] Claudia Cauli, Meng Li, Nir Piterman, and Oksana Tkachuk. 2021. Pre-deployment security assessment for cloud services through semantic reasoning. In Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I 33. Springer, 767–780.
[14] AWS Cloudformation. 2024. AWS CloudFormation Linter. https://github.com/aws-cloudformation/cfn-lint
[15] World Wide Web Consortium. [n. d.]. SPARQL 1.1 Query Language. https://www.w3.org/TR/sparql11-query/
[16] Rhino Security Labs. 2020. AWS Privilege Escalation Vulnerabilities. https://rhinosecuritylabs.com/aws/aws-privilege-escalation-methods-mitigation
[17] nccgroup. 2024. ScoutSuite. https://github.com/nccgroup/ScoutSuite
[18] Ilia Shevrin and Oded Margalit. 2023. Detecting Multi-Step IAM Attacks in AWS Environments via Model Checking. In 32nd USENIX Security Symposium (USENIX Security 23). USENIX Association, Anaheim, CA, 6025–6042. https://www.usenix.org/conference/usenixsecurity23/presentation/shevrin