Editor: **Tim Menzies**
North Carolina State University
tim@menzies.us

One-Click Formal Methods

John Backes, Pauline Bolignano, Byron Cook, Andrew Gacek, Kasper Søe Luckow, Neha Rungta, Martin Schaef, Cole Schlesinger, Rima Tanash, Carsten Varming, and Michael Whalen

FORMAL METHODS ARE mathematically based approaches for specifying, building, and reasoning about software. Despite 50 years of research and development, formal methods have had only limited impact in industry. While we have seen success in such domains as microprocessor design and aerospace (e.g., proofs of security properties for helicopter control systems[1]), we have not seen wide adoption of formal methods for large and complex systems, such as web services, industrial automation, or enterprise support software.

One of the key difficulties when proving the security, safety, and robustness of these systems is the problem of finding system architecture models necessary for analysis. Proving the system at its lowest level of detail is intractable, and, thus, we must reason at higher levels of abstraction. If written by hand, these models are expensive to build and hard to keep up to date with implementations. Another problem is that the size of the potential user community and the business value have typically not justified the creation of scalable and easy-to-use tools for the formal verification of those models.

Digital Object Identifier 10.1109/MS.2019.2930609
Date of current version: 22 October 2019

With the cloud, much of this has changed. Descriptions of cloud services provide accurate models of the system. That is to say, the application program interfaces (APIs) of cloud services are computer-readable contracts that establish and govern how the system behaves. In many cases, these models are amenable to formal analysis at scale.[2] Most importantly, since those models are utilized by a large user community, it is now economically feasible to build the tools needed to verify them.

The larger cloud providers are rapidly developing and applying formal method tools. At Amazon Web Services (AWS), for example, we have used cloud models to construct large-scale automated reasoning tools that can prove whether or not access controls meet governance rules and whether networks are properly secured. These tools are used millions of times daily and help AWS customers manage the security of their accounts.

This is the beginning of an era in which security, compliance, availability, durability, and safety properties can be proven about large-scale architectures. In this short column, we discuss the trend of constructing practical and scalable cloud-based formal methods and how they can easily be used by customers—sometimes with a single operation for one-click formal methods.

The Classical Approach (Where Formal Verification Was Hard)

Figure 1 shows a simplified, three-tier web application for uploading pictures developed in a traditional (noncloud) environment. The web tier has two REST resources:

- the Login API for users to authenticate with the service
- the Upload API to upload new pictures to the website.

The app tier consists of four microservices that interact with each other through a standardized API. The Auth Service processes authorization requests; the Session Service tracks stateful data relevant to the user's current visit to the website; the Upload Service receives photos from the user and stores them for future retrieval; and the Thumbnail Service creates thumbnails for the photos in the data store. The data tier has three databases: Auth DB for

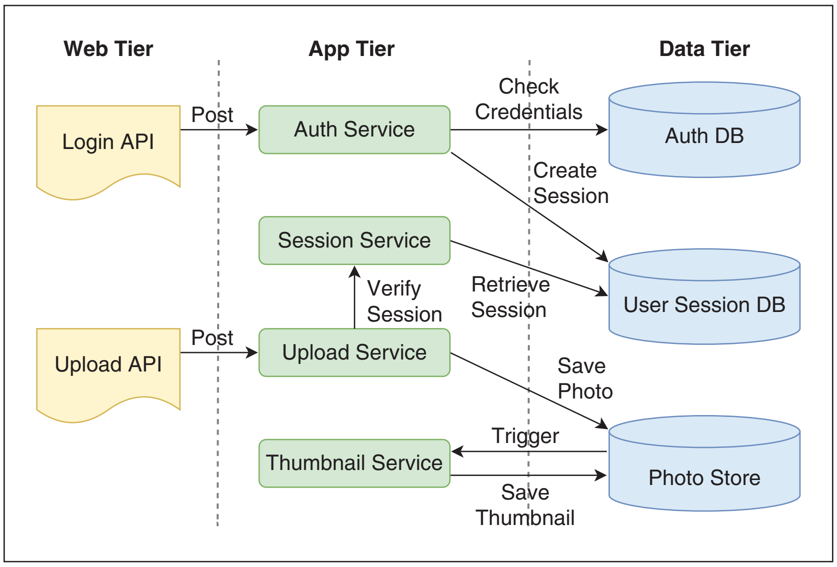

FIGURE 1. The architecture of a three-tier web application.

authorization credentials, User Session DB for user sessions, and Photo Store for user photos and thumbnails on the website.

Imagine that we want to prove least-privilege access to resources for the system. Toward this goal, we would have to prove the following requirements:

1. Only the Auth Service shall access the Auth DB.
2. The Auth Service shall not write to Auth DB. (For simplicity, we assume that users are added to the authentication database using an external mechanism.)
3. Resources in the web tier shall not directly access databases in the data tier.
4. The Thumbnail Service shall access only the Photo Store (no other databases) and shall write only to the thumbnail portion of the Photo Store.

In a noncloud computing environment, the problem is that the diversity of technologies composing the system makes it challenging to verify these end-to-end requirements. At the very least, we have to consider the following:

- *Network controls*: These are used to guard the compute nodes in each tier. Typically, controls are enforced through the use of hardware or software firewalls, which block packets from restricted Internet Protocol (IP) addresses and/or port ranges.
- *File system permissions*: These are employed to control and delegate user access to local data.
- *Database credentials*: These are utilized to restrict access to a set of privileged users, e.g., developers.
- *Cryptographic keys*: These are used to protect user credentials in the databases.

We also must reason about combinations of these access control mechanisms. For example, requirement 4 involves reasoning about network reachability, database access control, and, potentially, file system permissions, depending on how the Photo Store is implemented. We are reasoning simultaneously about both low-level implementation details and higher-level architectural design.

To reason end to end, we must either build new mechanisms and tools over a combined semantic model or determine how to decompose properties such that results from existing tools can be soundly combined. Also, for any model we build, we must check that it matches the behavior of the deployed system. Finally, maintaining and scaling the model as components are added or changed is a daunting and often-neglected task.

A New Approach (in the Cloud, Where Formal Verification Works Well)

Now consider the example from Figure 1 in the cloud. Cloud computing providers, such as AWS, give customers a comprehensive set of system services and features that are easy to plug in to each other. We will keep the same services in the app tier and use the provided database and storage facilities from AWS for the data tier. As we did before, imagine we are aiming to prove least-privilege access to resources of the system. In the cloud context, the proof in this example boils down entirely to reasoning about policies. This is because AWS defines a policy language that allows customers to configure access control across all services and resources, including APIs, compute instances, databases, alarms, logs, and metrics. This policy language governs access to all of the components in Figure 1. A common language allows us to reason about all of the disparate components and soundly compose the results, with no additional effort.

```
{
  "Statement": [
    …
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::website-photo-store/photos/*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::website-photo-store/thumbnails/*"
    },
    {
      "Effect": "Allow",
      "Action": "lambda:InvokeFunction",
      "Resource":
        "arn:aws:lambda:us-east-1:111122223333:function:CompressImage"
    },
    …
  ]
}
```

(a)

```
"Constraints": {
  "Actions": [ "s3:PutObject" ],
  "NotResources": [
     "arn:aws:s3:::website-photo-store/thumbnails/*"
  ]
}

"Constraints": {
  "Actions": [ "s3:GetObject" ],
  "NotResources": [
    "arn:aws:s3:::website-photo-store/thumbnails/*"
    "arn:aws:s3:::website-photo-store/photos/*"
  ]
}

"Constraints": {
  "Actions": [ "dynamodb:*" ]
}
```

(b)

```
{
  "Principal": "arn:aws:iam::123456789012:role/Thumbnail",
  "Action": "s3:GetObject",
  "Resource": "arn:aws:s3:::website-photo-store/backups/"
}
```

(c)

FIGURE 2. The policies and constraints in Zelkova: (a) a small portion of the access control policy for the Thumbnail Service from Figure 1, (b) three constraints representing violations of requirement 4, and (c) a representative violation report.

Also, the cost of creating the analysis is amortized across all of the platform's users, so we can invest in scalable and accurate analyses.

Figure 2(a) shows a policy for the Thumbnail Service. In this example, we have implemented the Photo Store using Amazon Simple Storage Service (Amazon S3) and the Auth and User Sessions databases in Amazon DynamoDB. The access control policy in Figure 2(a) determines the access rights for the Thumbnail Service. The first statement allows the service to read files from the photo directory. The second and third statements allow the Thumbnail Service to write to the thumbnail directory and invoke an external function to compress images.

At AWS, we have developed the Zelkova tool[3] to prove properties across examples like that in Figure 1. Zelkova encodes access control policies and properties into satisfiability modulo theories (SMT) logic. SMT is a language for checking propositional logic satisfiability extended with theories that allow reasoning about richer data, such as unbounded integers or real numbers. Zelkova uses the theories of strings, regular expressions, bit vectors, and integer comparisons. The SMT models generated by Zelkova can be analyzed by several efficient back-end tools.

We can now use automated reasoning to provide inexpensive and provable assurance to customers.

Suppose we wish to verify requirement 4 of the policy in Figure 2(a). We write constraints representing violations of this requirement, as shown in Figure 2(b). Informally, the constraints state that any write request outside of the thumbnail directory of the Photo Store is a violation, as well as any read request outside of the thumbnail and photo directory or any access to a DynamoDB database (which contains the Auth and User Sessions databases).

In the example involving Figure 2(a) and (b), if there were no further Allow statements on Amazon S3 resources, the tool would return a *valid*; it is not possible for the thumbnail account to read files from other locations in Amazon S3. Suppose, however, that the policy had an additional statement that allowed reading from the website-photo-store/backups directory. In this case, the result

ABOUT THE AUTHORS

JOHN BACKES is a senior software development engineer with the Amazon Inspector service team at Amazon Web Services, where he is the lead developer for the Tiros service, which performs semantic analysis of virtual private cloud networks. Contact him at jbackes@amazon.com.

PAULINE BOLIGNANO is a software engineer at Amazon Web Services. Her research focuses on providing quality assurance to customers through the verification of the various aspects of software and system development. Contact her at pln@amazon.

BYRON COOK is a professor of computer science at University College London and director of Automated Reasoning at Amazon. Contact him at byron@amazon.com.

ANDREW GACEK is an applied scientist with the Automated Reasoning Group at Amazon Web Services. His research interests include developing and applying automated formal verification at scale. Contact him at gacek@amazon.com.

KASPER SØE LUCKOW is a software development engineer with the Automated Reasoning Group at Amazon Web Services. His research interests include program analysis and automated verification at scale. Contact him at luckow@amazon.com.

NEHA RUNGTA is a principal applied scientist and the leader of the Formal Services team at Amazon Web Services. Her research interests include improving the customer experience in the cloud through the use of formal verification. Contact her at rungta@amazon.com.

MARTIN SCHAEF is a software engineer at Amazon Web Services Security. His research focuses on large-scale program analysis infrastructure. Contact him at schaef@amazon.com.

COLE SCHLESINGER is a senior applied scientist at Amazon Web Services. His research interests include programming languages and formal methods with a focus on building domain-specific models supporting automated verification. Contact him at awscole@amazon.com.

RIMA TANASH is the lead security engineer with the Amazon Security Hub service team, where she applies automated reasoning technologies to audit various access configurations. Her research interests include data privacy using machine learning. Contact her at tanashr@amazon.com.

CARSTEN VARMING is a senior software engineer with the Automated Reasoning Group at Amazon Web Services. His research interests include functional programming paradigms, logic, software architecture, and program analysis at scale. Contact him at varming@amazon.com.

MICHAEL WHALEN is a principal applied scientist and leader of the Proof Platforms team at Amazon Web Services. His research interests include scaling formal verification tools and their application to industrial problems. Contact him at mww@amazon.com.

would be an *invalid*, and Zelkova would report a violation involving the constraint shown in Figure 2(c).

It is possible to customize Zelkova for a variety of workflows. First, one may use the tool as a preventative control. These controls serve as gatekeepers in an automated workflow, enforcing a set of checks that, if unsuccessful, halt the workflow. As part of the pipeline, Zelkova controls ensure that only compliant access control policies are created and attached to production resources. It is also possible to use Zelkova for detective (i.e., auditing) and responsive (i.e., monitoring and alarming) controls. Such controls dynamically monitor, analyze, and respond to events in the cloud, including configuration changes, and can be equipped with Zelkova checks to detect policy compliance violations. In the case of a violation, several options are available. A notification email can be generated and sent to the user, or the system can revert to a known good state.

In practice, Zelkova is used millions of times a day by both internal and external customers, supporting preventative, detective, and reactive controls, and 99% of all Zelkova proofs complete in 160 ms or lower. Zelkova is currently integrated within AWS services, including Amazon S3, AWS Config, AWS IoT Device Defender, Amazon Macie, AWS Trusted Advisor, and Amazon GuardDuty. External customers, ranging from the financial industry to compliance regulators, use Zelkova to ensure that their access control policies are compliant with corporate governance rules.

Opportunities for Formal Methods in a Cloud Environment

Formal methods in the cloud are used for more than just access control. Tiros,[4] part of Amazon Inspector, uses the model provided by Amazon Elastic Compute Cloud (Amazon EC2) network configurations to perform proofs of network reachability without generating any network traffic. For example, a customer may check whether there exists any public IP address on the Internet that can access a local database server. Unlike packet-scanning approaches, Tiros will find any such access path and does not add load to the network. Other examples where we will investigate the common cloud model to perform proofs include the Internet of Things (IoT) (AWS IoT Core), build and deploy (AWS CodeStar), infrastructure as code (AWS CloudFormation), logging (AWS CloudTrail), monitoring (Amazon CloudWatch), and machine-learning frameworks (Amazon SageMaker).

One-Click Formal Methods—Try It Out!

We have constructed the Zelkova and Tiros tools so that this technology is available at the click of a button (or check of a checkbox).

- In Amazon S3 Block Public Access, when creating a storage bucket, the creation page includes a checkbox to deny public access to the bucket. If this option is selected, Zelkova will safeguard a policy, disallowing modifications that would allow public access.
- In Amazon Inspector, by enabling network reachability checks, Tiros will prove which servers are publicly accessible.
- In AWS Config, which assesses and audits resource configuration, enabling certain managed rules will use Zelkova to ensure that common corporate governance policies are followed.

More generally, we can now use automated reasoning to provide inexpensive and provable assurance to customers. We expect that this trend of building practical and scalable formal methods in the cloud will lead to environments where security, compliance, availability, durability, and safety properties can be proved about large-scale systems. For more information, check out our Amazon Provable Security webpage at https://aws.amazon.com/security/provable-security.

References

1. D. Cofer et al., "A formal approach to constructing secure air vehicle software," *Computer,* vol. 51, no. 11, pp. 14–23, 2018.
2. B. Cook, "Formal reasoning about the security of Amazon Web Services," in *Proc. Federated Logic Conference (FLoC)*, 2018. doi: 10.1007/978-3-319-96145-3_3.
3. J. Backes et al., "Semantic-based Automated Reasoning for AWS Access Policies using SMT," in *Proc. Formal Methods in Computer-Aided Design (FMCAD)*, 2018. doi: 10.23919/FMCAD.2018.8602994.
4. J. Backes et al., "Reachability analysis for AWS networks," in *Proc. Computer Aided Verification*, July 2019, pp. 231–241. doi: 10.1007/978-3-030-25543-5_14.