Continuously Generating Models of Cloud Systems with Applications in Privacy

Tristan Ravitch, Amazon Web Services

- Reasoning (verification, audit/compliance) about complex systems requires models
- Complex systems lack up-to-date models
- Work using code analysis for model synthesis is promising, but not enough by itself

Goal: Automatic and continuous generation of up-to-date models to support high-confidence system reasoning

Systems cannot be understood without considering their configuration and their infrastructure

Challenge: System Configuration

Complex systems almost always have configuration to control their behavior in different operational scenarios. If configuration values are not accounted for, analysis of code artifacts cannot account for their effects and cannot fully understand the system.

Privacy Example: Scalable cloud systems are regionalized and behave differently in different regions; they select close data stores (e.g., to reduce latency) and may have features enabled or disabled due to local regulations (e.g., GDPR).

Configuration can come in many forms including configuration files and configuration stores (e.g., configuration services). Configuration files have an enormous diversity of syntax and semantics, with some approximating or even exhibiting the characteristics of programming languages. Service based configuration is dynamic and enables great flexibility (e.g., run-time provisioning or adaptation), but poses challenges for static analysis.

Generalizing beyond configuration files, environment variables, scripts, and system processes can move sensitive data in ways not visible from a view of only application code. For example, system daemons can periodically move logs or data across network links.

Challenge: Component Organization

Both enterprise cloud systems and embedded cyber-physical systems are collections of components that communicate over networks (either WAN or local). Understanding and modeling the behavior of these systems requires modeling which components communicate with each other and what they communicate.

Privacy Example: Cloud data stores have user-configurable mechanisms to automatically replicate data; understanding the replication policy is critical to determine compliance with data sovereignty regulations. Modeling access permissions and data retention policies is also necessary to fully understand the privacy impact of a data store.

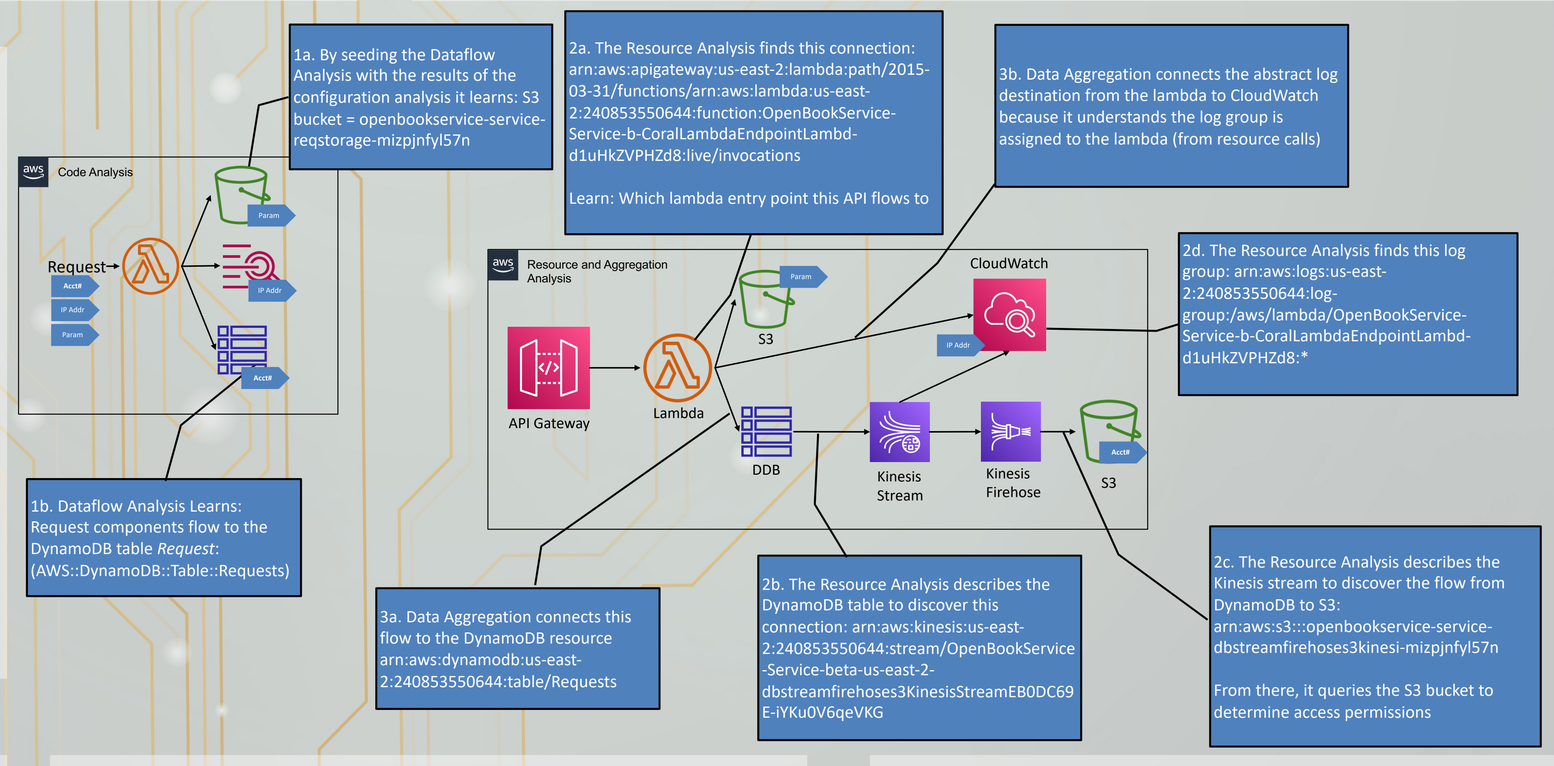
Ideally we could analyze Infrastructure as Code (IaC) configurations to understand how components connect. In some cases we can, but in others we may not be able to due to drift or incompleteness. Infrastructure drift occurs when manual or automated processes change infrastructure configurations after IaC runs, potentially leading to incorrect results. Incompleteness occurs when IaC systems generate additional infrastructure that is not syntactically obvious. The automation provided by IaC systems (like CloudFormation) is helpful, but obscures the true nature of the system. Replicating their logic with full fidelity is unsustainable, so other solutions are required.

(0.5) Configuration Analysis

We model the set of configuration sources for our use case including both static configuration files and query dynamic configuration. We unify configuration in a standard format that maps configuration APIs to the values they return for use in the code analysis. Configuration analysis includes modeling dependency injection, which is widely used to facilitate testing and component reuse.

We account for regionalized (or instanced) configuration and run one code analysis per concrete configuration.

Future Work: Monitor for run-time configuration changes, explore parameterized configurations to reduce the cost of re-analysis



(1) Code Analysis

The code analysis chosen for model generation depends on the nature of the models required for a given use case: it is a parameter.

We inline configuration values and use context-sensitive constant propagation to associate configuration with the critical components that they configure. We are experimenting with a value-set analysis to support *conditional* configuration that can be resolved later (3).

Privacy instantiates the analysis framework with taint analysis to track the flow of data. It produces a model of flows from inputs to outputs, which may include unresolved nodes (e.g., log groups). We track the flow of every data value through the service without attempting to categorize it as privacy-relevant or not, as data categorization at the level of code is not necessarily possible (e.g., the same service may handle private data or non-private data, depending on context).

A coarse architectural model may prefer a constant analysis, while a security model may require a typestate analysis.

(2) Infrastructure Analysis

In our setting, infrastructure is able to self-report its current configuration. We query live cloud infrastructure for both allocated resources (e.g., storage and compute) as well as the connections between them, producing a graph.

Example: The infrastructure analysis captures relationships that include 1) the log destinations for each component in the system, and 2) infrastructure-driven data flows.

Future Work: Detect drift by comparing against IaC.

In cases where drift is not possible, analysis of IaC would be simpler and more direct. IaC analysis requires implementing a semantics for each IaC language. In the embedded systems space, it may also require analysis of build systems.

(3) Aggregation

We combine the results of the infrastructure and code analysis to create a global data flow model. It matches unresolved node references from the code analysis results with their corresponding nodes in the infrastructure.

Key Observations: The configuration analysis gives the code analysis precise names for system entities that can be matched against the infrastructure analysis.

Privacy Use: We label privacy-relevant data values and compute graph reachability relations to build data mappings that enable reasoning about where data flows to, who has access, and how long it is retained.

This aggregation architecture could be used to generate SysML models. The analysis as described captures some behavioral aspects that can be represented in models; however, higher-fidelity abstractions are required to support model checking.

THE TWENTY-THIRD ANNUAL
HIGH CONFIDENCE SOFTWARE AND SYSTEMS CONFERENCE
MAY 8 -10, 2023 | http://cps-hcss.org