Qapla: Ensuring policy compliant queries in data retrieval systems

Aastha Mehta[1], Eslam Elnikety, Deepak Garg, Peter Druschel
MPI-SWS

Online data retrieval (ODR) systems typically collect and serve massive amounts of data including public web pages, blogs, personal emails, online social network profiles and activities, along with real-time data streams from microblogging sites and localization services. Examples of data retrieval systems include commercial providers such as Microsoft, Google, Facebook, Amazon and eBay. Private enterprises and government units often use open source or custom-built ODR systems internally.

ODR systems need to comply with complex data privacy and usage policies, as breaches due to policy violations are monetarily expensive. For instance, Google's users may restrict some of their private information (e.g., exact age and location) to their friends, while allowing the provider to use other attributes (e.g., their search history) only for personalization; however, accidental usage of their private information for personalization could lead to lawsuits, and loss of business and reputation for the provider.

However, ensuring policy compliance in such ODR systems is challenging and error prone due to several reasons. First, similar structured data is often grouped together in database cells or key-value tuples even if the individual data items in the group have different owners and usage policies. Second, policies are often implicit and scattered across code, and their specification and enforcement is spread across multiple components and layers of the application software stack. As a result, despite rigorous testing, it is difficult to eliminate all possible design flaws, software bugs and misconfigurations, and even to understand what the policy in effect is.

Qapla. The objective of this work is to provide a mechanism to enforce confidentiality, integrity, and compliance policies across *distributed, database-backed* ODR systems in a systematic and efficient manner. Qapla provides a declarative, query-aware policy language to specify policies on different data sources such as key-value pairs, tables, records, attributes, cells and files. Qapla enforces policies with the help of a reference monitor (QRM), which mediates execution of SQL-like queries and tags the application with the policies of all cells accessed in the query. The policies are propagated as "taints" over I/O channels such as files, pipes, and sockets. Finally, the policies accumulated in the taint set are evaluated at the application endpoint before releasing any data from the application.

We briefly describe the policy language, a few example policies and the enforcement mechanism.

Policy language. Our query-aware policy language specifies which data operations (join, selection, projection, aggregation, etc.) are allowed or disallowed on each data unit. The language can also disallow linking of two or more data items along a distributed data processing pipeline.

Example 1: Policy compliant analytics. In order to comply with their own privacy policy, data retrieval systems may stipulate that analysts can never link users' personally identifiable information (PII), such as their IP address, with their history. The PII and history columns hold a Qapla symmetric policy that prevents analysts from projecting them together in a single query or even across multiple queries.

Going further, analysts may not have access to user IP addresses at all and may be restricted to asking only stipulated aggregate queries on the user PII, such as "number of buyers from country C". To do this, the policy on the user IP address column prevents its direct flow into the query result, but allows joining it with a $\langle$IP, country$\rangle$ mapping to select the country.

Example 2: Policy compliant recommendations. A user's activity history can be used only for personalized recommendations and expires after a short time interval. The history column policy disallows queries if they are issued after the expiration time of the column.

Reference monitor. We prototype QRM by modifying the library-based SQLite database system. First, we separate the SQLite execution engine (backend) from its query parser and optimizer (frontend), and build the reference monitor into the execution engine. The frontend, which constitutes a significant fraction of the SQL codebase remains unmodified and is not in Qapla's the trusted computing base. Any bugs in the query or the frontend cannot compromise the enforcement of Qapla policies.

The execution engine is modified to perform a static analysis of every optimized query, to extract which SQL operations are performed on each column accessed by the query. When the query executes, this information is used to declassify individual database cells in accordance with their individual policies and a precise taint (also a policy) is added to the application process. A small kernel module propagates the taint during IPC and enforces all taint at system endpoints.

Aastha Mehta and Eslam Elnikety are students.
[1]Will present the poster.