Secure Cooperative Analytics

Qiang Cao and Jeff Chase
{qiangcao, chase}@cs.duke.edu

Project Silver

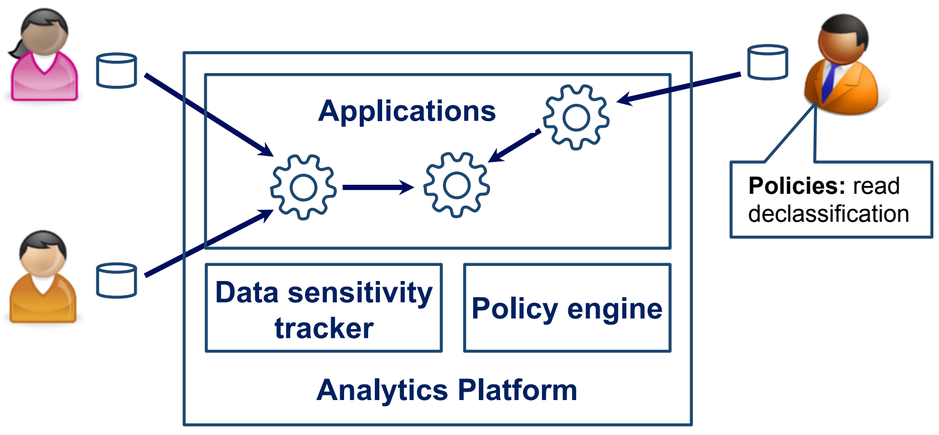
Motivation

Secure cooperative analytics: Enables large-scale analytics on data shared by multiple participants, e.g., users, institutions, and enterprises, without revealing their sensitive data to one another ("privacy-preserving").

Aggregating brings more utility and value from sensitive data:
- Collaborative threat monitoring
- Healthcare analytics
- Social sciences research

Our goals:
- Simplified model based on mutual trust in third-party cloud provider
- Extended access control model with compound principals (trusted code)
- Secure and fine-grained delegations of access and authority
- Enable support for statistical privacy (e.g., differential) at app layer

Design



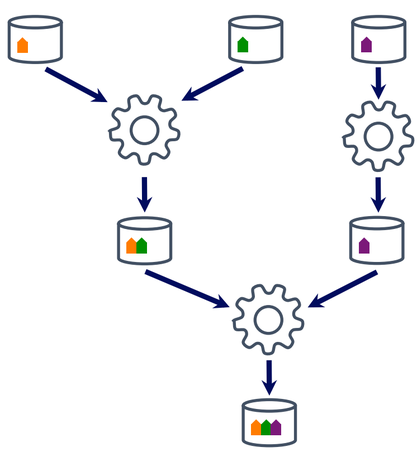
Main idea
- Controls **information flow** within the platform so that data disclosure to subjects and programs complies with each owner's policies

Data sensitivity tracker
- Tracks data sensitivity using its lineage
- Sensitivity of outputs can be quantified by statistical privacy models.

Policy engine
- Each data owner makes policies on how her data is to be shared.
- Platform tracks flow and enforces compliance with data policies.

Information flow control for analytics

Tracking data sensitivity
- Each data object is associated with a set of sensitivity tags.
- Tags propagate through the computations of a workflow; data sensitivity is preserved.



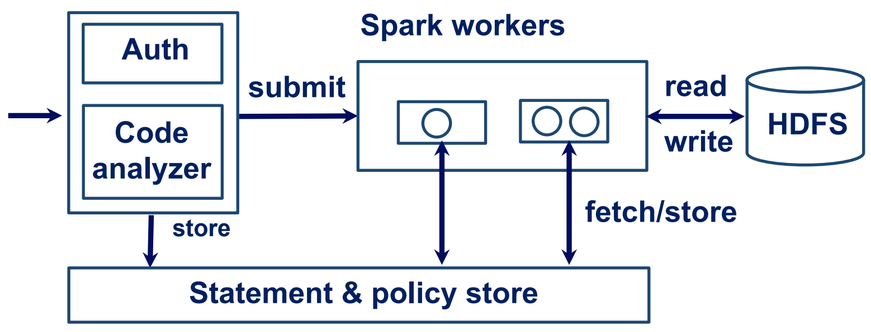
Access control
- A subject/program can access an object if it can access all of the source data, i.e., prove access for all of its tags.

Declassification
- Trusted subjects/programs have authority to remove one or more tags of an object.

Implementation

Integration with Apache Spark

Additional components:
- Dedicated job submitter for authentication and static code analysis
- An ACLs-protected HDFS domain to store data objects and tags



SAFE implementation of policy as logical trust rules

DIFC policy

```
defcon difcAccessPolicy(?User, ?File) :-
 spec('Local DIFC access policies'),
 {
  fileToAccess($File, $User).
  tagAccess(?Tag, ?User) :-
   fileToAccess(?File, ?User),
   ?FileOwner := rootPrincipal(?File),
   ?FileOwner: fileTag(?File, ?Tag).
  tagAccess(?Tag, ?User)?
  label('DIFCAccessPolicy/$File').
 }.
```

- If the owner of a file labels the file with multiple tags, the DIFC policy requires access to all the tags.
- Tags of the same file can be specified individually.
- We use inference to track all the tags associated to a file and enforce the DIFC policy on file access.

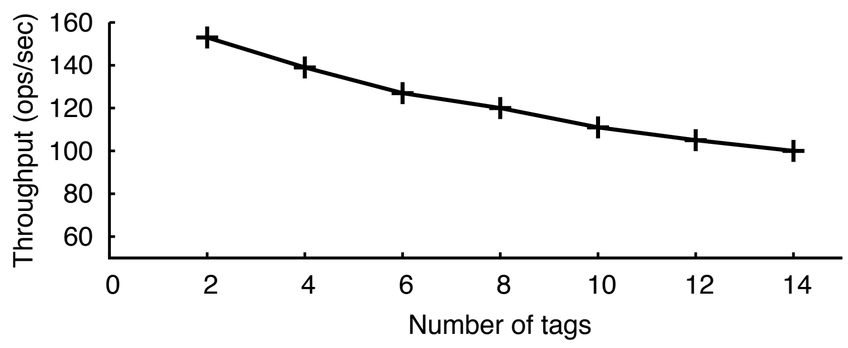
Access check using inference closure

```
defguard checkFileAccess(?User, ?File) :-
 spec('Check if a subject can access a file'),
 ?FileOwner := rootPrincipal(?File),
 ?FileTags := label(?FileOwner, "fileLabel/?File"),
 ?DifcRules := difcAccessPolicy(?User, ?File),
 ?Subgoals:= inferQuerySet(?DifcRules, ?FileTags),
 ?TagAccessPolicySet := label("tag-access-policy"),
 {
  link($BearerRef).
  link($Subgoals).
  link($TagAccessPolicySet).
 }.
```

- A guard approves a file access if the subject has access for each of the tags of the file.
- Using the DIFC policy set and the file's tag set, one can get an inference closure that contains the required tag-access in logic subgoals.
- The guard uses a credential context for access check that links to the subject's BearerRef, the logic sub-goal closure, and a standard tag access policy set.

Results

Throughput of compliance check with respect to the number of tags



Throughput of compliance check with respect to tag delegation depth