Information Flow Control on a Multi-Paradigm Web Application for SQL Injection Prevention

Meriam Ben Ghorbel-Talbi, François Lesueur, Gaetan Perrin

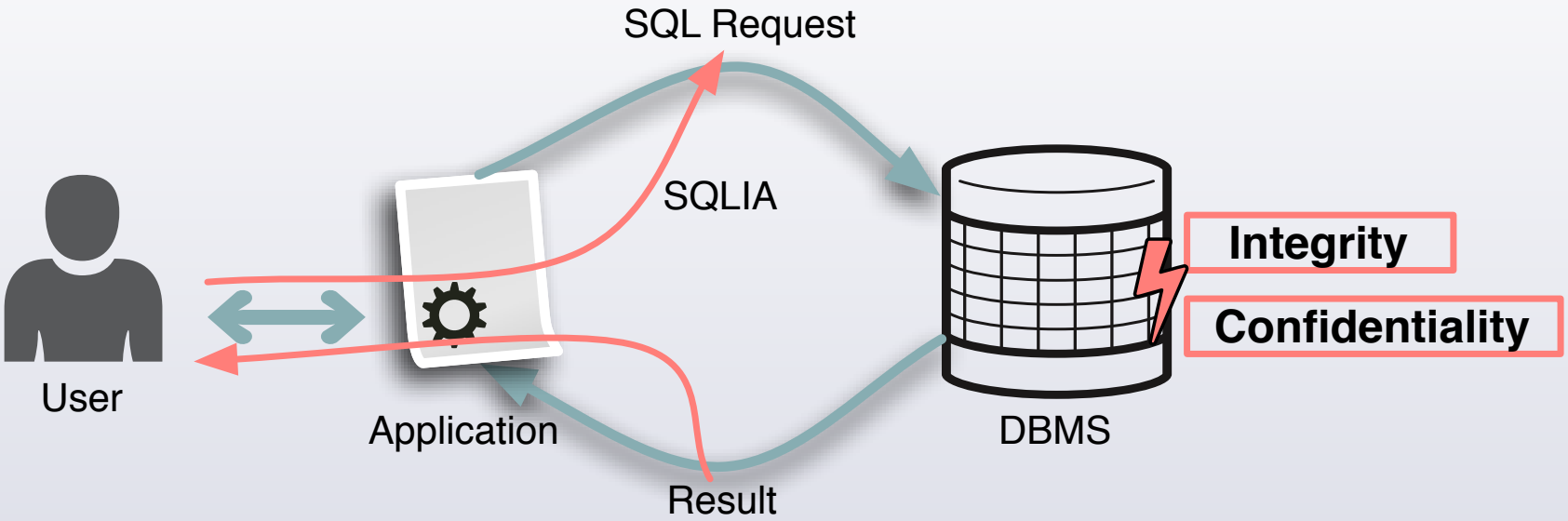
Laboratoire d'InfoRmatique en Image et Systèmes d'information

LIRIS UMR 5205 CNRS/INSA de Lyon/Université Claude Bernard Lyon 1/Université Lumière Lyon 2/École Centrale de Lyon

`http://liris.cnrs.fr`

ANR KISS project (ANR-11-INSE-0005)

SQL Injection Attacks



Example

```
uname = request.POST['username']
passwd = request.POST['password']
sql = "SELECT id FROM users WHERE username='" + uname
    + "' AND password='" + passwd + "'"
database.execute(sql)
```

SQLI Mitigations Status

A well-known vulnerability

- First public discussions in 1998
- Known mitigation techniques (special chars escaping, prepared statements)

But...

- Mitigations must be integrated during the development
- Requires competency and rigor during the *whole* development

⇒ Partial deployment of mitigations

- Widely deployed web applications are usually ok
- Internal or ad hoc applications are often vulnerable (depends on a *single* unprotected SQL query)

845 CVE the last 3 years, conforting internal empirical analysis

Generic approaches

Black/White list

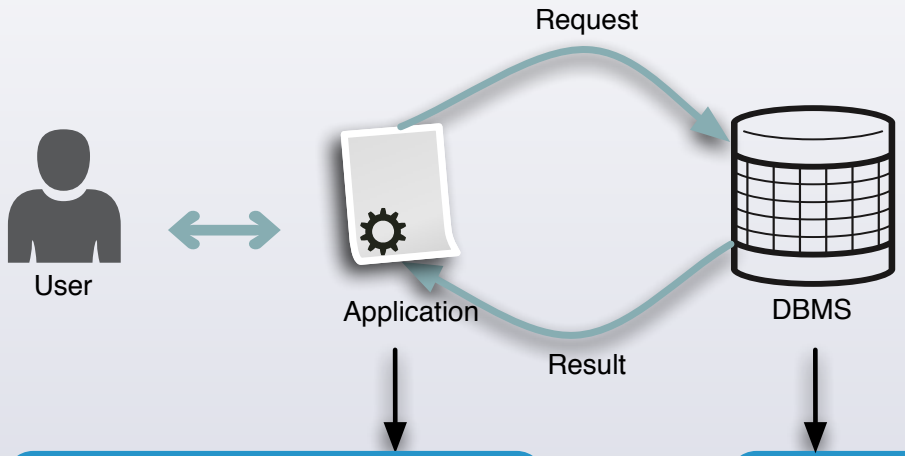
- Apache mod_security
- Oracle Database Firewall

Learning phase or signatures, false positives/negatives

Our proposition

- Do not try to detect/mitigate injection points
- SQL injection = information leakage (or tampering, but not considered here)
- Leverage Information Flow Control to block SQL injections

Existing Information Flow Control



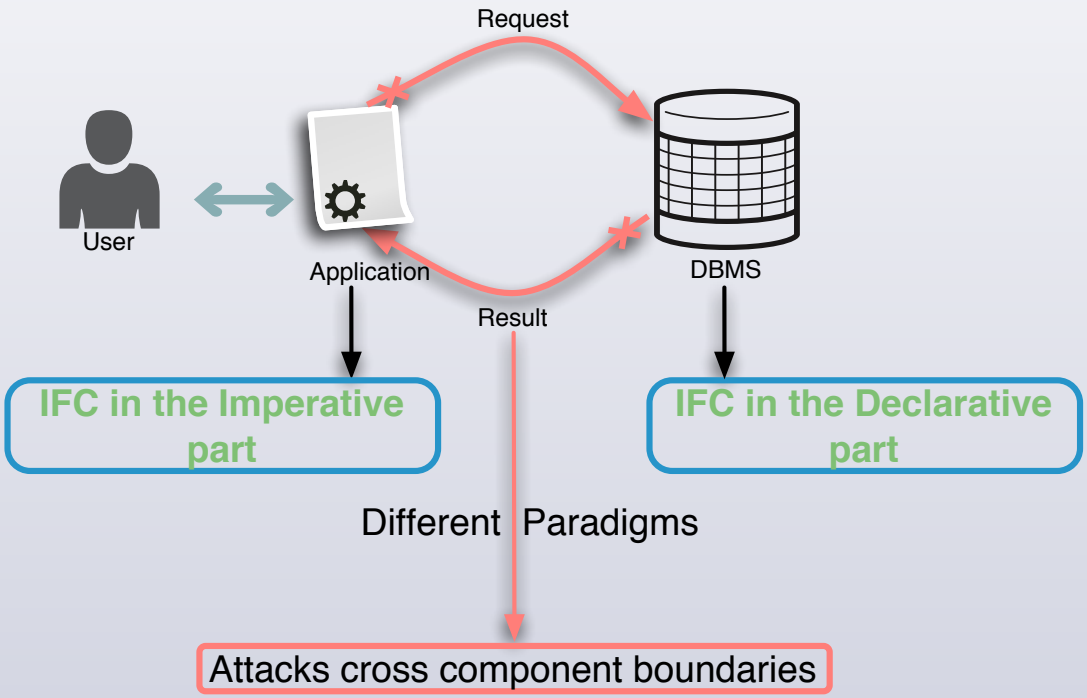
Imperative part

Application-wide tracking using security-typed languages with integrated information-flow control (tainted-untainted labels)

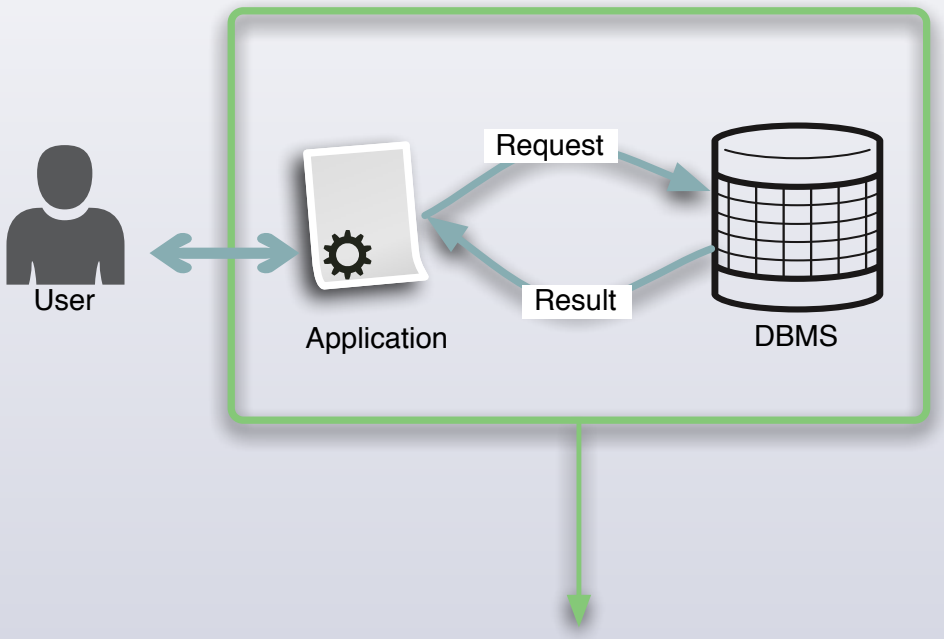
Declarative part

Database-wide flow tracking using multilevel secure (MLS) DBMSs, that enforce MAC policies in databases

Cross component attacks



End-to-end IFC



Tracking information across application-database boundaries

Our Proposition

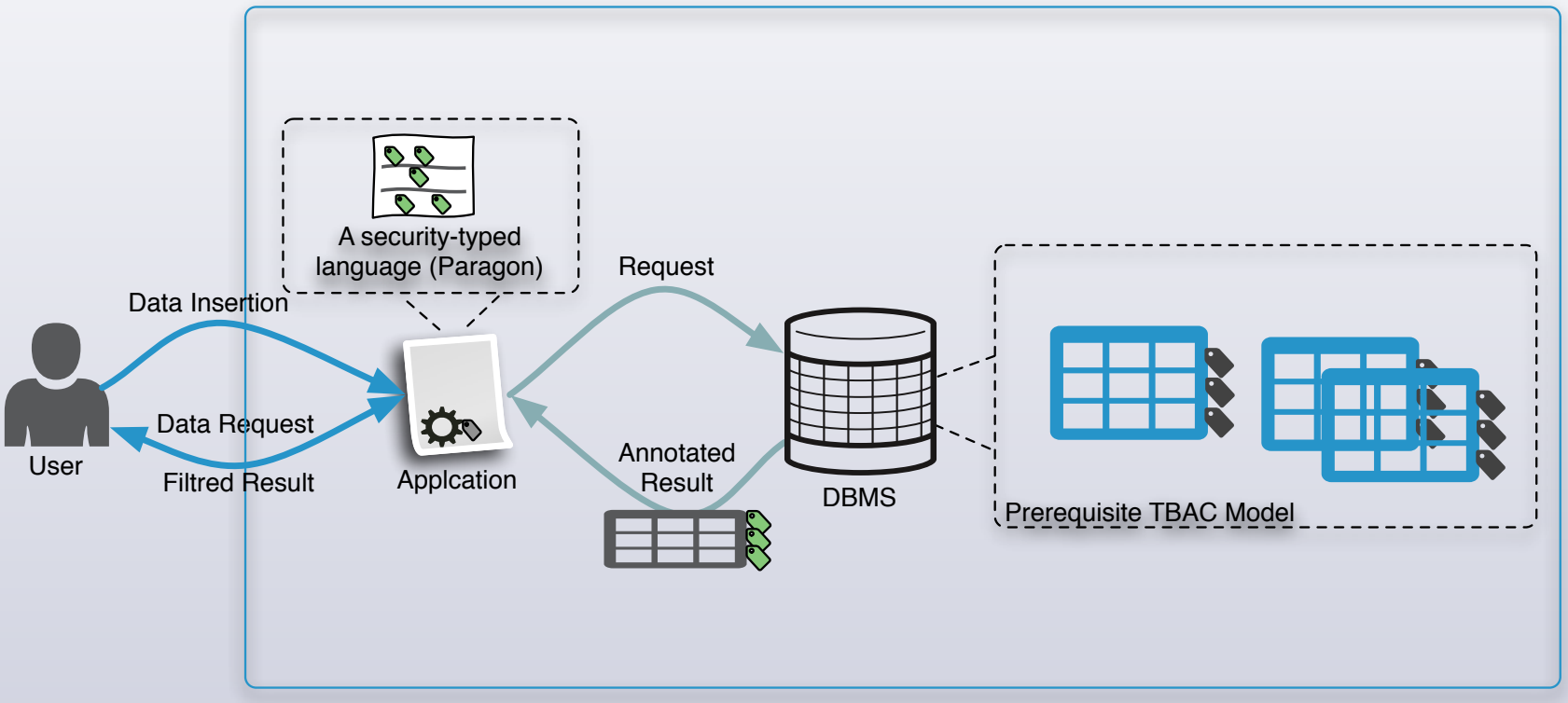
An integrated framework...

- Track information flows from the moment they enter the system until they leave it
- Dissemination Control to circumvent the threats of uncontrolled declassification

...allows to greatly reduce the burden of the developer

- Dynamic: to tag data entering the system rather than variables in the code (proxy service)
- End-to-end: to control output only when it leaves the system
- Dissemination control: data entering the system are tagged with their allowed ways of being declassified

System Architecture



Security Model

Security Model

The TBAC model

what? tuple-based fine-grained access control models

how? sticky policy paradigm

when? policies are combined and evaluated at query evaluation time

why? dissemination control, access is authorized in accordance with initial data producers

"One may access to a piece of information if he is authorized to access to the original tuples which contribute to it"

Decentralized IFC

- Systems with mutual distrust and decentralized authority
- JIF: an application of the DIFC to programming languages

The security Model

The Prerequisite TBAC

- A new instance to deal with declassification
- A user is allowed to access a data if and only if the prerequisites expressed by the data owners have been previously satisfied
- Each tuple t is annotated by an *s-tag*
  - An *s-tag* is a disjunction of atomic *tags*
  - $t_{auth}= ((p, U_v), U_r)$, p a set of prerequisites, U_v a set of validators, U_r a set of readers
  - An empty prerequisite means that readers can access this tuple without any conditions

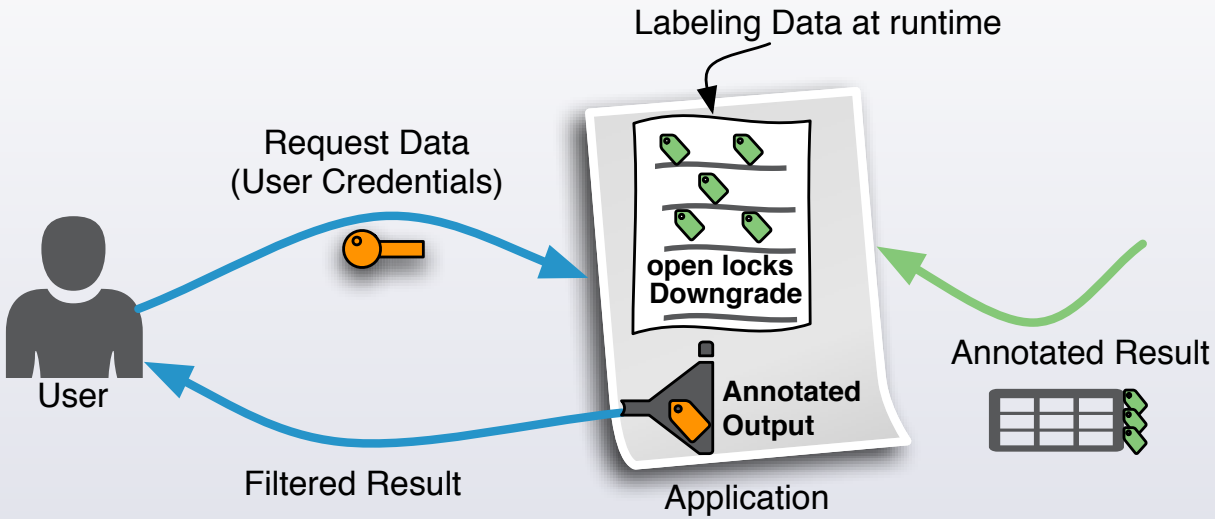
Application-side

Application-side

Paragon

- A security-typed extension to Java that is more general than JIF
- Builds on two basic components
  - Actors: principals or specific communication channels
  - Parameterized locks: boolean variables used to communicate the security relevant state of the program to the policy
- Paragon policy is similar to our policy definition: the prerequisite conditions and Locks are both used to specify how to declassify data

Application-side



- Runtime policy: used to instantiate variables policies using s-tags that are attached to the query result
- Downgrading: according to the system state locks are opened to declassify data policies
- Filtering: output channels are labeled with a security policy, only data that satisfy the security policy will flow from the application to the user

Database-side

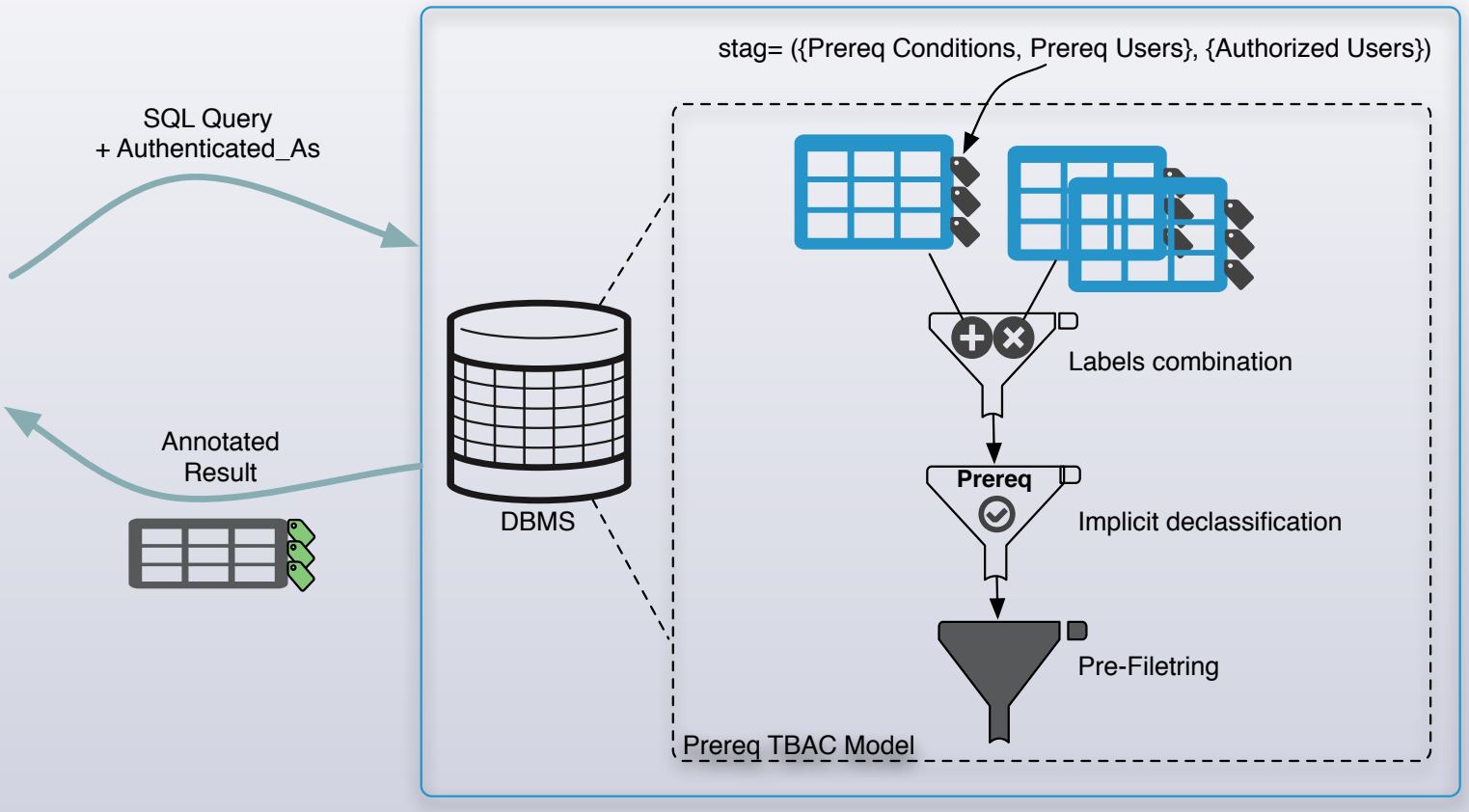
The Database-side

Policy Combination

- Combination of two tuples a and b
  - $a_{auth} = \{tag_{a_1} \vee \ldots \vee tag_{a_n}\}$
  - $b_{auth} = \{tag_{b_1} \vee \ldots \vee tag_{b_m}\}$
- If $t = a \bowtie b$, access to t requires access to both a and b
  - $t_{auth} = \{(tag_{a_1} \wedge tag_{b_1}) \vee \ldots (tag_{a_1} \wedge tag_{b_m}) \vee \ldots (tag_{a_n} \wedge tag_{b_m})\}$
  - $((p_{a_i}, U_{va_i}), U_{ra_i}) \wedge ((p_{b_j}, U_{vb_j}), U_{rb_j}) = \{((p, U_v), U_r) | p = p_{a_i} \cup p_{b_j}, U_v = U_{va_i} \cap U_{vb_j}, U_r = U_{ra_i} \cap U_{rb_j}\}$
- If $t = a \cup b$, access to t requires access to any of a and b
  - $t_{auth} = \{tag_{a_1} \vee \ldots \vee tag_{a_n} \vee tag_{b_1} \vee \ldots \vee tag_{b_m}\}$
  - Simplification must be applied for tags having the same prerequisite sets

$$\{((p, U_v), U_r) | U_v = U_{va_i} \cup U_{vb_j}, U_r = U_{ra_i} \cup U_{rb_j}\}$$

The Database-side



Proof-of-Concept

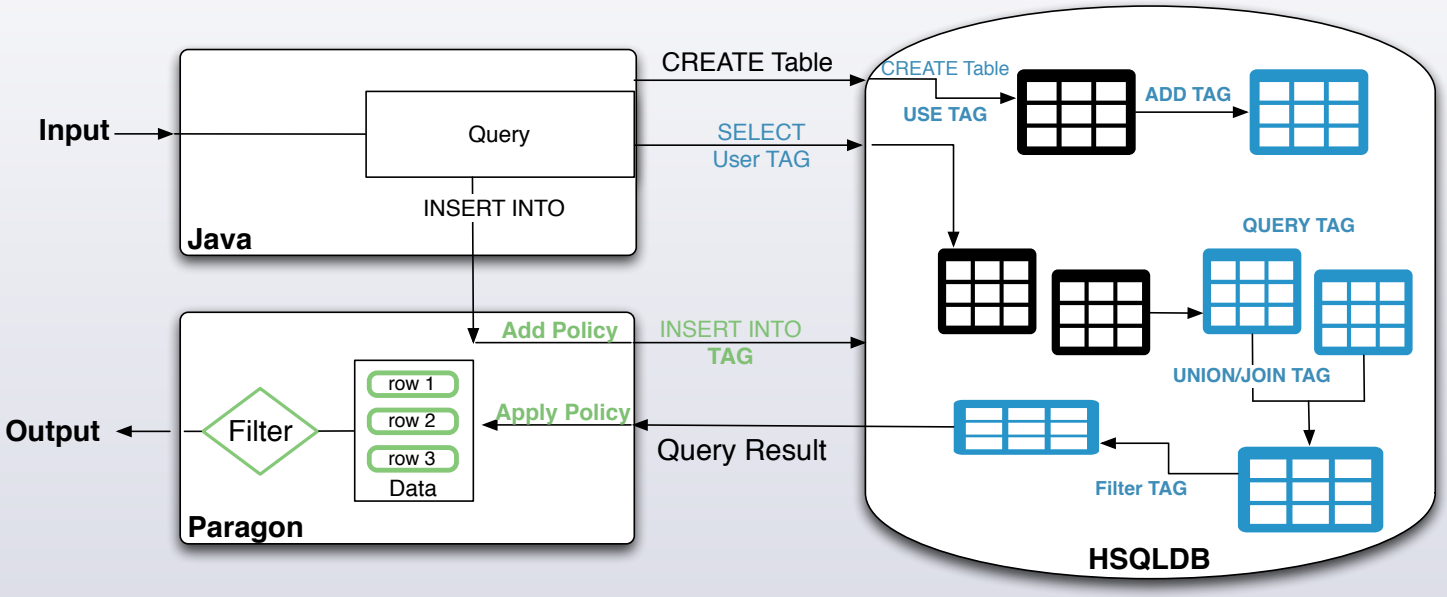
Database side

HSQLDB

A custom SQL parser that modifies all SQL queries at runtime

- Table creation
  - The USETAG command is added to the SQL query to automatically insert a new column called STAG
  - This column is used to store the security policy at the tuple level
- Data request
  - The UserTAG command is added to the SQL query to specify the current user credentials
  - The query result is intercepted to run our algorithm that combines s-tags according to the SQL query
  - The pre-filtering function decides which tuples can be sent to the application-side, according to the current user credentials

Architecture



Testing SQLIA

If the SQLI succeeds in the database-side and the whole table is returned, the output result shown to the user is filtered according to his credentials

Conclusion

- We focused on attacks threatening data confidentiality
- We proposed a proof of concept implementation to demonstrate that our approach is feasible
- Our aim was to let the application part as unchanged as possible
  - A custom SQL parser that modifies SQL queries at runtime to add and combine *s-tags*
  - A proxy service to label data entering the system and to dynamically propagate the *s-tags* from the database to the application
  - A filtering service to check outgoing data
- Future work
  - Prototype on a vulnerable third-party application
  - Evaluation of the performance

Information Flow Control on a Multi-Paradigm Web Application for SQL Injection Prevention

Meriam Ben Ghorbel-Talbi, François Lesueur, Gaetan Perrin

Laboratoire d'InfoRmatique en Image et Systèmes d'information

LIRIS UMR 5205 CNRS/INSA de Lyon/Université Claude Bernard Lyon 1/Université Lumière Lyon 2/École Centrale de Lyon

`http://liris.cnrs.fr`

ANR KISS project (ANR-11-INSE-0005)