ORACLE

Mimicry-Based Testing of Runtime SQLi Prevention Approaches

Anjana Perera, François Gauthier, Kostyantyn Vorobyov, Matthew Harris, Paddy Krishnan
Oracle Labs
Brisbane, Australia

Agenda

- Runtime Application Self Protection
  - Preventing SQL injections: Focus on data exfiltration
- Testing self protections
  - Are synthesized protections adequate?
- Fuzzing as test generator
- Results and discussions

Runtime Application Self Protection

- SQL injection is still prevalent.
- Most SQLi defense techniques detect *syntactic* anomalies.

- Our previous work[1] synthesized allowlists based on *information flow.*

```
SELECT fname, lname, email FROM members WHERE
    dept='HR';

SELECT dept, email FROM members WHERE
    lname='Baker' OR dept='Sales';
```

fname	lname	ssn	dept	email
John	Adams	XXXX	HR	j.adams@foo.com
Mary	Baker	XXXX	HR	m.baker@bar.com
Juan	Lopez	XXXX	Sales	j.lopez@foo.com
Amir	Zafar	XXXX	Sales	a.zafar@bar.com

- Based on the example queries, column `ssn` should not be disclosed.

1: K. Vorobyov, F. Gauthier, P. Krishnan, Synthesis of Allowlists for Runtime Protection against SQLi, ICSE 2024

Testing Synthesized Protections

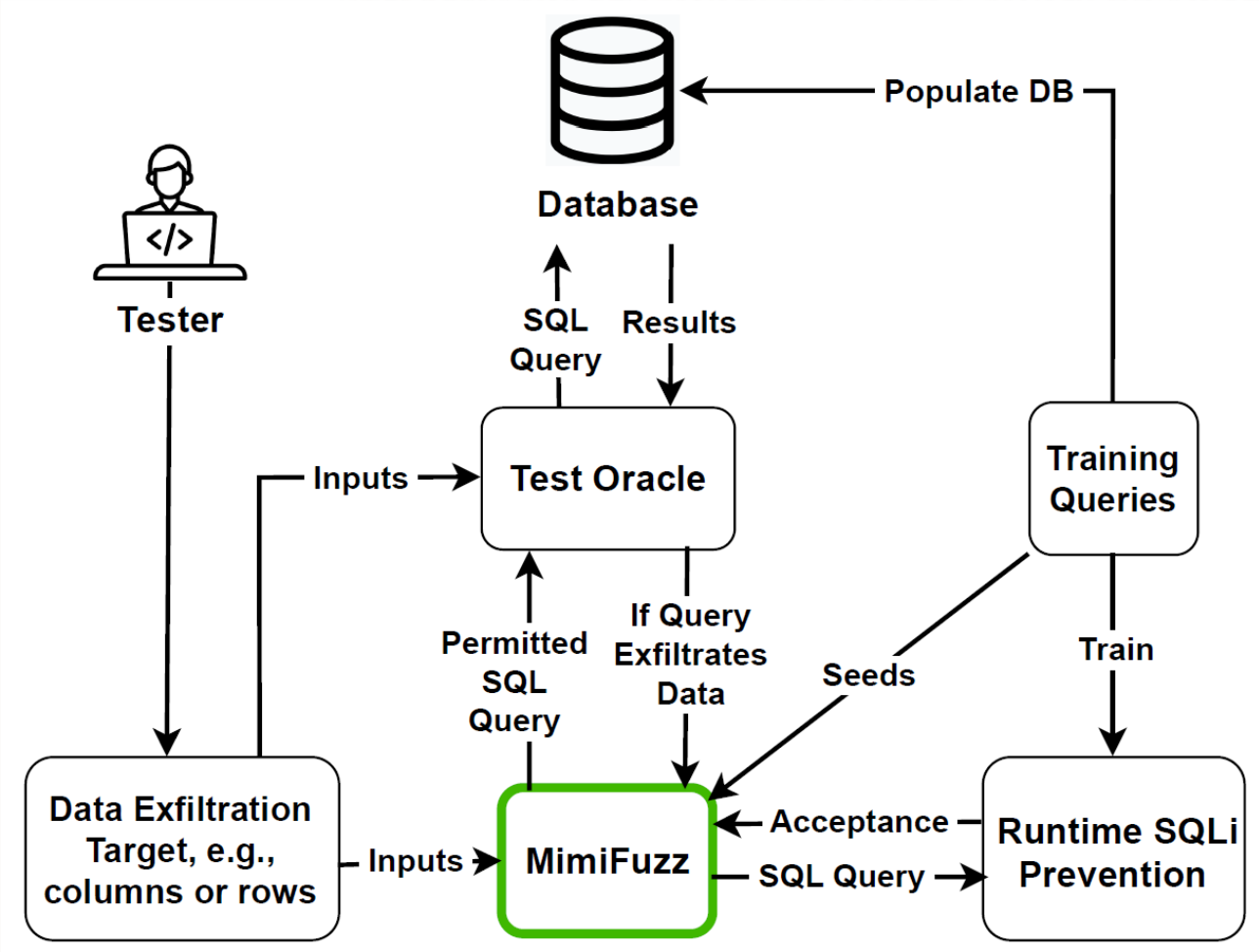
- Existing SQLi testing techniques are not effective for testing defenses.
  - Significantly change the structure of queries
  - Easily blocked by existing syntax-based tools, e.g., WAF, DB-Firewall

- Are existing SQLi defenses vulnerable to information flow-based attacks?
  - Keep the syntactic structure similar to known benign queries
  - Use mimicry to change the information flow (semantics) of the query

Approach and Architecture

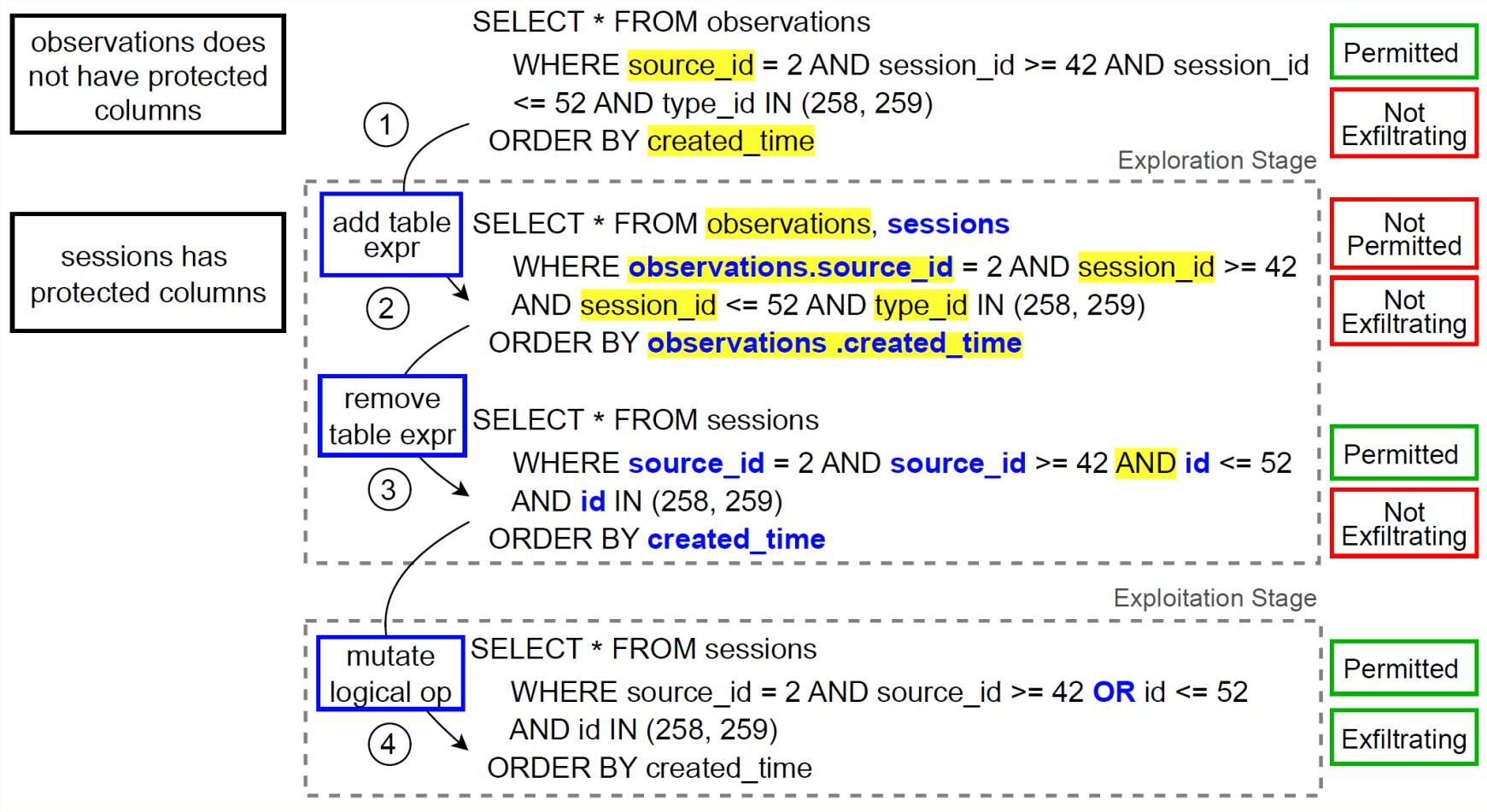
MimiFuzz

1. Tester sets the exfiltration target.
   - e.g., exfiltrate data from rows or columns
2. Training queries are used to:
   - Seed MimiFuzz with accepted queries
   - Build a test oracle of disclosable rows and/or columns
3. Queries are generated in two phases:
   - Exploration: To find accepted syntax
   - Exploitation: To exfiltrate data
4. Generated queries are:
   - Tested against protections for acceptance
   - Checked for exfiltration by the test oracle



Example

1. Benign training query.
2. Exploration expands query with new syntax: [rejected]
3. Exploration removes offending syntax and replaces a column name: [accepted]
4. Exploitation mutates the **`WHERE`** clause to match more values: [accepted and exfiltrating]



Results

- Compared MimiFuzz with SQLMap against state-of-the-art protections such as *SQLBlock* and *SOFIA*
  - Both *SQLBlock* and *SOFIA* use analysis of syntactic features
  - Sqlmap relies on coarse syntax transformations and *could not generate a single exfiltrating query*
  - MimiFuzz ran for 10 rounds and consistently generated exploits on all runs except two marked with *

Dataset from benchbase	# Data exfiltrating queries	
	SQLBlock	SOFIA
auctionmark	3	1
chbenchmark	3	3
epinions	-	1
otmetrics	-	1*
resourcestresser	3	-
seats	3	2*
smallbank	3	1
tatp	3	1
tpcc	3	1
tpch	3	-
twitter	3	1
voter	3	1
wikipedia	3	1
Ycsb	-	2

Conclusion

- Fuzzing that preserves structural information necessary for testing SQLi defences.
  - Compared our work with Sqlmap which does not preserve structure.
- Separating fuzzing into exploration and exploitation phases useful.
- Training queries provide good seeds (including disclosable data and structural information).

Future Work

- Focus on row exfiltration via mutating predicates
  - Structure preserved
- Large scale experiment to check for effectiveness of MimiFuzz
- Need ground truth: What is correct behaviour?

ORACLE