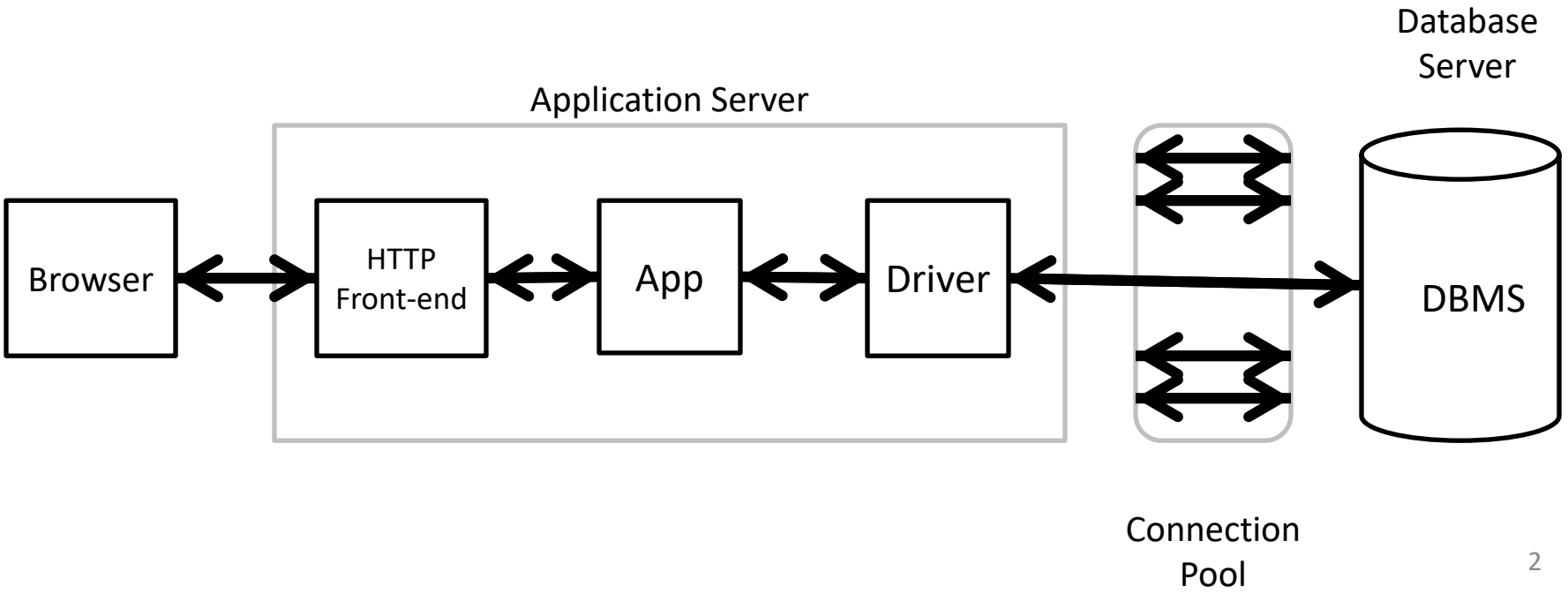
Ensuring Authorized Updates in Multi-user Database-Backed Applications

Kevin Eykholt, Atul Prakash, Barzan Mozafari

University of Michigan Ann Arbor

Database Backend

- Web Applications allow users to remotely access services
- Information stored in database



Symantec Patches High Risk Vulnerabilities in Endpoint Protection

By SecurityWeek News on March 21, 2016

Symantec has released an update for its Symantec Endpoint Protection (SEP) to resolve three High risk security vulnerabilities in the product.

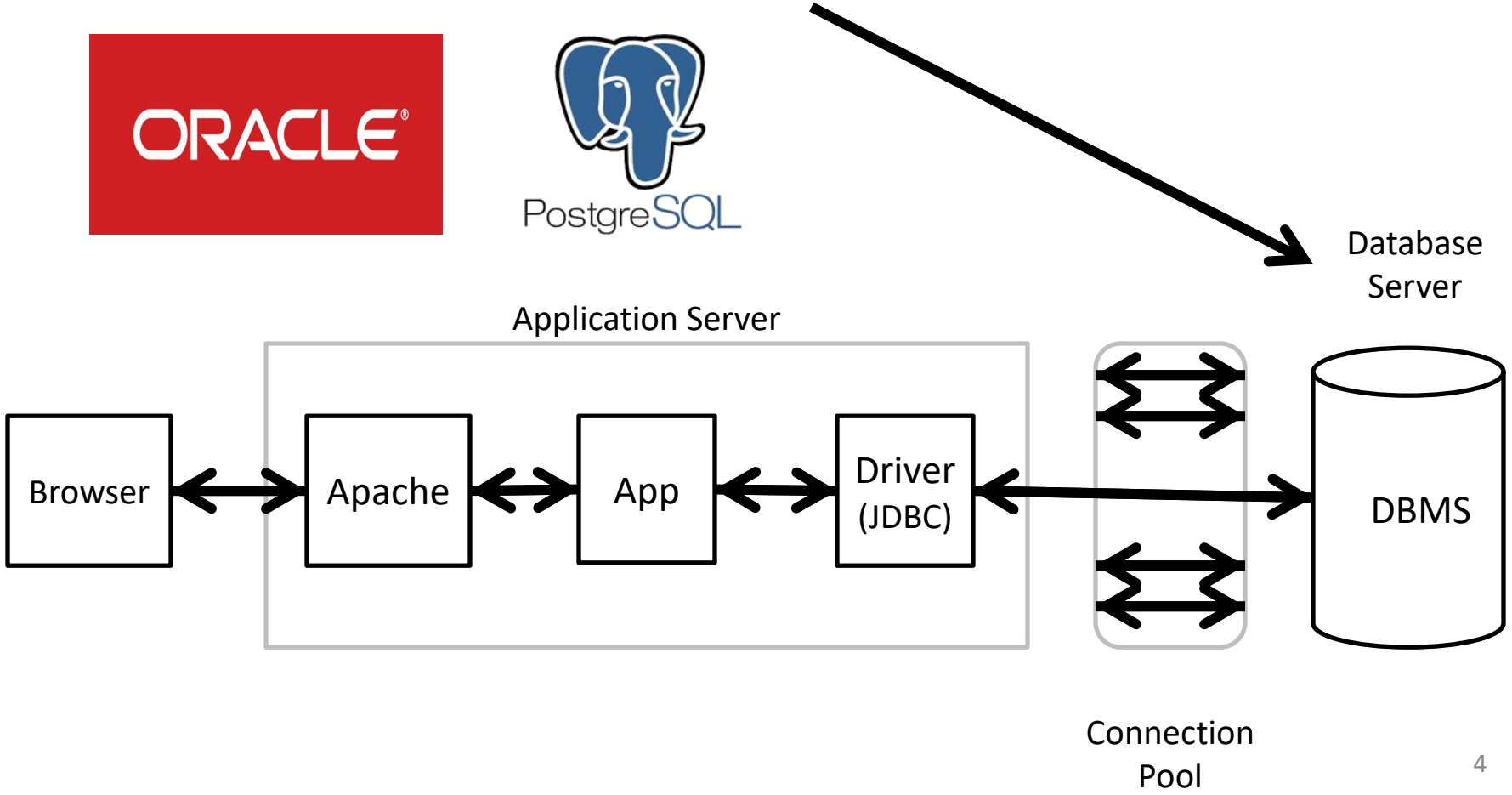
According to an **advisory** issued Mar. 17, the security flaws in Symantec Endpoint Protection could potentially result in authorized users with low privileges gaining elevated access to the Management Console. Moreover, the security firm warns that SEP Client security mitigations could be bypassed to achieve arbitrary code execution on a targeted client.

The first of the three security issues is a cross-site request forgery vulnerability in the management console for SEPM **(CVE-2015-8152)**, caused by an insufficient security check in SEPM. An authorized but less-privileged user could gain unauthorized elevated access to the SEPM management console by including arbitrary code in authorized logging scripts.

In addition to the CSRF issue, Symantec resolved an SQL injection vulnerability in SEPM (**CVE-2015-8153**). This security flaw can also be exploited by an authorized, logged-in user to potentially elevate access to administrative level on the application.

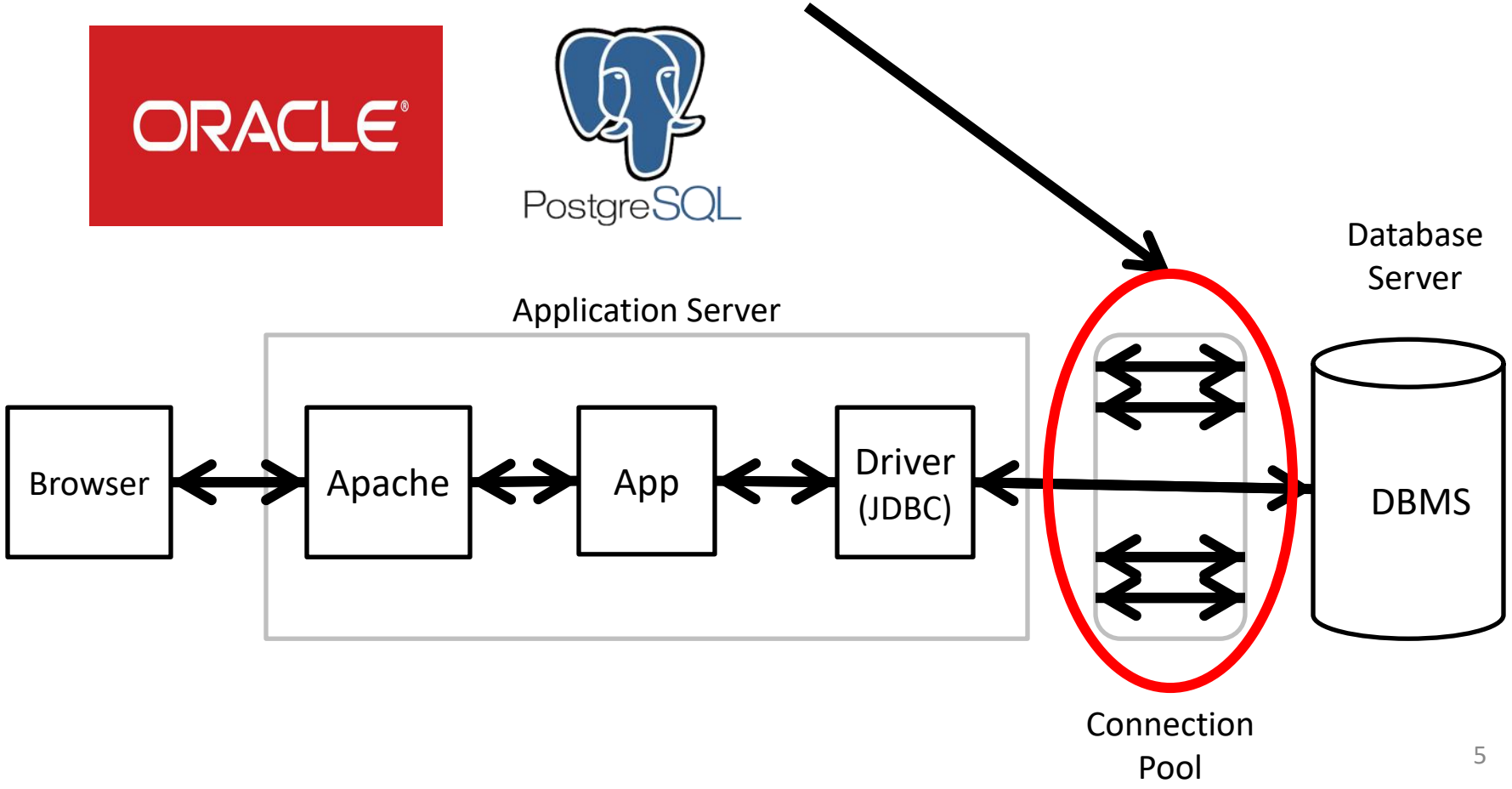
Built-in Database Access Controls

Some DMBS provide fine-grained access control based context of connected user



Built-in Database Access Controls

But database user is not the same as application user!



Application based Access Controls

- CLAMP and Nemesis both define per-user access control policies on each table

- Most existing work defines access control policies using **database views**

B. Parno, J. M. McCune, D. Wendlandt, D. G. Andersen, and A. Perrig. Clamp: Practical prevention of large-scale data leaks. In S & P, 2009.

M. Dalton, C. Kozyrakis, and N. Zeldovich. Nemesis: Preventing authentication & access control vulnerabilities in web applications. In USENIX, 2009.

Database View

- Database views restrict a user to a portion of the database using a SELECT query
- “Only allow customers to view their own orders”

```
SELECT O.*
FROM orders O
WHERE O.cust_id = $current_id
```

Database View

- Existing techniques use views to restrict read/write access
- The same query can also express the write policy: “Only allow customers to update their own orders”

```
SELECT O.*
FROM orders O
WHERE O.cust_id = $current_id
```

What if you can’t map a user to authorized rows in the table?

Problem

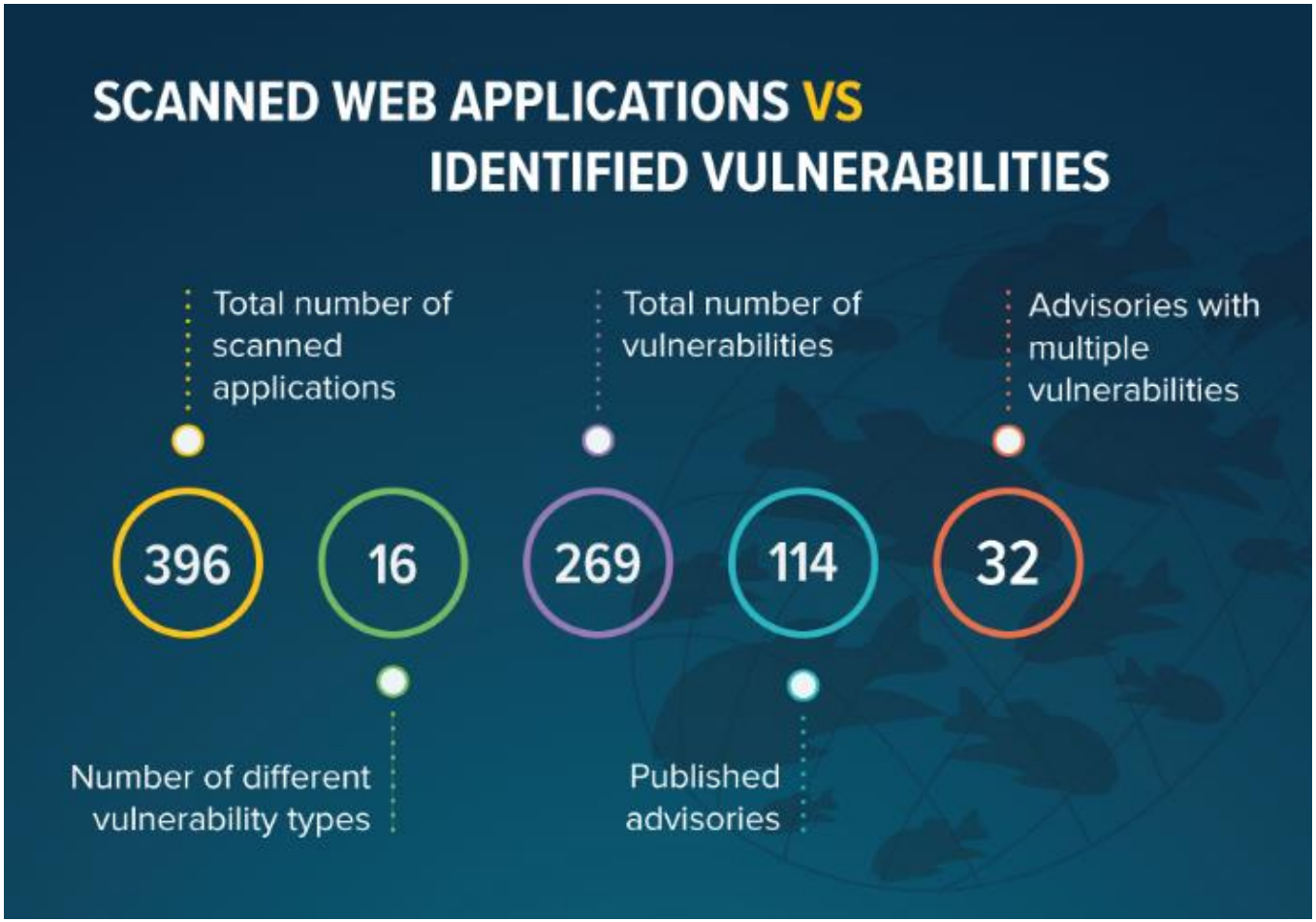
- “Customers can only leave reviews for items they have purchased”

```
SELECT R.*
FROM reviews R, orders_products P, orders O
WHERE O.cust_id = current_id AND
      O.oid = OP.oid AND
      OP.product_id = R.product_id AND
      R.cust_id = $current_id
```

Survey of Existing Web Applications

Web App	Total Tables	**Tables Requiring Join Policy**
Wordpress	12	**4 (33%)**
hotCRP	24	**6 (25%)**
LimeSurvey	36	**18 (50%)**
osCommerce	40	**4 (10%)**
MediaWiki	48	**10 (21%)**
WeBid	55	**5 (9%)**
Drupal	60	**12 (20%)**
myBB	75	**8 (11%)**
ZenCart	96	**18 (19%)**
Cyclos	185	**24 (13%)**
Average Percent		**21%**

Insecure Code



R. Abela. Infographic: Statistics about the security scans of 396 open source web applications. https://www.netsparker.com/s/r/bl/2016_statistics_open_source_web_application_scans.png

Insecure Code

Symantec Patches High Risk Vulnerabilities in Endpoint Protection

By SecurityWeek News on March 21, 2016

Symantec has released an update for its Symantec Endpoint Protection (SEP) to resolve three High risk security vulnerabilities in the product.

According to an **advisory** issued Mar. 17, the security flaws in Symantec Endpoint Protection could potentially result in authorized users with low privileges gaining elevated access to the Management Console. Moreover, the security firm warns that SEP Client security mitigations could be bypassed to achieve arbitrary code execution on a targeted client.

The first of the three security issues is a cross-site request forgery vulnerability in the management console for SEPM **(CVE-2015-8152)**, caused by an insufficient security check in SEPM. An authorized but less-privileged user could gain unauthorized elevated access to the SEPM management console by including arbitrary code in authorized logging scripts.

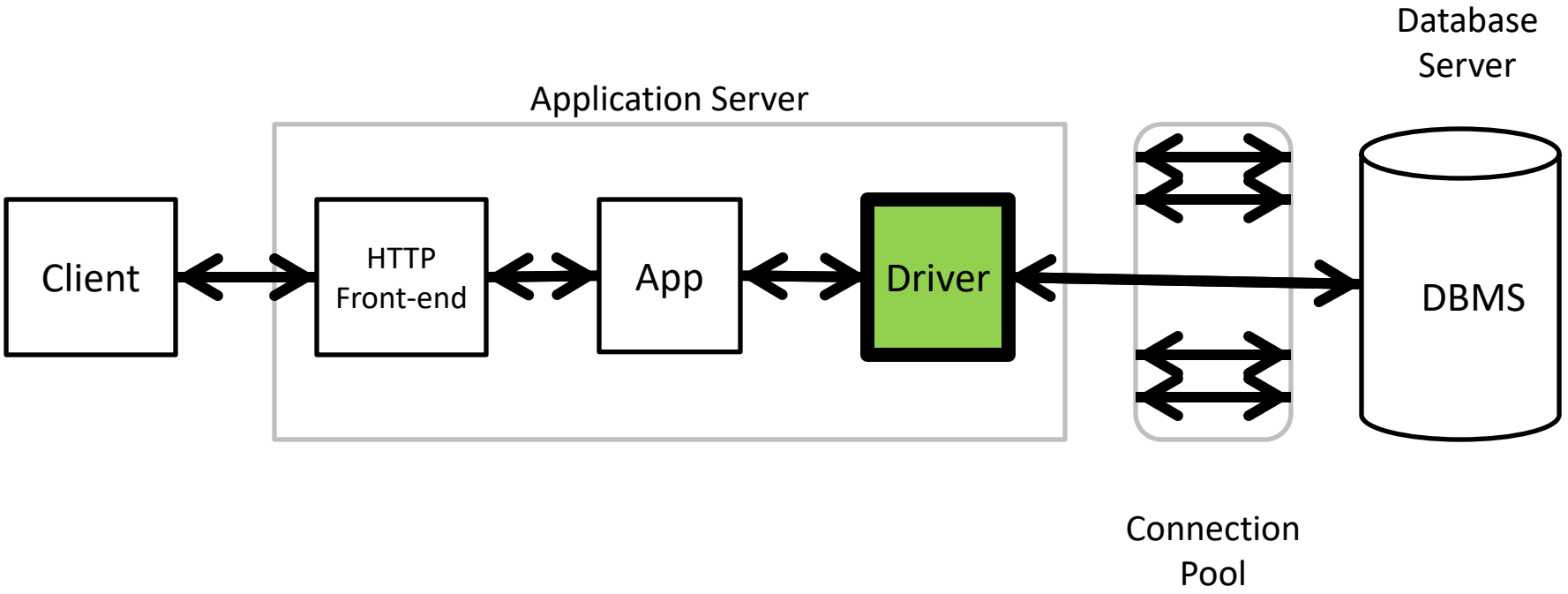
In addition to the CSRF issue, Symantec resolved an SQL injection vulnerability in SEPM (**CVE-2015-8153**). This security flaw can also be exploited by an authorized, logged-in user to potentially elevate access to administrative level on the application.

Symantec patches high risk vulnerabilities in endpoint protection.
http://www.securityweek.com/symantec-patches-high-risk-vulnerabilities-endpoint-protection

Design Goals

- Generality: Access control policy can be enforced read/write queries
- Correctness: Current user can only view/modify authorized information
- Architectural Compatibility:
  - Solution works with existing web applications without requiring major changes
  - Solution is not database specific
- Simple: Preliminary knowledge overhead is low

Proposed Solution



Implement the control in the database driver

Solution Outline

- **What is Query Safety?**
- How can Query Safety be enforced?
- Experiments with proposed methodology

Database Policies

- Two types of policies:
  - Read Policy
  - Write Policy
- A policy is composed of a set of rules for each database table

Policy Definitions

- Customers can only view/modify their own orders

Read/Write Policy

```
SELECT O.*
FROM orders O
WHERE O.cust_id = current_id
```

Policy Definitions

- Customers can view items available in the store

Read Policy

```
SELECT P.*
FROM products P
```

Write Policy

```
SELECT P.*
FROM products P
WHERE 1=0
```

Policy Definitions

- Customers can read any review, but only leave reviews for items they purchased

Read Policy

```
SELECT R.*
FROM reviews R
```

Write Policy

```
SELECT R.*
FROM reviews R, orders_products P, orders O
WHERE  O.cust_id = current_id AND
       O.oid = OP.oid AND
       OP.product_id = R.product_id AND
       R.cust_id = current_id
```

Query Safety

- A **safe** query is one that allows a user to only view/modify authorized tuples in the database identified by the security policy
  - A read query is safe if it is **read-safe**
  - A write query is safe if it is **write-safe**

Read-Safe Query

- A query is **read-safe** if the query's result is unchanged when executed on only the tables a user is authorized to access

Read-Safe Query

Read Policy

```
SELECT O.*
FROM orders O
WHERE O.cust_id = 1
```

Read-Safe

```
SELECT orders_id
FROM orders
WHERE cust_id = 1
```

Not Read-Safe

```
SELECT *
FROM orders
```

Write-Safe Query

- A query is **write-safe** if:
  1. The query is read-safe
  2. The query does not modify unauthorized tuples
     - The results of the query should not change if the query is restricted to modifying tuples in the write policy

Write-Safe Query

Write Policy

```
SELECT O.*
FROM orders O
WHERE O.cust_id = 1
```

Write-Safe Queries

```
DELETE FROM orders
WHERE cust_id = 1
```

Not Write-Safe Queries

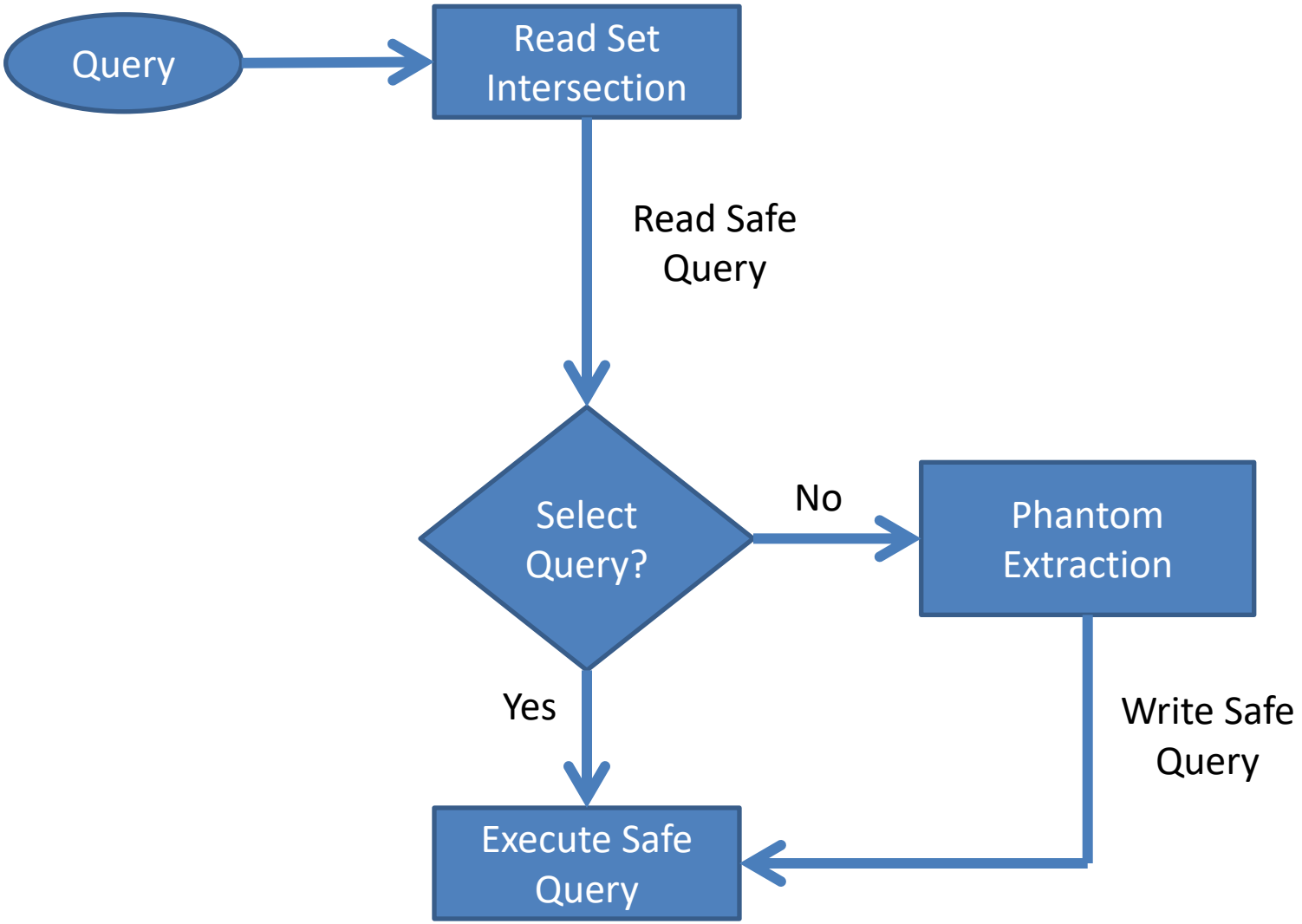
```
DELETE FROM orders
```

```
INSERT INTO orders
AS     SELECT *
       FROM orders
```

Solution Outline

- What is Query Safety?
- **How can Query Safety be enforced?**
- Experiments with proposed methodology

Ensuring Query Safety



Read Set Intersection

- Any query can be transformed to a read-safe query by adding additional conditions to the WHERE clause of any SELECT queries

Read Policy

```
SELECT O.*
FROM orders O
WHERE O.cust_id = 1
```

```
SELECT *
FROM orders
```

```
SELECT *
FROM orders
WHERE O.cust_id = 1
```

Phantom Extraction

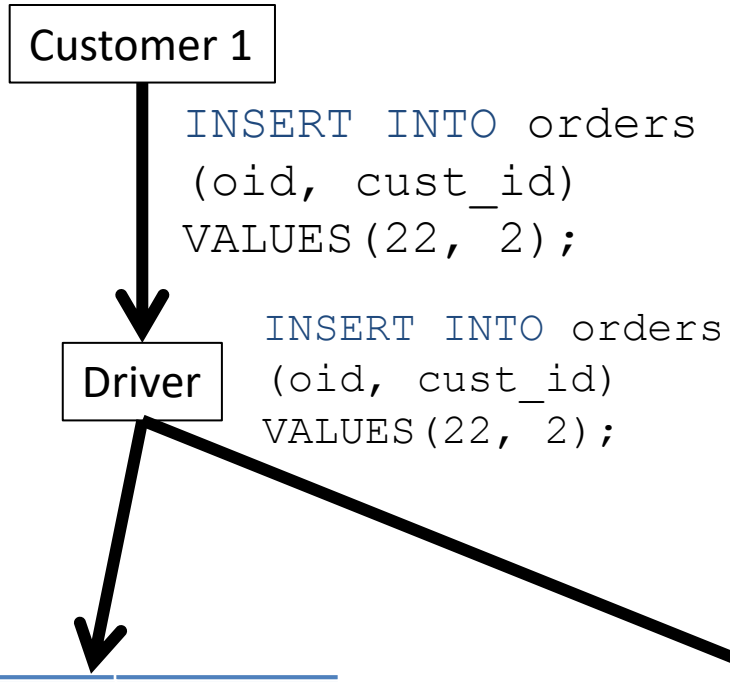
1. Transform the query into a read-safe query (Read Set Intersection)
2. Modify the resulting query to only update tuples authorized by the write policy

- Two strategies for step 2:
  - V-Copy
  - No-Copy

V-Copy

- Determines query safety using temporary tables
- Given a query, V-Copy always results in safe behavior

V-Copy



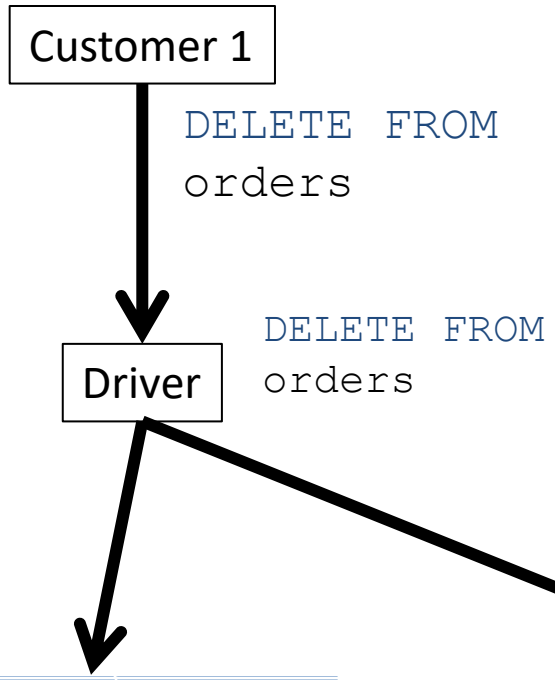
1. Copy schema of T
2. Execute the query on the copy
3. Check if result is allowed by the write policy
4. Execute Null query

oid	cust_id
4	1
12	2
19	3
21	2

oid	cust_id
22	2

```
SELECT O.*
FROM orders O
WHERE O.cust_id = 1
```

V-Copy



1. Copy authorized write tuples from T
2. Execute the query on the copy
3. Check if result is allowed by the write policy
4. Propagate changes

oid	cust_id
12	2
19	3
21	2

oid	cust_id

```
SELECT O.*
FROM orders O
WHERE O.cust_id = 1
```

No-Copy

- Can only use when:
  - Write Policy for table does not contains a join
  - Query is not a nested INSERT
  - And if the query is UPDATE, the SET clause only contains static values

No-Copy

- If the query is DELETE, append additional conditions to WHERE clause

Write Policy

```
SELECT O.*
FROM orders O
WHERE O.cust_id = 1
```

```
DELETE
FROM orders
```

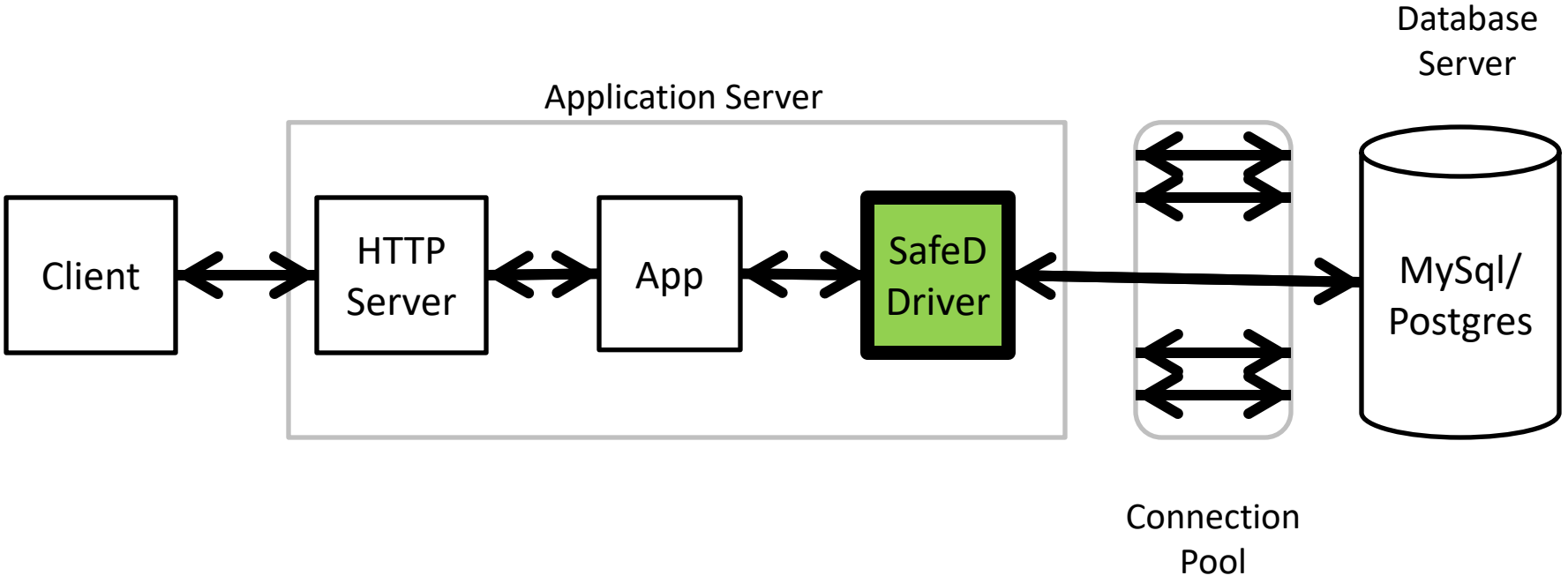
```
DELETE
FROM orders
WHERE O.cust_id = 1
```

Solution Outline

- What is Query Safety?
- How can Query Safety be enforced?
- **Experiments with proposed methodology**

SafeD

- We created SafeD, a custom JDBC driver, that implements both V-Copy and No-Copy



Benchmark

- TPC-C Benchmark
  - Provided in OLTPBenchmark
  - 5 transaction types
  - Scale factor: 20
  - Worker count:
    - 60 (PostgreSQL)
    - 100 (MySQL)
  - Phase Duration: 10 minutes

Access Roles and Policy

- Customer: Executes new order, order status, payment transactions
- Managers: Executes delivery and stock transactions

Table Name	Customer	Manager
Customer(C_ID, C_D_ID, C_W_ID)	=(CID,DID,WID)	Full Access
District(D_ID, D_W_ID)	=(DID,WID)	=(DID,WID)
Warehouse(W_ID)	=(WID)	Full Access
OOrder(O_C_ID, O_D_ID, O_W_ID)	=(CID,DID,WID)	=(DID,WID)
New_Order(NO_O_ID)	Contain (OID) in OOrder	Full Access
Order_Line(OL_O_ID)	Contain (OID) in OOrder	Full Access
History(H_C_ID, H_D_ID, H_W_ID)	=(CID,DID,WID)	Full Access
Item	Full Access	Full Access
Stock	No Access	Full Access

Performance Measures

- Average Latency

$$AL^{S,r} = \frac{\sum_{i=1}^{N} L_i^{S,r}}{N}$$

- Average Throughput

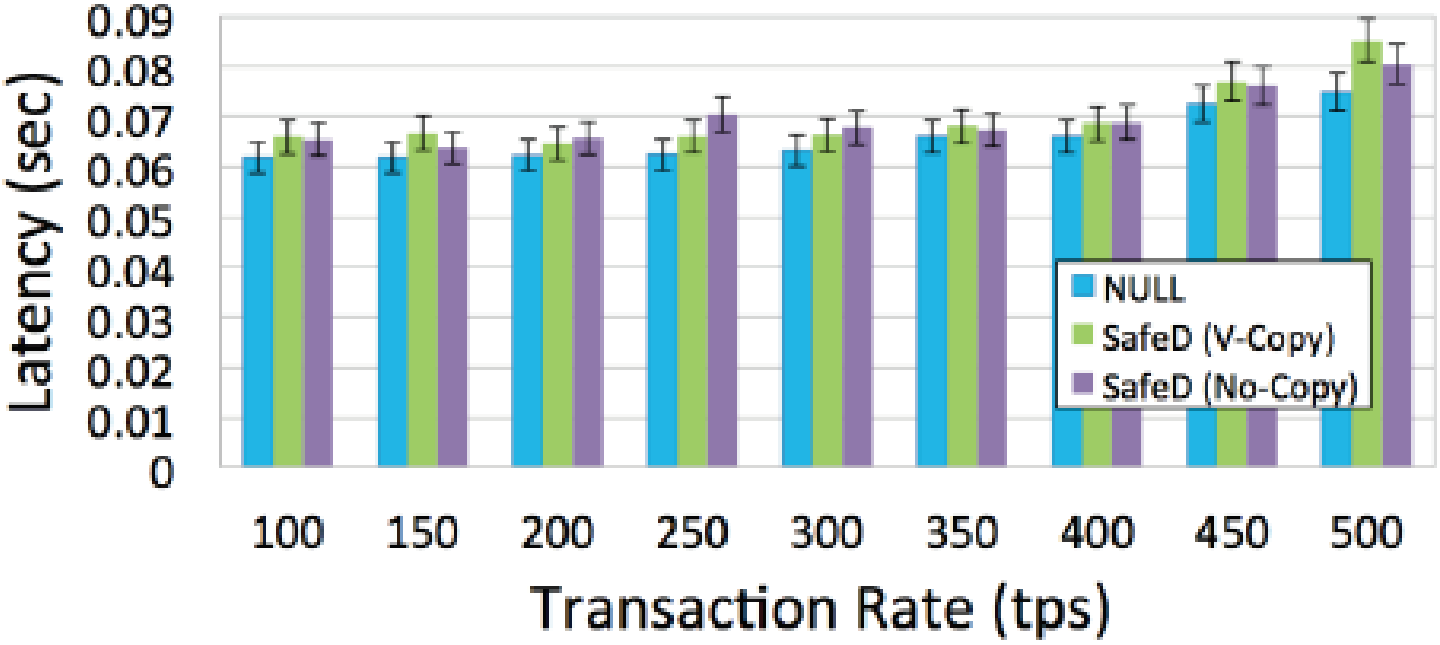
$$AT^{S,r} = \frac{\sum_{i=1}^{N} T_i^{S,r}}{N}$$

Performance Measurements

- When queries in the workload are safe
  - What is the performance overhead compared to a database without built-in access controls?
  - How does SafeD compare to an existing built-in access control mechanism
- How does overall performance vary as the number of unsafe queries in the workload increase?

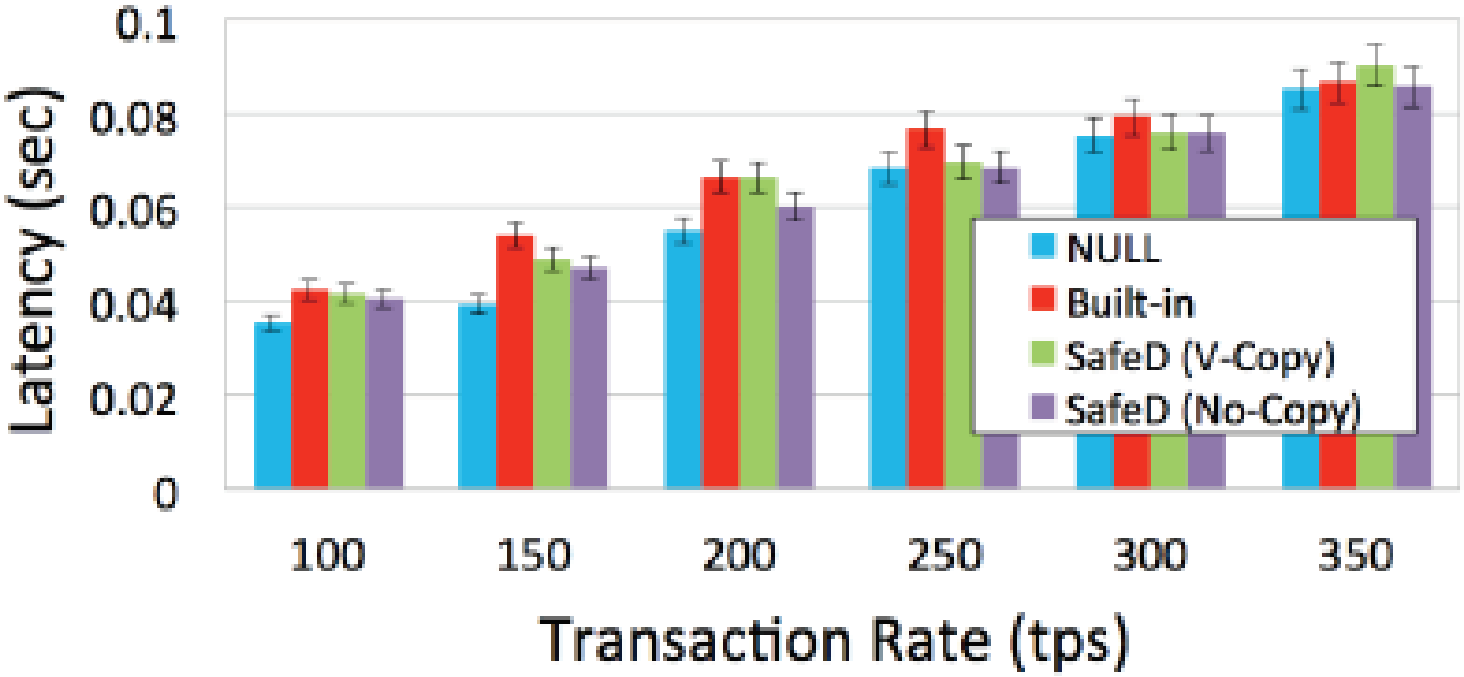
MySQL Performance Results

No-Copy has an average latency overhead of 5.9%
V-Copy has an average latency overhead of 6.1%



PostgreSQL Performance Results

SafeD access controls has comparable performance to built-in Postgres Access Control



Modified Security Policy

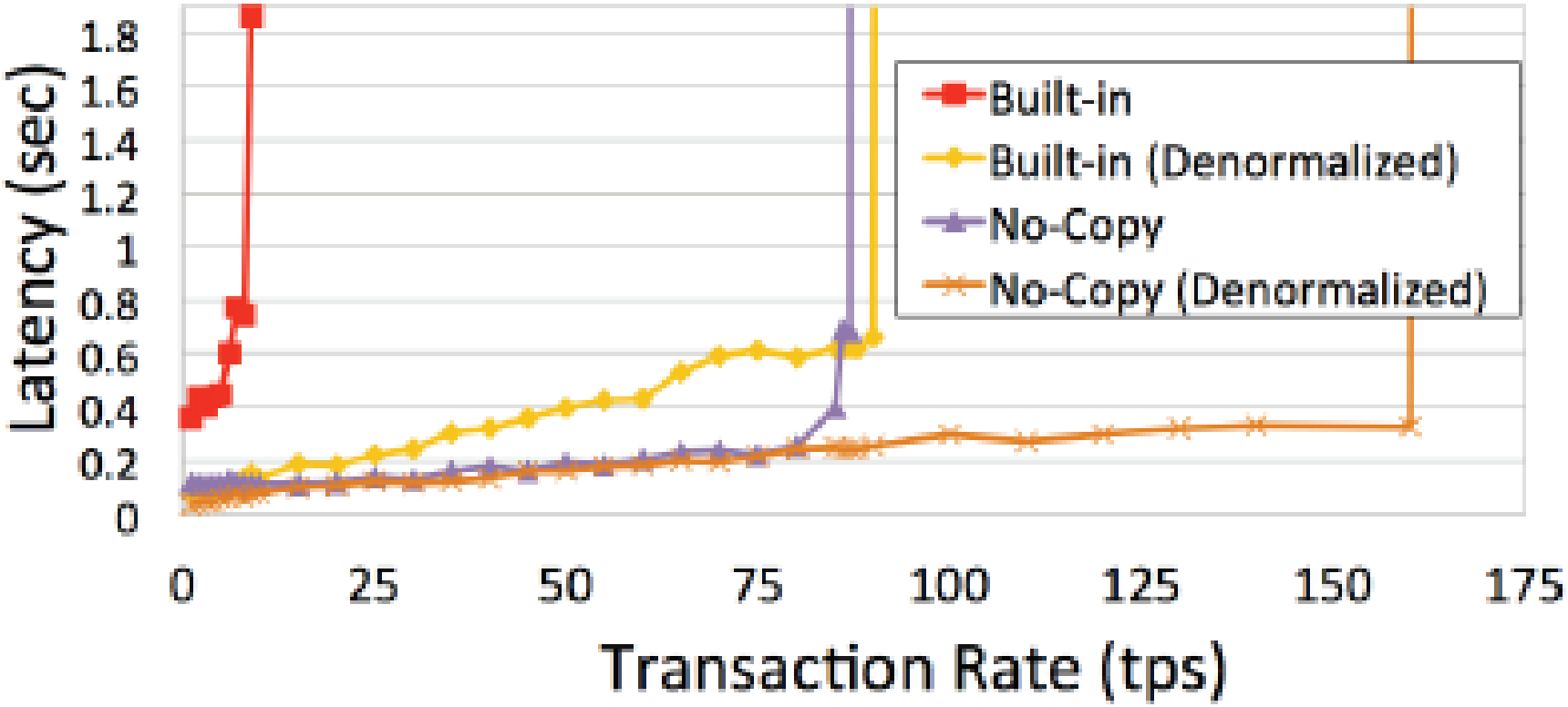
Table Name	Customer	Manager
Customer(C_ID, C_D_ID, C_W_ID)	=(CID,DID,WID)	Full Access
District(D_ID, D_W_ID)	=(DID,WID)	=(DID,WID)
Warehouse(W_ID)	=(WID)	Full Access
OOrder(O_C_ID, O_D_ID, O_W_ID)	=(CID,DID,WID)	=(DID,WID)
New_Order(NO_O_ID)	Contain (OID) in OOrder	Full Access
Order_Line(OL_O_ID)	Contain (OID) in OOrder	Full Access
History(H_C_ID, H_D_ID, H_W_ID)	=(CID,DID,WID)	Full Access
Item	Full Access	Full Access
Stock	No Access	Full Access

Security Policy – Policy 2

Table Name	Customer	Manager
Customer(C_ID, C_D_ID, C_W_ID)	=(CID,DID,WID)	Full Access
District(D_ID, D_W_ID)	=(DID,WID)	=(DID,WID)
Warehouse(W_ID)	=(WID)	Full Access
OOrder(O_C_ID, O_D_ID, O_W_ID)	=(CID,DID,WID)	$O\_C\_ID > 0$
New_Order(NO_O_ID)	Contain (OID) in OOrder	Contain (OID) in OOrder
Order_Line(OL_O_ID)	Contain (OID) in OOrder	Full Access
History(H_C_ID, H_D_ID, H_W_ID)	=(CID,DID,WID)	Full Access
Item	Full Access	Full Access
Stock	No Access	Full Access

PostgreSQL Performance Results

No-Copy can sustain a much higher transaction throughput with much lower latency

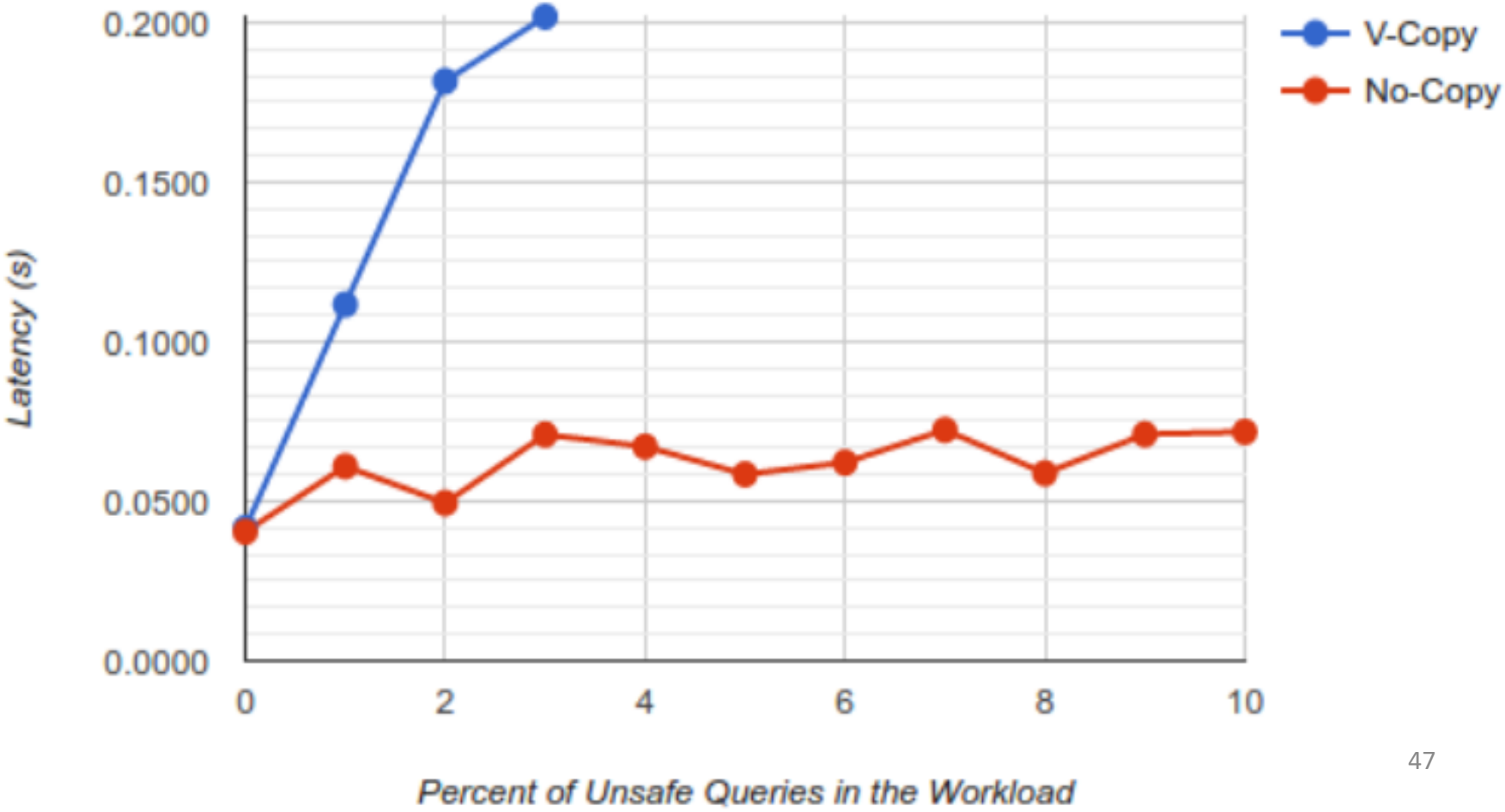


Unsafe Queries

- Previous performance numbers were measured when all the queries in the workload were safe
- In addition to the normal TPC-C queries, we added a mix of unsafe read and write queries to the workload

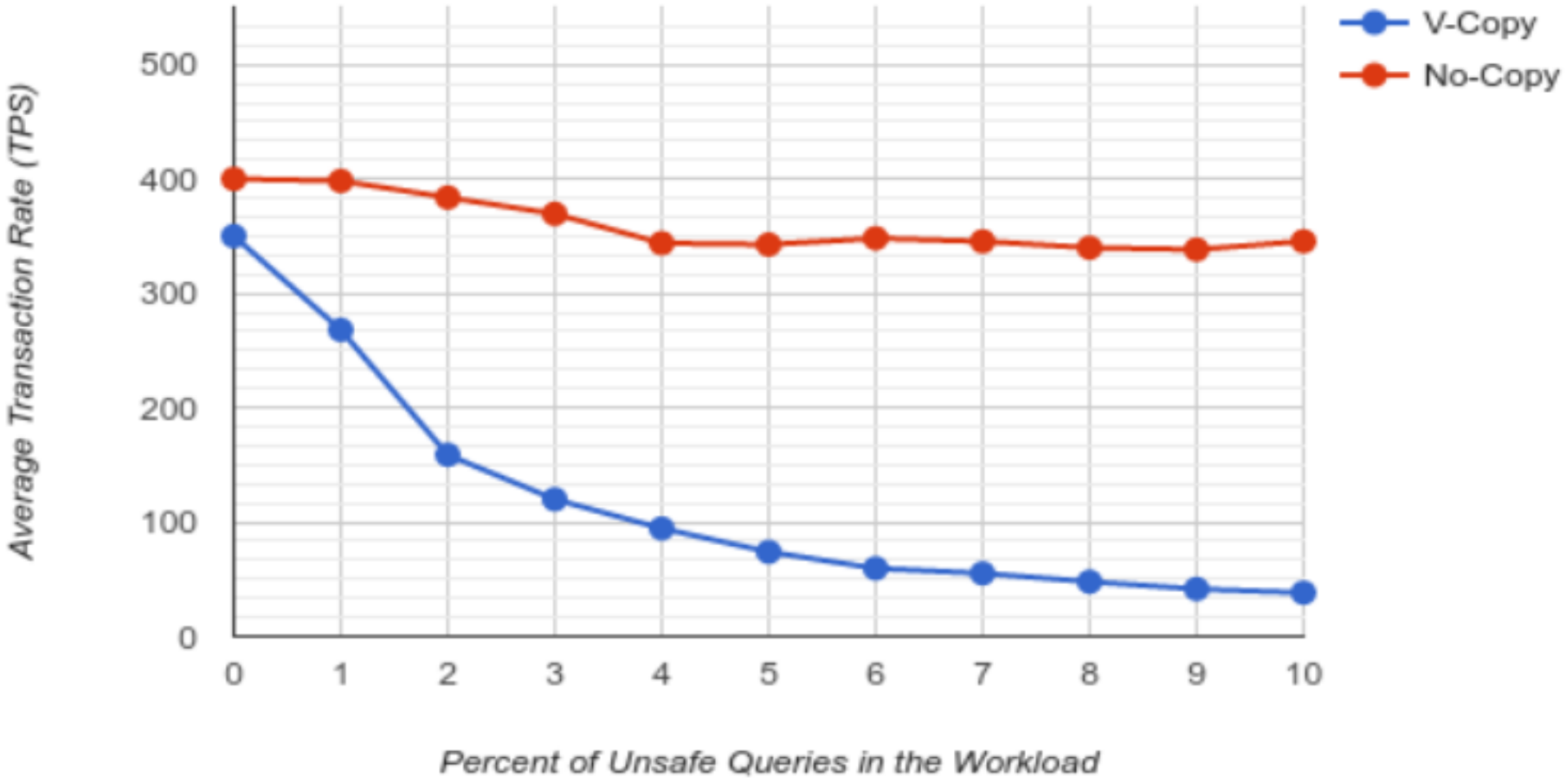
PostgreSQL Performance Results – Unsafe Queries with Policy 1

V-copy does not scale well as the number of unsafe queries increases



PostgreSQL Performance Results – Unsafe Queries with Policy 1

No-Copy sustains a higher average transaction rate



Syntax Knowledge Required

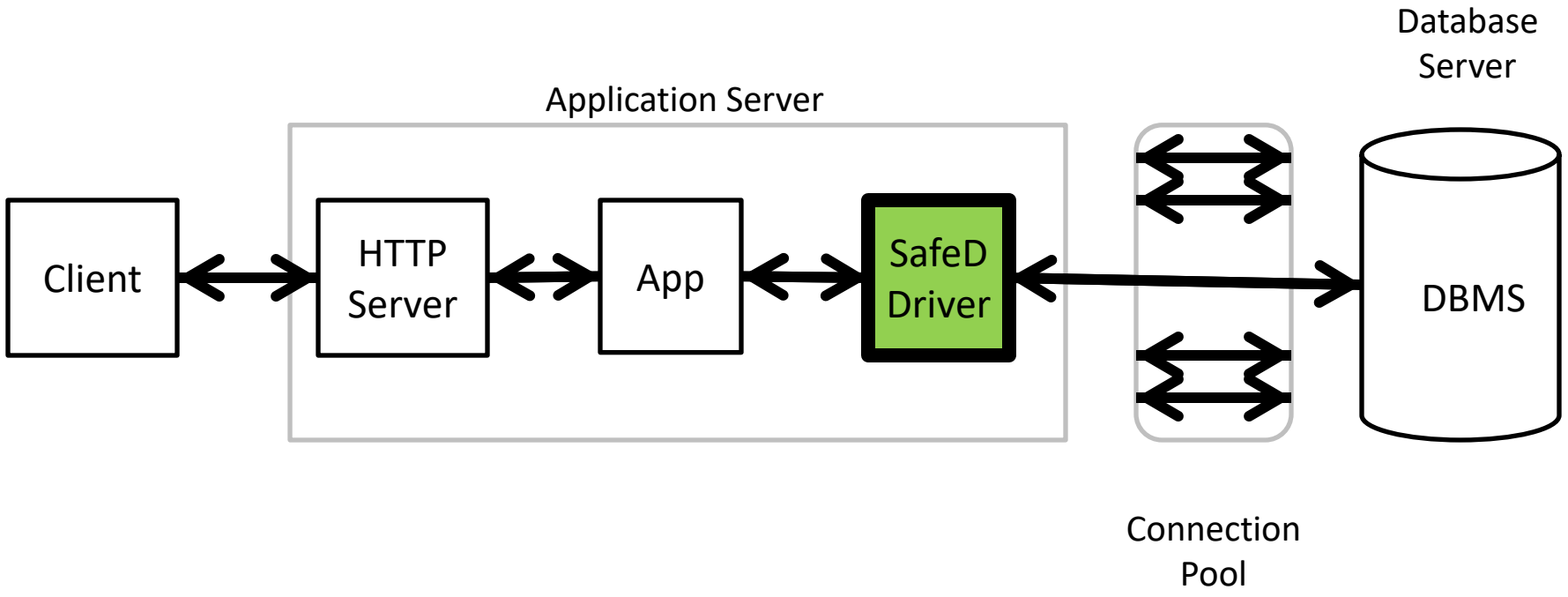
- SafeD
  - SELECT
- PostgreSQL
  - Policy and policy condition
  - Role
- Oracle
  - System Context
  - Login Trigger
  - Policy
  - Policy function

Developer Effort

- SafeD only requires basic SQL syntax to define
  - Policies defined as intuitive SELECT statements
- Postgres and Oracle both have database specific syntax for access control

Access Control Mechanism	LOC
SafeD	36
Postgres's Built-in Access Control	54
Oracle's Built-in Access Control (a.k.a. VPD)	544

Ensuring Authorized Updates in Multi-user Database-Backed Applications



Kevin Eykholt
keykholt@umich.edu