



Corso di Laurea in Computer Science

# Techniques to defeating SQL Injection Attacks

Mattia Prestifilippo Colombrino

# SQLRand

- ▶ SQL Injection is an attack in which the attacker injects malicious SQL commands into an input field

```
"SELECT * FROM utenti WHERE username = ' " + username + "'  
AND password = ' " + password + "'"
```



```
"SELECT * FROM utenti WHERE username = ' ' OR 1=1; -- AND  
password = 'anytext';
```

# SQLRand

- ▶ A tool reads an SQL query and adds a random number to all its SQL keywords.

```
"SELECT * FROM utenti WHERE username = '" + username + "'  
AND password = '" + password + "'"
```



```
"SELECT123 * FROM123 utenti WHERE123 username = '" +  
username + "' AND123 password = '" + password + "'"
```

# SQLRand

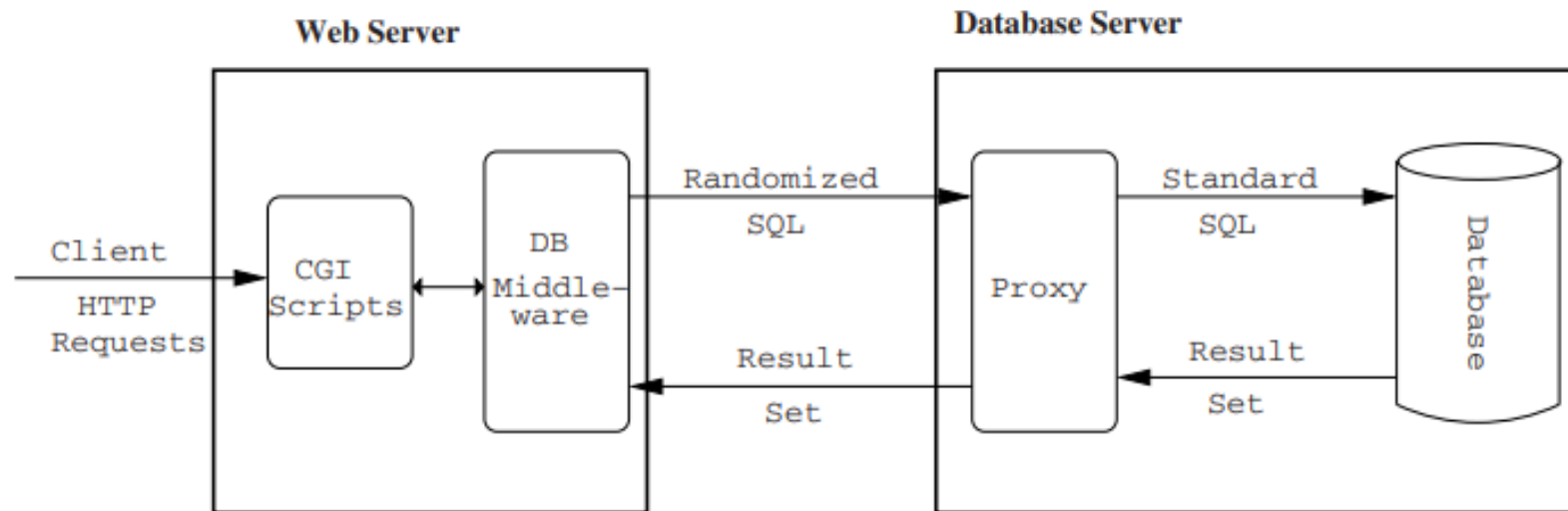
- ▶ An intermediary proxy intercepts the queries and verifies that each keyword has the random key
- ▶ If not, the code is considered syntactically incorrect
- ▶ The injected code will not contain the random key.

SELECT123 ✓ \* FROM123 ✓ utenti WHERE123 ✓ username =  
'user1' AND123 ✓ password = 'pass1'; → ✓

SELECT123 ✓ \* FROM123 ✓ utenti WHERE123 ✓ username = ''  
OR ✗ 1=1 -- AND123 password = 'anything'; → ✗

# SQLRand

- ▶ The web server redirects SQL packets addressed to the database to the proxy
- ▶ The proxy validates the query and translates it into standard syntax
- ▶ If valid, it forwards the query to the database for execution. If not, it sends an error packet to the web server



# SQLRand

- ▶ The key is randomized periodically in both the proxy and the web server.

- ▶ `SELECT Ab12Cd * FROM Ab12Cd utenti WHERE Ab12Cd username = 'user1' AND Ab12Cd password = 'pass1';`



`SELECT j3X5Fg * FROM j3X5Fg utenti WHERE j3X5Fg username = 'user1' AND j3X5Fg password = 'pass1';`

- ▶ The DBMS cannot expose the key through error messages.
- ▶ No proxy error message must leak

# SQL Injection Detection System

- ▶ For each SQL statement used, a rule is created that defines its syntactic structure
- ▶ The rules must cover all possible values of the expected queries

```
<Table reference> ::= "USERS";
```

```
<Table Column> ::= "user_id" | "user_level";
```

```
<Query specification> ::= "SELECT" <Select List> <From Clause> <Where Clause>;
```

```
<Select List> ::= <Table Column> ( "," <Table Column> )*;
```

```
<From Clause> ::= "FROM" <Table reference>;
```

```
<Where Clause> ::= "WHERE" <search condition> "AND" <search condition>;
```

```
<search condition> ::= <Table Column> "=" <Safe String>;
```

```
<Safe String> ::= "<AllowedChars>*" / ["OR", UNION, --, ''] ;
```

# SQL Injection Detection System

- ▶ Each query is split into token of characters and it is validated based on the associated syntactic rule
- ▶ An SQL statement is considered valid if it does not violate the syntactic rules defined in the specifications

`SELECT * FROM utenti WHERE username = ' ' × OR × 1=1 -- × AND  
password = 'anything';` → **×** La query contiene caratteri non permessi dalle regole



# SQL Injection Detection System

- ▶ The EMM is a proxy that intercepts queries and sends them to the VCM
- ▶ The VCM splits the query into tokens and validates it by comparing it with the corresponding syntactic rule
- ▶ If the query is valid, the EMM forwards it to the database; otherwise, it reports a potential injection

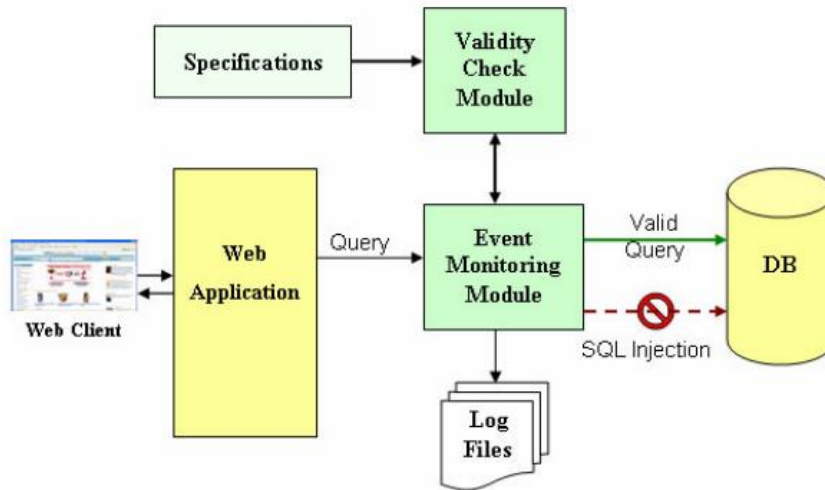



Figure 3: The architecture of SQL-IDS.

# Two Tier Defense Against SQL Injection

- ▶ First level: Client-side input validation → Checks data type, input length, and ensures it does not contain potentially malicious characters

Password: ' OR 1=1 ✗ Potentially harmful characters Eta: Ciao ✗ Incorrect data type

- ▶ Second level: Identity-based encryption  → When a user registers, a key is generated based on their credentials

Key = Hash('mario.rossi@example.com + SecurePassword123') 

- ▶ All data inserted into database queries by a given user will be encrypted with that key

Name: Mario Rossi → hV39Sdx5sW Address: ' OR 1=1 → s3F6d2Wq

# Two Tier Defense Against SQL Injection

- ▶ The **User Interface Module** captures and validates the input
- ▶ If the input does not meet the constraints, it is forwarded to the ASA Module ✖; Otherwise, it is sent to the IBC Module ✔
- ▶ The **Identity-Based Cryptography** Module generates an encryption key based on user credentials 🔑
- ▶ If the credentials match the data stored in the database, the connection to the DB is established ✔, otherwise, control is passed to the AA Module ✖.

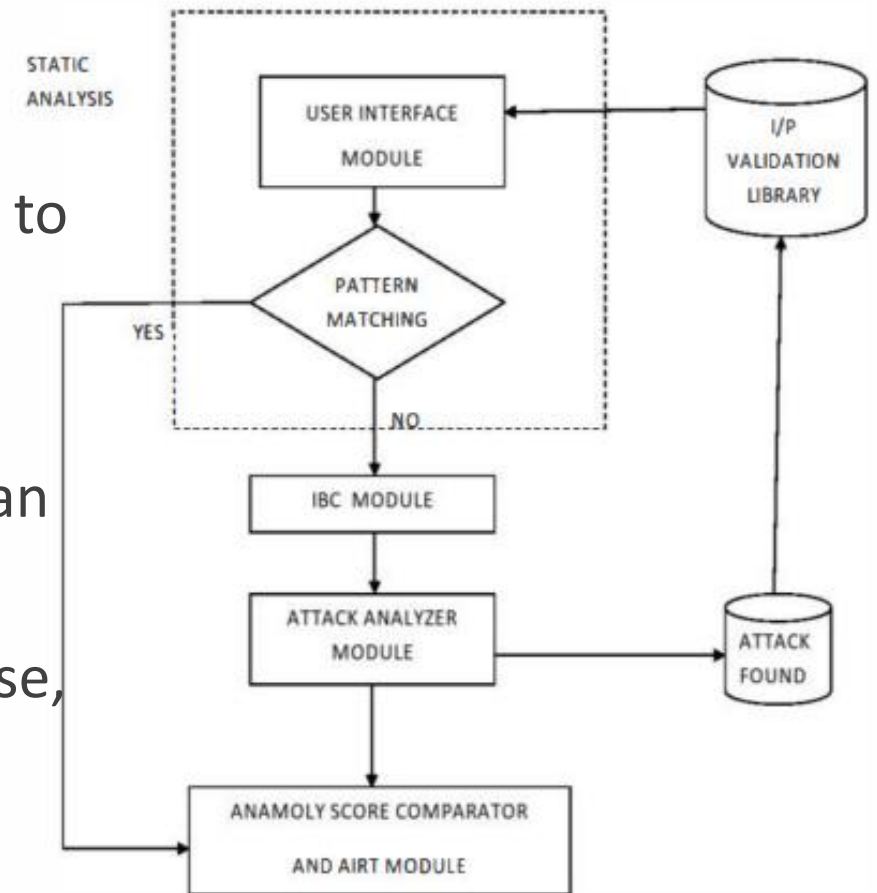


FIG. 2 PROPOSED ARCHITECTURE

# Two Tier Defense Against SQL Injection

- ▶ The **Analyzer Module** analyzes potentially malicious data and logs the attack, updating the validation library
- ▶ It then passes control to the ASCAA Module
- ▶ The **Anomaly Score Comparator and Airt Module** redirects the malicious user to the appropriate page
- ▶ It uses two tables that associate the user's IP address with an anomaly score and link it to a redirection page

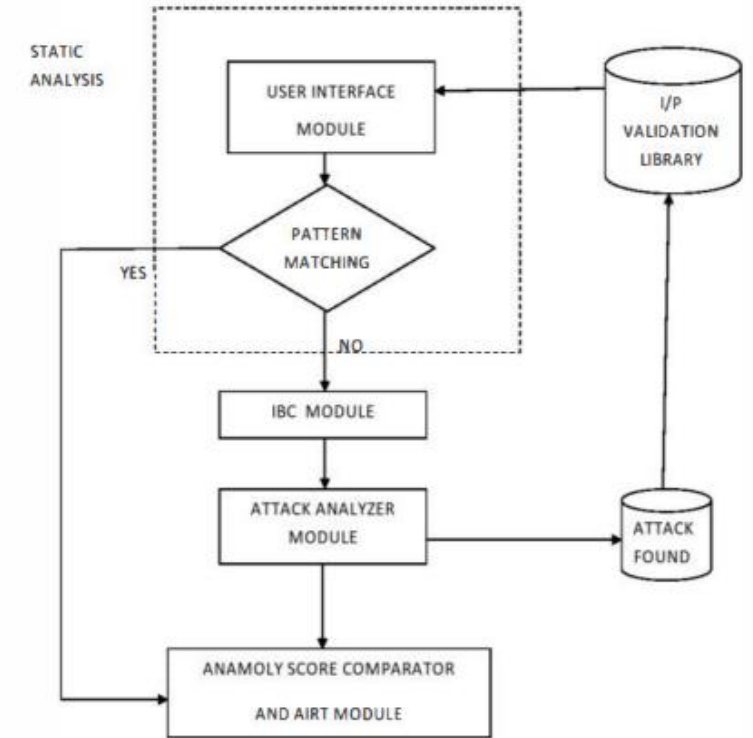


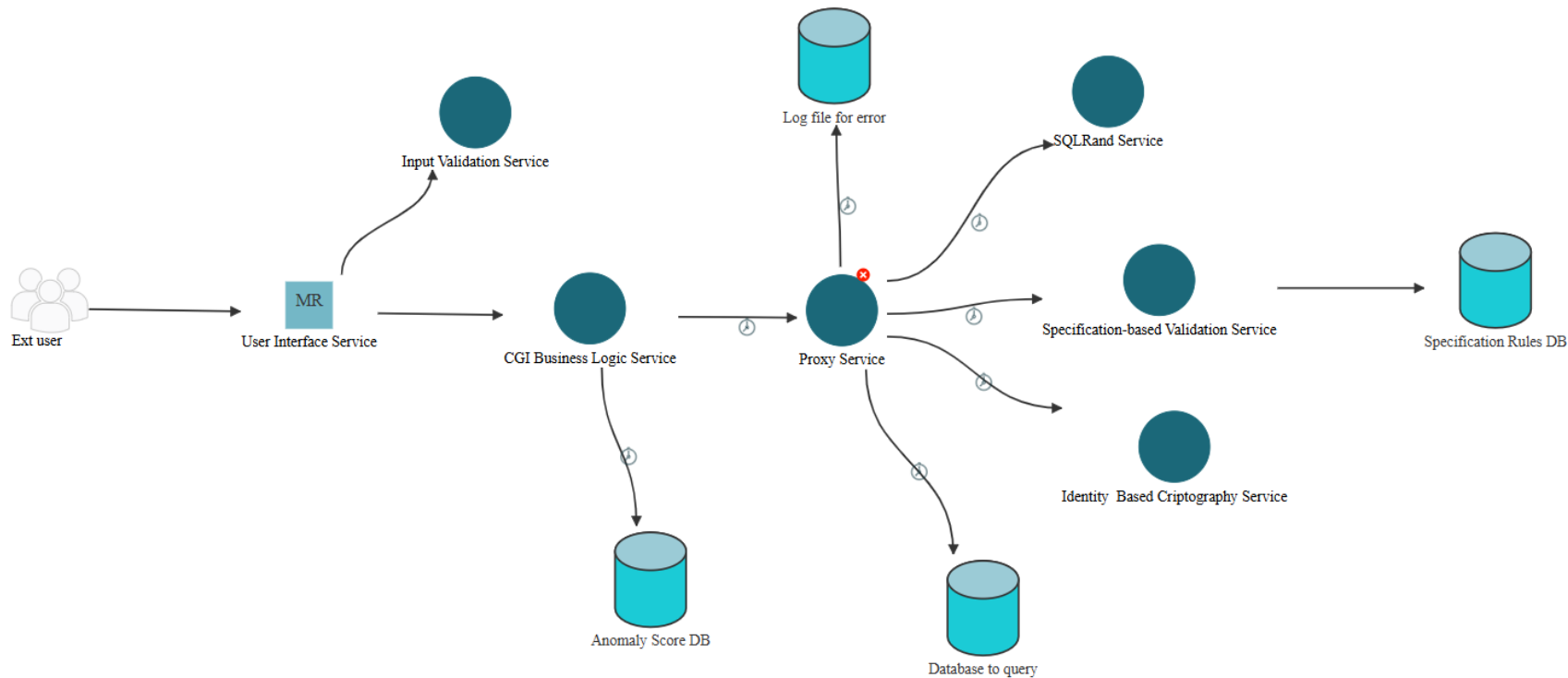
FIG. 2 PROPOSED ARCHITECTURE

TABLE 1 : REDIRECTION TABLE

RESPONSE REDIRECTION	ANOMALY SCORE
Alarm page	1-5
Email to admin	5-10
Block that IP address	More than 10

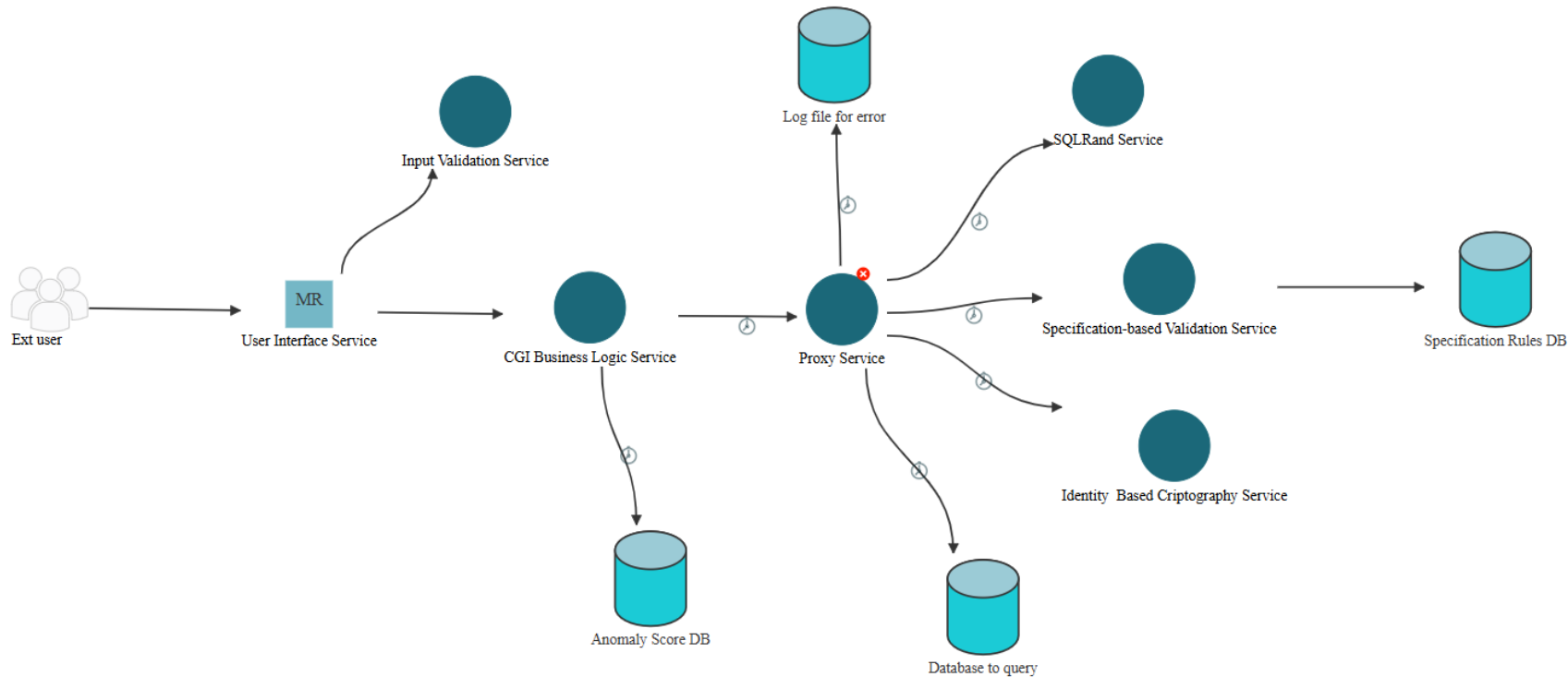
# A new approach

- It is possible to combine the three approaches to create a more robust technique
- A microservices-based architecture is used
- The **User Interface Service** receives input data from the client and sends it to the **Input Validation Service** for validation



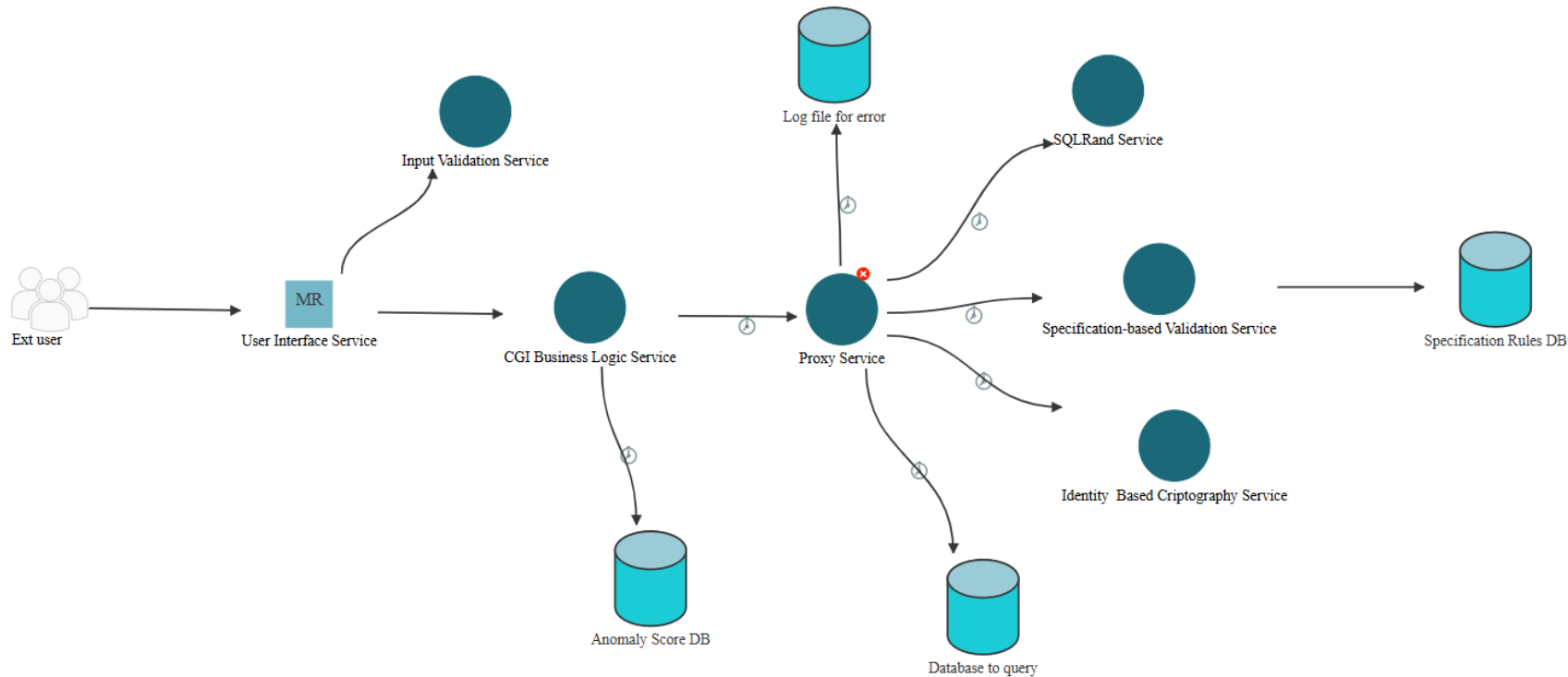
# A new approach

- The UI Service sends the input to the CGIBL Service for processing
- The **Business Logic** Service constructs a query by appending a randomized key as a suffix to SQL keywords and integrating the input
- It then sends the query to the Proxy Service



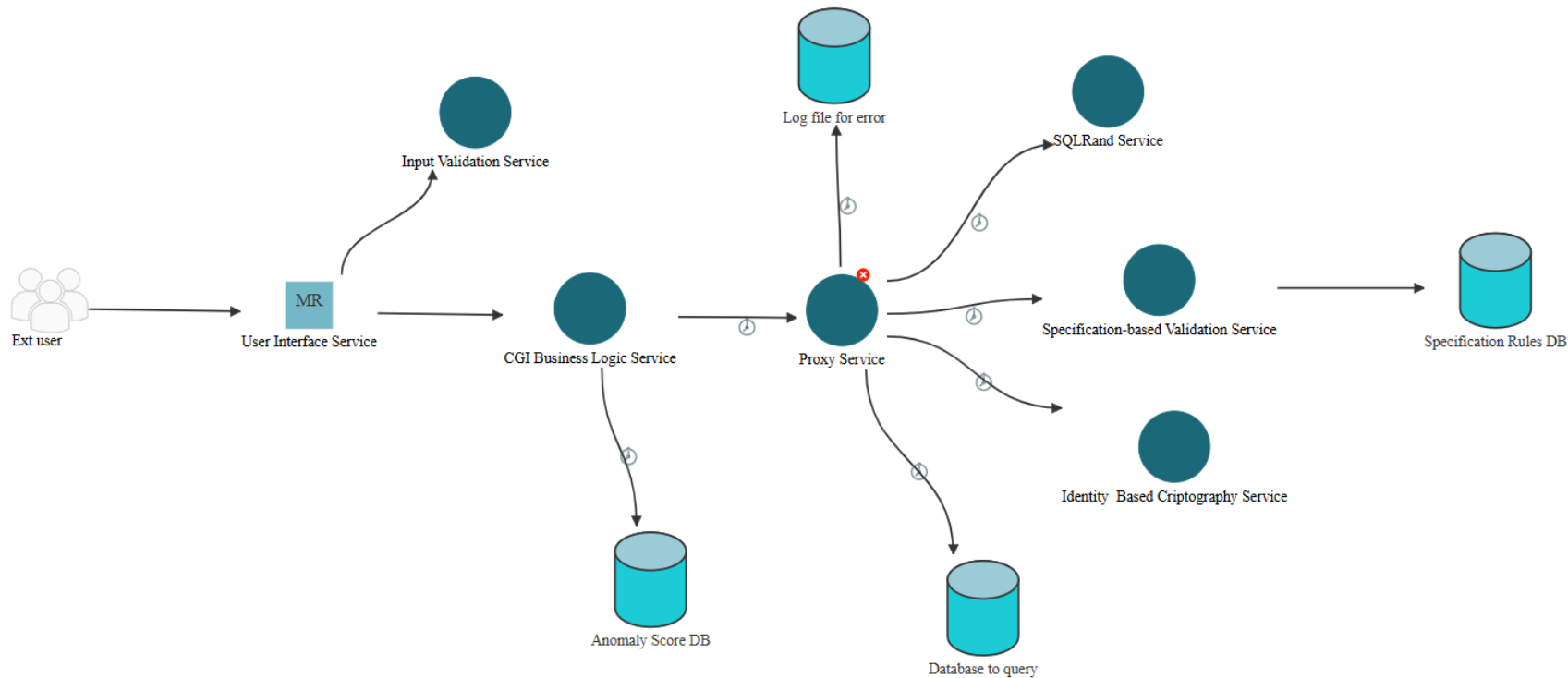
# A new approach

- The PS sends the query to the **SQLRand Service**, which checks that each SQL keyword correctly contains the random key in its suffix.
- If valid, it de-randomizes the query and returns it to the PS
- The PS then sends the de-randomized query to the **Specification-based Validation Service**, which ensures it complies with the associated syntax rule



# A new approach

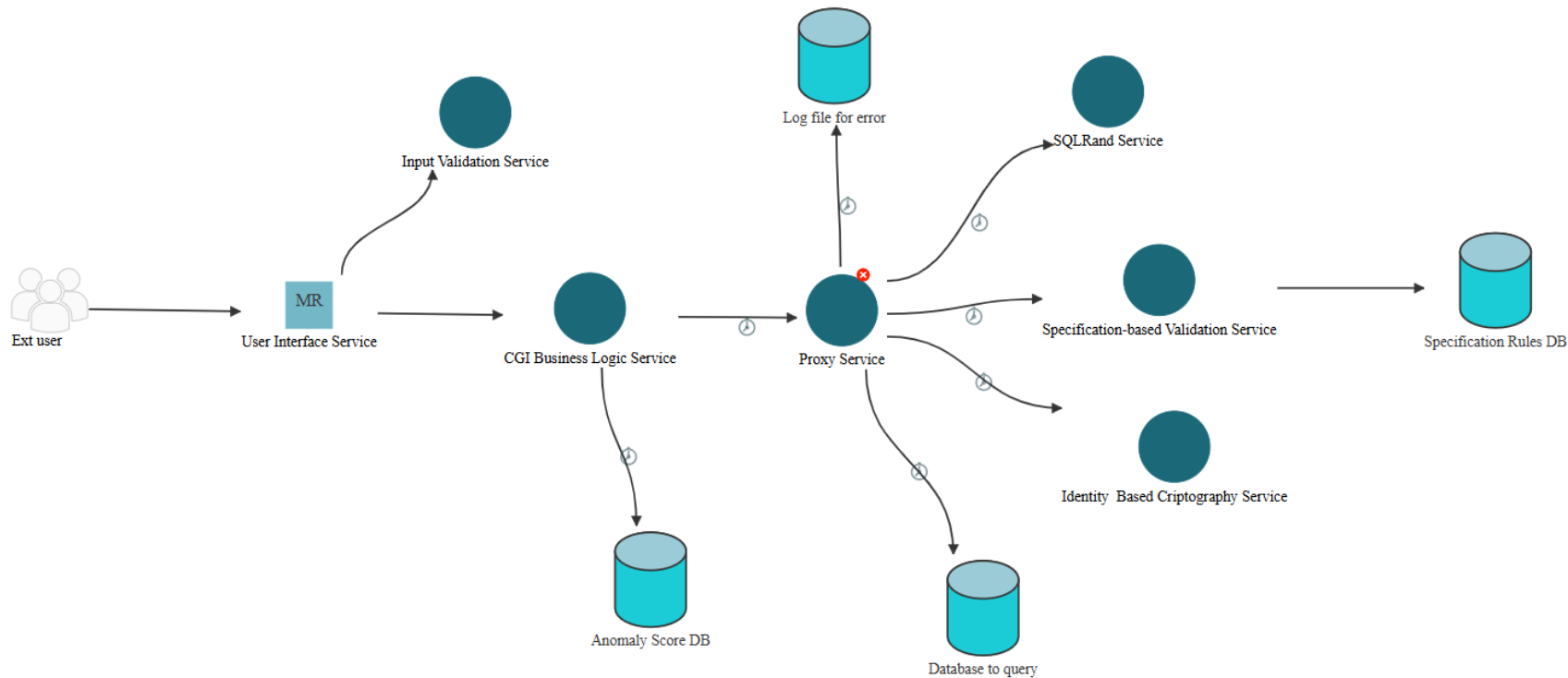
- The PS generates an identity-based key through the **Identity-Based Cryptography Service** and encrypts all input fields
- If malicious code bypasses the previous checks, it remains unreadable
- If all steps are successful, the PS executes the final query on the database





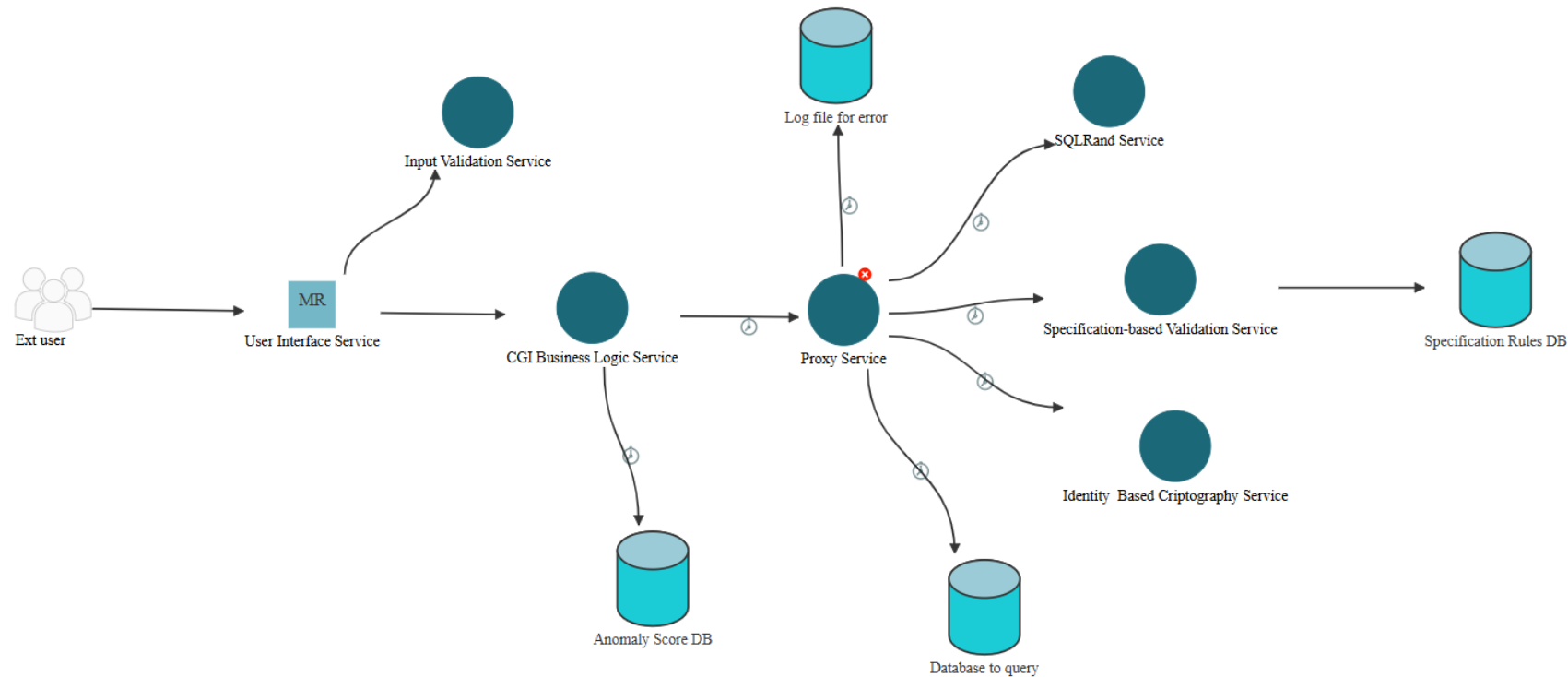
# A new approach: Handle error

- If an error occurs at any step, the PS logs the error details in a log file and returns an error message to the BL Service
- The BL Service increases the **Anomaly Score** associated with the suspicious user's IP address and, based on this score, redirects the user to the **appropriate page**



# A new approach: Problems

- Excessive overhead and redundancy in input field validation
- A first refactoring approach consists of **merging** the Input Validation Service and the Specification-Based Validation Service
- A second approach involves partially using the implemented techniques (e.g., SQLRand + Identity-Based Cryptography)



Users	Min	Max	Mean	Std
10	74	1300	183.5	126.9
25	73	2782	223.8	268.1
50	73	6533	316.6	548.8

## Benchmark

- **SQLRand** → 50 concurrent users × 5 queries: max 6 ms overhead per query
- **SQL-IDS** → 2030 clean + 420 with injection : max 20 ms overhead per query

Table 1: Performance results.

SQL Queries	min time overhead	max time overhead	average time overhead
2,450	~ 0 ms	20 ms	0.5 ms

Table 2: Effectiveness results.

Attacks Performed	Attacks Detected	False Negative Rate
420	420	0 %

- The Two-Tier Approach can only block 90% of attacks

Table 2: Chi Square Dataset

Web Application	No. of Inputs	No. of Attacks	No. of False Positives and Negatives	Total
Without Proposed Approach	50	100	40	190
With Proposed Approach	65	10	5	80
Total	115	110	45	270

## Bibliography

- Stephen W. Boyd and Angelos D. Keromytis, **SQLrand: Preventing SQL Injection Attacks**
- Konstantinos Kemalis and Theodoros Tzouramanis, **SQL-IDS: A Specification-based Approach for SQL-Injection Detection**
- Naresh Duhan, Bharti Saneja, **A Two Tier Defense Against SQL Injection**