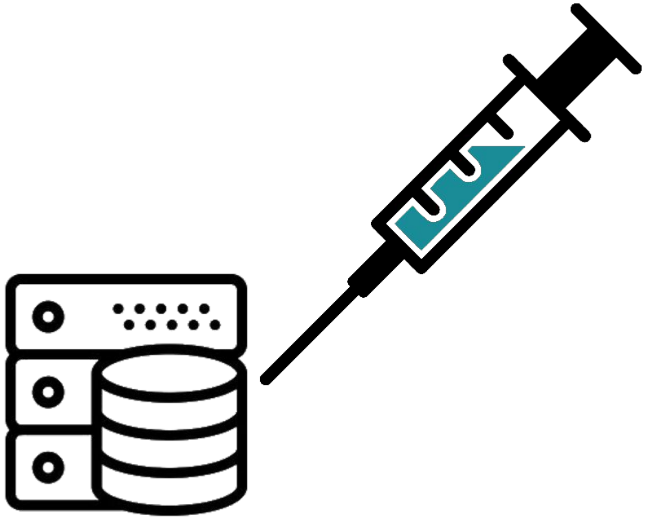


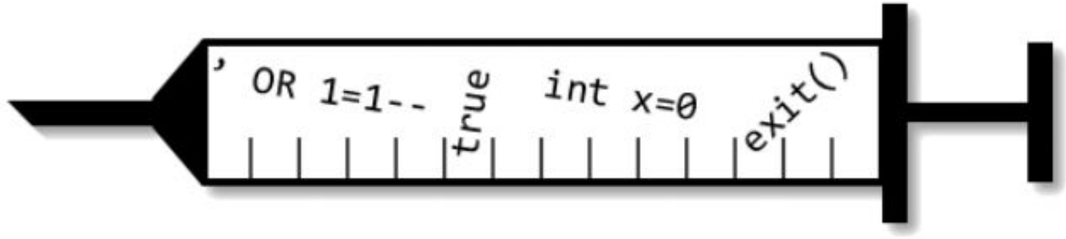
SQL Injection

Attacks & Mitigation

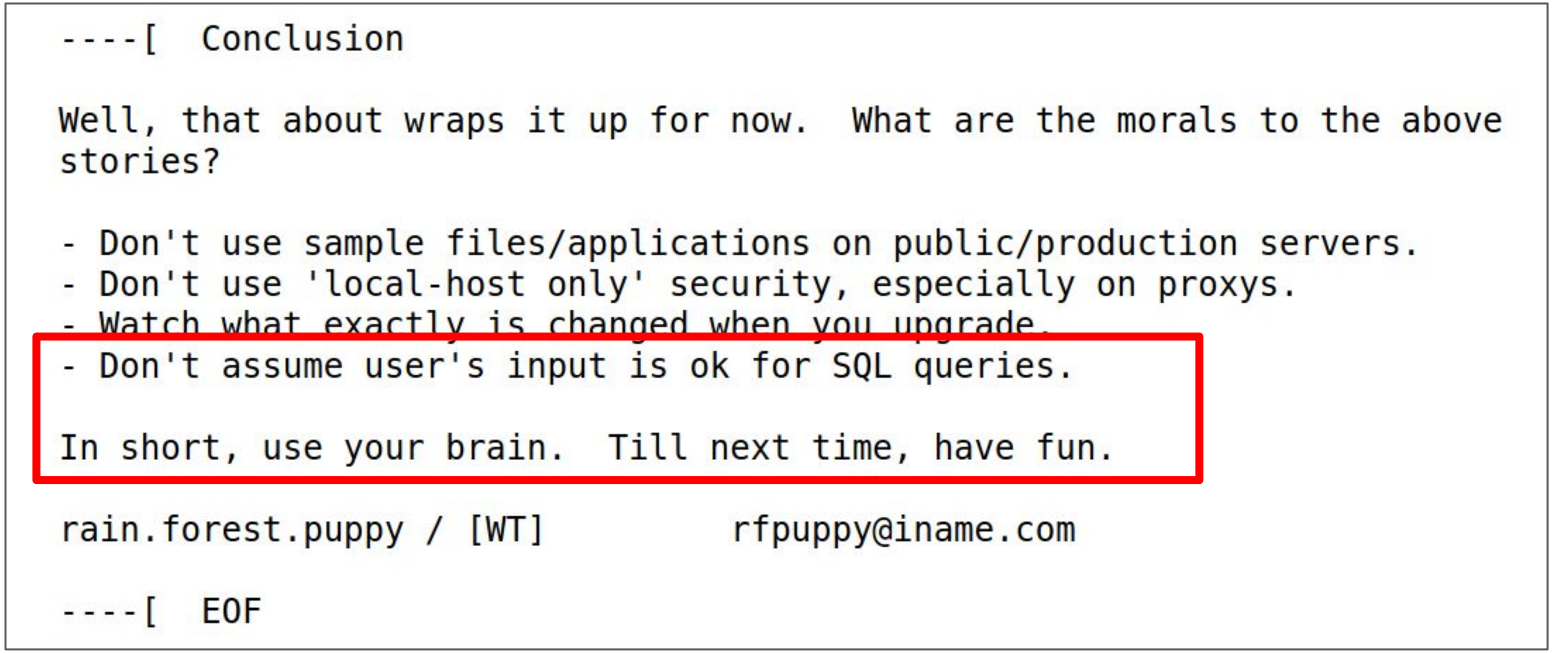
Dr Thomas Laurenson
thomas@thomaslaurenson.com

SQL Injection

- A type of injection attack
- A common attack against web applications
- Attacker forces application to accept malicious code
- Caused by bad programming



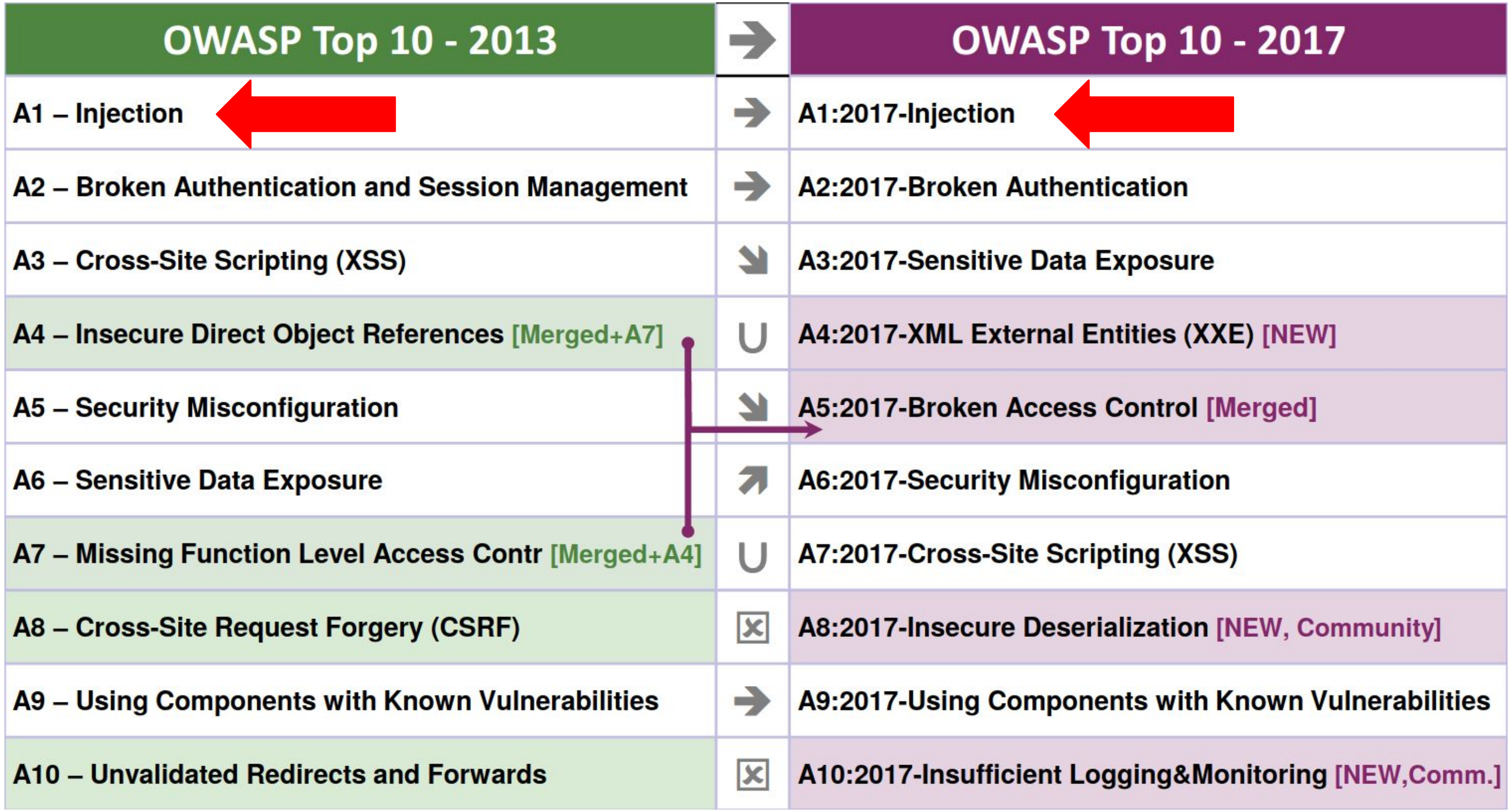
The Beginning of SQL Injection



```
----[  Conclusion

Well, that about wraps it up for now.  What are the morals to the above
stories?

- Don't use sample files/applications on public/production servers.
- Don't use 'local-host only' security, especially on proxys.
- Watch what exactly is changed when you upgrade.
- Don't assume user's input is ok for SQL queries.

In short, use your brain.  Till next time, have fun.

rain.forest.puppy / [WT]          rfpuppy@iname.com

----[  EOF
```

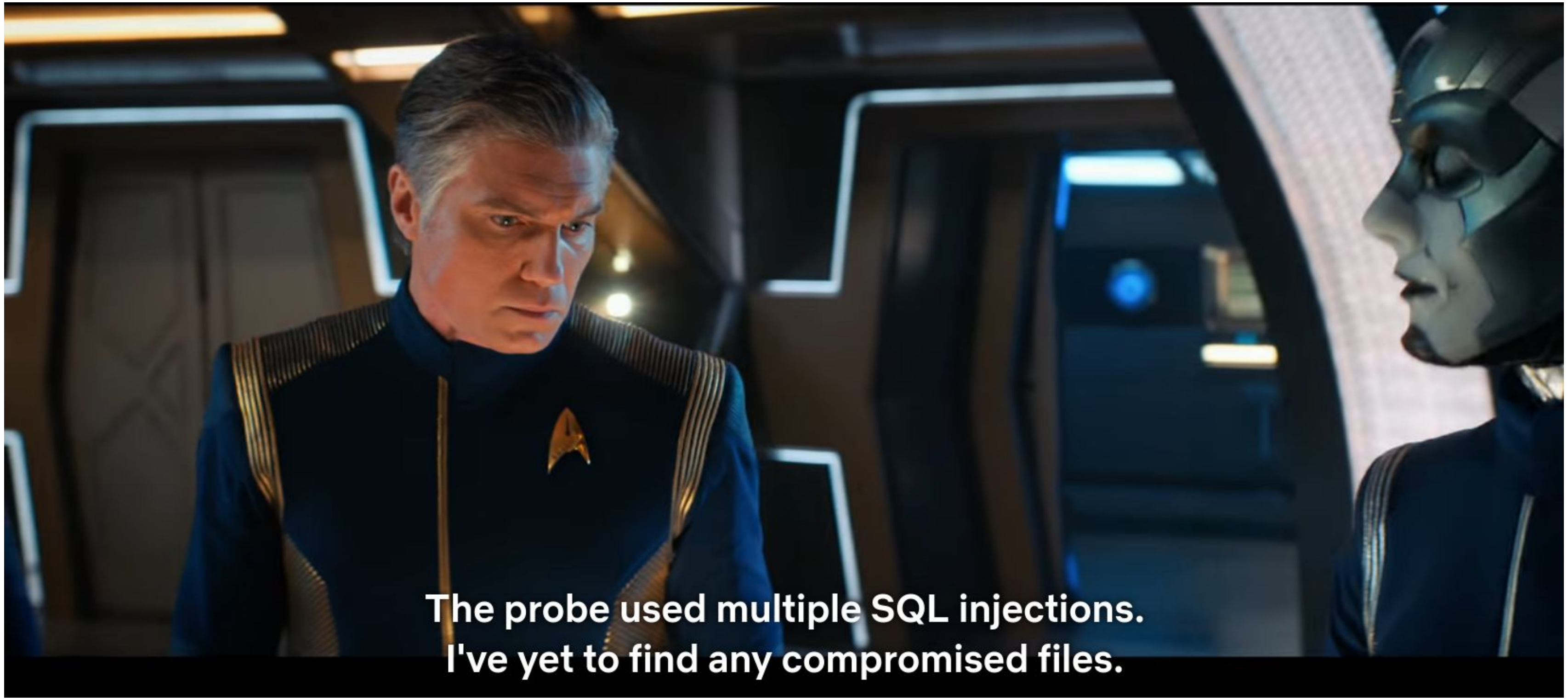
SQL Injection and OWASP

SQL Injection and OWASP Top 10

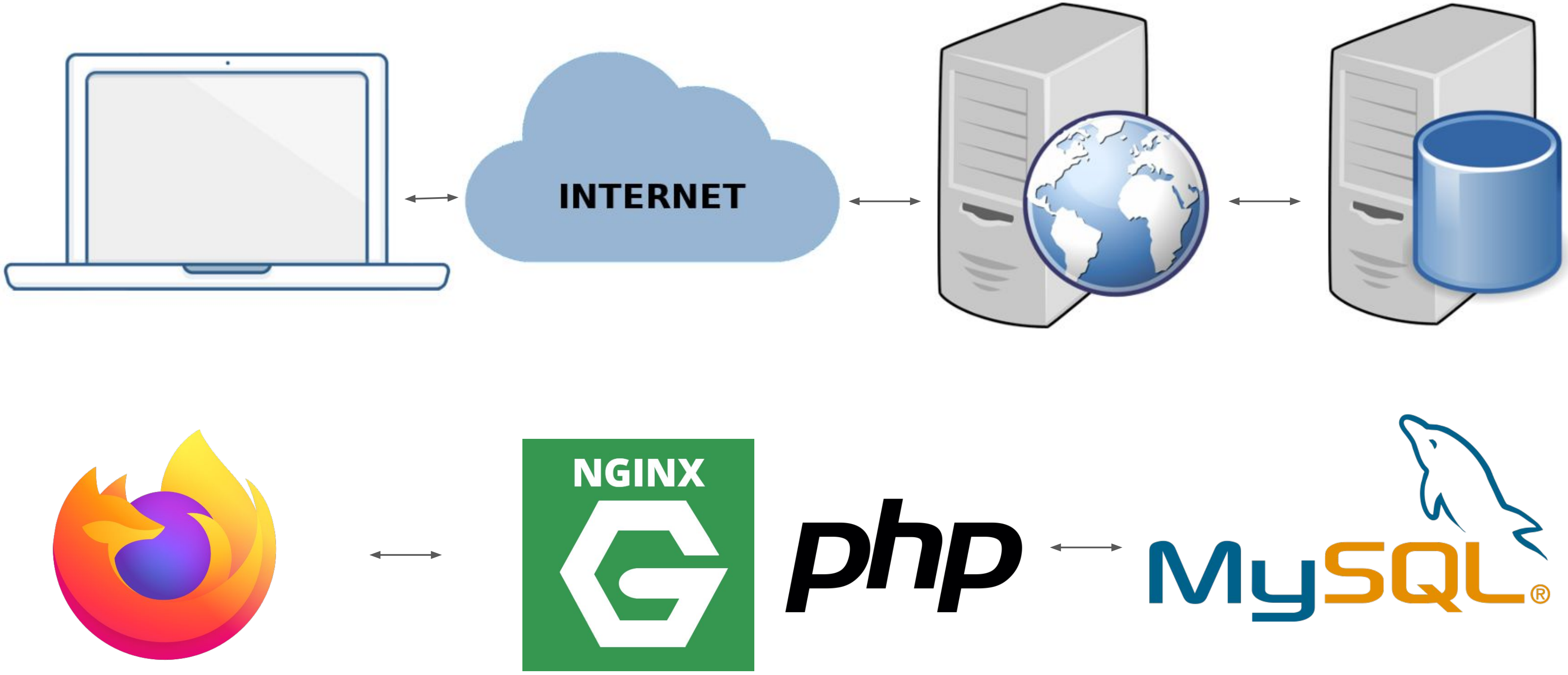


OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	∪	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	∪	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

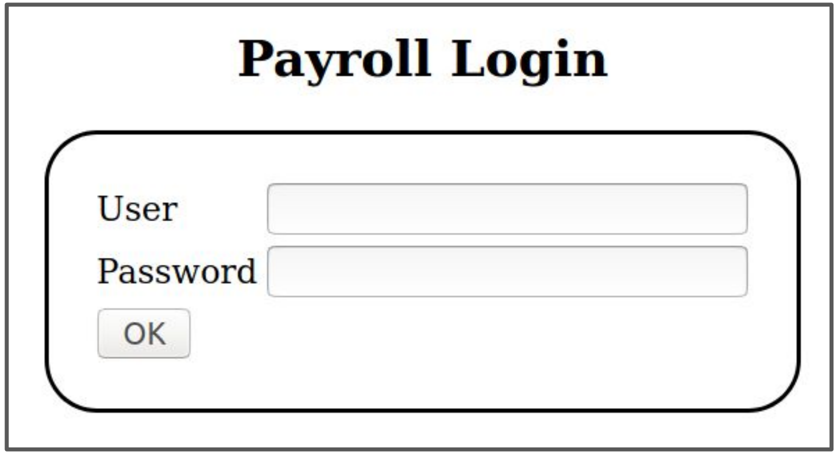
SQL Injection… still a problem in the 23rd century

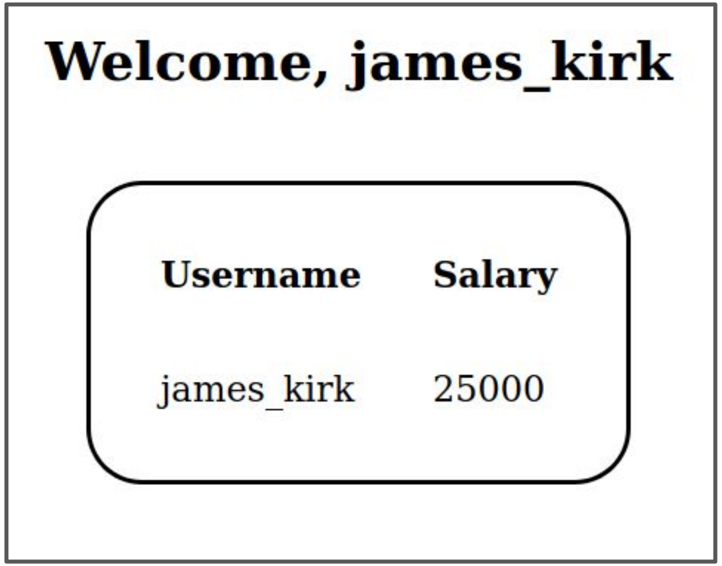

The probe used multiple SQL injections.
I've yet to find any compromised files.

SQL Injection and Web Applications



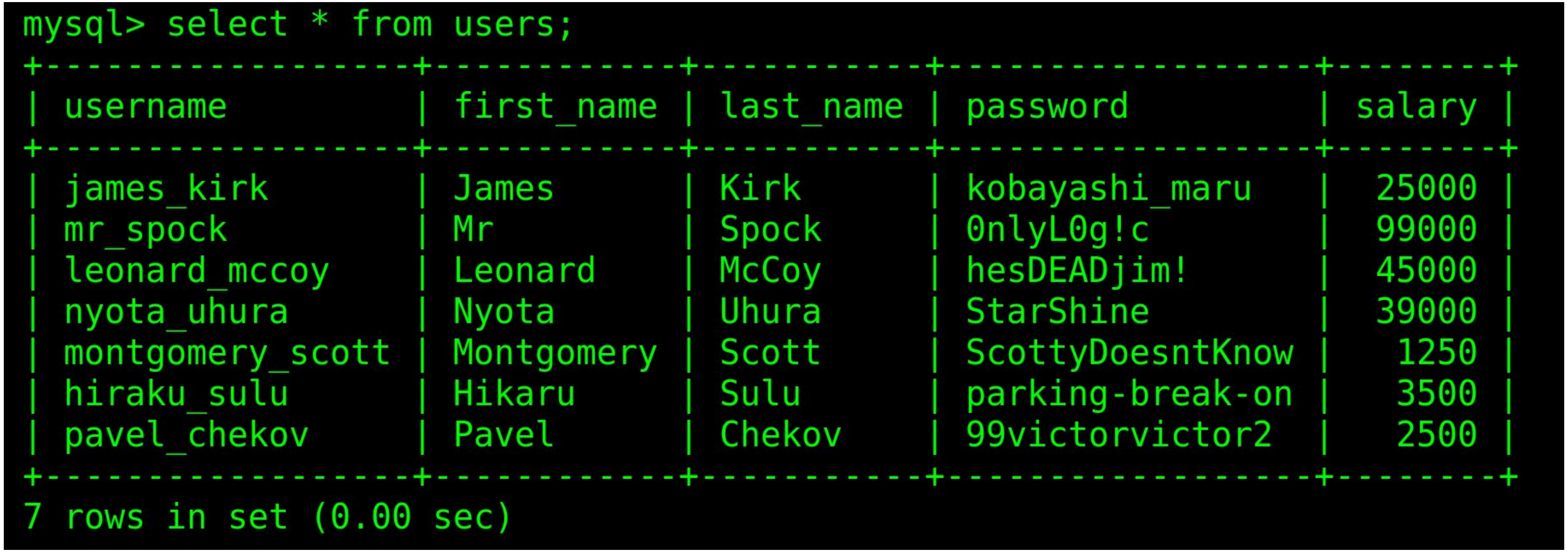
Web Application Example: Frontend




Welcome, james_kirk

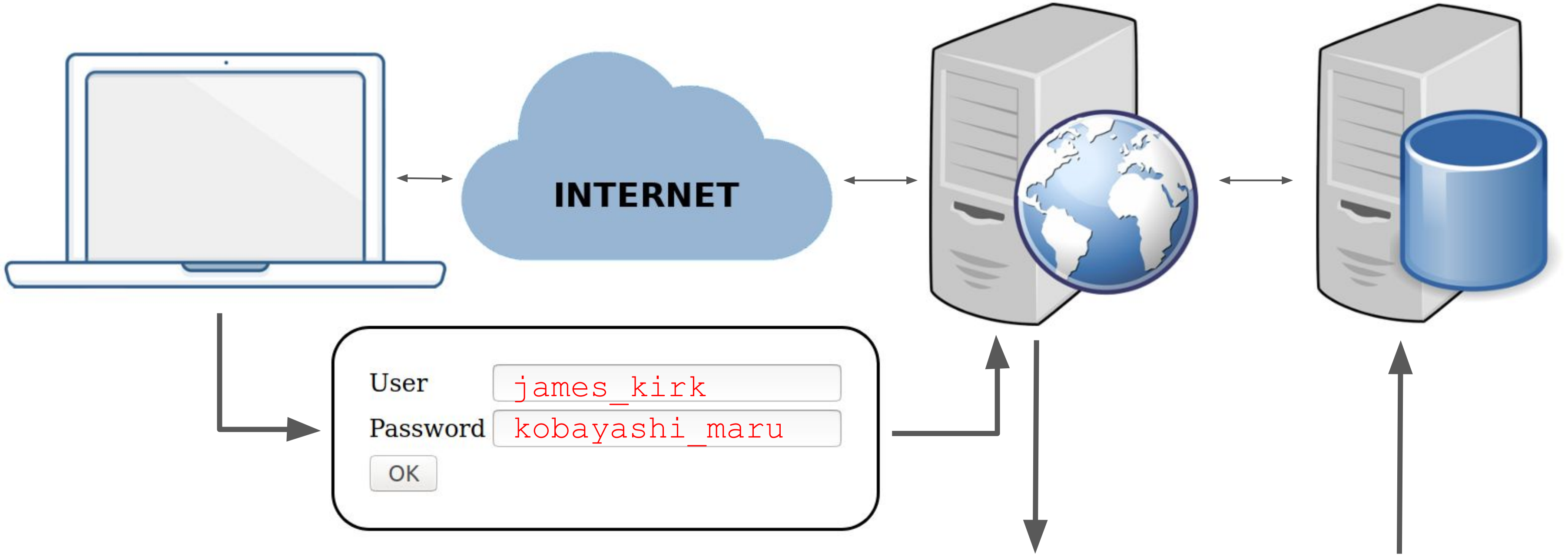
Username	Salary
james_kirk	25000

Web Application Example: Backend



```
mysql> select * from users;
+-------------------+------------+-----------+-------------------+--------+
| username          | first_name | last_name | password          | salary |
+-------------------+------------+-----------+-------------------+--------+
| james_kirk        | James      | Kirk      | kobayashi_maru    |  25000 |
| mr_spock          | Mr         | Spock     | 0nlyL0g!c         |  99000 |
| leonard_mccoy     | Leonard    | McCoy     | hesDEADjim!       |  45000 |
| nyota_uhura       | Nyota      | Uhura     | StarShine         |  39000 |
| montgomery_scott  | Montgomery | Scott     | ScottyDoesntKnow  |   1250 |
| hiraku_sulu       | Hikaru     | Sulu      | parking-break-on  |   3500 |
| pavel_chekov      | Pavel      | Chekov    | 99victorvictor2   |   2500 |
+-------------------+------------+-----------+-------------------+--------+
7 rows in set (0.00 sec)
```

Normal Web Application Query



```
SELECT username, salary FROM users
WHERE username = 'james_kirk' AND password = 'kobayashi_maru'
```

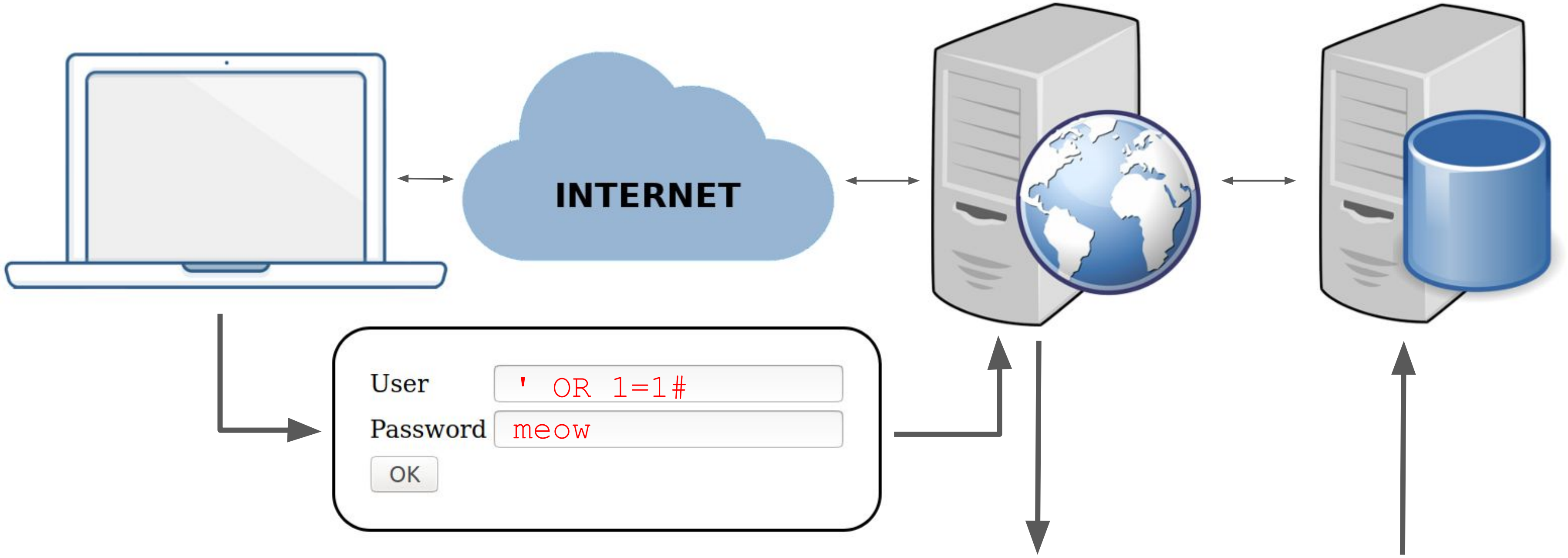
SQL Injection Vulnerability

```
# Get username/password from user
$user = $_POST['user'];
$pass = $_POST['password'];
```

```
# Construct SQL query
$sql = "SELECT username, salary FROM users
        WHERE username = '$user' AND password = '$pass'";
```

```
# Execute SQL query
$conn->multi_query($sql)
```

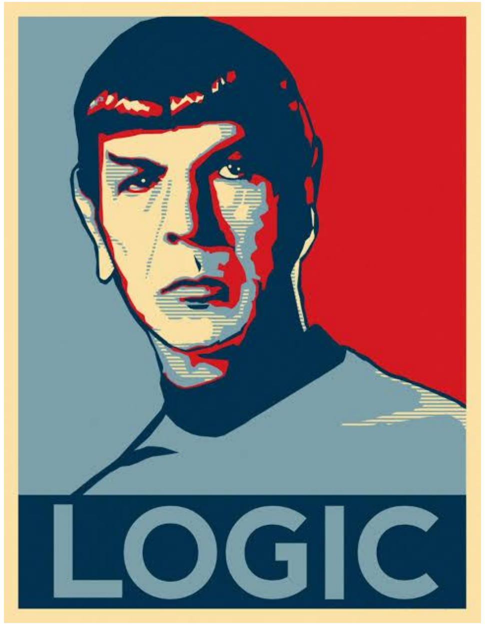
SQL Injection Attack



```
SELECT username, salary FROM users
WHERE username = '' OR 1=1#' AND password = 'meow'
```

SQL Injection Attack Logic

```
SELECT username, salary FROM users
WHERE username = '' OR 1=1#' AND password = 'meow'
```



`username = ''`	False
`1=1`	True

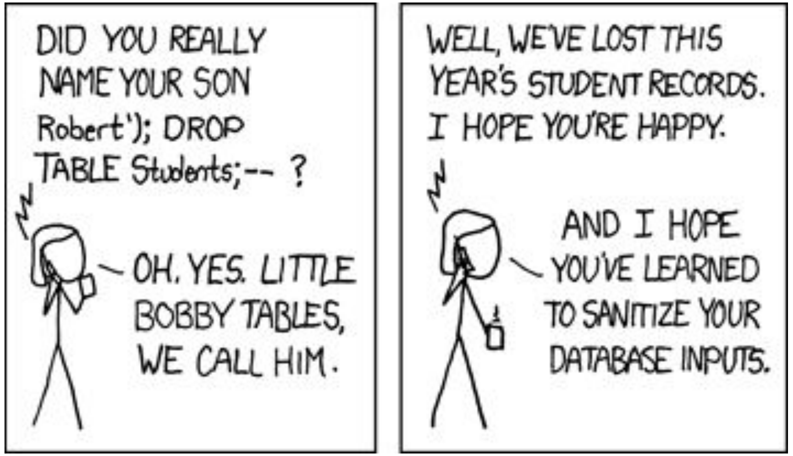
`username = '' OR 1=1`	True

`#`	PHP comment
`' AND password = 'meow'`	Not run

Demo: Attacking

SQL Injection Mitigation

- OWASP provide an SQL Injection Prevention Cheat Sheet
- Primary defences:
  - Use of Prepared Statements
  - Use of Stored Procedures
  - Whitelist Input Validation
  - Escaping All User Supplied Input


DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

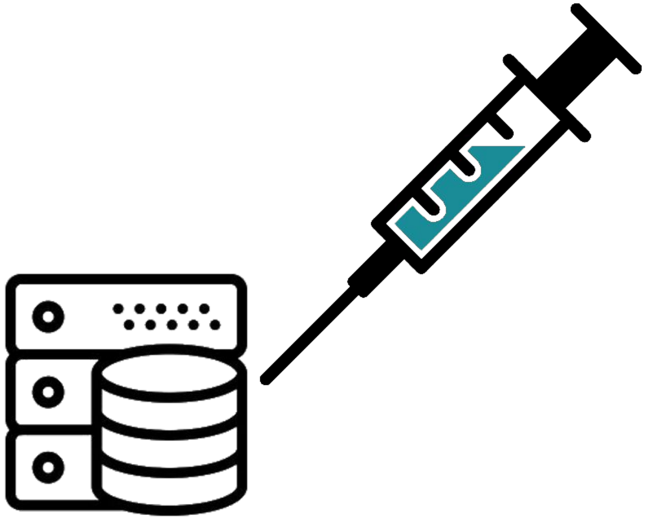
- Basic Premise: **SANITIZE USER INPUT!**

Demo: Mitigating

SQL Injection Mitigation: Example

```
# Construct and execute SQL query
$stmt = $pdo->prepare("SELECT username, salary FROM users
                   WHERE username = ? and password = ?");
$stmt->execute([$user, $pass]);
```

```
SELECT username, salary FROM users
WHERE username = '' OR 1=1#' AND password = 'meow'
```



Thanks!

Questions?

https://github.com/thomaslaurenson/startrek_payroll