

A Closer Look at SQL Injection Attack

+ What is a SQL Injection (SQLi) Attack?

- Many web applications take user input from a form
- Often this user input is used literally in the construction of a SQL query submitted to a database. For example:

```
SELECT productdata FROM table WHERE productname = 'user  
input product name' ;
```

- A SQL injection attack involves placing SQL statements in the user input

<https://www.helpnetsecurity.com/2021/01/11/sql-injection-bug/>

Test Your **CISSP KNOWLEDGE**
with **FREE** Interactive Flash Cards (ISC)[®]

ACCESS NOW ►

Featured news

Cyber criminals are targeting digital artists

VPN attacks up nearly 2000% as companies embrace a hybrid workplace

Cloud computing costs skyrocketing as businesses support a remote workforce

Top threats to consumer cyber safety

Vaccine passports challenged by data privacy and security implications



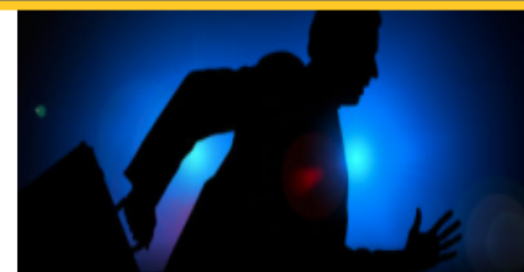
Matias Madou, CTO, Secure Code Warrior
January 11, 2021

Share



SQL injection: The bug that seemingly can't be squashed

If you're in a hands-on cybersecurity role that requires some familiarity with code, chances are good that you've had to think about SQL injection over and over (and over) again.



To identify cybersecurity vendor sustainability, start with the fundamentals

Why XSS is still an XXL issue in 2021

Are your cyber defenses stuck in the sandbox?

How do I select an ITSM solution for my business?



CCSP
Certified Cloud Security Professional
An (ISC)[®] Certification

A Guide to Becoming
CCSP CERTIFIED
Elevate Your Cloud

Structured Query Language (SQL)

- Widely used **database query language**

- Retrieve a set of records, e.g.,

SELECT * FROM Person WHERE Username='Lee'

- Add data to the table, e.g.,

**INSERT INTO Key (Username, Key) VALUES ('Lee',
Ifoutw2)**

- Modify data, e.g.,

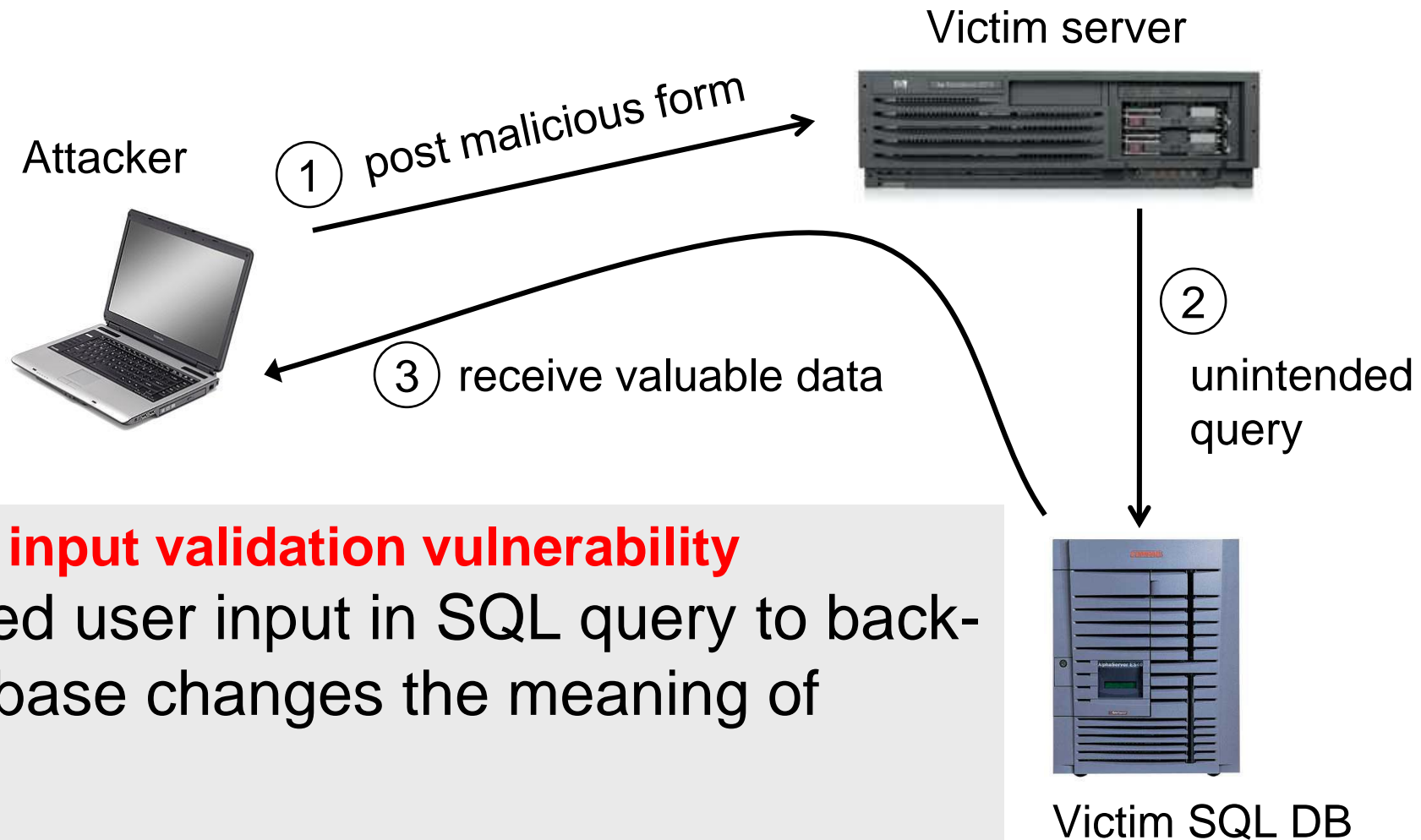
UPDATE Keys SET Key=ifoutw2 WHERE PersonID=8

Sample PHP Code

```
$selecteduser = $_GET['user'];  
$sql = "SELECT Username, Key FROM Key".  
      "WHERE Username='$selecteduser';  
$rs = $db->executeQuery($sql);
```

- What if **'user' is a malicious string** that changes the meaning of the query?

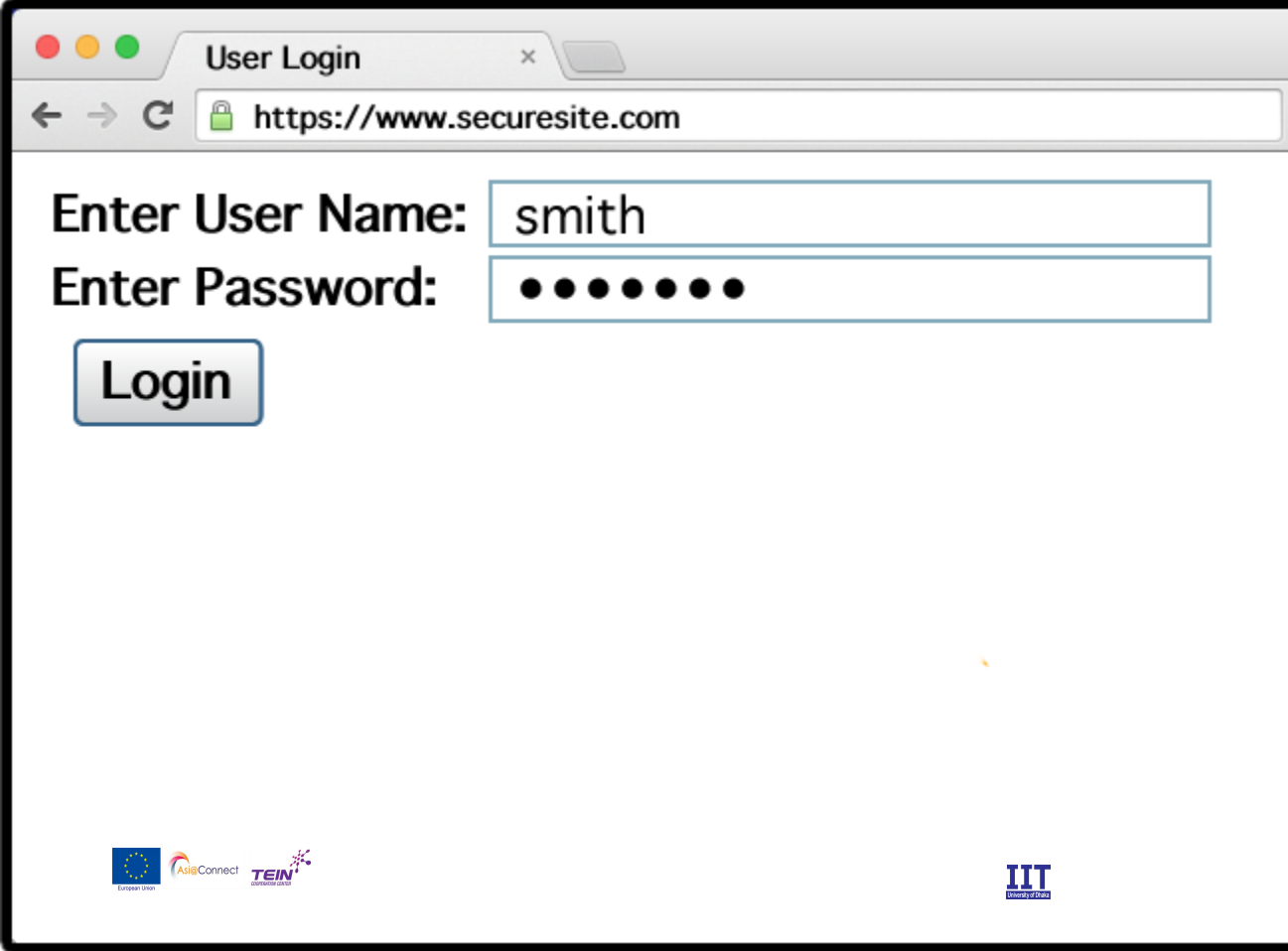
SQL Injection: Basic Idea



◆ This is an **input validation vulnerability**

Unsanitized user input in SQL query to back-end database changes the meaning of query

Example Login Prompt



A screenshot of a web browser window titled "User Login". The address bar shows "https://www.securesite.com". The login form contains two input fields: "Enter User Name:" with the text "smith" and "Enter Password:" with seven dots. A "Login" button is positioned below the password field. At the bottom of the page, there are logos for the European Union, AsiaConnect, TEIN, and IIT Bombay.

User Login

← → ↻  https://www.securesite.com

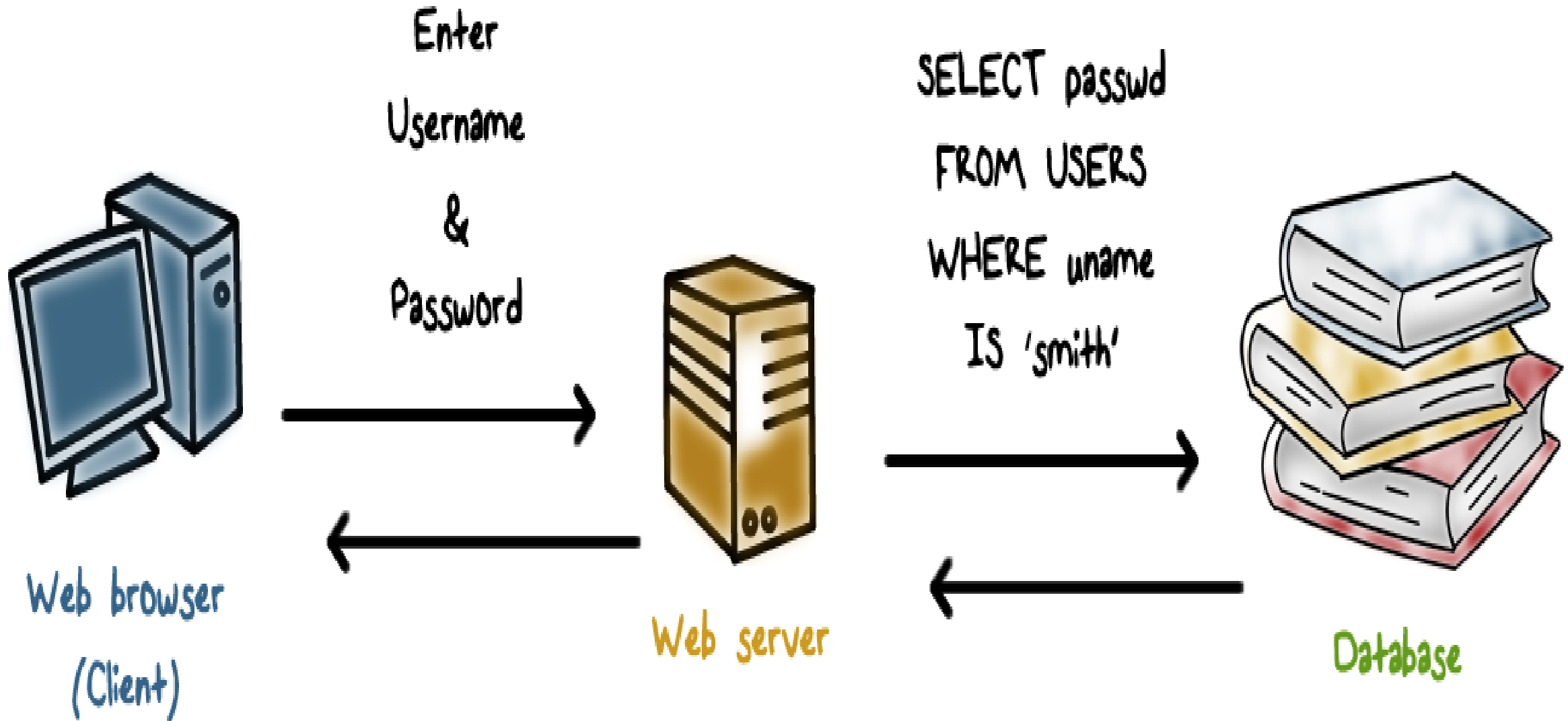
Enter User Name:

Enter Password:

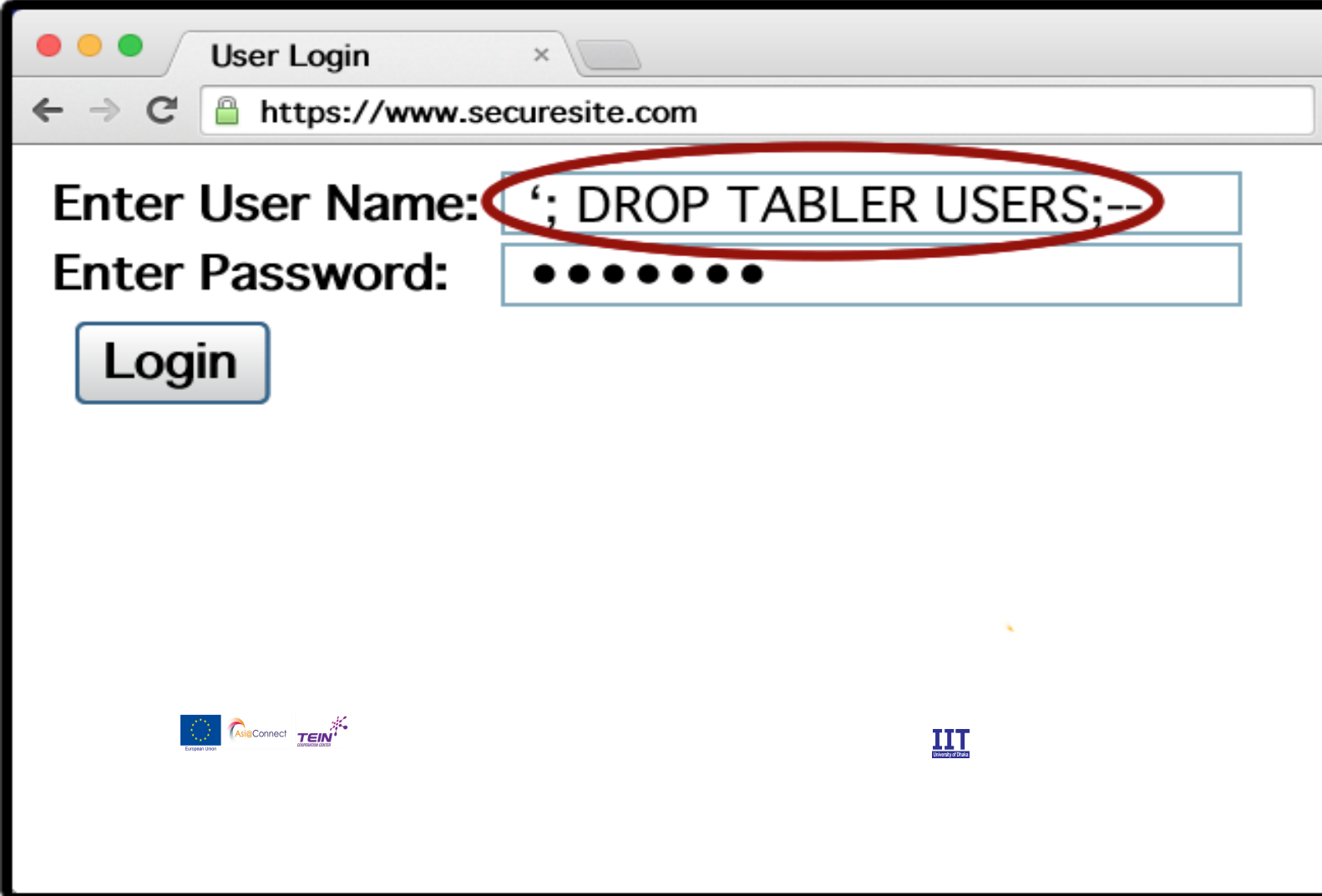
Login

Normal Login



Malicious User Input



The image shows a web browser window titled "User Login" with the address bar displaying "https://www.securesite.com". The login form contains two input fields: "Enter User Name:" and "Enter Password:". The "Enter User Name:" field contains the text "; DROP TABLER USERS;--", which is circled in red. The "Enter Password:" field contains seven black dots. Below the input fields is a "Login" button. At the bottom of the page, there are logos for the European Union, AsiaConnect, TEIN, and IIT Bombay.

User Login

https://www.securesite.com

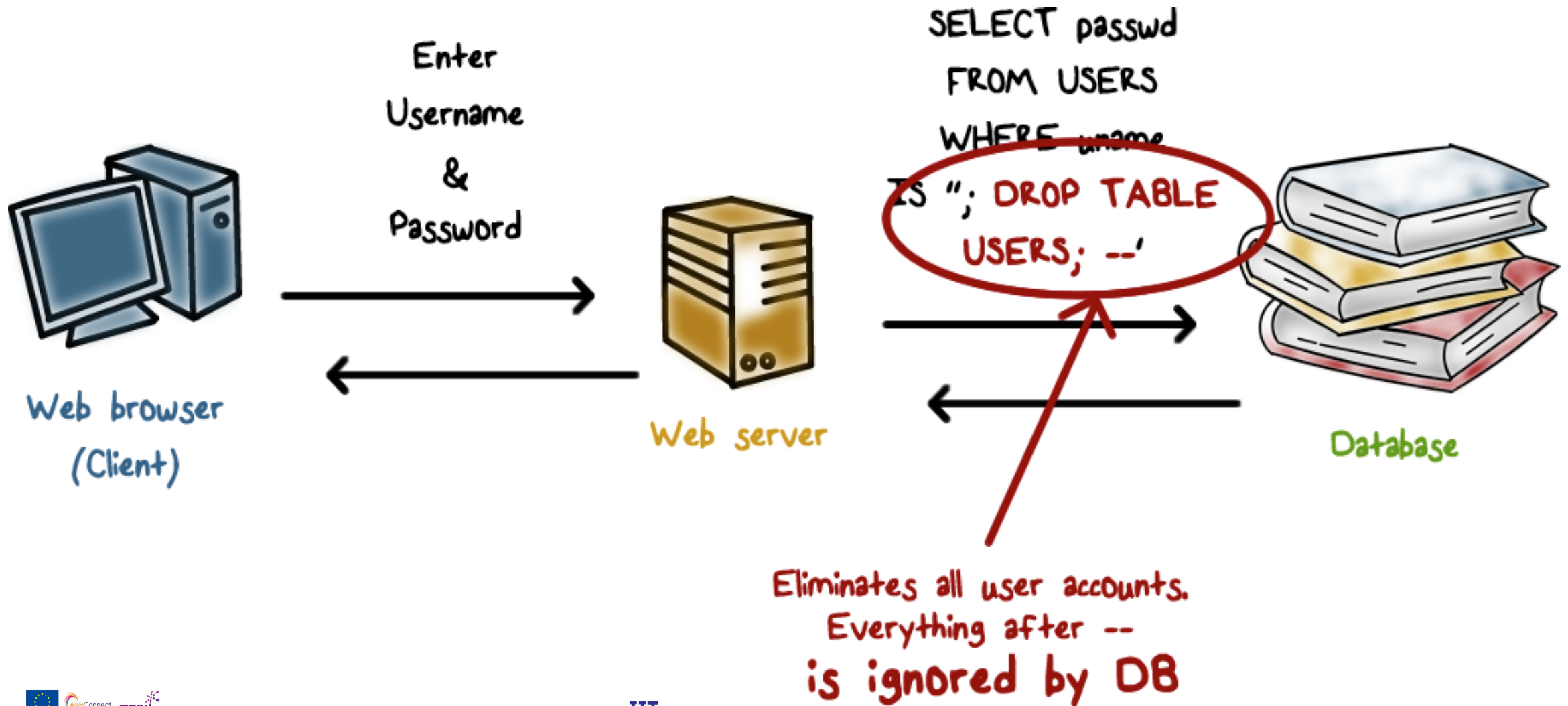
Enter User Name: ; DROP TABLER USERS;--

Enter Password: ●●●●●●●

Login

European Union AsiaConnect TEIN IIT Bombay

Example SQL Injection Attack





SQL Injection Attack #1

Unauthorized Access Attempt:

`password = ' or 1=1 --`

SQL statement becomes:

**`select count(*) from users where username = 'user' and
password = '' or 1=1 --`**

Checks if password is empty OR 1=1, which is always true, permitting access.



SQL Injection Attack #2

Database Modification Attack:

```
password = 'foo'; delete from table users  
where username like '%'
```

DB executes *two* SQL statements:

```
select count(*) from users where username = 'user' and password = 'foo' ;  
delete from table users where username like '%'
```



Another SQL Injection Example

[From Kevin Mitnick's "The Art of Intrusion"]

- User enters: ' OR WHERE pwd LIKE '%' as both name and passwd
- Server executes:

SELECT * WHERE user=' OR ~~WHERE~~ pwd LIKE '%' AND pwd=' OR
~~WHERE~~ pwd LIKE '%' ————— Wildcard matches any password

- Logs in with the credentials of the first person in the database
(typically, administrator!)



It Gets Better

- User gives username

' exec cmdshell 'net user badguy badpwd' / ADD --

- Web server executes query

set UserFound=execute(

SELECT * FROM UserTable WHERE

username= ' exec ... -- ...);

- Creates an account for badguy on DB server



More SQL Injection Attacks

- Create new users

```
'; INSERT INTO USERS ('uname','passwd','salt')  
VALUES ('hacker','38a74f', 3234);
```

- Reset password

```
'; UPDATE USERS SET email=hcker@root.org WHERE  
email=victim@yahoo.com
```

An example Vulnerable Website

<https://www.hacksplaining.com/exercises/sql-injection#/start>

+ How to Prevent SQLi attack?

- Check syntax of input for validity

- Many classes of input have fixed languages

- Email addresses, dates, part numbers, etc.
 - Verify that the input is a valid string in the language
 - Some languages allow problematic characters (e.g., ‘*’ in email); may decide to not allow these
 - Exclude quotes and semicolons

- Not always possible: consider the name Bill O'Reilly

+ How to Prevent SQLi attack?

■ Have length limits on input

- Many SQL injection attacks depend on entering long strings

■ Use provided functions for escaping strings

- Many attacks can be thwarted by simply using the SQL string escaping mechanism
 - ' → \' and " → \"
- mysql_real_escape_string() is the preferred function for this

■ Will not guard against all attacks

- Consider:
 - SELECT fields FROM table WHERE id = **23 OR 1=1**
 - No quotes here!



Overall Prevention Policy

- Prevent leakage of database schema and other information
- Limit privileges (defense in depth)
- Encrypt sensitive data stored in database
- Harden DB server and host OS
- Apply input validation

+ Lets Try some examples

Please Visit:

<http://sqlfiddle.com/#!9/1bff5c/1>



Further Practice

https://seedsecuritylabs.org/Labs_16.04/Web/Web_SQL_Injection/