Web Security

Major vulnerabilities:

- Injection
- Cross Site Request Forgery (CSRF)
- Cross Site Scripting (XSS)

Injection attack

Injection: ***Untrusted data*** is sent to an ***interpreter*** as a part of the command or query

- Code Injection
- SQL Injection

Code Injection

- An attacker could submit malicious code as an input to the web server.
- If the web server is not careful about validating user inputs, it may affect by Code Injection.
- For example, if a user types ***2+3*** in the website, the server will run ***eval('2+3')*** and return the result to the user.

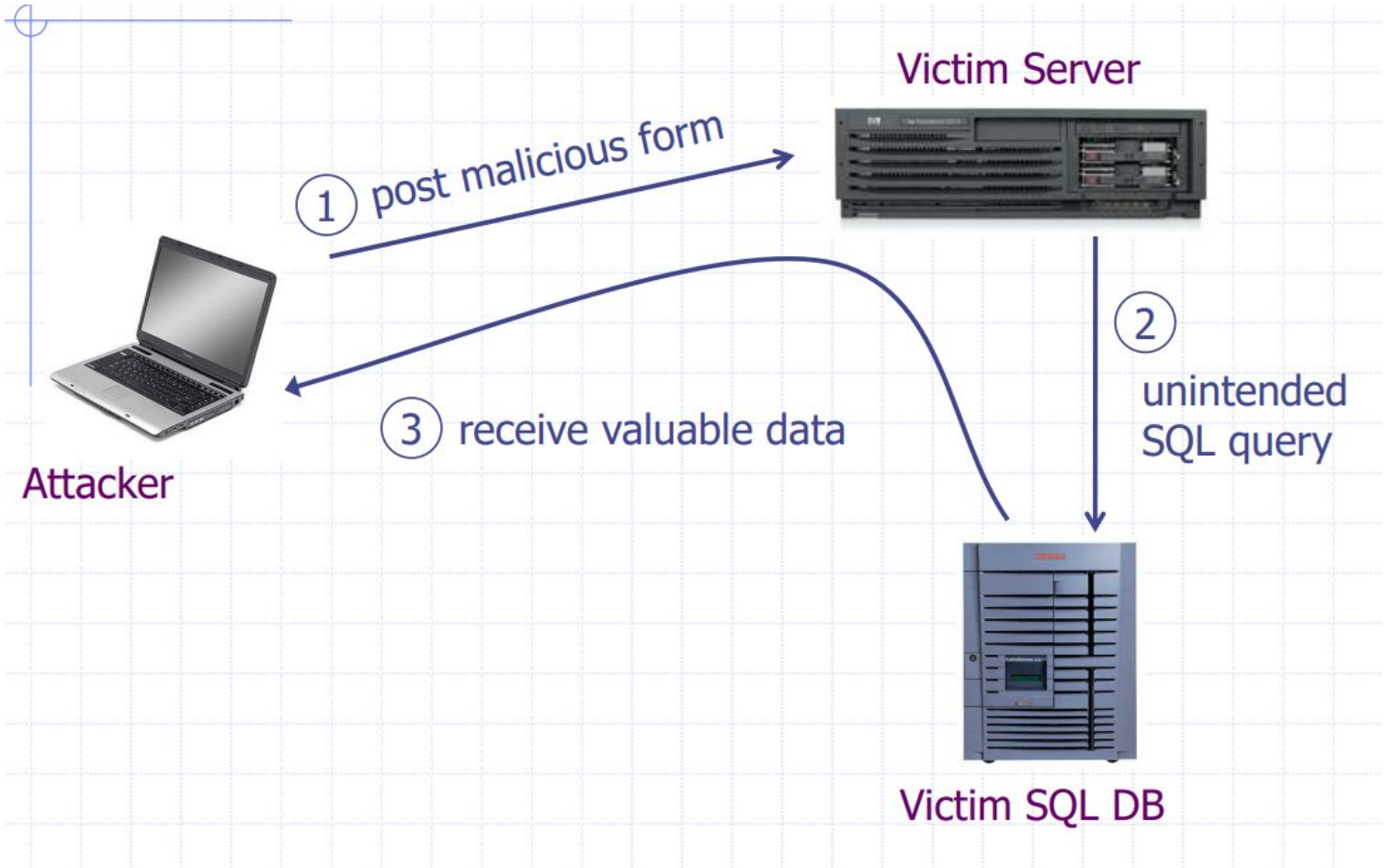
```
2+3"); os.system("rm -rf /
```

```
eval("2+3"); os.system("rm *.*")
```

Code Injection

- The general idea behind these attacks is that a web server uses user input as part of the code it runs. If the input is not properly validated, an attacker could create a special input that causes unintended code to run on the server.

SQL Injection



SQL Injection

Many modern web servers use SQL databases to store information such as user logins or uploaded files.

Course_rating

id	course	rating
1	CS101	4.5
2	CS201	4.4
3	CS301	4.6

http://www.Course.edu/Course_rating?course=**CS101**

SELECT rating FROM Course_rating WHERE course='CS101'

SQL Injection

- An attacker could create a special input that allows arbitrary SQL code to be run.

- 'garbage'; SELECT * FROM passwords WHERE username = 'admin'

- SELECT rating FROM Course_rating WHERE course = 'garbage'; SELECT * FROM passwords WHERE username = 'admin'

SQL Injection Strategies

Writing a malicious input that creates a syntactically valid SQL query can be tricky.

- garbage is a garbage input to the intended query so that it doesn't return anything.
- ' closes the opening quote from the intended query. Without this closing quote, the rest of our query would be treated as a string, not SQL code.
- ; ends the intended SQL query and lets us start a new SQL query.
- SELECT password FROM passwords WHERE username = "admin is the malicious SQL query we want to execute. Note that we didn't add a closing quote to "admin, because the intended SQL query will automatically add a closing quote at the end of our input.

SQL Injection

SELECT username FROM users WHERE username = ‘user1' AND password = 'password123‘

If the query returns more than 0 rows, the server registers a successful login.

Suppose we want to login to the server, but we don’t have an account, and we don’t know anyone’s username. How might we achieve this using SQL injection?

SQL Injection

First, in the username field, we should add a dummy username and a quote to end the opening quote from the original query:

```
SELECT username FROM users WHERE username = ‘User1'' AND password
= 'password123'
```

Next, we need to add some SQL syntax so that this query returns more than 0 rows. One trick for forcing a SQL query to always return something is to add some logic that always evaluates to true, such as OR 1=1:

```
SELECT username FROM users WHERE username = 'User1' OR 1=1' AND password = '_____'
```

SQL Injection

Next, we have to add some SQL so that the rest of the query doesn't throw a syntax error.
One way of doing this is to add a semicolon (ending the previous query) and write a dummy query that matches the remaining SQL:

SELECT username FROM users WHERE username = 'User1' OR 1=1; SELECT username FROM users WHERE username = 'User1' AND password = '_____'

SELECT username FROM users WHERE username = 'User1' OR 1=1; SELECT username FROM users WHERE username = 'User1' AND password = 'garbage'

SQL Injection

SELECT username FROM users WHERE username = 'alice' OR 1=1--'
AND password = 'garbage'
or
username = alice' OR 1=1--
password = garbage

Preventing SQL Injection

- Escape inputs
- Use parameterized/prepared SQL
- Use ORM framework