Software Security 09

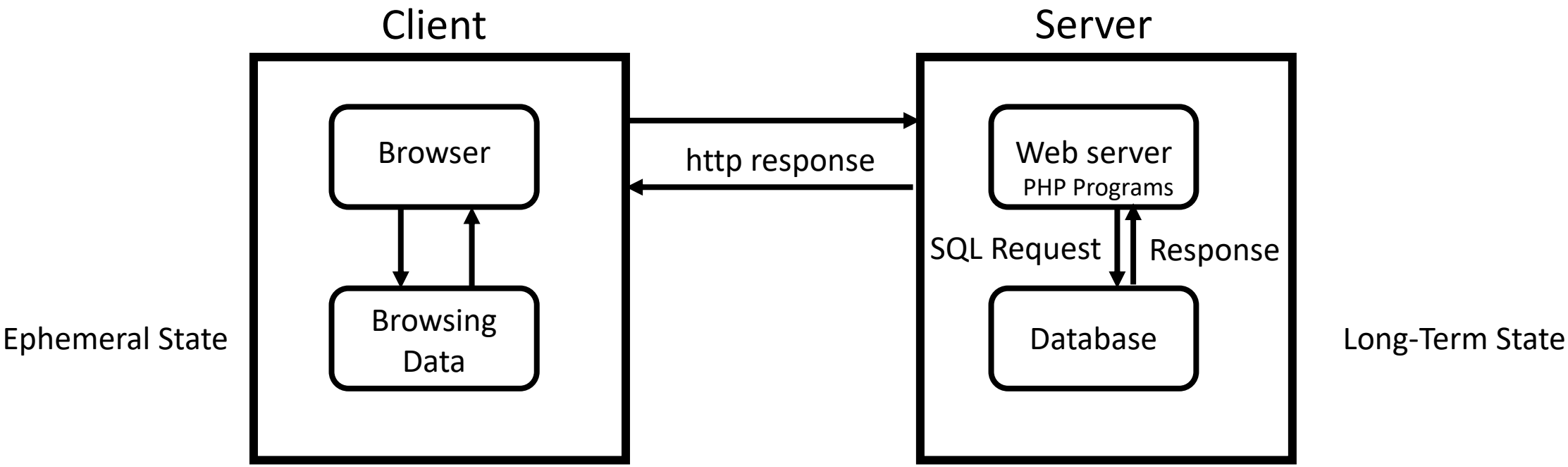
Sven Schäge

Learning Goals

- Understand modern web technology.
- Be able to detail in what stages the development progressed and why each step addresses a new specific demand for improvement.
- Understand the relevance of DBMSs and the SQL language and be able to craft simple SQL queries.
- Understand the relevance of PHP and be able to craft simple PHP files.
- Understand SQL injection and its practical relevance.
- Reflect on the underlying core problem of SQL injection and based on that provide countermeasures and argue for their effectiveness.
- Be able to abstract these countermeasures and transfer them to other technologies (in Kingdom 1).
- Be able to compare these countermeasures with countermeasures of other security vulnerabilities (of technologies in Kingdom 1).
- Reflect on the benefits, drawbacks, and application scenarios of blind SQL attacks.

SQL Injection

Websurfing and SQL Injections



Significance

The CWE Top 25

Below is a list of the weaknesses in the 2022 CWE Top 25, including the overall score of each. The KEV Count (CVEs) shows the number of CVE-2020/CVE-2021 Records from the CISA KEV list that were mapped to the given weakness.

Rank	ID	Name	Score	KEV Count (CVEs)	Rank Change vs. 2021
1	CWE-787	Out-of-bounds Write	64.20	62	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.97	2	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	22.11	7	+3 ▲
4	CWE-20	Improper Input Validation	20.63	20	0
5	CWE-125	Out-of-bounds Read	17.67	1	-2 ▼
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	17.53	32	-1 ▼
7	CWE-416	Use After Free	15.50	28	0
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.08	19	0
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.53	1	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	9.56	6	0
11	CWE-476	NULL Pointer Dereference	7.15	0	+4 ▲
12	CWE-502	Deserialization of Untrusted Data	6.68	7	+1 ▲
13	CWE-190	Integer Overflow or Wraparound	6.53	2	-1 ▼
14	CWE-287	Improper Authentication	6.35	4	0
15	CWE-798	Use of Hard-coded Credentials	5.66	0	+1 ▲
16	CWE-862	Missing Authorization	5.53	1	+2 ▲
17	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	5.42	5	+8 ▲
18	CWE-306	Missing Authentication for Critical Function	5.15	6	-7 ▼
19	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.85	6	-2 ▼
20	CWE-276	Incorrect Default Permissions	4.84	0	-1 ▼
21	CWE-918	Server-Side Request Forgery (SSRF)	4.27	8	+3 ▲
22	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	3.57	6	+11 ▲
23	CWE-400	Uncontrolled Resource Consumption	3.56	2	+4 ▲
24	CWE-611	Improper Restriction of XML External Entity Reference	3.38	0	-1 ▼
25	CWE-94	Improper Control of Generation of Code ('Code Injection')	3.32	4	+3 ▲

SQL Language in Examples

Table Accounts

UserID	Password	Balance
Alice	1234	100
Bob	5678	200

- Data stored in tables

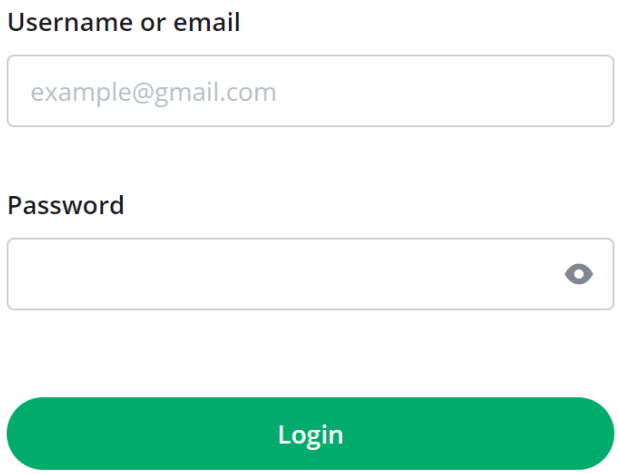
- SELECT Balance FROM Accounts WHERE (UserID='Bob' AND Password='5678');
  - Reads Bob's balance
- INSERT INTO Accounts Values('Charlie', '90AB', 200); -- A comment!
  - Adds new user Charlie together with his data
- UPDATE Accounts SET Balance='200' WHERE UserID='Alice';
  - Overwrites Alice's balance to 200
- DROP TABLE Accounts; /*Yet another, possibly multiline comment*/
  - Deletes entire table

PHP Example

Table Accounts in Database

UserID	Password	Balance
Alice	1234	100
Bob	5678	200

Website presented to User




```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);
…
$valid=mysql_query(“SELECT * from Accounts WHERE (UserID=‘$usernameField’ AND Password=‘$passwordField’);”);
…
$conn->close();
?>

<!DOCTYPE html>
…
<form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>" method="post">
<input type="text" name="username" class="form-control" value="<?php echo $usernameField; ?>">
…
</html>
```

SQL Injection

Table Accounts in Database

UserID	Password	Balance
Alice	1234	100
Bob	5678	200

- Attacker uses Username:
  - **ALICE' OR 0=0); --**
- **Resulting** SQL Query send to database:

SELECT * from Accounts WHERE (UserID='ALICE' OR 0=0); -- ' AND Password='$passwordField');

PHP Code

<?php …

$valid=mysql_query("

SELECT * from Accounts WHERE (UserID='$usernameField' AND Password='$passwordField');

");

… ?>

<!DOCTYPE html>…

<form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>" method="post">

<input type="text" name="username" class="form-control" **value="<?php echo $usernameField; ?>"**>

… </html>

SQL Injection

Table Accounts in Database

UserID	Password	Balance
Alice	1234	100
Bob	5678	200

- Attacker uses Username:
  - **ALICE' OR 0=0); DROP Table Accounts; --**
- **Resulting** SQL Query send to database:

SELECT * from Accounts WHERE (UserID='ALICE' OR 0=0); DROP Table Accounts; -- ' AND Password='$passwordField');

PHP Code

<?php …

$valid=mysql_query("

SELECT * from Accounts WHERE (UserID='$usernameField' AND Password='$passwordField');

");

… ?>

<!DOCTYPE html>…

<form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>" method="post">

<input type="text" name="username" class="form-control" **value="<?php echo $usernameField; ?>"**>

… </html>

Blind SQL

- Blind SQL injection
  - SQL injection attack where the attacker cannot directly query or see the results of the injected SQL statement.
- Classically, the attacker injects malicious SQL code that affects the logical flow of the application. The attacker then observes the application's response to determine if the injected condition is true or false.
- Boolean-based blind SQL injection:
  - attacker manipulates conditions to get a true or false response of the application.
- Time-based blind SQL injection:
  - attacker introduces **time delays** in the SQL query to infer true or false conditions, and

SELECT * FROM users WHERE username = 'input_username' AND password = 'input_password';

' OR IF(1=1, SLEEP(5), 0) --

SELECT * FROM users WHERE username = '' OR IF(1=1, SLEEP(5), 0) --' AND password = 'input_password';

SQL Injection Countermeasures

SQL Injection

Table Accounts in Database

UserID	Password	Balance
Alice	1234	100
Bob	5678	200

- Attacker uses Username:
  - **ALICE' OR 0=0); DROP Table Accounts; --**
- **Resulting** SQL Query send to database:

SELECT * from Accounts WHERE (UserID='ALICE' OR 0=0); DROP Table Accounts; -- ' AND Password='$passwordField');

PHP Code

<?php …

$valid=mysql_query("

SELECT * from Accounts WHERE (UserID='$usernameField' AND Password='$passwordField');

");

… ?>

<!DOCTYPE html>…

<form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>" method="post">

<input type="text" name="username" class="form-control" **value="<?php echo $usernameField; ?>"**>

… </html>

Root of Problem

- Input values are expected to be evaluated as data in an SQL command
- But: attackers can construct input values that would be evaluated to modify the SQL command

Countermeasures

- Input Validation
  - Similar to buffer overflow countermeasures
- Methods to validate input
  - Check and drop input if is does not have expected form; otherwise accept
    - Whitelisting
      - Rejection rather than fix
  - Sanitize, i.e. modify to have the correct from
    - Blacklisting: delete unwanted characters, e.g. –
    - Escaping: change problematic characters in a way that won‘t change the control flow
      - E.g. use \‘ for ‘, makes clear that apostrophe is used as data
      - Use libraries to do that!
    - Prepared Statements

Prepared Statements

- In PHP, SQL statements are prepared and thus syntactically fixed
  - Data variables are represented simply by ?
- In a second PHP command, the variables are filled with actual values (binding)
- This two-step approach clarifies the difference between data that belongs to the structure of the SQL statement and simple data
- For other languages that access databases prepared statements can also be expressed

Further Techniques

- Use Restrictive Acccess Control Policy to Important Tables
- Encrypt Sensitive Data