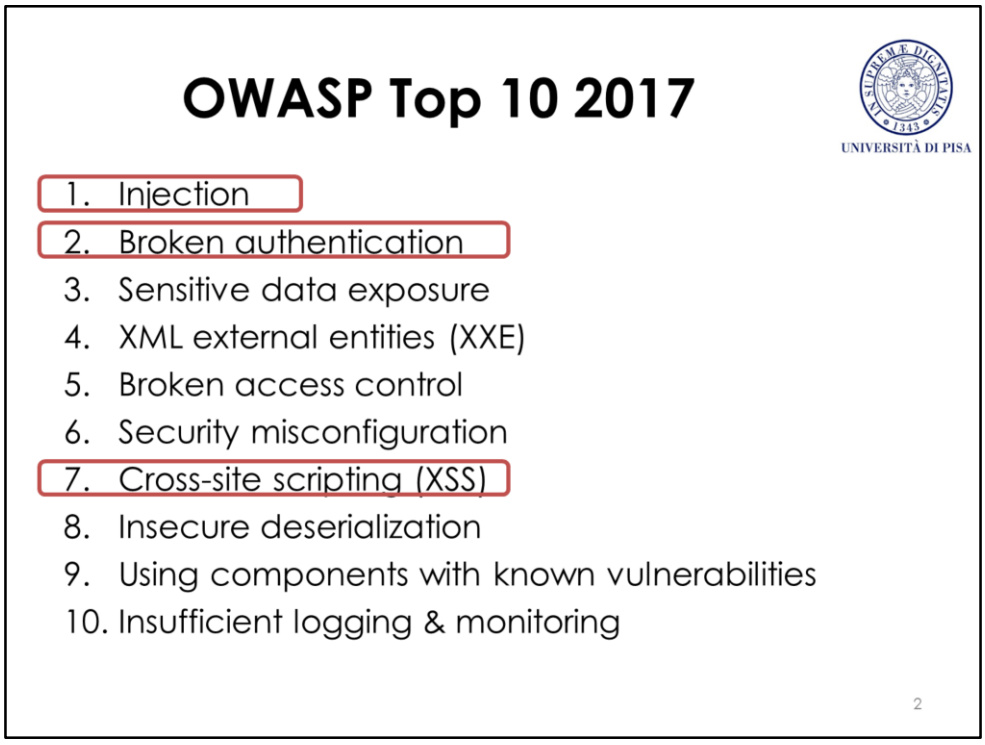
Cybersecurity
Web Security

Pericle Perazzo
Dept. of Information Engineering
University of Pisa
pericle.perazzo@iet.unipi.it
Version: 2018-04-19



This slide shows the OWASP Top 10 Web Application Security Risks of 2017, which is a list of the currently most dangerous web vulnerabilities in terms of prevalence (how much the vulnerability is widespread), exploitability, and technical impact. In these slides we will focus only on injection and broken authentication vulnerabilities.

Web Security

INJECTION

3

Catch the bug!

UNIVERSITÀ DI PISA

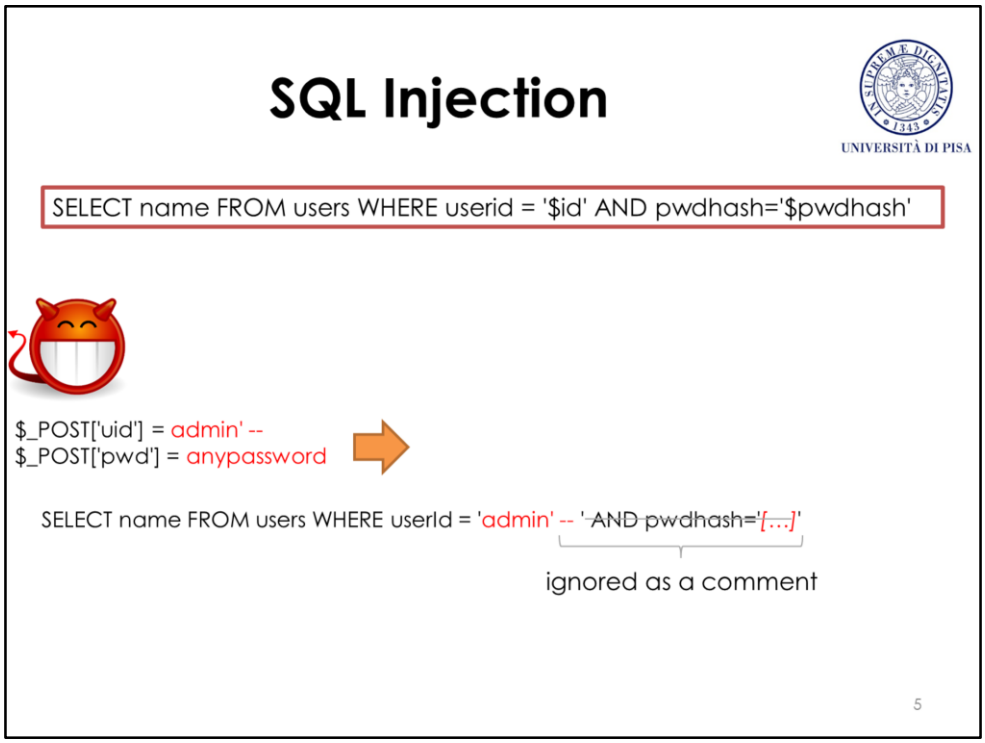
```
$id = $_POST['uid'];
$pwdhash = hash('sha256', $_POST['pwd']);
$res = mysqli_query($link,
  "SELECT name FROM users WHERE userid = '$id' AND pwdhash='$pwdhash'"
);
if (!$res) { die('Query error'); }
$row = mysqli_fetch_row($res);
if (!$row) { die('Invalid user ID or password'); }
echo "Hello, $row[0]";
// … Begin session
```

$_POST['uid'] = myuser
$_POST['pwd'] = pa55w0rd

SELECT name FROM users WHERE userId = 'myuser' AND pwdhash='56965e[…]'

4

This PHP code performs a simple login procedure by matching a user identifier and a (hashed) password provided by the client as HTTP POST parameters on a database of users. If the user-provided identifier <$id> and the hashed password <$pwdhash> do not match any tuple in the <users> table, then an authentication error will be raised. Otherwise, a personalized welcome message will be printed and a new active session will be registered. The query is built by inserting the values of <$id> and <$pwdhash> in a skeleton query string.



The code works normally if the inputs are those expected, but the behavior is unexpected if the input contains SQL symbols. Consider an attacker that sends the above malicious user ID, which contains SQL symbols that are interpreted as code by the victim system. The "'" character closes the string literal in the query string, and the "-- " (with the trailing space) symbol makes the SQL interpreter to ignore the successive query, thus bypassing the password check. The attacker gains access to the system with the privileges of the user "admin". This is an example of *SQL injection* attack. Despite such an attack is quite simple, it represents one of the most common and devastating attacks in the web.

SQL Injection

- Problem: part of user input is treated as SQL code
- Number #1(!) security vulnerability in web applications (2017)
- Risks:
  - Bypassing authentication / escalate privileges
  - Stealing data
  - Adding or modifying data
  - Partially or totally deleting a database

6

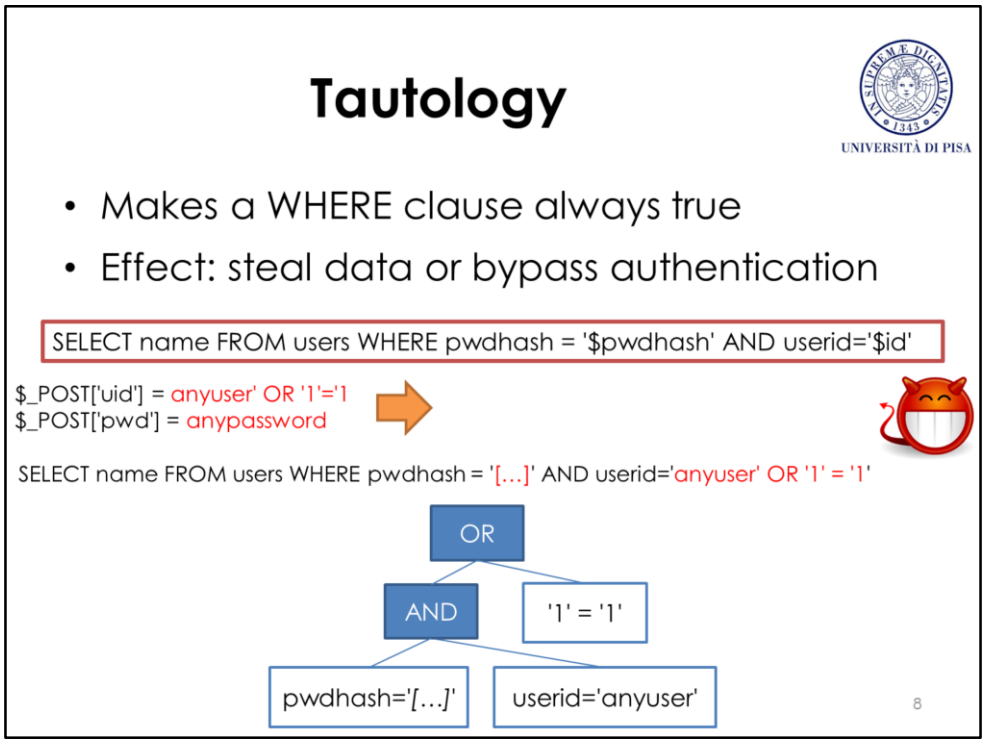
A SQL injection vulnerability occurs when a system builds a SQL query from some user input, in such a way that part of the user input can be treated as SQL code. Despite its simplicity, this is one of the most common and devastating attacks of the web according to OWASP. Bypassing authentication is not the only possible impact of a SQL injection. Other effects include escalating privileges, stealing data, adding or modifying data, partially or totally deleting a database.

SQL Injection Techniques

- Clause commenting (previous example)
- Tautology
- Union query
- Piggybacked query

7

There are many techniques for injecting SQL code. The most common ones are: clause commenting (the previous example), tautology, union query, and piggybacked query.



The *tautology* technique consists in injecting a condition wich always evaluates to true (e.g., 1=1) in a WHERE clause in order to bypass it somehow. Imagine that the WHERE condition of the previous example had the password and the userid clauses appearing in a different order, like shown in the above slide. In this case, the simple clause commenting technique cannot be applied. However, the attacker can bypass authentication anyway with the tautology technique, by injecting «anyuser' OR '1'='1» as the user ID. The query will return more than one tuple (actually all the tuples of the <users> table), thus the attacker can bypass the authentication.
The clause commenting and the tautology techniques can also be used to steal data. For example, a webpage that displays some information of the current user can be forced to display information of all the users by making the WHERE clause evaluate always to true.

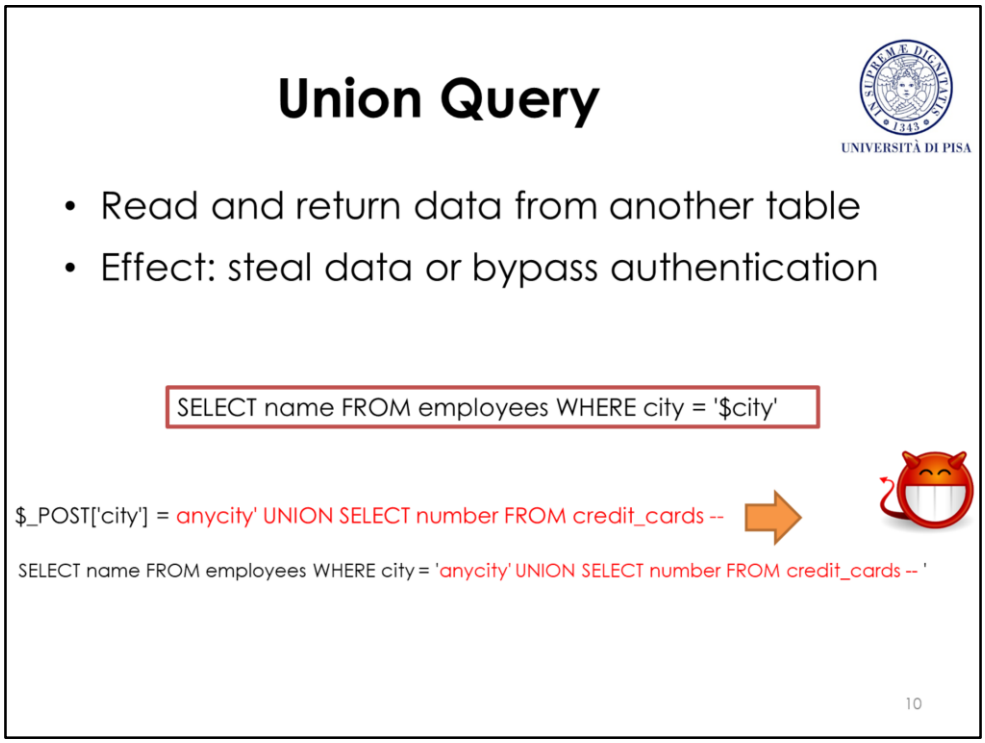
Catch the bug!

UNIVERSITÀ DI PISA

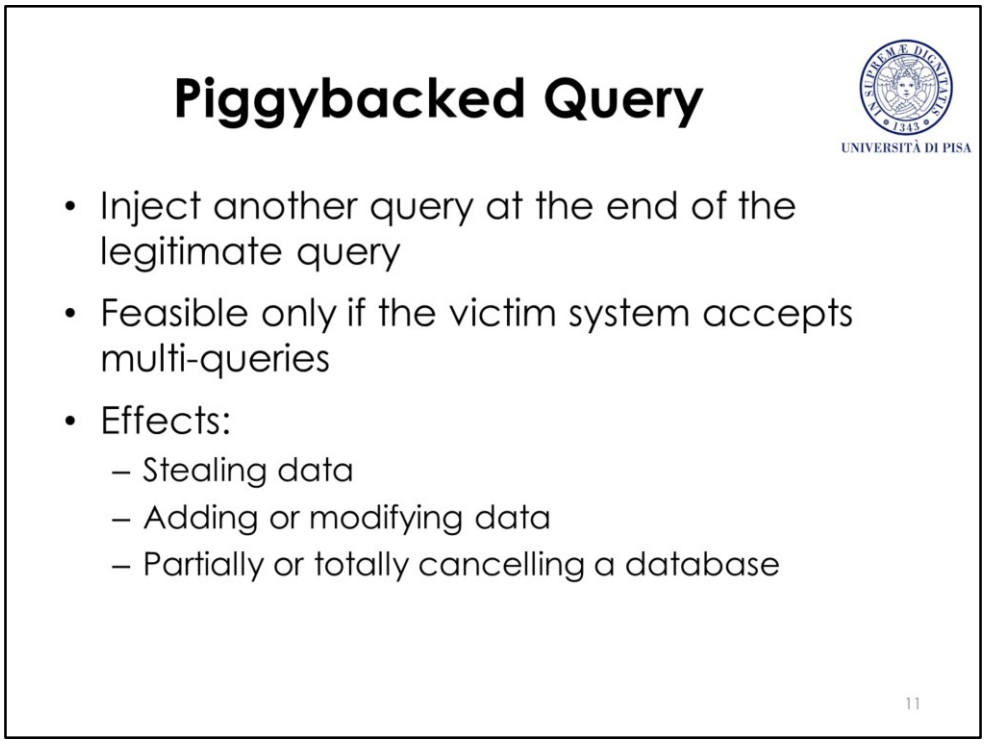
```
$city = $_POST['city'];
$res = mysqli_query($link,
  "SELECT name FROM employees WHERE city = '$city'"
);
if (!$res) { die('Query error'); }
while ($row = mysqli_fetch_row($res)) {
  echo $row[0] . '</br>';
}
```

9

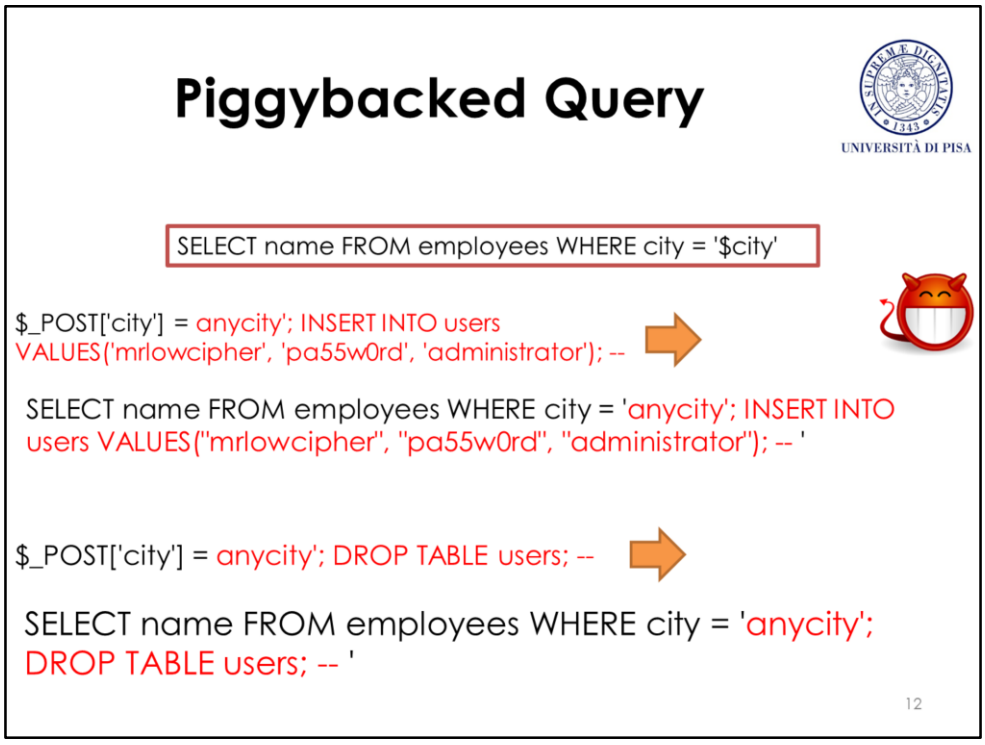
This PHP code displays the names of the all the employees working in a city specified by an HTTP POST parameter. Assume that the <employees> table contains no sensitive information.



With the *union query* technique, an attacker can deceive the system into returning data from a table different from the one specified by the SELECT statement. The attacker injects a UNION operand followed by another query. The UNION operand merges the tuples resulting from the first and the second query.
In the example code, all the tuples of the sensitive table <credit_cards> are returned to the attacker. Note that the final «-- » symbol is necessary to comment away the final «'» symbol, which would otherwise cause the query to fail due to syntax error.



With the *piggybacked query* technique, an attacker injects another arbitrary query after the legitimate query. This is possible only if the victim website performs vulnerable multi-queries on the SQL database, or if the SQL database is configured to accept multi-queries.



With piggybacked queries, it is possible to modify or add data, for example change the administrator password or add a new adversary-controlled administrator user. It is also possible to delete partially or totally a database, thus causing a denial of service and a loss of valuable information.

Injection Vectors

UNIVERSITÀ DI PISA

- User input (HTTP post/get)

```
SELECT name FROM users WHERE userid = '$_POST['id']' AND
pin=$_POST['pin']'
```

- Cookies

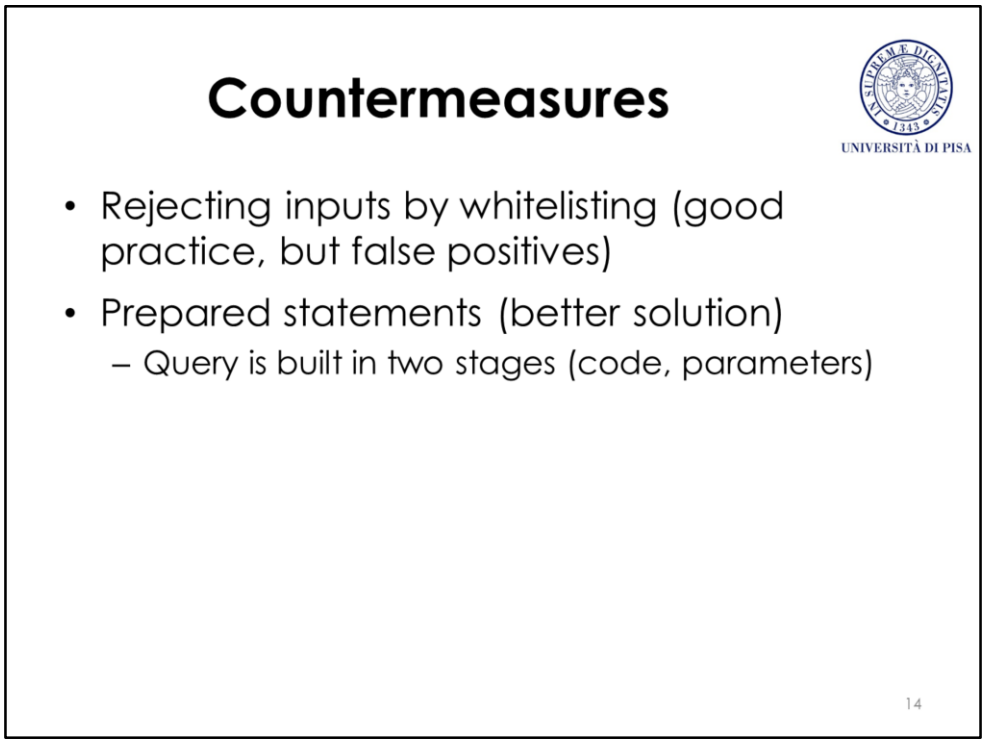
```
SELECT userid FROM sessions WHERE sid = '$_COOKIE['sid']'
```

- Server variables

```
INSERT INTO log VALUES ('$_SERVER['HTTP_USER_AGENT']',
'date(DATE_ATOM)')
```

13

An attacker can leverage multiple attack vectors to carry out a SQL injection. Apart from POST/GET parameters, other common attack vectors are cookies, and server variables for example HTTP header fields like the browser name and version (*user agent*). Injection attacks through server variables are particulary common in log operations, for example if a website stores in a log database the user agent and the date of each client (see slide above).



Whitelisting inputs and rejecting them if they contain invalid symbols is always a good practice, but it may be insufficient for SQL injection. Indeed, many SQL symbols appear in legitimate field values (e.g., the name «Randy O'Brian» contains the operator «and» and the symbol «'»). If we accept the «'» symbol, then the system will be vulnerable. If we reject it, then the system will reject legitimate names (false positive). A more definitive solution are *prepared statements*, which is a technique by which the query is built in two separate stages: SQL code first and then input parameters. Prepared statements were originally introduced for improving the performance of executing many times the same query with different parameters. As a secondary effect, prepared statements also make SQL injection attacks infeasible.

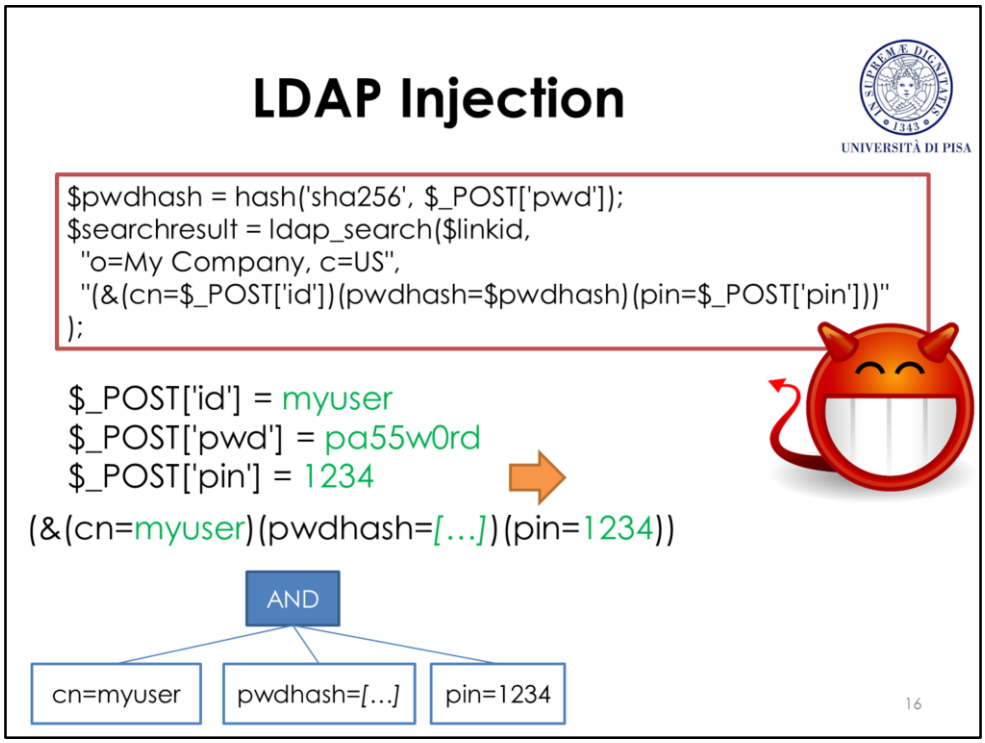
Prepared Statements

UNIVERSITÀ DI PISA

```
$id = $_POST['uid'];
$pwdhash = hash('sha256', $_POST['pwd']);
$stmt = mysqli_prepare($link,
  "SELECT name FROM users WHERE userid=? AND pwdhash=?");
mysqli_stmt_bind_param($stmt, 'ss', $id, $pwdhash));
if(!mysqli_stmt_execute($stmt)) { die('Query error'); }
$res = mysqli_stmt_get_result($stmt);
$row = mysqli_fetch_row($res);
if (!$row) { die('Invalid user ID or password'); }
mysqli_stmt_close($stmt);
echo "Hello, $row[0]";
// … begin session
```

15

This code snippet shows an example of prepared statement in PHP language with MySQL. The function mysqli_prepare() prepares a SQL query defined up to two parameters, indicated by two «?» symbols in the query string. The parameters are inserted in the prepared query afterwards, with the function mysqli_stmt_bind_param(), where 'ss' stands for «two parameters, both of string type». The mysqli_stmt_bind_param() function correctly escapes the parameters in such a way that no SQL injection attack is possible. The following mysql_stmt_execute(), mysqli_stmt_get_result(), and mysqli_stmt_close() functions respectively executes the query, retrieves the result, and deallocates the prepared statement.



SQL is not the only language subject of injection. Other examples are LDAP filters (*LDAP injection*), shell commands (*OS command injection*), Hibernate queries (*Hibernate injection*), XPath expressions (*XPath injection*), etc.
LDAP (Lightweight Directory Access Protocol), which is a protocol for accessing and maintaining a directory service, which is a sort of non-relational database that binds names to resources. LDAP databases are less powerful than relational databases as they cannot process complex transactions. However, they offer high performances when reading data. This makes LDAP very used to maintain databases of users and credential, which are often read and seldom updated. An LDAP database simply associates names (e.g., a user name) to resources (e.g., a password hash). It can be queried with an LDAP search operation, which takes a search filter as a parameter. An LDAP search filter is specified in a particular string format, which could be subject to injection (*LDAP injection*) if not properly protected.
In the above example, an LDAP search filter is used to check if a given user with given a password is present in the database. It follows a Polish notation (i.e., first the operator «&», followed by a list of operands), and it reads as follows: «cn (common name) is myuser and pwdhash is […] and pin is 1234».

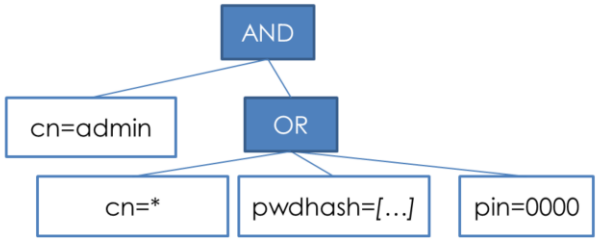
LDAP Injection

UNIVERSITÀ DI PISA

(&(cn=$_POST['id'])(pwdhash=$pwdhash)(pin=$_POST['pin']))

$_POST['id'] = admin)(|(cn=*
$_POST['pwd'] = anypassword
$_POST['pin'] = 0000)

(&(cn=admin)(|(cn=*)(pwdhash=*[…]*)(pin=0000)))



17

The above slide shows a possible LDAP injection attack by tautology technique, after which the search filter always evaluates to true.

LDAP Injection


```
$pwdhash = hash('sha256', $_POST['pwd']);
$escp_id = ldap_escape($_POST['id'], null, LDAP_ESCAPE_FILTER);
$escp_pwdhash = ldap_escape($pwdhash, null, LDAP_ESCAPE_FILTER);
$escp_pin = ldap_escape(_POST['pin'], null, LDAP_ESCAPE_FILTER);
$searchresult = ldap_search($linkid,
  "o=My Company, c=US",
  "(&(cn=$escp_id)(pwdhash=$escp_pwdhash)(pin=$escp_pin))"
);
```

18

This PHP code snippet shows how to correctly escape LDAP filter arguments before using them.

Web Security

BROKEN AUTHENTICATION

19

Improper Authentication

UNIVERSITÀ DI PISA

```
if (!$_COOKIE['loggedin']) {
  // … perform authentication
  if(/* authentication fail */) { die('Auth fail'); }
  setcookie('loggedin', 'true');
  echo "Hello, user!";
}
// … continue session
```

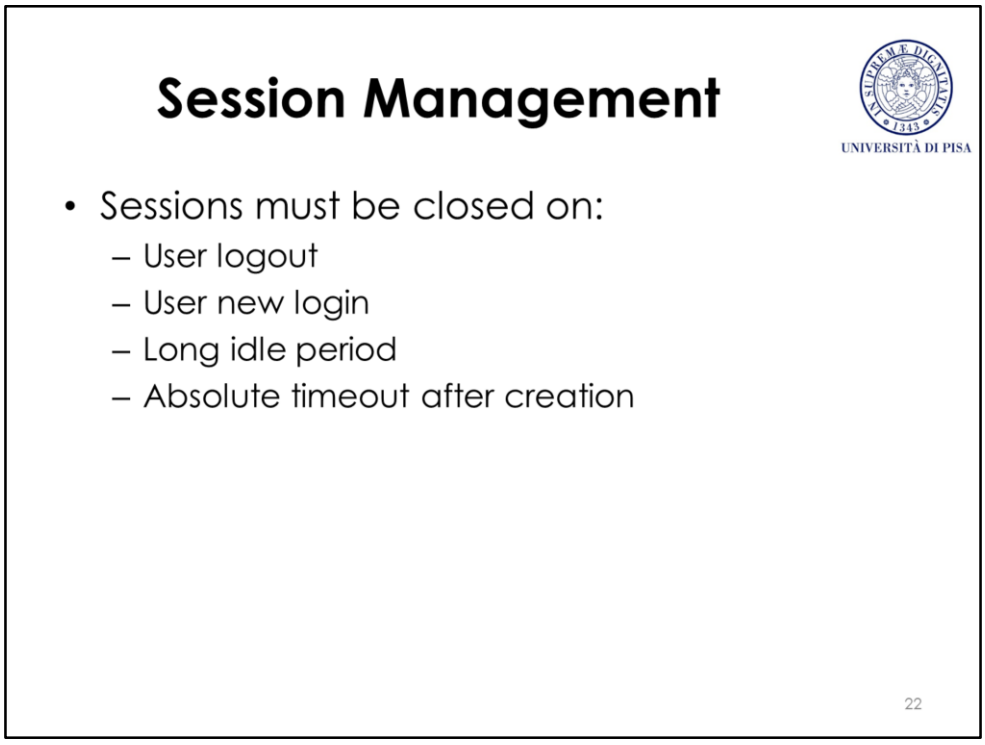
«*Broken authentication*» is a general term that includes a family of different vulnerabilities involving the authentication and the session management mechanisms. This PHP code snippet shows an example of (obviously) improper way to manage authentication. The logged/unlogged status of a user is maintained by a cookie. If such a cookie is not present, then the user undergoes the authentication process, after which the cookie is set. However, cookies are client-side data, and they can be easily created or modified by the user via browser. To bypass authentication, a malicious user can simply set a cookie named «loggedin».

Improper Authentication

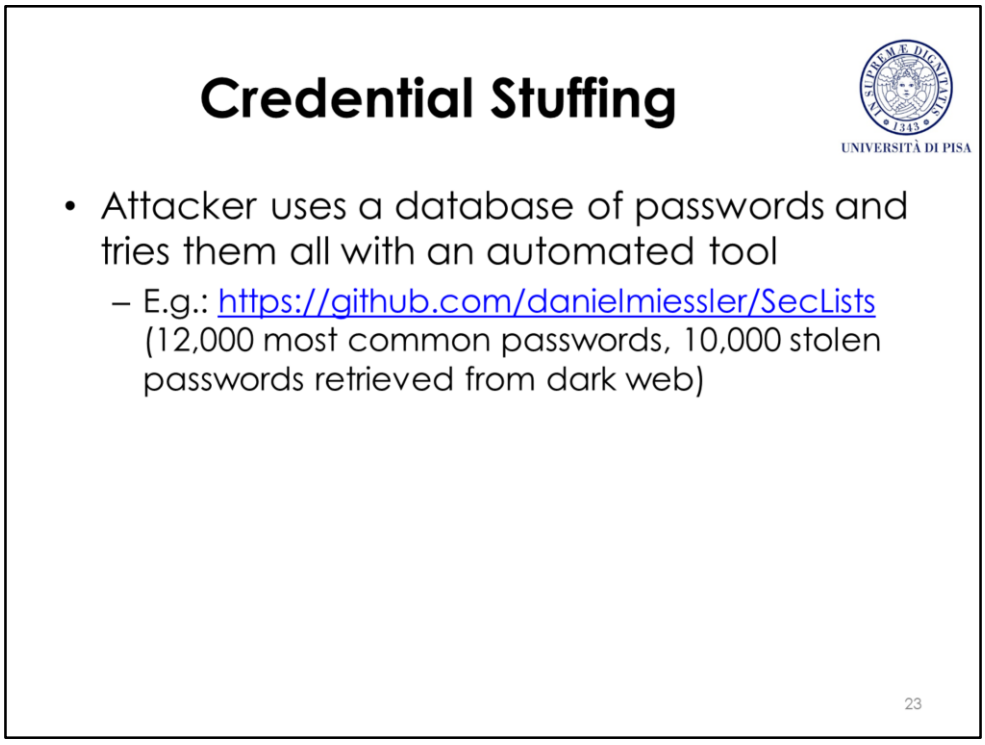
UNIVERSITÀ DI PISA

```
session_start();
if (!isset($_SESSION['AUTHTIME'])
    || time() - $_SESSION['AUTHTIME'] >= 600) {
  session_unset();
  session_destroy();
  // … perform authentication
  if(/* authentication fail */) { die('Auth fail'); }
  session_start();
  $_SESSION['AUTHTIME'] = time();
  echo "Hello, user!";
}
// … continue session
```

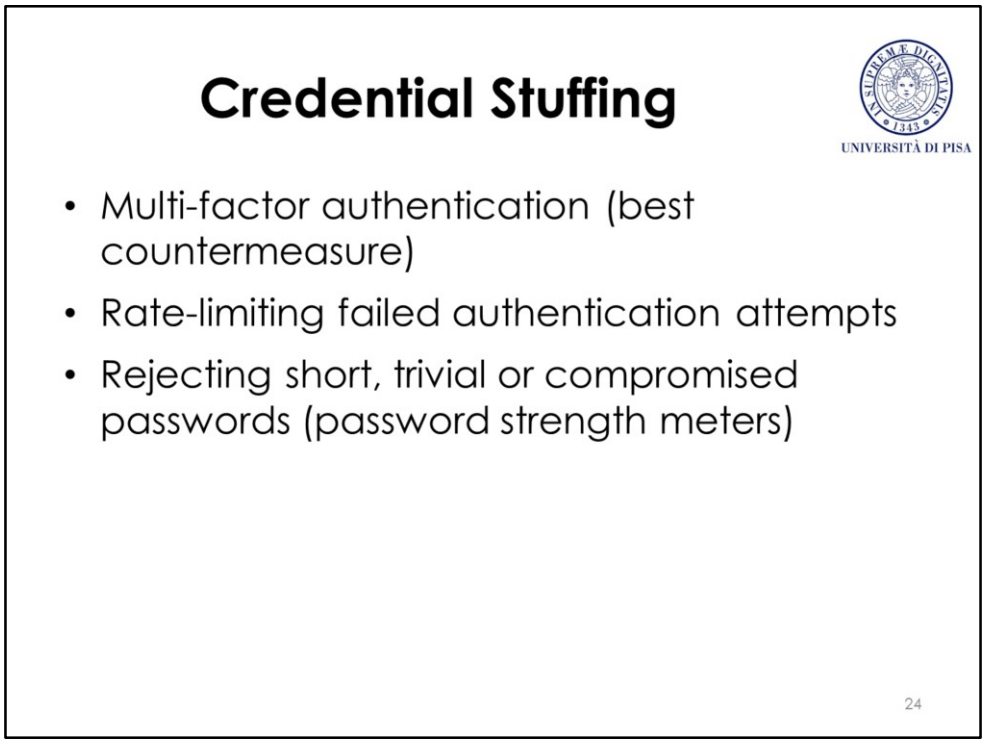
A generally accepted technique to manage sessions in web applications involves *session IDs*. A session ID is a cryptographically secure random number generated by the server. The session ID is stored both on the client (typically, by a cookie) and on the server (typically, in a temporary file). The client specifies the session ID for each request within the session. At the logout or after a timeout, the server cancels the session ID. This PHP code snippet shows how to implement a simple session with the session ID technique. The function session_start() establishes a new session with the current client, or resumes an existing one. The associative array <$_SESSION> can be used to store values about the current session established with the client, for example the session start timestamp. Unlike $_COOKIE variables, $_SESSION variables are stored at the server-side, so the client cannot modify them. The code snippet performs authentication if no authentication timestamp exists, or the authentication is too old (10 minutes). Before performing a new authentication, it closes the old session, by clearing all session data and destroying it with the functions session_unset() and session_destroy().



Users often forget to logout after their work session ended. If the client's browser is used by multiple users (e.g., in case of a public computer), a malicious user can continue the session established by a previous legitimate user. For this reason, sessions must always be closed after an absolute timeout after creation and after a long idle period.



Credential stuffing is an attack by which the attacker uses a database of common or stolen passwords and tries them all against a website with an automated tool. The attacker can also try permutations or variations of dictionary words.

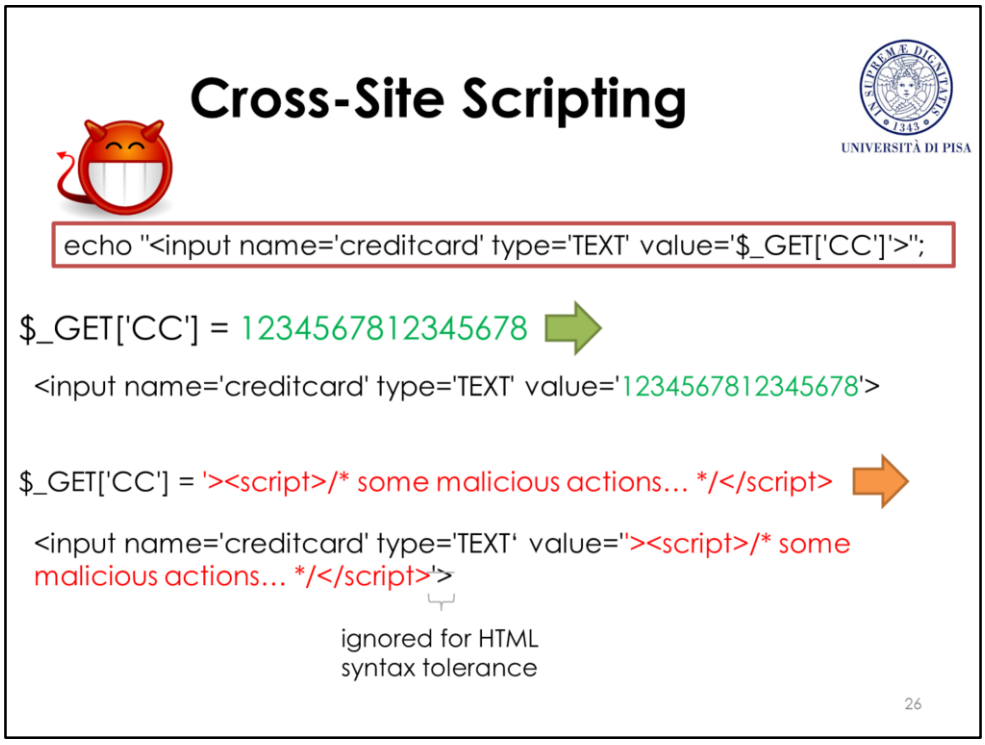


Credential stuffing is generally hard to avoid, because it leverages the weakness of the passwords chosen by the user. Passwords are often the weakest link of the security chain in every application. This is because users cannot memorize too complex and too many passwords. The best countermeasure is to perform *multi-factor authentication*, i.e., confirming the user identity on the basis of multiple pieces of evidence of different types, for example a password and a smart card. Typically, authentication factors are categorized in three types: what the user knows (e.g., a password), what the user has (e.g., a smart card or a token generator), what the user is (e.g., his/her fingerprints). However, multi-factor authentication is expensive and cannot be applied in every application. Another common and cheaper countermeasure consists in limiting the rate of failed authentication attempts, in order to slow-down automated credential stuffing. The user is forced to wait some time before retrying after a failed login attempt. The amount of time to wait must increase with the number of failed attempts. A good practice is also rejecting to weak passwords at signup time. Using a password strength meter which measures (typically with a set of heuristics) the strength of a password also incentives the user in choosing a good password.

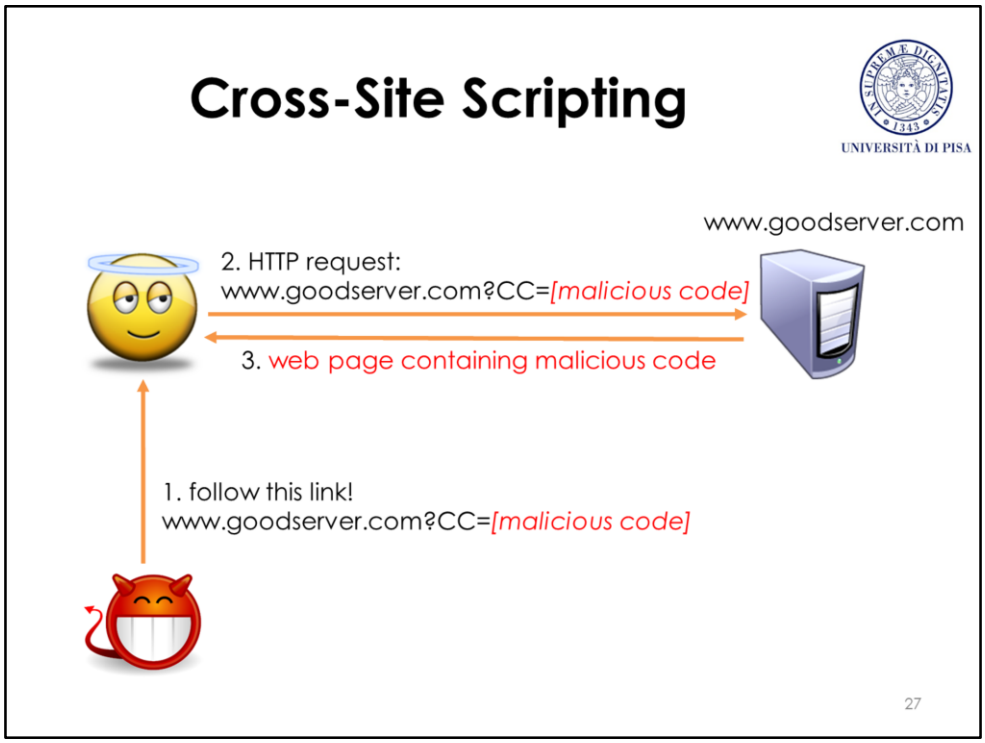
Web Security

CROSS-SITE SCRIPTING (XSS)

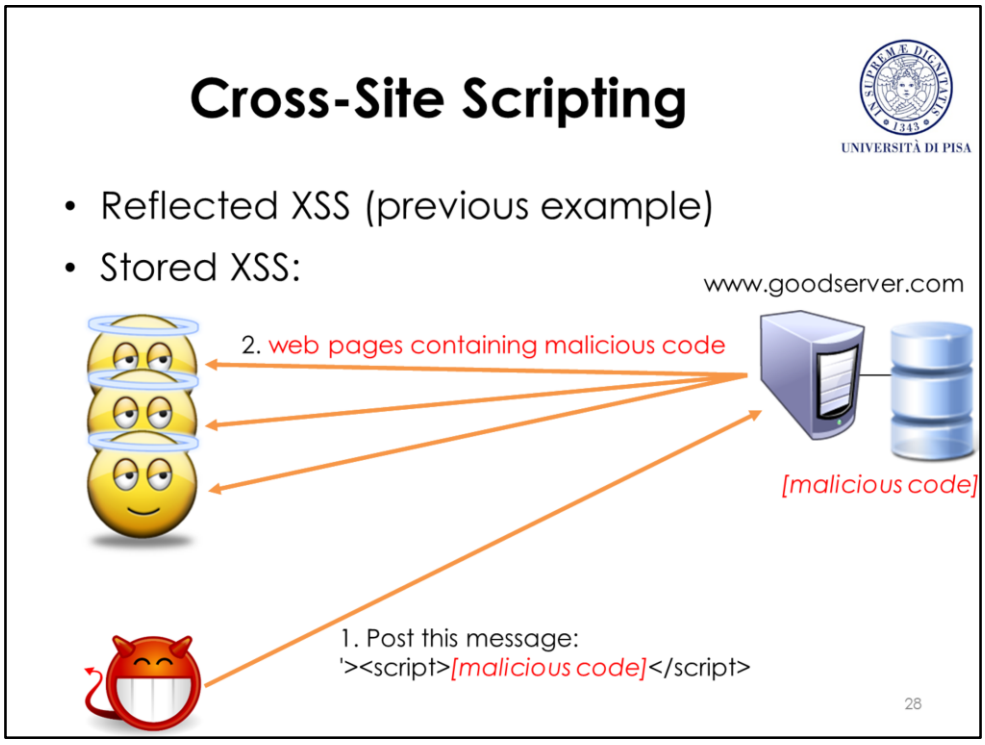
25



Cross-site scripting (XSS) is an attack by which an attacker injects some malicious client-side code (e.g., Javascript code) in a web page provided by a legitimate server. Such malicious code is then executed by the victim client.
In the example above, the server shows an input field where the user can input a credit card number. The default value of such a field is provided by the client through an HTTP GET parameter. However, such a parameter could contain HTML code, so that an attacker can inject a <script> tag containing some malicious Javascript code.



The attack follows the above schema. The adversary deceives a victim into following a particular link to a honest server, which contains the injected code as an HTTP GET parameter. The honest server then forms a web page containing the malicious code, which is finally executed by the victim. The malicious code can do various things. Typically, it sends somehow to the attacker the session ID that the victim user has established with the honest server, in such a way the attacker can do operations (e.g., money transferts) on behalf of the victim user (*session hijacking*). Cross-site scripting is particularly dangerous because the client browser trusts the honest server, so it could be configured the execute scripts sent by the server without any protection.



The previous example shows the *reflected XSS* technique. Another, more devastating, technique is the *stored XSS*. In the stored XSS, the attacker makes the server store the injected malicious code, so that the server builds web pages containing it to many clients.

Countermeasure

UNIVERSITÀ DI PISA

```
$escp_cc = htmlspecialchars($_GET['CC'], ENT_QUOTES);
echo "<input name='creditcard' type='TEXT' value='$escp_cc'>";
```

$_GET['CC'] = '><script>/* some malicious actions… */</script>

<input name='creditcard' type='TEXT' value='&#x27&gt&lt[…]'>

29

A general countermeasure is escaping all the untrusted inputs before including them in the HTML code. A way to do that in the PHP language is to use the htmlspecialchars() function. Such a function translates the «'» character into «&#x27», the «>» character into «&gt», etc. so that the untrusted input is totally interpreted as data.

Laboratory Time

UNIVERSITÀ DI PISA

Laboratory Time

- In VictimOS, attack the website available at http://intranet.dev/sqli/index.html with SQL injection
- Login form:
  - Bypass authentication as the user «Batman» through clause commenting
  - Bypass authentication as any user through tautology
  - Steal credit card numbers through union query
- Signup form:
  - Cancel a table through piggybacked query
- Correct the vulnerabilities of the website