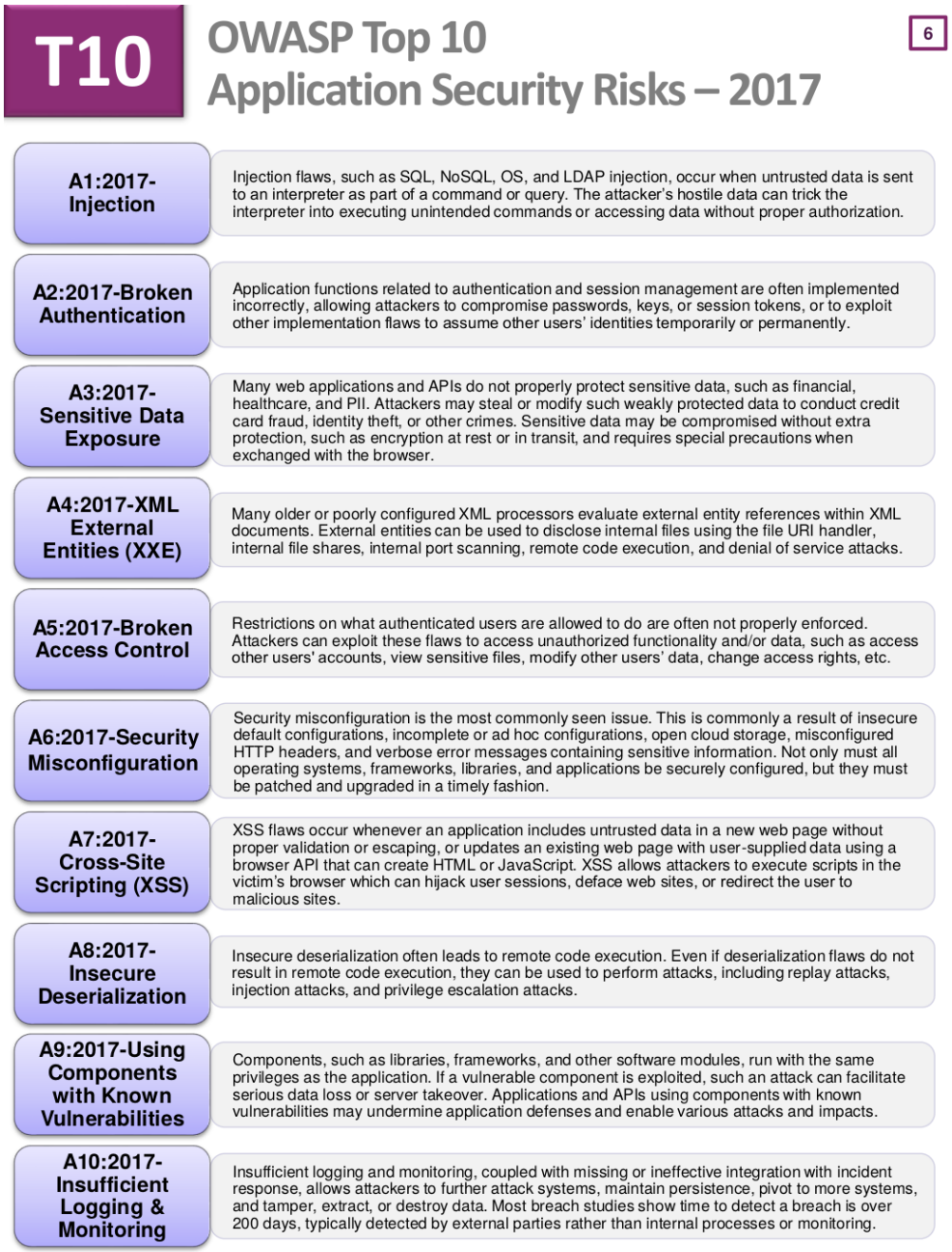
Web security: server-side attacks

Myrto Arapinis
School of Informatics
University of Edinburgh

March 25, 2019

OWASP top 10 (2017)



- OWASP is an open community dedicated to enabling organizations to conceive, develop, acquire, operate, and maintain applications that can be trusted.
- OWASP develops tools, documents, forums, and chapters for improving application security.

Injection attack

OWASP definition

Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

We are going to look at:

- command injection attacks
- SQL injection attacks

Command injection: a simple example

- Service that prints the result back from the linux program `whois`
- Just like a phone directory, `whois` is a lookup tool that allows querying information about Domain names (e.g. `google.com`), IP address (e.g. `216.58.208.78`), and ASN (Autonomous System Numbers) (e.g. `AS15169`)
- Invoked via URL like (a form or Javascript constructs this URL):

  `http://www.example.com/content.php?domain=google.com`

- Possible implementation of `content.php`

```
<?php
  if ($_GET['domain']) {
    <?  echo system('whois '.$_GET['domain']); ?>
  }
?>
```

Command injection: a simple example cont'd

- This script is subject to a **command injection attack**! We could invoke it with the argument `www.example.com; rm *`

  `http://www.example.com/content.php?`
  `domain=www.google.com; rm *`

- Resulting in the following PHP

```
<?  echo system('whois www.google.com; rm *'); ?>
```

Defense: input escaping

```
<?  echo system('whois'.escapeshellarg($_GET['domain'])); ?>
```

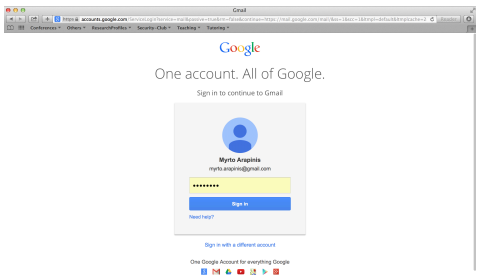
escapeshellarg() adds single quotes around a string and quotes/escapes any existing single quotes allowing you to pass a string directly to a shell function and having it be treated as a single safe argument

GET INPUT	Command executed
`www.google.com`	`whois 'www.google.com'`
`www.google.com; rm *`	`whois 'www.google.com rm *'`

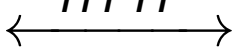
Command injection recap

- Injection is generally caused when data and code share the same channel:
  - `"whois"` is the code and the filename the data
  - **But** `';'` allows attacker to include new command
- **Defenses** include input validation, input escaping and use of a less powerful API
- **Defenses** include applying the principle of least privilege: the web server should be operating with the most restrictive permissions as possible (read, write, and execute permissions only to necessary files)

Web applications

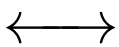


Client
(HTML, JavaScript)



Google

Server
(PHP)



Database
(SQL)

Databases

- A database is a system that stores information in an organised way, and produces report about that information based on queries.
- DBs often contain confidential information, and are thus frequently the target of attacks.
- Web server connects to DB server:
  - Web server sends **queries** or **commands** according to incoming HTTP requests
  - DB server returns associated values
  - DB server can **modify/update** records
- SQL: commonly used database query language - supports a number of operations to facilitate the access and modification of records in DB

username	password
alice	01234
bob	56789
charlie	43210

user_accounts

SQL SELECT

To express queries, retrieve a set of records from DB:

```
SELECT field FROM table WHERE condition # SQL comment
```

returns the value(s) of the given field in the specified table, for all records where condition is true

Example:

username	password
alice	01234
bob	56789
charlie	43210

user_accounts

`SELECT password FROM user_accounts WHERE username='alice'` returns the value `01234`

SQL INSERT

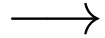
To create new records in DB:

```
INSERT INTO table VALUES record # SQL comment
```

Example:

username	password
alice	01234
bob	56789
charlie	43210

user_accounts



username	password
alice	01234
bob	56789
charlie	43210
eve	98765

user_accounts

```
INSERT INTO user_accounts VALUES ('eve', 98765)
```

Other SQL commands

- `DELETE FROM table_name WHERE condition`: deletes existing records satifying the condition
- `DROP TABLE table`: deletes entire specified table
- Semicolons separate commands:
  Example:

```
INSERT INTO user_accounts VALUES ('eve', 98765);
SELECT password  FROM user_accounts
                 WHERE username='eve'
```

  returns `98765`

SQL injection: a simple example

The web server logs in a user if the user exists with the given username and password.

```
login.php:
$conn = pg_pconnect("dbname=user_accounts");
$result = pg_query(conn,
                   "SELECT * from user_accounts
                   WHERE username = " '.$_GET['user'].'"
                   AND password = '".$_GET['pwd']."';");
if(pg_query_num($result) > 0) {
   echo "Success";
   user_control_panel_redirect();
}
```

It sees if results exist and if so logs the user in and redirects them to their user control panel

SQL injection: a simple example

Login as admin:

SQL injection: a simple example

Login as admin:

http://www.example.com/login.php?user=admin'#&pwd=f

```
pg_query(conn,
         "SELECT * from user_accounts
         WHERE username = 'admin' # ' AND password = 'f';");
```

SQL injection: a simple example

Login as admin:

http://www.example.com/login.php?user=admin'#&pwd=f

```
pg_query(conn,
         "SELECT * from user_accounts
         WHERE username = 'admin' # ' AND password = 'f';");
```

Drop `user_accounts` **table:**

SQL injection: a simple example

Login as admin:

http://www.example.com/login.php?user=admin'#&pwd=f

```
pg_query(conn,
         "SELECT * from user_accounts
         WHERE username = 'admin' # ' AND password = 'f';");
```

Drop `user_accounts` table:

http://www.example.com/login.php?user=admin';
DROP TABLE user_accounts #&pwd=f

```
pg_query(conn,
         "SELECT * from user_accounts;
         WHERE user = 'admin'; DROP TABLE user_accounts;
         # ' AND password = 'f';");
```

Defense: sanitising the input

- SQL injection vulnerabilities are the result of prgrammers failing to sanitise user input before using that input to construct database queries.
- Most languages have built0in functions that strip input of dangerous characters:
  PHP provides function `mysql_real_escape_string` to escape special characters.

Defense: prepared statements

- Idea: the query and the data are sent to the database server separately
- Creates a template of the SQL query, in which data values are substituted
- Ensures that the untrusted value is not interpreted as a command

```
$result =  pg_query_params(
             conn,
             SELECT * from user_accounts  WHERE username = $1
                                          AND password = $2,
             array($_GET['user'], $_GET['pwd']));
```