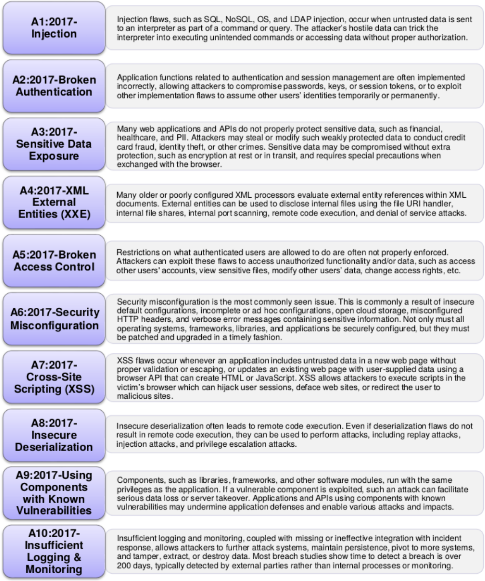
Injection attacks

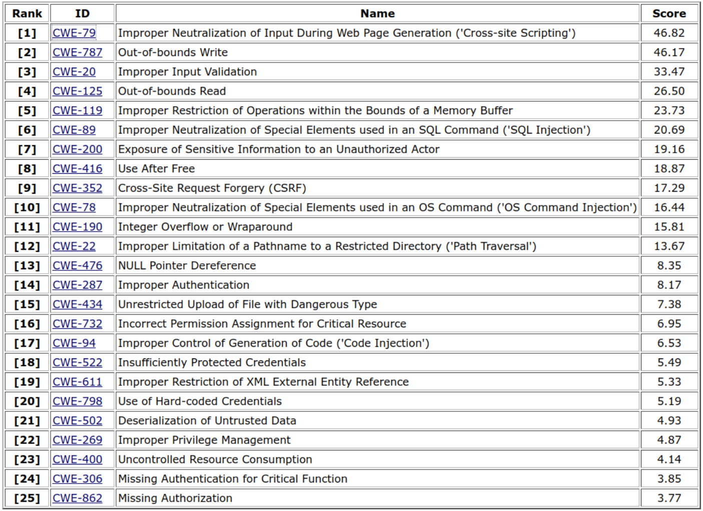
Myrto Arapinis
School of Informatics
University of Edinburgh

OWASP top 10 (2017)



A1:2017-Injection	Injection flaws, such as SQL, NoSQL, 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.
A2:2017-Broken Authentication	Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.
A3:2017-Sensitive Data Exposure	Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.
A4:2017-XML External Entities (XXE)	Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.
A5:2017-Broken Access Control	Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.
A6:2017-Security Misconfiguration	Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched and upgraded in a timely fashion.
A7:2017-Cross-Site Scripting (XSS)	XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
A8:2017-Insecure Deserialization	Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.
A9:2017-Using Components with Known Vulnerabilities	Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.
A10:2017-Insufficient Logging & Monitoring	Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

CWE top 25 (2020)



Rank	ID	Name	Score
[1]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.82
[2]	CWE-787	Out-of-bounds Write	46.17
[3]	CWE-20	Improper Input Validation	33.47
[4]	CWE-125	Out-of-bounds Read	26.50
[5]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	23.73
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	20.69
[7]	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	19.16
[8]	CWE-416	Use After Free	18.87
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	17.29
[10]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	16.44
[11]	CWE-190	Integer Overflow or Wraparound	15.81
[12]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	13.67
[13]	CWE-476	NULL Pointer Dereference	8.35
[14]	CWE-287	Improper Authentication	8.17
[15]	CWE-434	Unrestricted Upload of File with Dangerous Type	7.38
[16]	CWE-732	Incorrect Permission Assignment for Critical Resource	6.95
[17]	CWE-94	Improper Control of Generation of Code ('Code Injection')	6.53
[18]	CWE-522	Insufficiently Protected Credentials	5.49
[19]	CWE-611	Improper Restriction of XML External Entity Reference	5.33
[20]	CWE-798	Use of Hard-coded Credentials	5.19
[21]	CWE-502	Deserialization of Untrusted Data	4.93
[22]	CWE-269	Improper Privilege Management	4.87
[23]	CWE-400	Uncontrolled Resource Consumption	4.14
[24]	CWE-306	Missing Authentication for Critical Function	3.85
[25]	CWE-862	Missing Authorization	3.77

Injection attacks

OWASP definition

Injection flaws, such as SQL, NoSQL, 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.

Key steps to injection attack:

1. Attacker provides malicious input
2. Server accepts input without validating it first
3. Server run's attacker's arbitrary code

Command injection

Command injection attacks are possible when an attacker passes data to a system shell.

- The injection is generally caused when data and code share the same channel.

Command injection: a simple example

- Service that prints the result back from the linux program `whois`

- `whois` is a lookup tool that allows querying information about Domain names, IP address, and ASNs

- Invoked via URL like (a form or JS 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

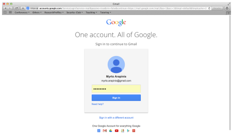
```
<?  echo system('whois'.escapeshellarg($_GET['domain'])); ?>
```

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 *'`

SQL injection

Web applications



Google

One account. All of Google.

Sign in to continue to Gmail

Myrto Arapinis
myrto.arapinis@gmail.com

•••••••••

Need help?

Sign in with a different account

One Google Account for everything Google

HTTP

Client
(HTML, JavaScript)

Server
(PHP)

Database
(SQL)

- 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

Databases

- A database is a system that stores information in an organised way, and produces report about that information based on queries.
- 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 cont'd

SQL injection: a simple example cont'd

Login as admin:

SQL injection: a simple example cont'd

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 cont'd

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 cont’d

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’;");
```

Heartland Payment Systems breach (2008)

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 built-in functions that strip input of dangerous characters:
  PHP provides function `mysql_real_escape_string` to escape special characters.
  - prepends backslashes to special characters: \x00, \n, \r, \, ’, ” and \x1a

  -
    ```
    SELECT * from user_accounts;
    WHERE user = ’admin\’; DROP TABLE user_accounts;’
    ```

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']));
```

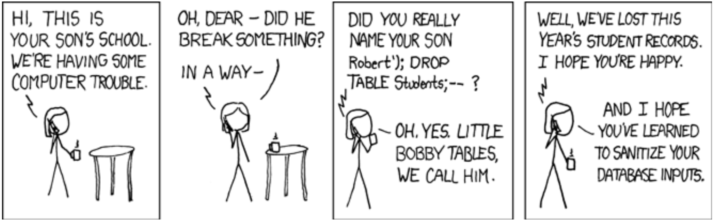
Injection recap

Injection - is generally caused when data and code share the same channel:

- command injection
- sql injection
- ...

Defenses

- include input validation, and input escaping
- 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)


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Injection attacks