(22)

WEB SECURITY:
INJECTION ATTACKS

## 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']); ?>
  }
?>
```

Input

'system' is really powerful!

Use 'execve' instead



## 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 *'); ?>`

data & code / share same channel

rm * is a command tht. deletes all files in current directory

## 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 *'` |



## 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

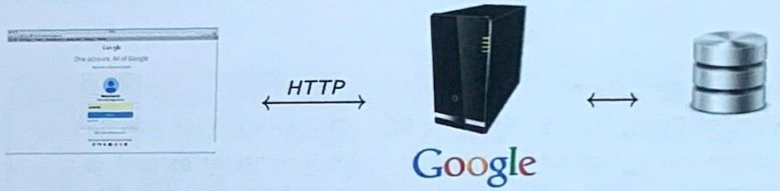
| username | password |
|---|---|
| alice | 01234 |
| bob | 56789 |
| charlie | 43210 |

user_accounts



# (2) SQL INJECTION

## Web applications




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



## 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"); //connect to dbase
$result = pg_query(conn,
                   "SELECT * from user_accounts
                   WHERE username = '".$_GET['user']."'
                   AND password = '".$_GET['pwd']."';");
if(pg_query_num($result) > 0) {
   echo "Success";
   user_control_panel_redirect();
}
```

INPUT SUPPLIED BY USER

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

1) **Login as admin:**

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

can be anything

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

2) **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';");
```

The "--" characters denote a comment in MySQL, which results in the rest of the line being ignored.

## 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)