All living is an obeying.
[…] **The one who cannot obey himself is commanded**.
[…] **commanding is harder than obeying**.
And not only that the commander bears the burden of all obeyers,
and that this burden easily crushed him:
In all commanding it seems to me there is **an eperiment and a risk**;
and always when it commands, the living risks itself in doing so.
Indeed, when it **commands itself**, even then it must pay for its commanding.
It must become the judge and avenger and victim of its own law.

Thus spoke Zarathustra (1885)
Friedrich Nietzsche

Web Cybersecurity – L1

Marco Rocchetto
marco@v-research.it

Mattia Pacchin
mattia@v-research.it

V-ReseΛrch edu

Research & Development for Cybersecurity Engineering

https://v-research.github.io/edu/

Agenda

Summary of the previous lesson & some terrible consequences

Cybersecurity Topic #1 - SQL-injection [theory 30m + lab 2h]
- Intro to Databases and DBMS [15m]
- Introduction to SQLmap with examples [15m]
- WebGoat lesson (A1 - Injection, path traversal excluded) [2h]

Coffee break [10m]

Open Discussion [30m]

Cybersecurity Topic #2 - Path Traversal [lab 30m]
- WebGoat lesson (A1 - Injection -> Path Traversal) [30m]

Summary of the Previous Lesson & Some Terrible Consequences

Slide only for ITS@311Verona 2nd year students in 2020

1. **The VERY IMPORTANT concepts in L0** are:
   1. CIA [https://v-research.github.io/edu/second_year_2020/lesson_0/l0_slide.pdf]
   2. For the LAZY ones:
      1. *Weakness* [1] The definition given by the MITRE in of weakness is: "a type of mistake that, in proper conditions, could contribute to the introduction of vulnerabilitieswithin that product. This term applies to mistakes regardless of whether they occur in implementation, design, orother phases of a product lifecycle."
      2. *Vulnerability* [2] as defined in [30], is a "weakness in an information system, system se-curity procedures, internal controls, or implementation that could be exploited by a threat source".
      3. *Attack*: as defined by the International StandardISO/IEC 27000, is an "attempt to destroy, expose, alter, disable, steal or gain unauthorized access to or makeunauthorized use of an asset"
2. **The death of path traversal**
   1. After the slides, HTTP basic + SQL injection, i.e. will skip path traversal this year :(

[1] FAQ – What is the difference between a software vulnerability and software weakness? Sept.17, 2019. URL: https://cwe.mitre.org/about/faq.html#A.2 (visited on 02/03/2020).

[2] Committee on National Security Systems (CNSS)."Glossary No 4009". In:National Information Assur-ance (IA) Glossary(Apr. 6, 2015). URL: https://rmf.org/wp-content/uploads/2017/10/CNSSI-4009.pdf

DataBase

- A database is an organized collection of structured data
- A database is usually controlled by a database management system (DBMS)
- Data within the most common types of databases in operation today is typically modeled in rows and columns in a series of tables to make processing and data querying efficient

DBMS

- A Database Management System (DBMS) is software designed to store, retrieve, define, and manage data in a database
- DBMS software primarily functions as an interface between the end user and the database, simultaneously managing the data, the database engine, and the database schema in order to facilitate the organization and manipulation of data

SQL

- SQL (Structured Query Language) is a standard language for storing, manipulating and retrieving data in databases.
- SQL can:
    1. execute queries against a database
    2. retrieve data
    3. insert, update and delete records
    4. create new databases, new tables, stored procedures and views
    5. set permissions on tables, procedures and views

SQL

```sql
-- CREAZIONE TABELLE
DROP TABLE Auto;
CREATE TABLE Auto (
    Targa VARCHAR(7),
    Marca VARCHAR(10),
    Cilindrata INT,
    Potenza INT,
    CodF VARCHAR(5),
    CodAss VARCHAR(5),
    PRIMARY KEY(Targa)
);

-- POPOLAZIONE TABELLE
INSERT INTO Auto(Targa, Marca, Cilindrata, Potenza, CodF, CodAss) VALUES ('AB123CC', 'Toyota', 2200, 180, 'PCC21', 'CC123');

-- QUERY
SELECT targa, Marca
FROM Auto
WHERE Cilindrata > 2000 OR Potenza > 120;

SELECT a.Targa, p.Nome
FROM Auto a
INNER JOIN Proprietari p
ON a.CodF = p.CodF
WHERE Cilindrata > 2000 OR Potenza > 120;
```

Detailed Examples

SQL Statement:

```
SELECT * FROM Proprietari
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 3

CodF	Nome	Residenza
PCC21	Gigio	Treviso
PCC22	Gigi	Milano
PCC23	Astrubale	Roma

SQL Statement:

```
SELECT targa, Marca
FROM Auto
WHERE Cilindrata > 2000 OR Potenza > 120;
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 2

Targa	Marca
AB123CC	Toyota
AB125CC	Honda

SQL Statement:

```
SELECT targa, Marca
FROM Auto
WHERE Cilindrata > 2000 OR Potenza > 120;
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 2

Targa	Marca
AB123CC	Toyota
AB125CC	Honda

SQL Statement:

```
SELECT a.Targa, p.Nome
FROM Auto a
INNER JOIN Proprietari p
ON a.CodF = p.CodF
WHERE Cilindrata > 2000 OR Potenza > 120;
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 2

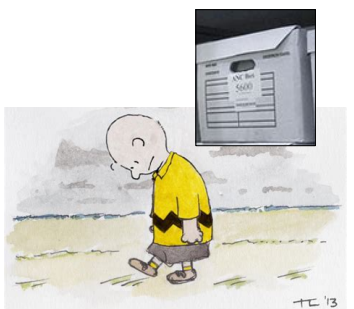
Targa	Nome
AB123CC	Gigio
AB125CC	Gigi

A DBMS from the point of view of a USER

1

2

SELECT * *FROM* users
WHERE
user = marco AND
psw = sesquipedale



etc

A DBMS from the point of view of a USER

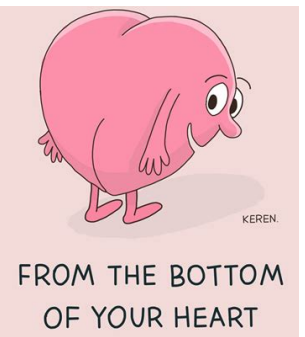
1 I Love you, DBMS
From the bottom of my heart!

?

3 Can you guess what the DBMS would think?

A DBMS from the point of view of a USER

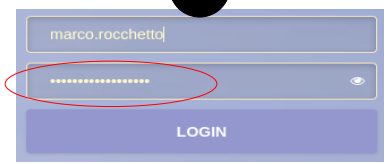
1 I Love you, DBMS
From the bottom of my heart!



3 The DBMS interprets it literally!

A DBMS from the point of view of a USER

1



2

*SELECT * FROM* users
WHERE
user = marco
AND
psw = " "
OR
1=1

A DBMS from the point of view of a HACKER

A = does it exists?
0 = no
1 = yes

A	AND	B
1	1	1
1	0	0
0	0	1
0	0	0

A	OR	B
1	1	1
1	1	0
0	1	1
0	0	0

*SELECT * FROM* users
WHERE
{user = marco
AND
psw = " "}
OR
1=1

user = marco	AND	Psw = " "	OR	1=1
1	0	0	1	1

SQL-injections - Types

- **Authentication bypass attack**: the intruder bypasses an authentication checkthat a web app performs by querying a database.
- **Data extraction attack**: the intruder obtains data from the database that she should not be able to obtain.

In a **Boolean-Based SQLi** (BB), an intruder inserts into an HTTP parameter, which is used by a web app to write a SQL query, one or more valid SQL statements that make the WHERE clause of the SQL query evaluate to true or false. By interacting with the web app and comparing the responses, the intruder can understand whether or not the injection was successful. In this way, an intruder can perform both authentication bypass and data extraction attacks.

Time-Based SQLi (TB) is quite similar to BB: the only difference is that TB does not need the web app to have a Boolean behavior. The intruder appends a timing function to the validity value of a Boolean clause. Thus, after the submission of the query by the web app, the database waits for a predefined amount of time for a tuple as a response to the query; the intruder can then inferwhether the Boolean value of the query was true or false observing a delay in the response. In real case scenarios, a BB is preferable as it is faster than a TB

When error pages are exposed to the Internet, some error messages of thedatabase could be exposed, thus giving an intruder the possibility of exploiting an **Error-Based SQLi** (EB). In this type of injection, the intruder tricks the database into performing operations that result in an error and then he extracts information from the error messages produced by the database. EB is generallyused to perform a data extraction attack by inducing the generation of an error that contains some information stored in the database

SQL-injections - Types

A **UNION Query-Based SQLi** (UQ) is a technique in which an intruder injects a SQL UNION operator to join the original query with a malicious one.The aim is to overwrite the values of the original query and thus, in order to extract information, UQ requires the web app to print the result of the query within the returned HTML page. This behavior allows the intruder to actually extract information from the database by reading it within the web app itself.

Second-Order SQLi (SO) is an injection that has no direct effect when submitted but that is exploited in a second stage of the attack. In some cases, aweb app may correctly handle and store a SQL statement whose value depends onthe user input. Afterwards, another part of the web app that doesn't implement a control against SQLi might use the previously stored SQL statement to executea different query and thus expose the web app to a SQLi. Automated scanners generally fail to detect this type of SQLi

With a **Stacked Queries SQLi** (SQ), an intruder can execute an arbitrary query different from the original one. The semicolon character ";" enables the intruder to concatenate a different SQL query to the original one. By doing so, the intruder can perform data extraction attacks as well as execute whatever operation is allowed by the database. With a SQ, an intruder can perform any of the SQLis described above. Thus, whenever we refer to all the SQLis in our categorization, we exclude SQ as it is already covered by the other ones.

SQL-injections - Protections

A **sanitization function** takes the input provided by the user and removes (i.e., escapes) all the special characters that could be used to perform a SQLi. Sanitization functions are not the best option when dealing with SQLi because they might not be properly implemented or do not consider some cases.

Prepared statements are the best option for preventing SQLis. They are mainly used to execute the same query repeatedly maintaining efficiency. However, due to their inner execution principle (if properly implemented) they are immune to SQLi attacks. The execution of a prepared statement consists mainly in two steps: preparation and execution. In the preparation step, the query is evaluated and compiled, waiting for the parameters for the instantiation. During the execution step, the parameters are submitted to the prepared statement and handled as data and thus they cannot be interpreted as SQL commands.