**Sniffing-**

What is Sniffing?

Sniffing is a process of monitoring and capturing all data packets passing through given network. Sniffers are used by network/system administrator to monitor and troubleshoot network traffic. Attackers use sniffers to capture data packets containing sensitive information such as password, account information etc. Sniffers can be hardware or software installed in the system. By placing a packet sniffer on a network in promiscuous mode, a malicious intruder can capture and analyze all of the network traffic.

There are two types:

Active Sniffing:

Sniffing in the switch is active sniffing. A switch is a point to point network device. The switch regulates the flow of data between its ports by actively monitoring the MAC address on each port, which helps it pass data only to its intended target. In order to capture the traffic between target sniffers has to actively inject traffic into the LAN to enable sniffing of the traffic. This can be done in various ways.

Passive Sniffing:

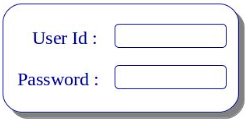
This is the process of sniffing through the hub. Any traffic that is passing through the non-switched or unbridged network segment can be seen by all machines on that segment. Sniffers operate at the data link layer of the network. Any data sent across the LAN is actually sent to each and every machine connected to the LAN. This is called passive since sniffers placed by the attackers passively wait for the data to be sent and capture them.

Introduction to SQL Injection

SQL injection is an attack where the hacker makes use of unvalidated user input to enter arbitrary data or SQL commands; malicious queries are constructed and when executed by the backend database it results in unwanted results. The attacker should have the knowledge of background database and he must make use of different strings to construct malicious queries to post them to the target.

For Example, in user login screen, username and password are the dynamic fields where users enter the data. Depending upon the user’s inputs dynamic queries will be constructed; the usual query will be user id password query,Select \* from users table where username=’Username.txt’ and password=’Password.txt’.

will be



If the input fields are not sanitized properly, then the malicious user can enter some data like this

Username = blah’ or 1=1—

Password = password

Here both username and password are incorrect. But the query which is constructed will be

Select \* from users where username=’blah’ or 1=1—and password=’password’

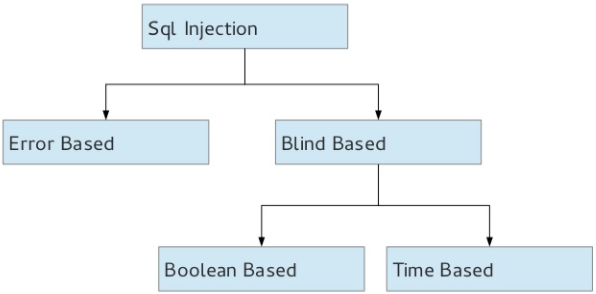
The query will run and the user will be granted access. This is because the first part of the query is

Select \* from users where username=’blah’ or 1=1—

Because – is a comment line in SQL, everything following that will be ignored. The query will only validate between username=’blah’ or 1=1.

Because 1=1 is always true, the user will be granted access.

**Types of SQL Injection-**



**Error based Injection**:

The attacker sends some malicious query to the database which results in errors. The errors should be very generic, otherwise, they may give useful hints to the attacker.

**Comment-Line:** Using comment line to cause the database to ignore a part of a valid query.

E.g. Select \* from stores where product\_id = blah’ or 1=1– (everything after this will be neglected)

**Tautology:** There are a lot of strings which always evaluates to be true, like ‘1’ = ‘1’ ‘a’ = ‘a’, etc., using them in the query to create constantly true conditions.

E.g. Select \* from users where username=’blah’ or ‘a’=’a’ — and password=’pass’

Union Based SQL injection:

Using union command in SQL query to execute additional queries; thereby, modifying/inserting/deleting or dropping the contents of the table.

E.g. Select \* from stores where product\_id=1 union select 1,database(),user(),4#

**Stored procedures:** Creating malicious inputs to execute malicious queries.

**Incorrect queries:** Coming up with logically incorrect queries to see the error messages to get more information about the target database.

Select \* from stores where id=1’

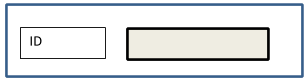
The above query will result in a syntax error and might reveal the backend database type.

**Blind SQL injection**:

This is a type of SQL injection where we don’t have a clue as to whether the web application is vulnerable to injection attack or not.

Types:

**Boolean:** Only correct queries show the result, wrong queries do not return anything. Attackers should try to generate logically correct queries.



If suppose the original query to the database is

Select \* from users where id=’id.txt’

If we give blah’ and 1=1# as input which evaluates to be a right query

Select \* from users where id=’blah’ or 1=1#, we will see the user results.

If we give blah’ and 1=2# as input which is a wrong query then we don’t see any results.

Select \* from users where id=’blah’ or 1=2#

**Time delay:** Depending on some conditions, setting a time delay. If that condition is satisfied, we can observe the time delay; thereby, concluding that the input we gave produced a positive result. This is a time consuming process.

Tools:

SQLMAP, Marathon tool.

Perimeter tools (IDS) Evasion Techniques:

* Use encryption.
* Obfuscate string to avoid pattern matching.
* Use Concatenation to confuse the IDS.
* Use encoding like ASCII encoding, hexadecimal encoding to avoid detection.
* Insert inline comments between query.