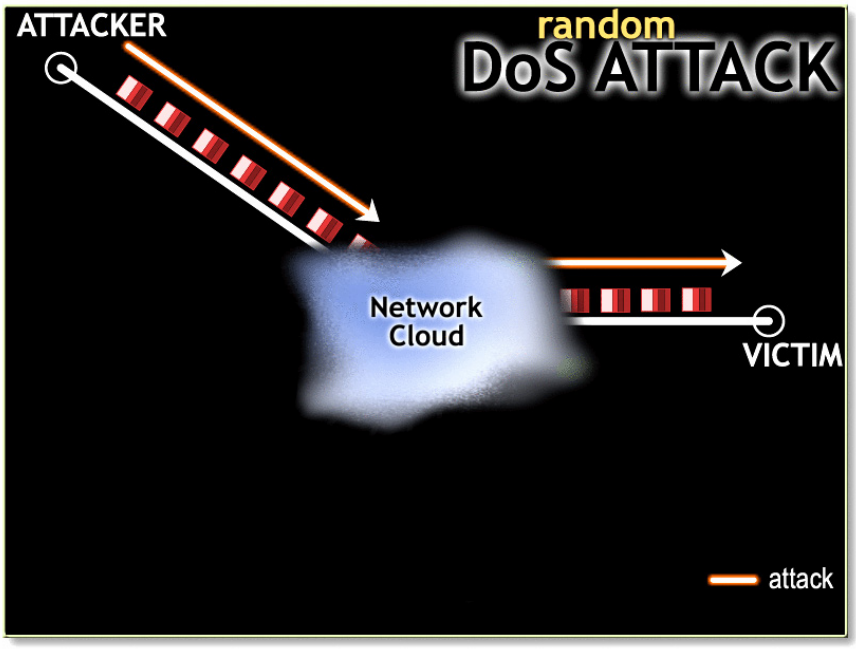
Some basic points about the mechanism of attacking SQL Injection and DDoS

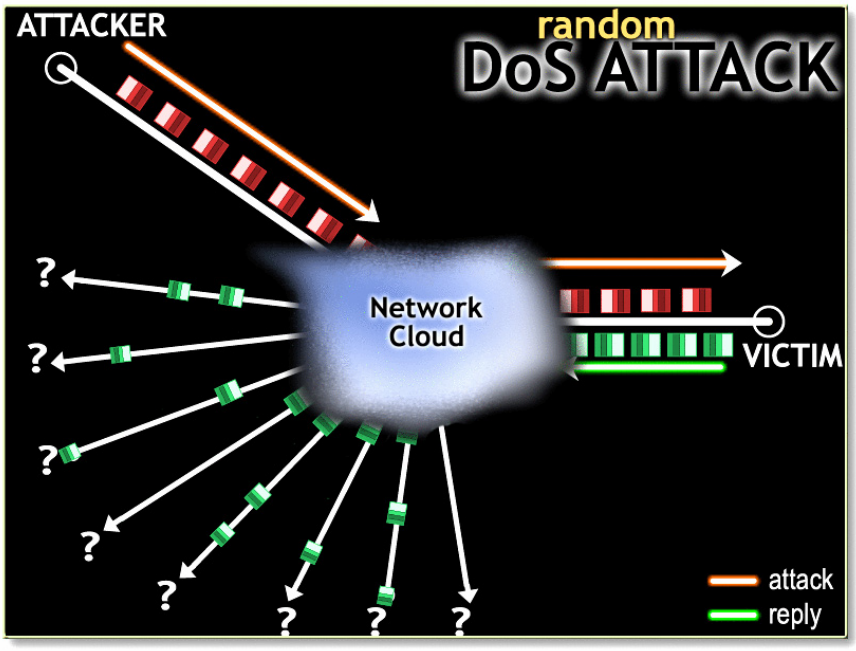
In most of our users, many people have heard of the concept of attacking and hijacking websites with the method of SQL Injection - SQLI and (Distributed) Denial of Service - DDoS.

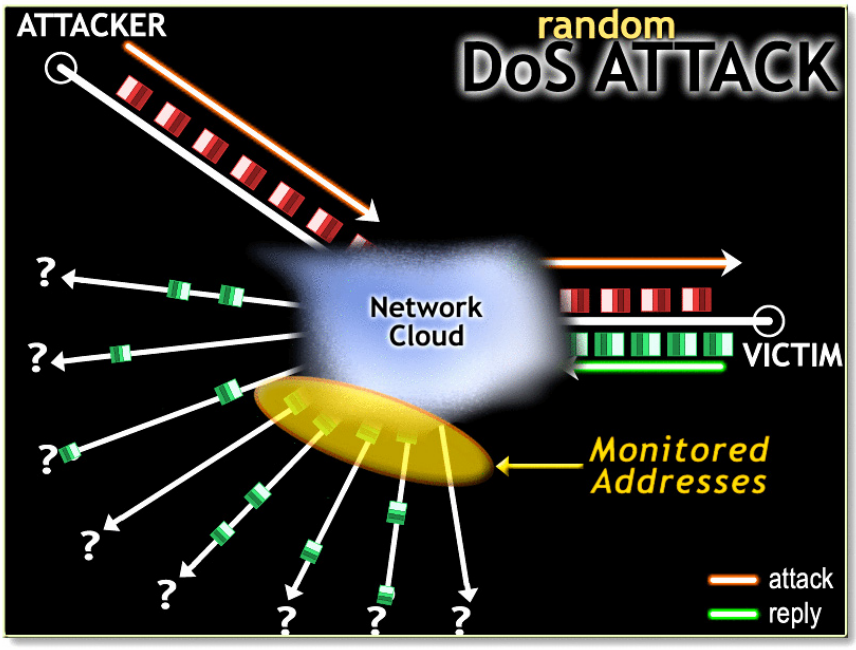
TipsMake.com - In most of our users, many people have heard of the concept of attacking, hijacking websites by the method of SQL Injection - SQLI and (Distributed) Denial of Service - DDoS. So really this process is how hackers use, based on where hackers can identify security holes on the website to attack there . In the article below, TipsMake.com will Introducing a few basic points for people to understand more about the nature of this fairly common attack method.

Denial of Service Attack:

For easier visualization of **DDoS** , please refer to the illustration below:







Nature:

Denial of service (also known as **distributed denial of service** or **DDoS**) is a form of attack that usually occurs when a system (here we are referring to the **web server**) receives too many internal access requests 1 time, so the system will fall into overload, and lead to unstable operation, generating many other problems. The general purpose of this **DDoS** attack method is to make the victim's website hosted on the server exceed normal traffic.

Operation mechanism:

To understand this process, we will take a look at a common example.

For example, there are thousands of people (taking on the attack - the main role of a **hacker**) joining in the process of interfering with any company X business by contacting, calling the support center. Support, take care of their customers. Specific times are determined at 9 am on Tuesday, and in most cases, X's phone system will not be able to withstand millions of calls at 9:00, quite different. compared to weekdays. In addition, customers who really want to call are unable to make calls . and of course, the reputation of that company X will be greatly affected.

In essence, the **DDoS** attack process is similarly conducted. Because there is almost no way to determine the origin of the access requests of hackers or real customers until the server handles them promptly, so the efficiency and success rate of the This method is very high.

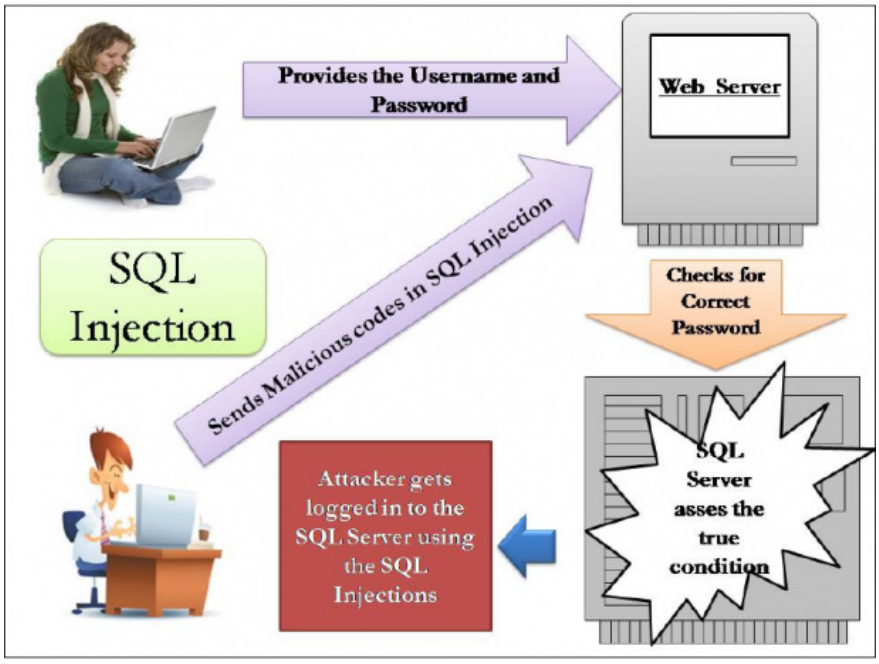
Implementation process:

Due to the extremely 'violent' nature of **DDoS** , hackers have to prepare a lot of computers to perform an attack on the target at the same time. Going back to Company X's example above, the key point of hackers is to simultaneously call the customer care center right at 9am. But from a technical perspective, the process is simple and easy when all the victim's computers are in a zombie state - that is, infected and affected by the hacker virus.

And as we all know, there are many variations of **malware** and **trojans** that have appeared in the past. After successful penetration into the system, they will silently wait for the right time to operate. For example, the command to execute the attack request will be sent to hackers, **Trojans** , and hackers by company X's **webserver** system at 9 am on Tuesday. can create a lot of attack commands from many different locations corresponding to computers that have been infected by **malware** .

The benefit of leveraging malicious computer systems proved to be very effective, as hackers will have many opportunities to hide their whereabouts, on the other hand, the effectiveness and success rate when conducting attacks. public is much more assured.

SQL Injection Attack:



Nature:

SQL injection attack mechanism - **SQLI** is a way of utilizing or exploiting the defects and shortcomings in terms of the technology used to build **websites** , and usually hackers will combine with process holes. database security. If successful in this intrusion, hackers can impersonate the user's official account, access the database and steal personal information. Unlike the approach of **DDoS** , **SQLI is** completely preventable if the administrator is aware of the importance of database security.

Implementation process:

Every time a user proceeds to log in to an online account, they will have to provide information about the Username and Password. In the process of checking and verifying the legality of that account, the corresponding system or web application will run 1 query statement in the following format:

SELECT UserID FROM Users WHERE UserName = 'myuser' AND Password = 'mypass';

Note that the value sections in the **SQL** query **statement** must end with a single quote (') and that's why they appear around the value entered.

The process requires a match between the entered account name (myuser) and password (mypass) with the record storing the information in the **Users** table, and then returns the **UserID** value. If it does not match, no **UserID** value will be returned, and user input information is incorrect.

Next, let's take a look at the account validation statement template that replaces the value of users entering on the web.

SELECT UserID FROM Users WHERE UserName = '[user]' AND Password = '[pass]'

At first glance, many people will think that such account validation process is perfectly reasonable and accurate. However, just a small change in this structure can be a sign of **SQLI** intrusion attack.

For example, **'myuser'–'** is the value entered in the **Username** field and **'wrongpass'** is the password, the alternative query statement pattern will look like this:

SELECT UserID FROM Users WHERE UserName = 'myuser' - 'AND Password =' ??wrongpass'

The key point and identity feature here are two consecutive hyphens **(- -)** , this is the starting point of the notification for the **SQL statement** , whatever is behind these 2 dashes will ignored. And so, the above statement will be verified by the database to:

SELECT UserID FROM Users WHERE UserName = 'myuser'

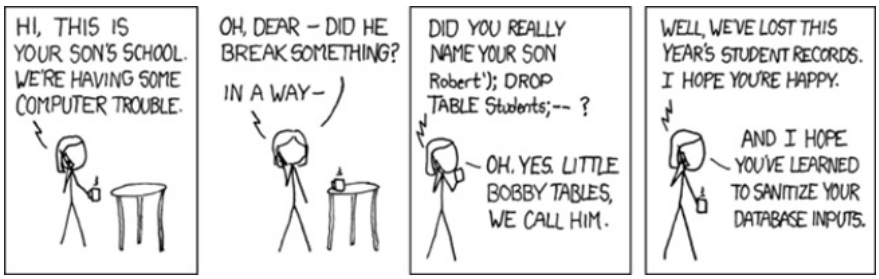
The biggest shortcoming here is the password check, with only 2 dashes as if the hacker could completely pass the test process, and 'blatantly' logged in with the account name '**myuser** ' without having to enter the corresponding password. Next will be the next **SQL injection** attacks.

Extent of damage:

In essence, the origin of **SQL injection** attacks is negligence in the application encryption process, but passwords can still be prevented, but the level of damage is unpredictable and depends on scale of each database. In the process, any web application can communicate with the backend database, it will have to provide login information to a database (this process is different from when the user proceeded to log into the website via the form. login). Depending on the corresponding level of authorization that the web application requires, the account in the database can use any editing permissions, from simple reading and writing to the available tables for to full authority.

Also based on the above example, we see that by entering an account name (eg **"youruser '-", "admin' -"** or whatever) the test can be logged directly into the **database** without know the **password** . Once inside, the system will not be able to know whether the account is a normal account or has the corresponding rights. In essence, the decentralization of the **database** does not provide corresponding and secure exact permissions in this process, because normally a website must at least have read or write access to the corresponding data base.

Suppose that our website system has full access, including deleting data, adding or deleting tables, creating new accounts . and in fact, there are quite a lot of programs to install and use must be decentralized at the highest level, so be careful when assigning permissions.



To further illustrate the level of damage that can be caused in this situation, we will rely on the above figure, and enter the following information in the **Username** field:

"Robert '; DROP TABLE Users; -"

With only a few changes, the query statement becomes:

SELECT UserID FROM Users WHERE UserName = 'Robert'; Users 'DROP TABLE - -' AND Password = 'wrongpass'

Note that commas in **MySQL** are used to end the old clause and start a new clause.

The above command will be made into the database:

SELECT UserID FROM Users WHERE UserName = 'Robert'

Users DROP TABLE

As simple as that, we have successfully implemented a small **SQLI** attack phase with the main purpose of deleting the entire **Users** data table.

How to prevent SQL Injection:

As we mentioned in the previous section of the article, **SQL Injection is** completely preventable. And one of the rules that should not be ignored is not to trust absolutely any information that users enter (similar to the situation we simulated above).

And the **SQLI** model can be easily prevented by a step called Sanitization - understandably the filtering of input - input information from users. Very simply, this process will handle any parentheses that appear in the **SQL** query **statement** .

For example, if you want to find the **'O'neil'** record in the database, it is not possible to use the above simulation method because the 'after' sign will cause the syntax to be interrupted halfway. Instead, we can separate this special character from the database with a sign. And so, the **'O'neal'** record will turn into **'O'neil'** .

To better understand this process, let's go back to the example above:

myuser '- / wrongpass:

SELECT UserID FROM Users WHERE UserName = 'myuser' - 'AND Password =' ??wrongpass'

Because the comma behind myuser is secure, the database will proceed to search for the **UserName** information of **"myuser '-"** . Besides, dashes **(- -)** are included with the data string, they will become part of the search information string and compiled by the corresponding **SQL** syntax.

Robert '; DROP TABLE Users; - / wrongpass:

SELECT UserID FROM Users WHERE UserName = 'Robert'; Users 'DROP TABLE - -' AND Password = 'wrongpass'

Only by separating parentheses behind **Robert** , commas and dashes include within the UserName search string, and so the database will search all records by information: **"Robert '; DROP Users - - "** instead of deleting the same data table.

Good luck!

You finished reading the article "**Some basic points about the mechanism of attacking SQL Injection and DDoS**" edited by the **TipsMake** team. We hope this article has provided you with many useful tech tips and tricks. You can search for similar articles on tips and guides. Thank you for reading and for following us regularly.
