

Chapter 7

Web and Database Security

Overview

Security in web applications is the most important concern when it comes to processing transactions in the web. One of the major issues is the security and privacy of data and information transferred, stored and processed through at real time. These days, many online transactions between client and server are executed at the cloud data centers, where such sensitive data run on virtual resources. Like Several other systems, web-based systems, Cloud Web applications are vulnerable and prone to various types of web Injection attacks which result from transferring untrusted content from web to the server side so a secure communication should be satisfied to prevent web security threats. This chapter will introduce the types of attacks that target web applications. In addition, several examples on many attack scenarios are introduced.

Knowledge Section

In the knowledge section, we will cover the types of web and database attacks and many other scenarios on those attacks.

Web Applications

Those days the big trend is to offer software services on the web. Many services such as online banking, shopping, government, bill payment, tax prep, customer relationship management. In general, application code split between client and server.

Client code executes Java script. Server code includes, PHP, Ruby, Java, or ASP. Security is rarely the main concern when it comes to web applications and vulnerabilities such as poorly written scripts, inadequate input validation, and inadequate protection of sensitive data are the main reasons for web attacks.

The Two Sides of Web Security

1. Web browser: The software that is used to browse website and it can be attacked by any web site it visits. Attacks on Web browser may results in Malware installation (keyloggers, Botnets, Document theft from corporate network and Loss of private data)
2. Web application code: this code runs at web site, e.g. Banks, e-merchants, blogs and it can be written in PHP, ASP, JSP, and Ruby. There are many potential bugs that leads to stolen and defaced sites, such as
 - (a) Cross Site Scripting(XSS): Malicious code injected into a trusted context (e.g., malicious data presented by a trusted website interpreted as code by the user's browser)
 - (b) Cross Site Request Forgery(XSRF): bad website forces the user's browser to send a request to a good website
 - (c) SQL injection: Malicious data sent to a website is interpreted as code in a query to the website's back-end database

In the following section, we will focus on the types of those attacks

Web Threat Models

There are several threat models to initiate attacks. This can happen at three levels, web-attacks, network attacks and malware attacks.

- Web attacker: Usually the objective here is to control the target website, obtain SSL/TLS certificate for the targeted website, or force users to visit the targeted website.
- Network attacker: in this form, the attacks can be categorized as active or passive. Example of passive attacks is the wireless Eavesdropper. Example on active attacks is the DNS poisoning
- Malware attacker: In this type of attacks the Attacker escapes browser sandbox

Cross Site Scripting

XSS can be initialized using different techniques, for instance, the attacker may inject malicious code into a benign website by loading it onto a valid server as part of a client review or a web-based email. Alternatively, the code may be injected into a URL and sent to user as an email. When the user taps the URL, the content will be transmitted to the benign sever and then returned as part of a request of user credentials.

In Table 7.1a, a phishing email asks the user to click on a URL. The URL is not human-readable, but it can be translated into a readable form after mapping the hexadecimal characters, as shown in Table 7.1b. Then, the JavaScript code embedded into the search query will be executed upon visiting the target website, which will inject the HTML code (fetched from www.very-bad-site.com) into the code the user’s browser would normally render.

In the cross site scripting attack, malicious markup and script is entered in the web pages that are viewed by other users. If proper care is not taken to filter this malicious piece of markup, the script gets stored in the system and also rendered on web pages.

Another example on XSS is given in the Figs. 7.1 and 7.2. As noticed in Fig. 7.1, the user can enter any text in the textbox and the textarea, including HTML markup

Table 7.1 (a) The URL sent in a Phishing email. (b) Translating URL into human readable form

<pre>http://www.well-known-financial-institution.com/?q=%3Cscript%3Edocument.write%28%22%3Ciframe+src%3D%27http%3A%2F%2Fwww.very-bad-site.com%27+FRAMEBORDER%3D%270%27+WIDTH%3D%27800%27+HEIGHT%3D%27640%27+scrolling%3D%27auto%27%3E%3C%2Fiframe%3E%22%29%3C%2Fscript%3E&...=&...</pre>	<pre>http://www.well-known-financial-institution.com/?q=<script>document.write("<iframe src='http://www.very-bad-site.com' FRAMEBORDER='0' WIDTH='800' HEIGHT='640' scrolling='auto'></iframe>")</script>&...=&..."></pre>
--	---

Your Name :

Your comment :

Fig. 7.1 Textbox and text area in a web form

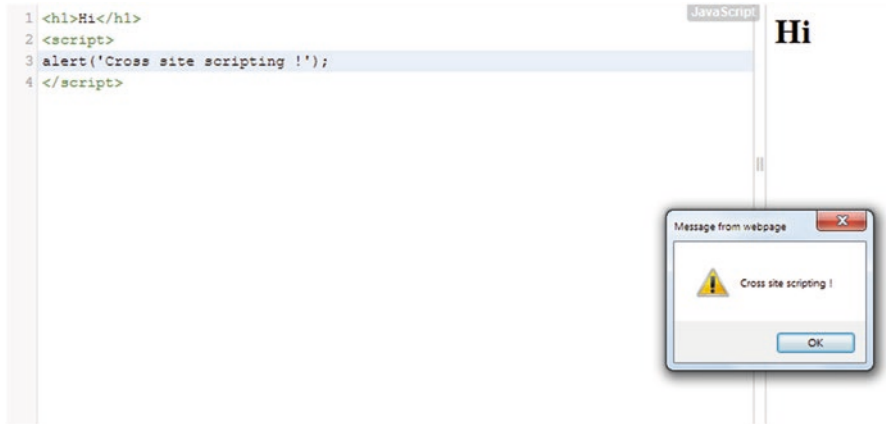


Fig. 7.2 Textbox and text area in a web form

Fig. 7.3 Cookies associated with a specific webpage

```

HTTP/1.1 303 See Other
Location: www.facebook.com
Set-Cookie: user=me@yegor256.com; Secure; HttpOnly

```

Fig. 7.4 Sever response when the cookie header is correct

```

HTTP/1.1 303 See Other
Location: www.facebook.com
Set-Cookie: user=me@yegor256.com

```

tags and script fragments! Once the form is submitted the posted data is saved in the database. The form is submitted to the SaveData action method. The SaveData () saves the data in a SQL Server database table named Comments.

Now assume that a use enters the following text in the comments textarea in Fig. 7.2:

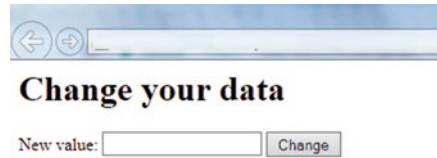
When such a user posts the above content it gets saved in the database. Later when this saved content is rendered on a web page it executes the script. There are yet other types of XSS attack that is associated with cookie-based authentication, based on a browser's ability to expose all cookies associated with a web page to JavaScript executed inside it (see Fig. 7.3).

An attacker may inject some malicious JavaScript code into the page; this will happen only if your entire HTML rendering is done wrong, and this code will gain access to the cookie. Then, the code will send the cookie somewhere else so the attacker can collect it. To prevent this, there is another flag to prevent this type of attacks. The presence of this flag will tell the browser that this particular cookie can be transferred back to the server only through HTTP requests. JavaScript won't have access to it. The server matches the provided information with its records and decides what to do (Fig. 7.4). If the information is invalid, it returns the same login

```
HTTP/1.1 303 See Other
Location: www.facebook.com
Set-Cookie: user=b1ccafd92c568515100f5c4d104671003cfa39
```

Fig. 7.5 Cookie encryption to prevent XSS

Fig. 7.6 A web page on which is vulnerable to CSRF



page, asking you to enter it all again. If the information is valid, the server returns something like this:

If the server trusts any browser request with a user email in the Cookie header, anyone would be able to send my email from another place and get access to my account. The first step to prevent this is to encrypt the email with a secret encryption key, known only to the server (Fig. 7.5). Nobody except the server itself will be able to encrypt it the same way the server needs to decrypt it. The response would look like this, using an example of encryption by XOR cipher with bamboo as a secret key:

XSS has several impacts

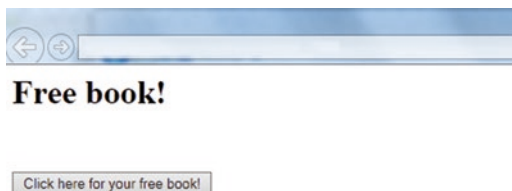
1. Several types of attacks including Denial-of-service attacks, crash users browser, pop-up-flooding, redirection
2. Access to Users' machine using ActiveX objects to control machine and uploading local data to attacker's machine
3. Spoil public image of company, Load main frame content from other locations, and redirect to dialer download

Cross Site Request Forgery

CSRF is yet another type of injection attacks that can be initiated as part of phishing campaigns. The attacker sends emails to victims to lure them into visiting a web page that is under attacker control (Blatz 2007, Nagar and Suman 2016). The attacker hides several executable elements in his page (e.g., Java scripts blocks) which will make a request to the target application. This automatically appends session token to the request when the victim is logged in to the application at that time. The application will automatically perform whatever action the attacker requested.

The Target Page To execute this attack, the victim will access a specific page that looks like the page shown on Fig. 7.6

Fig. 7.7 A web page which is vulnerable to CSRF



The markup to create the page shown in Fig. 7.6 looks like the following:

```
<h1>Change your data</h1>
<form method='post'>
New value: <input type='password' name='newpassword' />
<input type='submit' value='Change'>
</form>
```

This renders to page like this, hosted on a domain http://www.mydomain.com/change_your_data.html

When a logged in user hits the submit button, the browser knows it has to post the new password to greatdomainname.com. By convention, it will send the session ID along with the post request. When the user hits the submit button, the browser will send the new password to mydomain.com. In addition, the session ID will be send with post request.

The attacker will design a page such as the one shown in Fig. 7.7.

```
<h1>Free book!</h1>
<form method="post" action=http://www.mydomainname.com/change_data.html>
  <input type="hidden" name="newpassword" value="h4ck3d!" /><br />
<br >
<input type="submit" value="Click here for your free book!" />
</form>
```

Notice that the attacker has instructed the form to submit to another domain then it is hosted on. The data in the attacker form will be directed to the attacker page. The user now logged in mydomain.com, and when he browses the attacker page. When the attacker clicks the button “click here” for your free book, this composes a request with a valid session ID, but the attacker has chosen his own password by using a hidden input field with the same name.

Database Security

Database is a collection of interrelated data and a set of programs to access the data. Database provides a convenient and efficient processing of data. A database management system is usually used to read, update and delete database components.

Database security refers to the collective measures used to protect and secure a database or database management software from illegitimate use and malicious threats and attacks. Database security measures aim at protecting data from unauthorized disclosure, denial of service attacks, and unauthorized modification. There are several security measures to protect databases including:

- Security Policy
- Access control models
- Integrity protection
- Privacy problems
- Fault tolerance and recovery
- Auditing and intrusion detection

Access Control

Access control ensures that all direct accesses to objects are authorized. It protects against accidental and malicious threats by regulating the read, write and execution of data and programs. There are several access control mechanisms to protect databases:

- Grant and Revoke
- Security through Views
- Stored Procedures
- Query modification
- Multilevel Security

Grant and Revoke

```
GRANT <privilege> ON <relation>
To <user>
[WITH GRANT OPTION]
```

- GRANT SELECT * ON *Registration* TO Matthews
- GRANT SELECT *, UPDATE(CLASS_NAME) ON *REGISTRATION* TO FARKAS
- GRANT SELECT(NAME) ON *Student* TO Brown

GRANT command applies to base relations as well as views.

Security through Views

This can be done by Assigning rights to access predefined views

```
CREATE VIEW Outstanding-Student
AS SELECT NAME, COURSE, GRADE
```

```
FROM Student
WHERE GRADE > B
```

The problem when using view for security is that views are difficult to maintain

Stored Procedures

In this type of countermeasures, we Assign rights to execute compiled programs using the following command

```
GRANT RUN ON <program> TO <user>
```

However, the problem with this kind of countermeasures is that programs may access resources for which the user who runs the program does not have permission

Query Modification

Query modification is done according to the privilege identified to each user. For example if the following privileges are given GRANT SELECT (NAME) ON *Student* TO Blue WHERE COURSE="CSCE 590" and Blue write the following query:

- SELECT *
FROM *Student*
Then the Modified query becomes
- SELECT NAME
FROM *Student*
WHERE COURSE="CSCE 590"

Multilevel Security

Multilevel security is to present users at different security level, different versions of the database. The main problem of this approach is that different versions of the database need to be kept consistent and coherent without downward signaling.

Privacy Problems in Databases

Statistical database is a database which provides statistics on subsets of records. Statistics may be performed to compute SUM, MEAN, MEDIAN, COUNT, MAX AND MIN of records. Given those statistics, there are several types of data compromises in the database as follows:

Name	Age	Has diabetes
Alice	25	Y
Bob	34	Y
Frank	33	Y
Ivy	42	N
Jim	12	Y

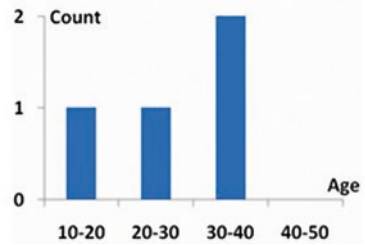


Fig. 7.8 How to compromise data in statistical database

	Total Income		Mr. White's Income
	Before the Move	After the Move	
Raw	\$50,000,000	\$49,000,000	\$1,000,000
Privatized	\$49,082,500	\$48,734,148	\$348,352

Fig. 7.9 How to compromise data in statistical database

- Exact compromise – a user is able to determine the exact value of a sensitive attribute of an individual
- Partial compromise – a user is able to obtain an estimator for a sensitive attribute with a bounded variance
- Positive compromise – determine an attribute has a particular value
- Negative compromise – determine an attribute does not have a particular value
- Relative compromise – determine the ranking of some confidential values

As an example Fig. 7.8 shows how the data is compromised in statistical databases, assume we have a database of medical records from a specific hospital, where ‘Y’ or ‘N’ denotes whether a person has diabetes or not. If the hospital intends to directly release a statistical report on the age distribution of diabetic patients in the database (see the histogram below), the privacies of individuals may be leaked. For example, if someone (often termed as an adversary) happens to know Alice is the only person who is between 20 and 30 years old in the hospital, he can confirm that Alice has diabetes.

The sources of this problem is due to adding or deleting records as shown in Fig. 7.9

Suppose that Mr. White is in a specific income database, let’s assume the total income in his original neighborhood is \$50 million. After he leaves, this Figure drops to \$49 million. Therefore, one can infer that his true income is \$1 million. To keep his income private, we have to ensure the query response is noisy enough to ‘hide’ this information

There are several methods to avoid data compromise, including:

- Query restriction
- Data perturbation/anonymization
- Output perturbation

Table 7.2 A database table with NoC as a private attribute

Name	Position	Age	NoC
George	Instructor	66	5
Heidi	Assistant professor	20	6
Holly	Associate professor	37	2
Leonard	Instructor	70	5
Massey	Instructor	33	3

Table 7.3 Query answered/Not answered based on QSOC

Query	Can be answered based on? (Yes, No)	Query result
Sum_NoC (position = Instructor)	Yes	13
Count (age < 60 & position = Instructor)	Yes	1
Sum_NoC (age > 60 & position = Instructor)	No	10

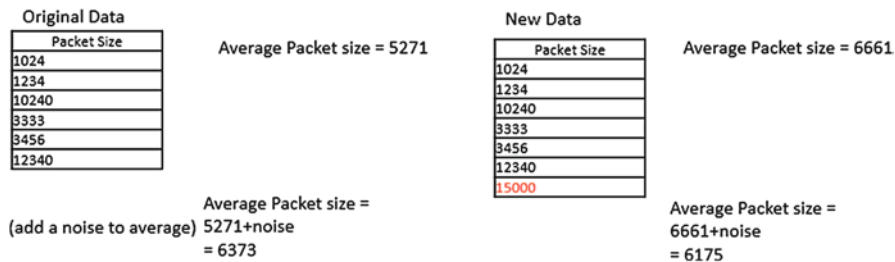


Fig. 7.10 Output perturbation example

Query restriction Based on Query Size A prevention mechanism that answers queries only when the size of the intersection of the query set and each previous query set is smaller than some parameter r . Consider Table 7.2 with $r = 2$.

Assume that NoC (number of children) is a private attribute. Based on **Query-set-overlap control (QSOC)** and **inference attacks** on statistical databases, if the queries in the Table 7.3 are to be executed, the first two queries can be answered without any privacy concerns. The last query cannot be answered since it will reveal the Number of Massey’s Children.

Output Perturbation In this approach, the noise added to the output. For example in Fig. 7.10, without noise if the attacker knows the average packet size before the new packet is added, it is easy to Figure out the packet’s size from the new average. With noise: One cannot infer whether the new packet is there.

Data Perturbation/Anonymization In this approach, the data itself is perturbed. For example, in the aggregation approach Fig. 7.11, the raw (individual) data is grouped into small aggregates before publication. The average value of the group

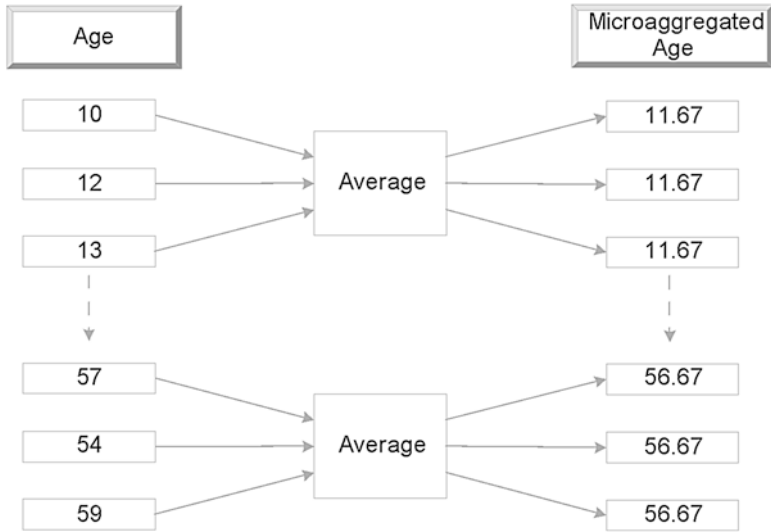


Fig. 7.11 Output perturbation example

replaces each value of the individual. Data with the most similarities are grouped together to maintain data accuracy. This approach helps to prevent disclosure of individual data.

SQL Injection Attacks

What is a SQL Injection Attack? Many web applications take user input from a form and this is often a user input that is used literally in the construction of a SQL query submitted to a database. For example: `SELECT product data FROM table WHERE productname = 'user input product name'`. A SQL injection attack involves placing SQL statements in the user input as shown in Figure. “Many, many sites have lost customer data in this way. SQL Injection attacks are often automated and many website owners may be blissfully unaware that their data could actively be at risk. These attacks can be detected and businesses should be taking basic and blanket steps to block attempted SQL Injection, as well as the other types of attacks we frequently see. Yahoo Voices was hacked in July 2012. The attack acquired 453,000 user email addresses and passwords. The perpetrators claimed to have used union-based SQL injection to break in. In addition, LinkedIn.com leaked 6.5 million user credentials in June. A class action lawsuit alleges that the attack was accomplished with SQL injection. SQL injection was documented as a security threat in 1998, but new incidents still occur every month. Making honest mistakes, developers fail to defend against this means of attack, and the security of online data is at risk for all of us because of it (Fig. 7.12).

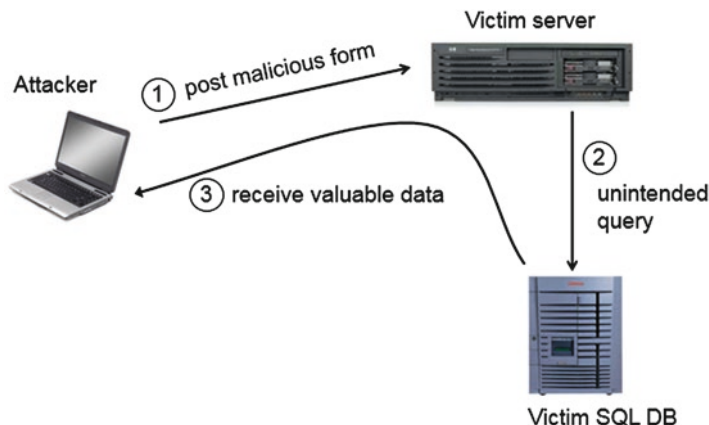


Fig. 7.12 Initiating SQL injection attack

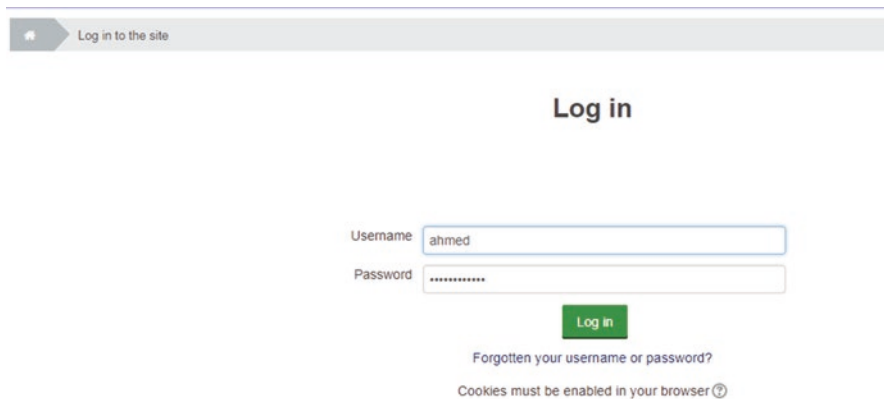


Fig. 7.13 Typical login prompt

This is Input validation vulnerability as shown in Figs. 7.13, 7.14, 7.15, and 7.16. Unsensitized user input in SQL query to back-end database changes the meaning of the query.

Using SQL Injection to Steal Data

SQL Injection is a web based attack used by hackers to steal sensitive information from organizations through web applications. It is one of the most common application layer attacks used today. We will focus on three operators that are used to execute SQL injection attacks

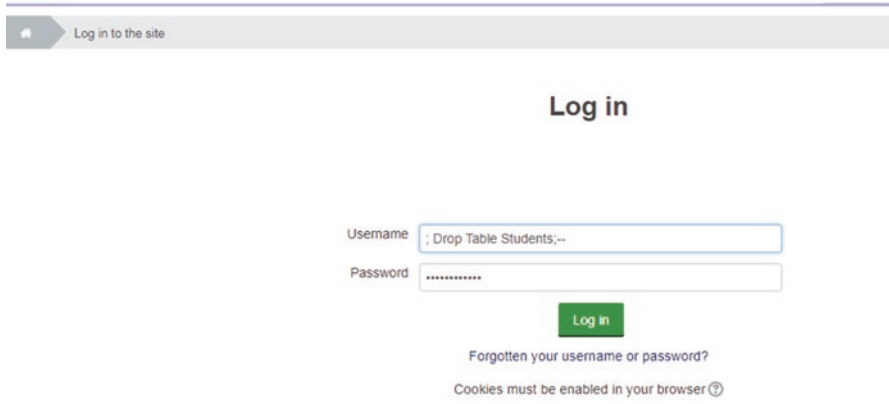


Fig. 7.14 Malicious user input

Fig. 7.15 Back end query formulation

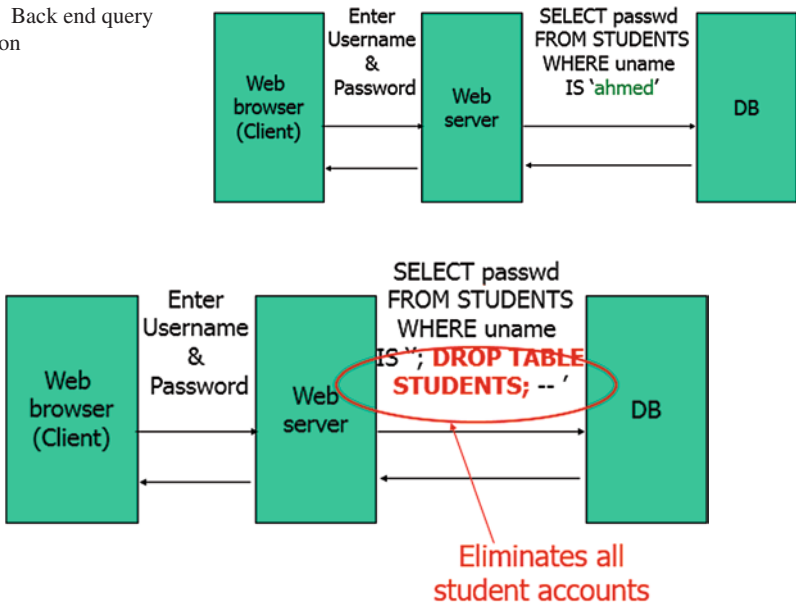


Fig. 7.16 Unsanitized user input in SQL query execution

A. Steal Data (Example) using OR operator

- User gives username ' OR 1=1 –
- Web server executes query
SELECT * FROM UserTable WHERE
- username=' OR 1=1 -- ...);
- Everything after -- is ignored!

- Now all records match the query
- This returns the entire database

B. Steal Data (Example) using LIKE operator

- To authenticate logins, server runs the following SQL command against the user database:
SELECT * from UserT WHERE user='name' AND pwd='passwd'
- User enters ' OR WHERE pwd LIKE '% as both name and passwd
- Server executes
SELECT * from UserT WHERE user=" OR WHERE pwd LIKE '%'
AND pwd=" OR WHERE pwd LIKE '%'
- Logs in with the credentials of the first person in the database (typically, administrator!)

C. Another Malicious User Input to Drop Tables

The attacker might use malicious database command to drop the table as follows:
Query'; DROP TABLE prodinfo; --

This Results in the following SQL: SELECT prodinfo FROM prodtable WHERE prodname = 'blah'; DROP TABLE prodinfo; --'

Notice that that last comment (--) consumes the final quote which causes the entire database to be deleted. The success of this attack depends on knowledge of table name. This is sometimes exposed to the user in debug code called during a database error. To overcome this type of attacks there is a need to use non-obvious table names, and never expose them to user.

Mitigation of SQL Injection Attacks

The main mitigation of SQL injection attacks is to sanitize the input. Overall, web application developers often simply do not think about “surprise inputs”, but security people do (including the bad guys). Sanitizing user inputs to insure that they do not contain dangerous codes, whether to the SQL server or to HTML itself. One’s first idea is to strip out “bad stuff”, such as quotes or semicolons or escapes

Ex. An email address only contains “A B C D E F G H I J K L M N O P Q R S T U
V W Y Z 0 1 2 3 4 5 6 7 8 9 @ . -_”

Skills Section

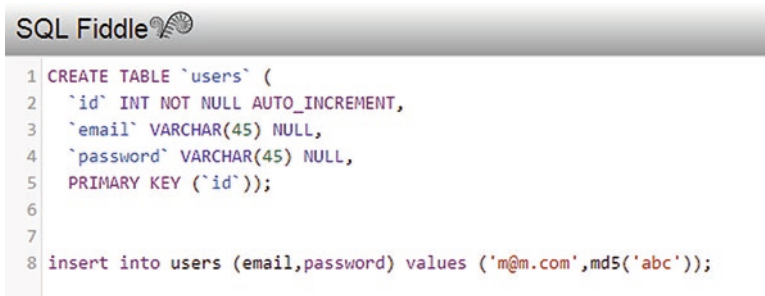
In this section we will introduce a complete example on SQL injection. The following web application accepts the email address, and password then submits them to a [PHP](#) file named index.php.

Consider a simple PHP web application with a login form. The code for the HTML form is shown below.

In this section we will focus on SQL injection Attacks and how they are initiated

```
>form action='index.php' method="post">  
>input type="email" name="email" required="required"/>  
>input type="password" name="password"/>  
>input type="checkbox" name="remember_me" value="Remember me"/>  
>input type="submit" value="Submit"/>  
</form>
```

We will illustrate SQL injection attack using SQLfiddle. The following schema will be created as part of this example:



```
SQL Fiddle  
1 CREATE TABLE `users` (  
2   `id` INT NOT NULL AUTO_INCREMENT,  
3   `email` VARCHAR(45) NULL,  
4   `password` VARCHAR(45) NULL,  
5   PRIMARY KEY (`id`));  
6  
7  
8 insert into users (email,password) values ('m@m.com',md5('abc'));
```


Once the schema is created we will insert a DB record as shown below

id	email	password
1	m@m.com	900150983cd24fb0d6963f7d28e17f72

Let's suppose an attacker use the following input in the email address field.
xxx@xxx.xxx' OR 1 = 1 LIMIT 1 -- ']

The generated dynamic statement will be as follows.

SELECT * FROM users WHERE email = 'xxx@xxx.xxx' OR 1 = 1 LIMIT 1 -- ']
AND password = md5('1234');



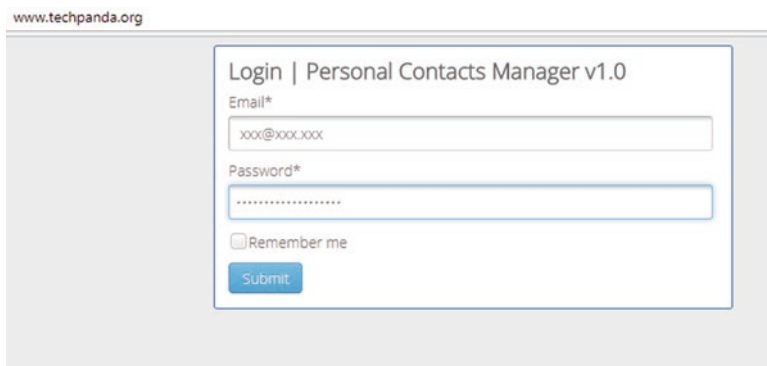
```
1 SELECT * FROM users  
2 WHERE email = 'xxx@xxx.xxx'  
3 OR 1 = 1 LIMIT 1 -- ' ] AND password = md5('1234');
```

The statement xxx@xxx.xxx ends with a single quote which completes the string quote. The OR 1 = 1 LIMIT 1 is a condition that will always be true and limits the returned results to only one record. The statement -- ' AND ... is a SQL comment that eliminates the password part. The query will retrieve the following record.

id	email	password
1	m@m.com	900150983cd24fb0d6963f7d28e17f72

SQL Inject a Web Application

A simple web application at <http://www.techpanda.org/> that is vulnerable to SQL will be used to demonstrate injection attacks for demonstration purposes only.



The generated SQL statement will be as follows:

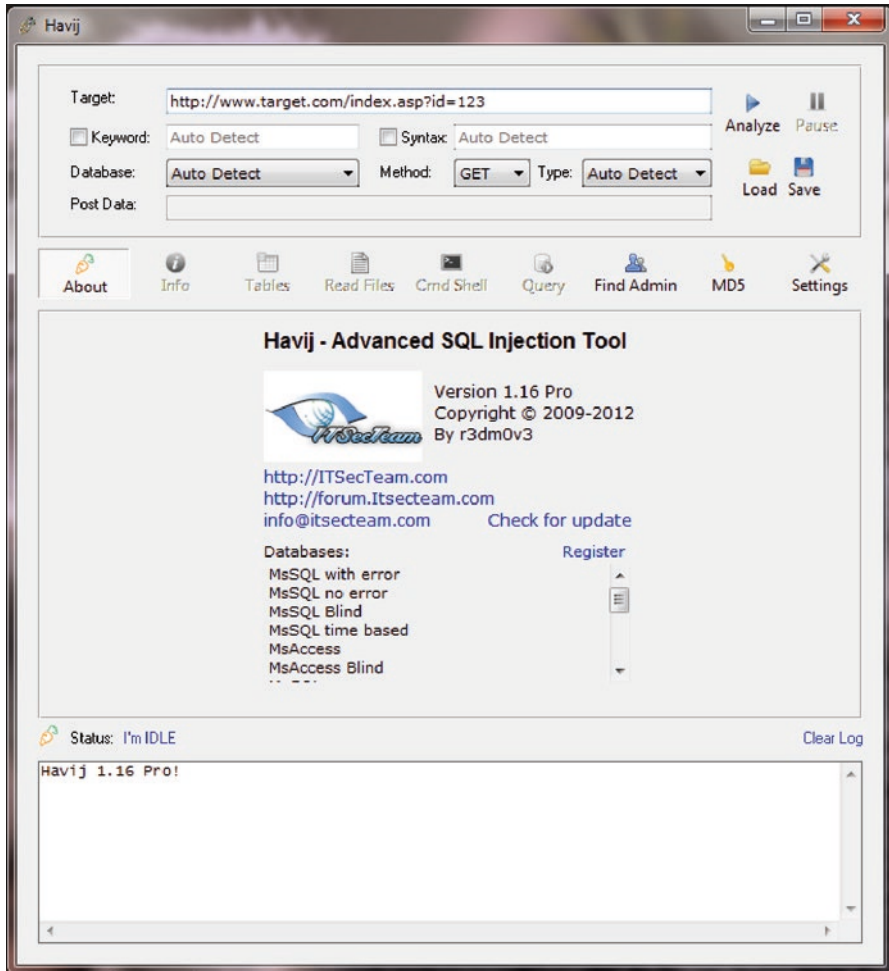
```
SELECT * FROM users WHERE email = 'xxx@xxx.xxx' AND password = md5('xxx') OR 1 = 1 -- ]');
```

Other Forms of SQL Injection Attacks

SQL Injections can do more harm than just by passing the login algorithms. Some of the attacks include

- Deleting data
- Updating data
- Inserting data

In this practical scenario, we are going to use Havij Advanced SQL Injection program to scan a website for vulnerabilities.



Applications Section

Applications for Encryption Techniques for Network Security

- You are expected to implement specific web application and then test it against web attacks such as SQL injection and XSS and CSRF

Applications of Website Vulnerability Scanners

- You are expected to search for scanners that can discover website vulnerabilities.
- Install and run those scanners on particular websites

Applications for Detecting Network Attacks Using Traffic Analysis

- You are expected to select one of the known vulnerable websites.
 - Check if the has any SQL injection vulnerabilities
 - Classify those vulnerabilities into insert update and delete vulnerabilities

Questions

- In the scope of web security, describe the differences between XSS and CSRF
- Following is a script that has been written for a login page in a specific web application `txtUserId = getRequestString("UserId");`
- `txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;`
- Assume that there is an SQL vulnerability in this application. Write down a query that return ALL rows from the "Users" table
- If you have the following part of a web application `uName = getRequestString("username");`
`uPass = getRequestString("userpassword");`
`sql = 'SELECT * FROM Users WHERE Name ="' + uName + '" AND Pass ="' + uPass + '"'`

User Name:

Password:

- Describe two vulnerabilities that can lead to XSS and SQL injection attacks
- Describe two techniques to mitigate SQL injection attacks
- What is the main purpose of query modification when an unauthorized query is written.

References

Application for testing and sharing. SQL queries. <http://sqlfiddle.com>

Cross site Scripting attacks. <http://deadlytechnology.com/web-development/xss/>

SQL Injection. https://www.w3schools.com/sql/sql_injection.asp

SQL Injection practical Example. <https://www.guru99.com/learn-sql-injection-with-practical-example.html>

SQL Injection practical Example. <http://www.techpanda.org/>