Web Application Security

Rajendra Kachhwaha
rajendra1983@gmail.com

October 26, 2015

Outline

Attacking Data Source:

1. Basics
2. Bypassing a Login
3. Injecting into SQL (Insert, Update, Union)

Basics:

1. Nearly all applications rely on a data store to manage data that is processed within the application.
2. In many cases this data drives the core application logic, holding user accounts, permissions, application configuration settings, and more.
3. Data stores have evolved to become significantly more than passive containers for data.
4. Most hold data in a structured format, accessed using a predefined query format or language, and contain internal logic to help manage that data.

Basics:

5. Applications use a common privilege level for all types of access to the data store and when processing data belonging to different application users.
6. If an attacker can interfere with the application's interaction with the data store, to make it retrieve or modify different data, he can usually bypass any controls over data access that are imposed at the application layer.
7. SQL injection is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution.

Bypassing a Login:

1. The process by which an application accesses a data store usually is the same, regardless of whether that access was triggered by the actions of an unprivileged user or an application administrator.
2. The web application functions as a access control to the data store, constructing queries to retrieve, add, or modify data in the data store based on the users account and type.
3. A successful injection attack that modifies a query can bypass the application's discretionary access controls and gain unauthorized access.

Bypassing a Login:Example:

1. Many applications that implement a forms-based login function use a database to store user credentials and perform a simple SQL query to validate each login attempt.
   SELECT * FROM users WHERE username = 'marcus' and password = 'secret'
2. If an attacker knows that the username of the application administrator is admin, he can log in as that user by supplying any password and the following username:admin'- -
3. This causes the application to perform the following query:
   SELECT * FROM users WHERE username = 'admin'- -' AND password = 'foo'
   The comment sequence (- -) causes the remainder of the query to be ignored,

Bypassing a Login:Example:

1. Generally, attacker does not know any username??

Bypassing a Login:Example:

1. Generally, attacker does not know any username??
2. An attacker can log in as the first user in the database by supplying the username:'OR 1=1- -
   SELECT * FROM users WHERE username = ' 'OR 1=1- -'
   AND password = 'foo'

Injecting into SQL:

1. Almost every web application employs a database to store the various kinds of information it needs to operate.
2. For ex., a web application deployed by an online retailer might use a database to store the following information:
   - User accounts, credentials, & personal information
   - Descriptions and prices of goods for sale
   - Orders, account statements, & payment details
   - The privileges of each user within the application
3. The means of accessing information within the database is Structured Query Language (SQL).
4. SQL can be used to read, update, add, and delete information held within the database.
5. Web applications commonly construct SQL statements that incorporate user-supplied data. If this is done in an unsafe way, the application may be vulnerable to SQL injection.

Injecting into SQL:

Exploiting a Basic Vulnerability

1. SELECT author, title, year FROM books WHERE publisher = 'Wiley' and published=1

Injecting into SQL:

Exploiting a Basic Vulnerability

1. SELECT author, title, year FROM books WHERE publisher = 'Wiley' and published=1
2. SELECT author,title,year FROM books WHERE publisher = 'O'Reilly' and published=1
   Unclosed quotation mark before the character string '
3. SELECT author, title, year FROM books WHERE publisher = 'Wiley' OR 1=1- -' and published=1

Injecting into SQL:Insert Statement:

1. For ex., an application may allow users to self-register, specifying their own username and password, and may then insert the details into the users table with the following statement:
   INSERT INTO users (username, password, ID, privs) VALUES ('daf', 'secret', 2248, 1)
2. For example, injecting into the username field, the attacker can supply the following:
   foo', 'bar', 9999, 0)- -

Injecting into SQL:Insert Statement:

1. For ex., an application may allow users to self-register, specifying their own username and password, and may then insert the details into the users table with the following statement:
   INSERT INTO users (username, password, ID, privs) VALUES ('daf', 'secret', 2248, 1)
2. For example, injecting into the username field, the attacker can supply the following:
   foo', 'bar', 9999, 0)- -
3. This creates an account with an ID of 9999 and privs of 0. Assuming that the privs field is used to determine account privileges, this may enable the attacker to create an administrative user.

Injecting into SQL:Update Statement:

1. For ex., when a user changes her password, the application might perform the following query:
   UPDATE users SET password = 'newsecret' WHERE user = 'marcus' and password = 'secret'
2. An attacker can bypass the existing password check and update the password of the admin user by entering the following username:admin'- -
3. Attack vector to resets the value of every user's password:

Injecting into SQL:Update Statement:

1. For ex., when a user changes her password, the application might perform the following query:
   UPDATE users SET password = 'newsecret' WHERE user = 'marcus' and password = 'secret'
2. An attacker can bypass the existing password check and update the password of the admin user by entering the following username:admin'- -
3. Attack vector to resets the value of every user's password:
   UPDATE users SET password='newsecret' WHERE user = 'admin' or 1=1

Injecting into SQL:Union operator:

1. The UNION operator is used in SQL to combine the results of two or more SELECT statements into a single result set.
2. For ex., Searching for books published by Wiley causes the application to perform the following query:
   SELECT author,title, year FROM books WHERE publisher = 'Wiley'
3. For example, entering the search term:
   Wiley' UNION SELECT username, password, uid FROM users- -
4. Causes the application to perform the following query:
   SELECT author, title, year FROM books WHERE publisher = 'Wiley' UNION SELECT username, password, uid FROM users- -'