5.6. SQL Injection

SQL injection and buffer overflows are hacking techniques used to exploit weaknesses in applications. When programs are written, some parameters used in the creation of the application code can leave weaknesses in the program. SQL injection and buffer overflows are covered in the same chapter because they both are methods used to attack application and are generally caused by programming flaws. Generally, the purpose of SQL injection is to convince the application to run SQL code that was not intended.

SQL injection is a hacking method used to attack SQL databases, whereas buffer overflows can exist in many different types of applications. SQL injection and buffer overflows are similar exploits in that they're both usually delivered via a user input field. The input field is where a user may enter a username and password on a website, add data to a URL, or perform a search for a keyword in another application. The SQL injection vulnerability is caused primarily by unverified or unsanitized user input via these fields.

Both SQL Server injection and buffer overflow vulnerabilities are caused by the same issue: invalid parameters that are not verified by the application. If programmers don't take the time to validate the variables a user can enter into a variable field, the results can be serious and unpredictable. Sophisticated hackers can exploit this vulnerability, causing an execution fault and shutdown of the system or application, or a command shell to be executed for the hacker.

SQL injection and buffer overflow countermeasures are designed to utilize secure programming methods. By changing the variables used by the application code, weaknesses in applications can be greatly minimized. This chapter will detail how to perform a SQL injection and a buffer overflow attack and explore the best countermeasures to prevent the attack.

As a CEH(Certified Ethical Hacker), it's important for you to be able to define SQL injection and understand the steps a hacker takes to conduct a SQL injection attack. In addition, you should know SQL Server vulnerabilities, as well as countermeasures to SQL injection attacks.

SQL injection occurs when an application processes user-provided data to create a SQL statement without first validating the input. The user input is then submitted to a web

application database server for execution. When successfully exploited, SQL injection can give an attacker access to database content or allow the hacker to remotely execute system commands. In the worst-case scenario, the hacker can take control of the server that is hosting the database. This exploit can give a hacker access to a remote shell into the server file system. The impact of a SQL injection attacks depends on where the vulnerability is in the code, how easy it is to exploit the vulnerability, and what level of access the application has to the database. Theoretically, SQL injection can occur in any type of application, but it is most commonly associated with web applications because they are most often attacked. During a web application SQL injection attack, malicious code is inserted into a web form field or the website's code to make a system execute a command shell or other arbitrary commands. Just as a legitimate user enters queries and additions to the SQL database via a web form, the hacker can insert commands to the SQL Server through the same web form field. For example, an arbitrary command from a hacker might open a command prompt or display a table from the database. A database table may contain personal information such as credit card numbers, social security numbers, or passwords. SQL Servers are very common database servers and used by many organizations to store confidential data. This makes a SQL Server a high-value target and therefore a system that is very attractive to hackers.

Finding a SQL Injection Vulnerability

Before launching a SQL injection attack, the hacker determines whether the configuration of the database and related tables and variables is vulnerable. The steps to determine the SQL Server's vulnerability are as follows:

1. Using your web browser, search for a website that uses a login page or other database input or query fields (such as an "I forgot my password" form). Look for web pages that display the POST or GET HTML commands by checking the site's source code.
2. Test the SQL Server using single quotes (''). Doing so indicates whether the user input variable is sanitized or interpreted literally by the server. If the server responds with an error message that says *use 'a'='a'* (or something similar), then it's most likely susceptible to a SQL injection attack.

3. Use the SELECT command to retrieve data from the database or the INSERT command to add information to the database.

Here are some examples of variable field text you can use on a web form to test for SQL vulnerabilities:

Blah' or 1=1- Login:blah' or 1=1-

Password::blah' or 1=1--

http://search/index.asp?id=blah' or 1=1--

These commands and similar variations may allow a user to bypass a login depending on the structure of the database. When entered in a form field, the commands may return many rows in a table or even an entire database table because the SQL Server is interpreting the terms literally. The double dashes near the end of the command tell SQL to ignore the rest of the command as a comment.

Here are some examples of how to use SQL commands to take control: To get a directory listing, type the following in a form field:

Blah';exec master..xp_cmdshell "dir c:*.* /s >c:\directory.txt"-To create a file, type the following in a form field:

Blah';exec master..xp_cmdshell "echo hacker-was-here > c:\hacker.txt"-To ping an IP address, type the following in a form field:

Blah';exec master..xp_cmdshell "ping 192.168.1.1"--

The Purpose of SQL Injection

SQL injection attacks are used by hackers to achieve certain results. Some SQL exploits will produce valuable user data stored in the database, and some are just precursors to other attacks. The following are the most common purposes of a SQL injection attack:

Identifying SQL Injection Vulnerability The purpose is to probe a web application to discover which parameters and user input fields are vulnerable to SQL injection.

Performing Database Finger-Printing The purpose is to discover the type and version of database that a web application is using and "fingerprint" the database. Knowing the type and

version of the database used by a web application allows an attacker to craft database specific attacks.

Determining Database Schema To correctly extract data from a database, the attacker often needs to know database schema information, such as table names, column names, and column data types. This information can be used in a follow-on attack.

Extracting Data These types of attacks employ techniques that will extract data values from the database. Depending on the type of web application, this information could be sensitive and highly desirable to the attacker.

Adding or Modifying Data The purpose is to add or change information in a database.

Performing Denial of Service These attacks are performed to shut down access to a web application, thus denying service to other users. Attacks involving locking or dropping database tables also fall under this category.

Evading Detection This category refers to certain attack techniques that are employed to avoid auditing and detection.

Bypassing Authentication The purpose is to allow the attacker to bypass database and application authentication mechanisms. Bypassing such mechanisms could allow the attacker to assume the rights and privileges associated with another application user.

Executing Remote Commands These types of attacks attempt to execute arbitrary commands on the database. These commands can be stored procedures or functions available to database users.

Performing Privilege Escalation These attacks take advantage of implementation errors or logical flaws in the database in order to escalate the privileges of the attacker.

SQL Injection Using Dynamic Strings

Most SQL applications do a specific, predictable job. Many functions of a SQL database receive static user input where the only variable is the user input fields. Such statements do not change from execution to execution. They are commonly called static SQL statements.

However, some programs must build and process a variety of SQL statements at runtime. In many cases the full text of the statement is unknown until application execution. Such statements can, and probably will, change from execution to execution. So, they are called dynamic SQL statements.

Dynamic SQL is an enhanced form of SQL that, unlike standard SQL, facilitates the automatic generation and execution of program statements. Dynamic SQL is a term used to mean SQL code that is generated by the web application before it is executed. Dynamic SQL is a flexible and powerful tool for creating SQL strings. It can be helpful when you find it necessary to write code that can adjust to varying databases, conditions, or servers. Dynamic SQL also makes it easier to automate tasks that are repeated many times in a web application.

A hacker can attack a web-based authentication form using SQL injection through the use of dynamic strings. For example, the underlying code for a web authentication form on a web server may look like the following:

SQLCommand = "SELECT Username FROM Users WHERE Username = '"

SQLCommand = SQLComand & strUsername

SQLCommand = SQLComand & "' AND Password = '"

SQLCommand = SQLComand & strPassword

SQLCommand = SQLComand & "'" strAuthCheck = GetQueryResult(SQLQuery)

A hacker can exploit the SQL injection vulnerability by entering a login and password in the web form that uses the following variables:

Username: kimberly

Password: graves' OR ''='

The SQL application would build a command string from this input as follows:

SELECT Username FROM Users

WHERE Username = 'kimberly'

AND Password = ‘graves’ OR ‘’=’’

This is an example of SQL injection: this query will return all rows from the user’s database, regardless of whether kimberly is a real username in the database or graves is a legitimate password. This is due to the OR statement appended to the WHERE clause. The comparison ‘’=’’ will always return a true result, making the overall WHERE clause evaluate to true for all rows in the table. This will enable the hacker to log in with any username and password.

SQL Injection Countermeasures

The cause of SQL injection vulnerabilities is relatively simple and well understood: insufficient validation of user input. To address this problem, defensive coding practices, such as encoding user input and validation, can be used when programming applications. It is a laborious and time-consuming process to check all applications for SQL injection vulnerabilities.

When implementing SQL injection countermeasures, review source code for the following programming weaknesses:

- Single quotes
- Lack of input validation

Buffer Overflows

The first countermeasures for preventing a SQL injection attack are minimizing the privileges of a user’s connection to the database and enforcing strong passwords for SA and Administrator accounts. You should also disable verbose or explanatory error messages so no more information than necessary is sent to the hacker; such information could help them determine whether the SQL Server is vulnerable. Remember that one of the purposes of SQL injection is to gain additional information as to which parameters are susceptible to attack.

Another countermeasure for preventing SQL injection is checking user data input and validating the data prior to sending the input to the application for processing.

Some countermeasures to SQL injection are

- Rejecting known bad input
- Sanitizing and validating the input field

As a CEH, you must be able to identify different types of buffer overflows. You should also know how to detect a buffer overflow vulnerability and understand the steps a hacker may use to perform a stack-based overflow attack. We'll look at these topics, as well as provide an overview of buffer-overflow mutation techniques, in the following sections.

Types of Buffer Overflows and Methods of Detection

Buffer overflows are exploits that hackers use against an operating system or application; like SQL injection attacks, they're usually targeted at user input fields. A buffer overflow exploit causes a system to fail by overloading memory or executing a command shell or arbitrary code on the target system. A buffer overflow vulnerability is caused by a lack of bounds checking or a lack of input-validation sanitization in a variable field (such as on a web form). If the application doesn't check or validate the size or format of a variable before sending it to be stored in memory, an overflow vulnerability exits.

The two types of buffer overflows are stack based and heap based.

> The *stack* and the *heap* are storage locations for user-supplied variables within a running program. Variables are stored in the stack or heap until the program needs them. Stacks are static locations of memory address space, whereas heaps are dynamic memory address spaces that occur while a program is running. A heap-based buffer overflow occurs in the lower part of the memory and overwrites other dynamic variables

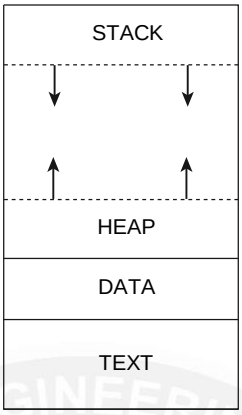

Fig: Stack versus Heap Memory

A call stack, or *stack*, is used to keep track of where in the programming code the execution pointer should return after each portion of the code is executed. A stack-based buffer overflow attack (occurs when the memory assigned to each execution routine is overflowed. As a consequence of both types of buffer overflows, a program can open a shell or command prompt or stop the execution of a program. The next section describes stackbased buffer overflow attacks.

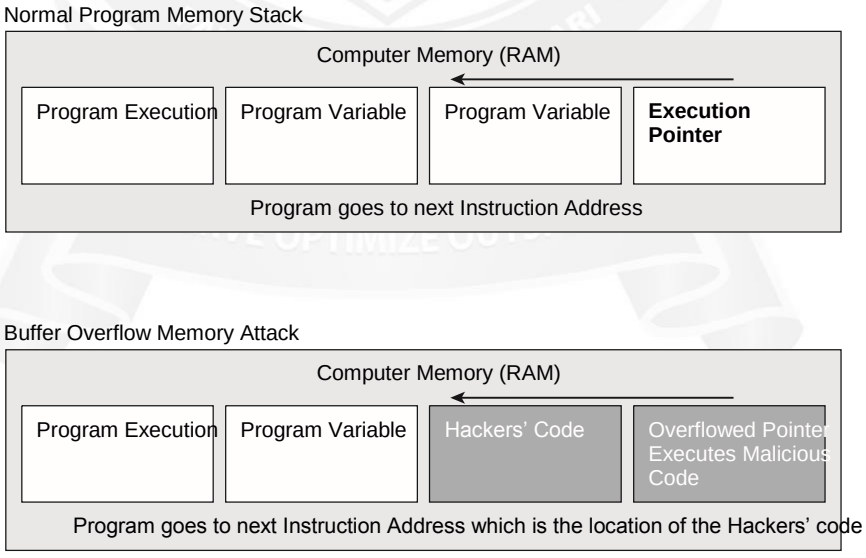

Fig: A stack-based buffer overflow attack

To detect program buffer overflow vulnerabilities that result from poorly written source code, a hacker sends large amounts of data to the application via a form field and sees what the program does as a result.

The following are the steps a hacker uses to execute a stack-based buffer overflow:

1. Enter a variable into the buffer to exhaust the amount of memory in the stack.
2. Enter more data than the buffer has allocated in memory for that variable, which causes the memory to overflow or run into the memory space for the next process. Then, add another variable, and overwrite the return pointer that tells the program where to return to after executing the variable.
3. A program executes this malicious code variable and then uses the return pointer to get back to the next line of executable code. If the hacker successfully overwrites the pointer, the program executes the hacker's code instead of the program code.

Most hackers don't need to be this familiar with the details of buffer overflows. Prewritten exploits can be found on the Internet and are exchanged between hacker groups.