1.1 Buffer Overflow Attack

Buffer: A buffer, in terms of a program in execution, is a region of computer's main memory that has certain boundaries. For example: char buff[10]. Here, 'buff' represents an array of 10 bytes where buff[0] is the left boundary and buff[9] is the right boundary of the buffer.

What is Buffer Overflow?

A buffer is said to be overflow when the data gets written beyond the left or the right boundary of the buffer. i.e. the data gets written to a portion of memory which does not belong to the program variable that references the buffer. For example: char buff[10]; buff[10] = 'a';. Here, the, index 10 was used to store the value 'a'. At this point the buffer overflow happens because data gets written beyond the right boundary of the buffer. Although buffer overflow is a bad programming practice, it can cause your program to crash or produce unexpected results.

1. Consider a scenario where you have allocated 10 bytes on heap memory:

```
char *ptr  = (char*) malloc(10);
```

Now, if you try to do something like this : ptr[10] = 'c';. Then this may lead to crash in most of the cases. The reason is that a pointer is not allowed to access heap memory that does not belong to it.

2. Consider another scenario where you try to fill a buffer (on stack) beyond it's capacity :

```
char buff[10] = {0};
strcpy(buff, "This String Will Overflow the Buffer");
```

The strcpy() function will write the complete string in the array 'buff'. But as the size of 'buff' is less than the size of string so the data will get written beyond the right boundary of array. Now, depending on the compiler you are using, for some compilers this will get unnoticed during compilation and would not crash during execution. So in these kinds of scenarios, buffer overflow corrupts the neighboring memory and if the corrupted memory is being used by the program then it can cause unexpected results.

Buffer Overflow Attacks

When attacker comes to know about a buffer overflow in your program and he/she exploits it as follows:

```
#include <stdio.h>
#include <string.h>
int main(void)
{
   char buff[15];
   int pass = 0;
   printf("\n Enter the password : \n");
   gets(buff);
   if(strcmp(buff, "qwertyuiop"))
   {       printf ("\n Wrong Password \n");          }
   else
   {       printf ("\n Correct Password \n");
            pass = 1;                                 }
   if(pass)
   {      /* Now Give root or admin rights to user*/
      printf ("\n Root privileges given to the user \n");
   }
   return 0;
}
```

Let's the run the program with correct password i.e. 'qwertyuiop' :

Enter the password: qwertyuiop
Correct Password
Root privileges given to the user

Buffer									Integer pass		
q	w	e	r	t	y	u	i	O	p	pass = 0	

This works as expected. The passwords match and root privileges are given. But there is a possibility of buffer overflow in this program. The gets() function does not check the array bounds and can even write string of length greater than the size of the buffer. Here is an example:

Enter the password: hhhhhhhhhhhhhh
Wrong Password
Root privileges given to the user

Buffer									Integer pass		
h	h	h	h	h	h	h	h	h	h	pass = h	h

In the above example, even after entering a wrong password, the program worked as if you gave the correct password.

SQL Injection Attack

SQL injection is a code injection technique in which malicious SQL statements are inserted into an input field of form that are later passed to SQL Server for parsing and execution. The purpose of this attack is to bypass authenticity check and to gain an unauthorized access to a database. The main vulnerability in web applications is: insufficient validation of user input. SQL injection attack exploits this security vulnerability, for example, user input may be incorrectly filtered for **string literal escape characters** embedded in SQL statements.

Techniques for discovering SQL injection vulnerabilities

1. **Testing by inference**-Identify all data entries accepted by the server application.
2. **Database errors**-Analyze the kinds of errors that applications typically display when the back-end database fails to execute a query.
3. **Application Response**-Sometimes the errors are not directly displayed in the browser i.e. applications react differently when they receive an error from the database. In this case we have to analyze the application response.

Example: Following is an example application that contains SQL injection vulnerability.
Front Web page: login.html

```
<html>
<body>
 <form action="http://172.16.4.59:8080/is/login.jsp" method="post">
 <h1 style="text-align:center;">Login Form</h1>
<table width="27%" align="center">
 <tr> <td >UserName:<input type="text" name="username"></td></tr>
 <tr> <td>Password:<input type="password" name="password"></td></tr>
 <tr><td><input type="submit" name="submit" value="LOGIN">  
 <input type="reset" name="reset" value="RESET"/></td></tr>
</table>
</form>
</body>
</html>
```

Back-end code: Jsp page for authentication:login.jsp

```
String login, password, pin, query
username = request.getParameter("username");
password = request.getParameter("password");
Connection conn.createConnection("MyDataBase");
query = "SELECT * FROM login WHERE username='"+username+"'AND pass='"+password+"';
ResultSet result = conn.executeQuery(query);
if (result.next())
// Display User's account information
else
out.println("Login Failed");
```

//

Now, if a user submits username and password as "famt" and "1234", the application dynamically builds and submits the query as SELECT * FROM users WHERE username='famt' AND pass='1234'. The famt's account information is returned and then displayed to the user. If there is no match in the database, an appropriate error message will be displayed.

Performing SQL Injection

Now, suppose an attacker submits " ' or 1=1" for the username input field (the input submitted for the password field is irrelevant). The resulting query is: SELECT * FROM users WHERE username='' or 1=1 -- AND pass=''. The code injected in the condition (OR 1=1) transforms the entire WHERE clause into a tautology. The database uses the condition as the basis for evaluating each row and deciding which ones to return to the application. Because the conditional is a tautology, the query evaluates to true for each row in the table and returns all of them. In our example, the returned set evaluates to a non-null value, which causes the application to conclude that the user authentication was successful. Therefore, the application would show all of the accounts in the set returned by the database.

How the SQL Injection attack works?

Cross Site Scripting

Cross-site Scripting (or XSS) allows an attacker to inject malicious scripts into trusted web sites viewed by other users. This attack is a type of code injection attack. The script written by an attacker is executed on the client-side (in the end user's web browser) rather than on the server-side. The end user's browser has no way to know that the script is not trusted, and will execute the script.

Who is Vulnerable to 'Cross-Site Scripting (XSS)'?

The websites that are allowing end user to supply input and if they do not ensure that all user supplied input is properly validated, before including that input in the output page, then such websites are vulnerable to CSS. Without proper output validation, such input will be treated as active content in the browser.

Actors in an XSS attack

In general, an XSS attack involves three actors: the website, the victim, and the attacker.

- **The website** serves HTML pages to users who request them. The website's database is a database that stores some of the user input included in the website's pages.
- **The victim** is a normal user of the website who requests pages from it using his browser.
- **The attacker** is a user of the website who infects a trusted web page with his malicious scrip.

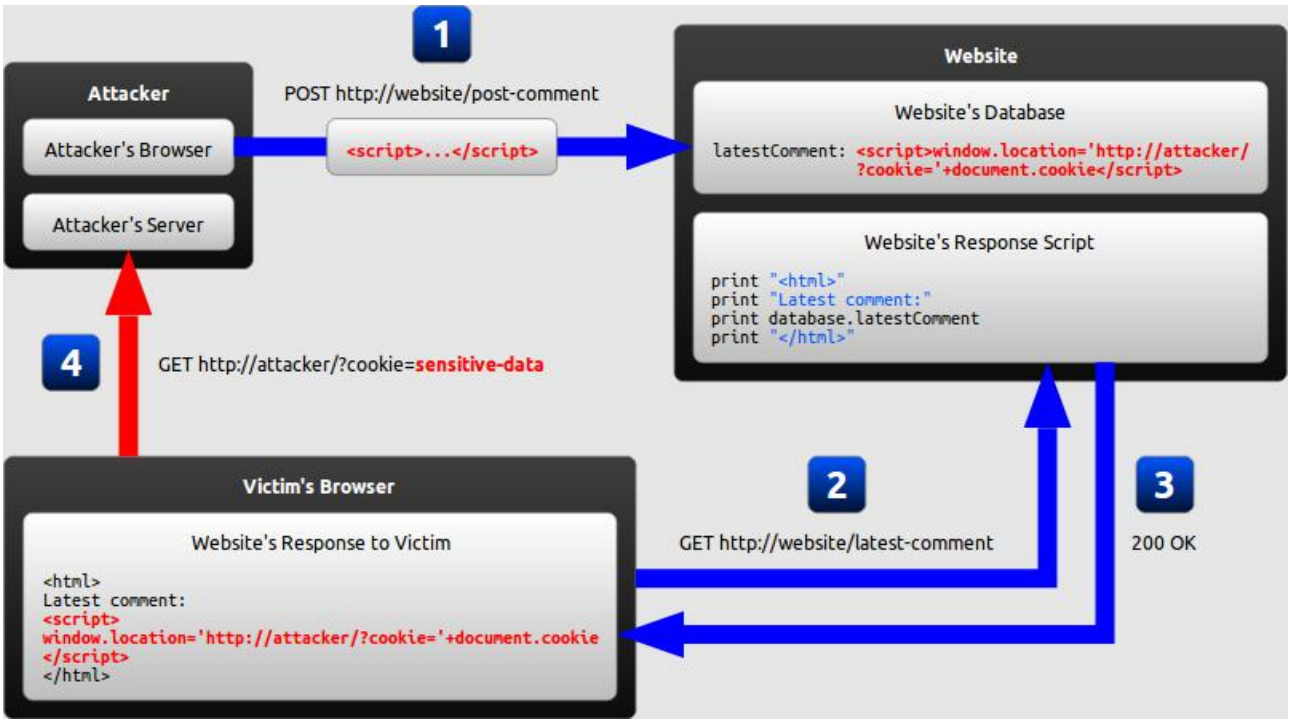
Types of CSS

Non-Persistent XSS

Here, the injected script is not permanent and just appears for the short time the user is viewing the page.

Persistent XSS

In this attack, the aim of an attacker is to somehow store the malicious script on the sites server. Some websites allow user to write contents on their websites that will be permanently stored in server database. For example: Online shopping websites allow users to write reviews about the product. So, an attacker can write contents including malicious script in it and the injected script is permanently stored on the target server database. If the server fail to sanitize the input provided, it results in execution of injected script. When another user views the site, the script will be executed in his browser. If the injected code is cookie stealing code, then it will steal cookie of users who read the post.



2. Malware/Malicious Software

A malware is a piece of software that is designed to do something “malicious/unwanted”. A malware exhausts system resources. Malware can be divided into

1. Malware that can operate independently and those that need a host program.
2. Malware that can replicate themselves and those that cannot replicate themselves.

Types of Malware: Virus, Worm, Trojan, Botnets, Spyware, Keyloggers, Rootkits etc.

2.1 Virus

A virus is a malicious computer program that replicates by attaching itself to another object. Almost all viruses are attached to an executable file, which means the virus may exist on your computer but it actually cannot infect your computer unless you run or open that executable file. It is important to note that a virus cannot be spread without a human action such as running an infected program.

Example: A virus is attached to email attachment. When you download that attachment and open it the attached virus will get spread on your machine.

2.2 Worm

It is a malware program that spreads copies of itself without the need to inject itself in other programs or no need to attach them to a program. Thus, worms are technically not viruses since they don't infect other programs. Rather they replicate themselves and send copies from computer to computer across network connections. Worms are similar to viruses but unlike a virus it has the capability to travel without any human actions.

Example: Suppose worm spreads itself through an email and when you open such email the worm spreads itself by sending same email to the people in your address book.

2.3 Trojan horse

A Trojan horse is computer program that will appear to be useful software, but also contains a hidden and potentially malicious function that performs some unwanted or harmful function (e.g. launches a keylogger). Trojan horses are often installed by a user or administrator, either deliberately or accidentally. Trojan horse programs can be used to accomplish functions indirectly that an unauthorized user could not accomplish directly.

Example: You might download what you think is a new game, but when you run it, it installs a keylogger on your computer, or next time when you start the game, the program emails your saved passwords to another person.

2.4 Bot

Bot Program on an infected machine that is controlled externally to perform malicious attacks on other machines. **Botnets** are collection of infected machines centrally controlled by a server. It can be used to launch distributed denial of service (**DDOS**) attack.

2.5 Logic Bomb

A **logic bomb** is a piece of code intentionally inserted into a software system that will perform an unauthorized action when certain logic conditions are met.

For example, a programmer may insert a piece of code in bank software to transfer some small amount from every account to some specific account.

2.6 Rootkits

A rootkit modifies the operating system to hide its existence and enable continued privileged access to a computer. The term *rootkit* is a concatenation of "root" (the traditional name of the privileged account on Unix operating systems) and the word "kit" (which refers to the software components).

Rootkit installation can be automated, or an attacker can install it once they have obtained root access. Obtaining root access is a result of direct attack on a system i.e., exploiting a known vulnerability or obtaining password.

Detection and Removal: Rootkit detection is difficult because a rootkit may be able to modify the software that is intended to find it. Detection methods include using an alternative and trusted operating system. Removal can be complicated or practically impossible, especially in cases where the rootkit resides in the kernel;

Solution: Reinstallation of the operating system is the only available solution to the problem.