# MODULE-6: System Security

VIT | Vidyalankar Institute of Technology
Accredited A+ by NAAC

**Prepared by Prof. Amit K. Nerurkar**

## Buffer Overflow

A buffer is a temporary area for data storage. When more data (than was originally allocated to be stored) gets placed by a program or system process, the extra data overflows. It causes some of that data to leak out into other buffers, which can corrupt or overwrite whatever data they were holding.

Since buffers are created to contain a defined amount of data, the extra data can overwrite data values in memory addresses adjacent to the destination buffer unless the program includes sufficient bounds checking to flag or discard data when too much is sent to a memory buffer.

Buffer overflows are categorized according to the location of the buffer in the process memory, the two main types being stack-based overflow and heap-based overflow.

The stack is a continuous space in memory used to organize data associated with function calls, including function parameters, function local variables and management information, such as frame and instruction pointers.

The heap is a memory structure used to manage dynamic memory. Programmers often use the heap to allocate memory whose size is not known at compile time, where the amount of memory required is too large to fit on the stack or where the memory is intended to be used across function calls.

Consider this example :

```
#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, "thegeekstuff"))

    {

        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;
}
```

The program above simulates scenario where a program expects a password from user and if the password is correct then it grants root privileges to the user.

Let's the run the program with correct password ie 'thegeekstuff' :

```
$ ./bfrovrflw

Enter the password :
thegeekstuff

Correct Password

Root privileges given to the user
```

This works as expected. The passwords match and root privileges are given.

But do you know that 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 to which the string is written. Now, can you even imagine what can an attacker do with this kind of a loophole?

Here is an example :

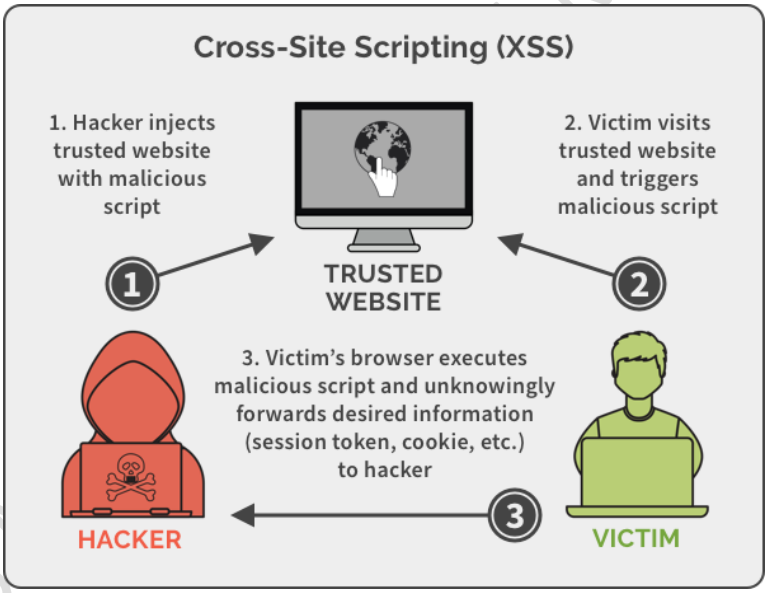
```
$ ./bfrovrflw

Enter the password :

hhhhhhhhhhhhhhhhhhhh

Wrong Password

Root privileges given to the user
```

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

There is a logic behind the output above. What attacker did was, he/she supplied an input of length greater than what buffer can hold and at a particular length of input the buffer overflow so took place that it overwrote the memory of integer 'pass'. So despite of a wrong password, the value of 'pass' became non zero and hence root privileges were granted to an attacker.

## Cross-site scripting

Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. It allows an attacker to circumvent the same origin policy, which is designed to segregate different websites from each other. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.




Cross-site scripting works by manipulating a vulnerable web site so that it returns malicious JavaScript to users. When the malicious code executes inside a victim's browser, the attacker can fully compromise their interaction with the application.

There are three main types of XSS attacks. These are:

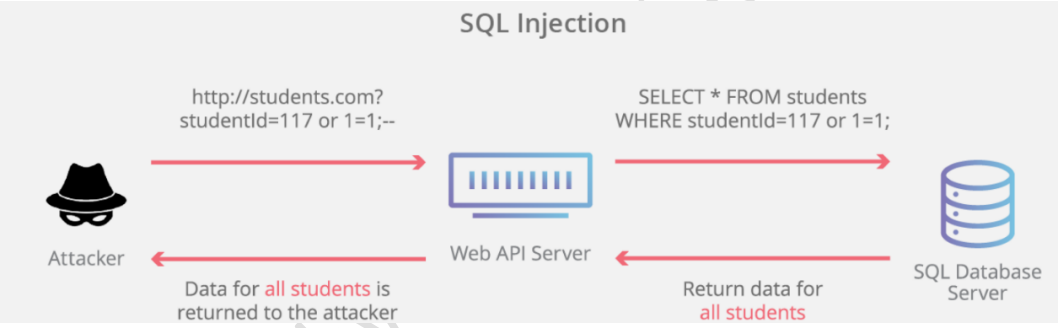
**Reflected XSS,** where the malicious script comes from the current HTTP request.

**Stored XSS**, where the malicious script comes from the website's database.

**DOM-based XSS,** where the vulnerability exists in client-side code rather than server-side code.

## SQL injection,

- SQL injection is a code injection technique that might destroy your database.
- SQL injection is one of the most common web hacking techniques.
- SQL injection is the placement of malicious code in SQL statements, via web page input.

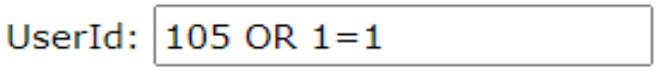



- SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will unknowingly run on your database.
- Look at the following example which creates a SELECT statement by adding a variable (txtUserId) to a select string. The variable is fetched from user input (getRequestString):

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

- Look at the example above again. The original purpose of the code was to create an SQL statement to select a user, with a given user id.

- If there is nothing to prevent a user from entering "wrong" input, the user can enter some "smart" input like this:




Then, the SQL statement will look like this:

```
SELECT * FROM Users WHERE UserId = 105 OR 1=1;
```

- The SQL above is valid and will return ALL rows from the "Users" table, since

OR 1=1 is always TRUE.

- What if the "Users" table contains names and passwords?

Here is an example of a user login on a web site:

Username:
John Doe

Password:
myPass

### Example

```
uName = getRequestString("username");
uPass = getRequestString("userpassword");

sql = 'SELECT * FROM Users WHERE Name ="' + uName + '" AND Pass ="' + uPass + '"'
```

### Result

```
SELECT * FROM Users WHERE Name ="John Doe" AND Pass ="myPass"
```

A hacker might get access to user names and passwords in a database by simply inserting " OR ""=" into the user name or password text box:

User Name:

" or ""="

Password:

" or ""="

The code at the server will create a valid SQL statement like this:

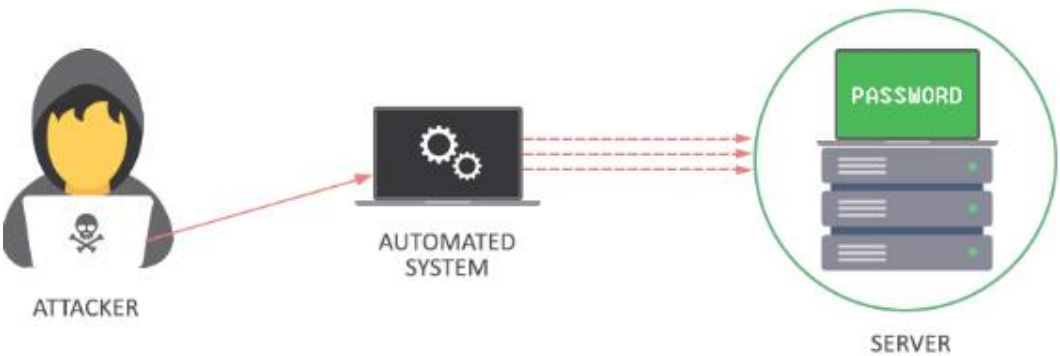
Result

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

## Brute Force Attack

A brute force attack (also known as brute force cracking) is the cyber attack equivalent of trying every key on your key ring, and eventually finding the right one.

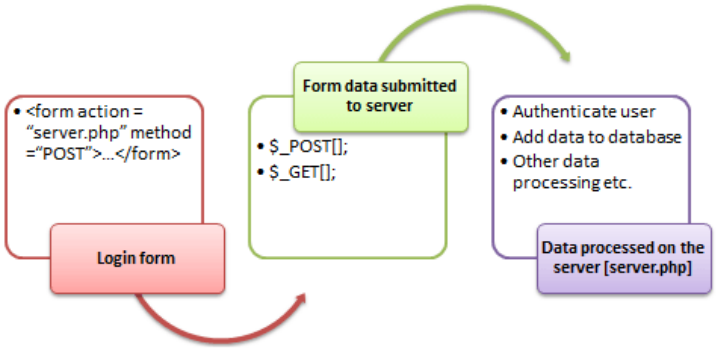
Attackers let a computer do the work – trying different combinations of usernames and passwords, for example – until they find one that works.



## Forms and Scripts

While dealing with the forms, information can be submitted and transferred to same or another page. To send submitted data through form, one can use GET & POST method to do that in PHP.

As in GET method key values are passed in the Url while in POST, the information transfers in a hidden manner.




**FORM SUBMISSION GET METHOD**

```
<form action="registration_form.php" method="GET">
    First name: <input type="text" name="firstname"><br>
    Last name: <input type="text" name="lastname">
    <br>
    <input type="hidden" name="form_submitted" value="1"/>
    <input type="submit" value="Submit">
</form>
```

**SUBMISSION URL SHOWS FORM VALUES**

localhost/tuttis/registration_form.php?firstname=Smith&lastname=Jones&form_submitted=1

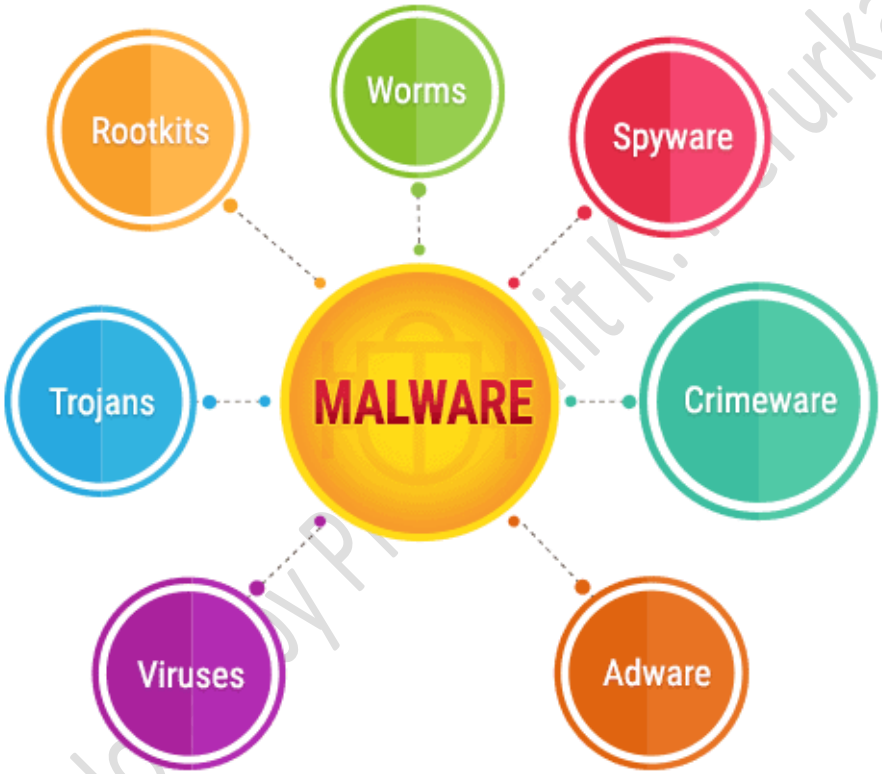
***FORM SUBMISSION POST METHOD***

```
<form action="registration_form.php" method="POST">
    First name: <input type="text" name="firstname"><br>
    Last name: <input type="text" name="lastname">
    <br>
    <input type="hidden" name="form_submitted" value="1"/>
    <input type="submit" value="Submit">
</form>
```

***Submission URL does not show form values***

localhost/tuttis/registration_form.php

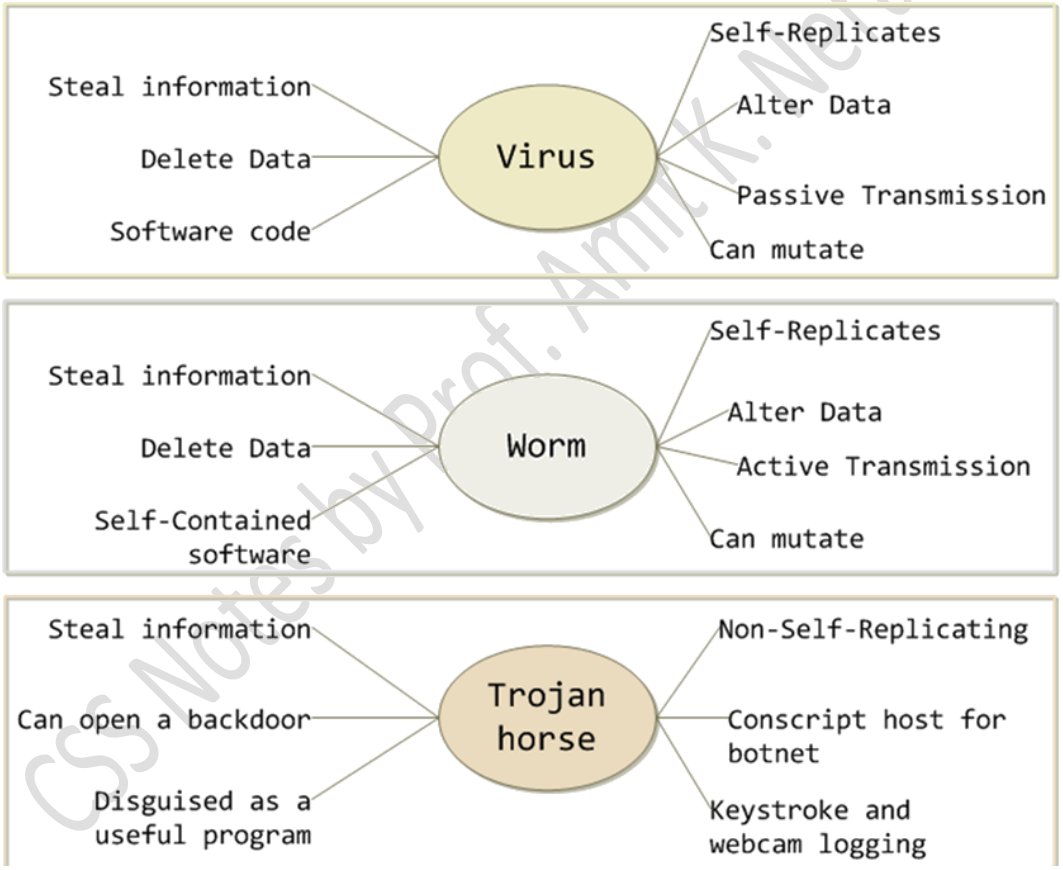
**Malware :** Malware is short for malicious software. Malware is the name given to any type of software that could harm a computer system, interfere with and gather a user's data, or make the computer perform actions without the owner's knowledge or permission.



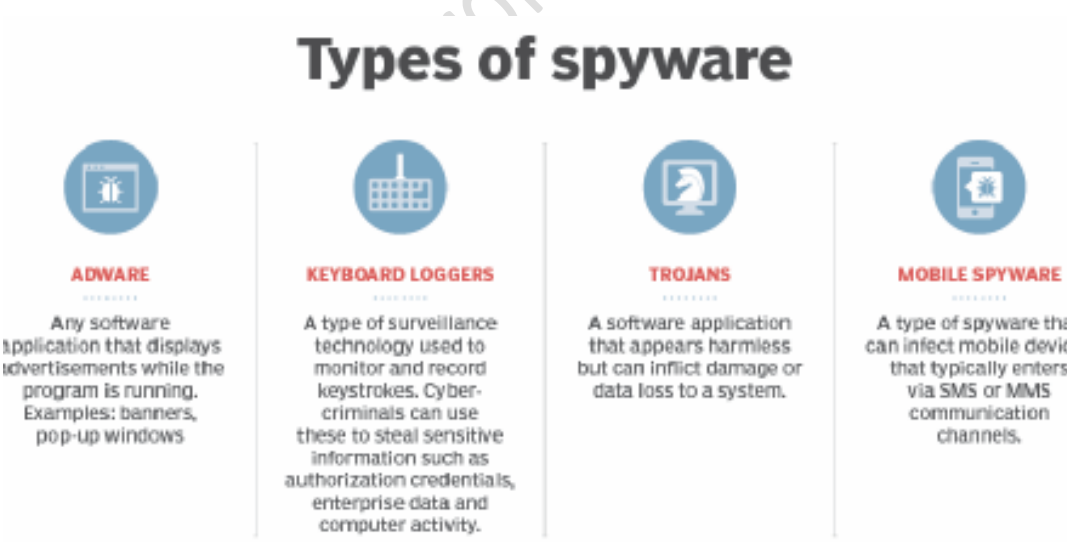

**Virus :** "A computer virus is a program that may disturb the normal working of a computer system". Virus attaches itself to files stored on floppy disks, USBs, email attachments and hard disks. A file containing a virus is called infected file. If this file is copied to a computer, virus is also copied to the computer.

**Trojan horse :** A type of malware that uses malicious code to install software that seems ok, but is hidden to create back doors into a system typically causing loss or theft of data from an external source.

**Worm :** Unlike a virus, a worm, is a standalone piece of malicious software that replicates itself in order to spread to other computers. It often uses a computer network to spread itself, relying on security flaws on the target system to allow access.



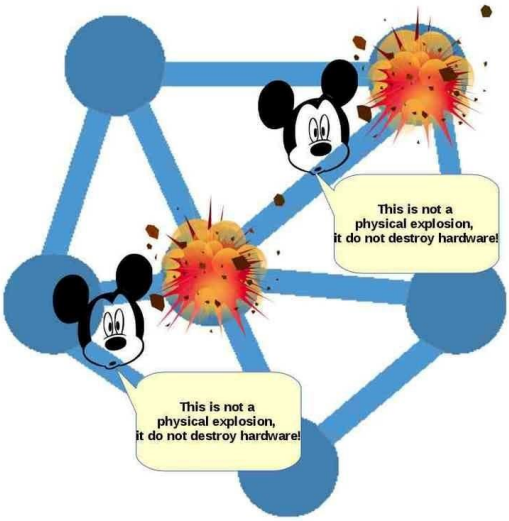
**Spyware :** Spyware is software that aids in gathering information about a person or organization without their knowledge, they can monitor and log the activity performed on a target system, like log key strokes, or gather credit card and other information.



**Adware :** Adware is software which can automatically causes pop-up and banner adverts to be displayed in order to generate revenue for its author or publisher. A lot of freeware will use Adware but not always in a malicious way, if it was malicious, it would then be classed as spyware or malware.

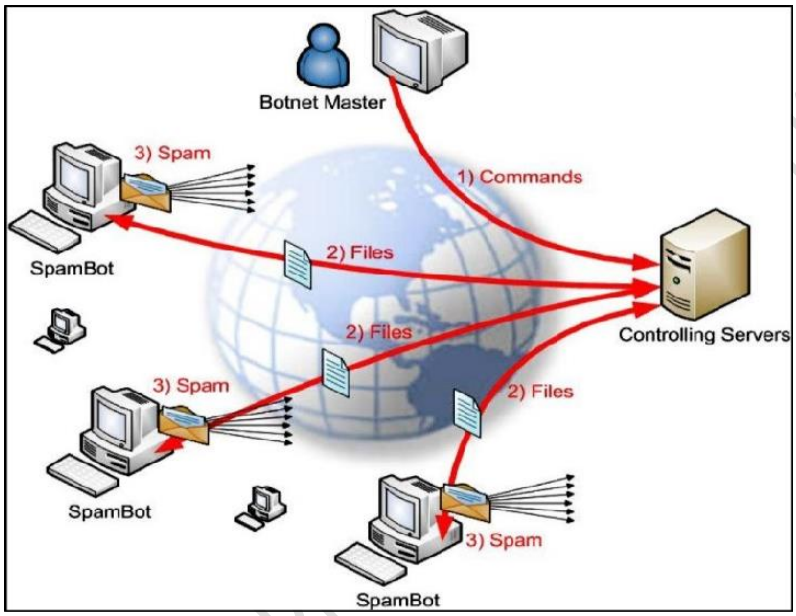

## Logic Bomb (slag code)

A logic bomb is a malicious program that is triggered when a logical condition is met, such as after a number of transactions have been processed, or on a specific date (also called a time bomb). Malware such as worms often contain logic bombs, behaving in one manner, then changing tactics on a specific date and time.



## Bots

A malicious bot is self-propagating malware designed to infect a host and connect back to a central server or servers that act as a command and control (C&C) centre for an entire network of compromised devices, or "botnet."




## Rootkits

A rootkit is a computer program designed to provide continued privileged access to a computer while actively hiding its presence. The term rootkit is a connection of the two words "root" and "kit." Originally, a rootkit was a collection of tools that enabled administrator-level access to a computer or network. Root refers to the Admin account on Unix and Linux systems, and kit refers to the software components that implement the tool. Today rootkits are generally associated with malware – such as Trojans, worms, viruses – that conceal their existence and actions from users and other system processes.

A rootkit allows someone to maintain command and control over a computer without the computer user/owner knowing about it. Once a rootkit has been installed, the controller of the rootkit has the ability to remotely execute files and change system configurations on the host machine. A rootkit on an infected computer can also access log files and spy on the legitimate computer owner's usage.

**References**


1. https://www.geeksforgeeks.org/buffer-overflow-attack-with-example/
2. https://searchsecurity.techtarget.com/definition/buffer-overflow
3. https://spanning.com/blog/cross-site-scripting-web-based-application-security-part-3/
4. https://portswigger.net/web-security/cross-site-scripting
5. https://www.thegeekstuff.com/2013/06/buffer-overflow/