CHAPTER 6

Application and Web Security

Introduction

As we have explored in earlier chapters, security applies to all the components of the systems including physical infrastructure like building, electricity, cables, and so on; hardware; network; software; tools / utilities; human beings including resources internal to the organization and contractors / suppliers who may be working from within the organization or outside the organization. Any part of the entire chain of components can be ignored from security perspective only at the peril of an organization.

Infrastructure is protected through physical security including fences. Hardware is protected through logical access control systems which are smart card based or biometric based or similar mechanisms or by physically securing the system, such as laptops locked to the desk through a locking cable. Physical infrastructure and hardware are relatively easy to secure (even though they have their own challenges like securing laptops during travel etc.). However, ensuring security of the software (including operating system, applications, tools / utilities) and networks are most difficult because of various possibilities as to what can go wrong. Some of the typical vulnerabilities they are exposed to are: misconfiguration, not validated inputs, defects / errors in coding, man in the middle attacks, man in the browser attacks, session hijacking, weak encryption keys, weak / default passwords, weak authentication mechanisms, SQL Injection, and Buffer Overflows.

Increased usage of software and web based applications over the internet has increased the exposure of these various kinds of attacks. Hence, securing them effectively should be the focus of all the designers and developers of such systems. People may believe in "If something has to go wrong it will go wrong." It may be so. But because something can go wrong - not acting on the possible issues is akin to a king inviting the enemy with a red carpet without even fighting the battle, which can only cost the king dearly. However, if we look at the history and data from the history, it is very clear that we are far behind in the race of attacks on software applications and web applications. We are yet to wake up and catch up with the best practices on application development and infrastructure set up, so that we have a fairly good chance of winning the race or at least giving the best possible fight.

Software Applications

More and more work, which was being done earlier by hardware components, is being carried out currently by software or software applications. For e.g. look at the functions carried out by the embedded systems in any automobile. It has substantially increased and many of the functions of a car or an airplane are now directed or controlled by software or software applications. Hardware can fail because of mechanical faults or component aging or component manufacturing defects or weaknesses of the material etc. However, in case of well thought out and well-designed hardware, such chances are less and also may be possible to be detected relatively easily.

Think of medical devices. More and more medical devices are moving to software based from mostly hardware based platforms. Think of an embedded device which is implanted in the human body. Imagine that there is a defect in the software or firmware on it. You need to patch the device to overcome this defect. You have to communicate wirelessly with the device to patch the device. Imagine you update a wrong patch and the problem gets aggravated rather than resolved. For all practical purposes, we will include firmware here as part of software. Think of somebody else tampering with the implanted device wirelessly[1].

Recently there was an incident of Air India Dreamliner flying from Melbourne to Delhi which was required to make an emergency landing at Kuala Lumpur (software malfunction in the cockpit). Earlier in the day, it was reported that the cockpit computers stopped working and the aircraft landed without any navigation aid (which was later denied)[2, 3]. Luckily the pilots were experienced and could navigate the flight effectively and ensure an emergency safe landing. Think of a situation a few years from now if pilots become so tuned to the helpful cockpit and navigation systems that they may become handicapped without these systems. They may not be able to navigate the flight even for a few minutes without these systems and the passengers' life may be at high risk. It may be possible for a passenger to wirelessly impact the cockpit system or for a maintenance engineer to be part of the terrorist team and adversely program the cockpit systems.

Further, the software can get corrupted, can get hacked, data can be manipulated, data can be overwritten, algorithms may be wrong or wrongly implemented, exceptions may not be handled effectively, and errors may not be handled effectively.

A software application is as good as what it does. It should not do more than what it is expected to do. At the same time, it should not fail to perform what it is expected to do. The following eight characteristics are fundamental from the perspective of an application's security (see Figure 6-1):

1. Completeness of the Inputs
2. Correctness of the Inputs
3. Completeness of Processing
4. Correctness of Processing
5. Completeness of the Updates
6. Correctness of the Updates
7. Maintenance of the Integrity of the Data in Storage
8. Maintenance of the Integrity of the Data in Transmission

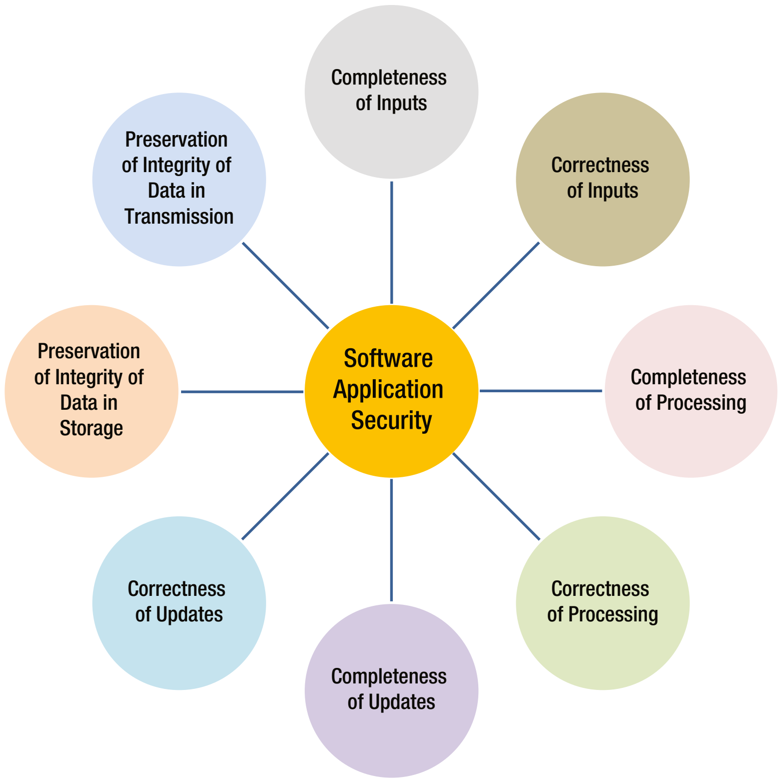
Software Application Security
Completeness of Inputs
Correctness of Inputs
Completeness of Processing
Correctness of Processing
Completeness of Updates
Correctness of Updates
Preservation of Integrity of Data in Storage
Preservation of Integrity of Data in Transmission

Figure 6-1. *Aspects of Software Application Security*

Completeness of the Inputs

Let us take an example from the field of banking to illustrate this characteristic. Every application has both master data as well as transaction data. Master data is almost fixed and may change only periodically depending upon the changes to the policies or changes to the rules, including Types of Deposits, and Rates of Interest for various tenures of deposits. Types of deposits can be Current Account, Savings Bank Account, Fixed Deposit, Cumulative Deposit, and Recurring Deposit. The rates of interest can be different for various types of eligible institutions and individuals and for various periods of deposit. Transaction data is related to a particular transaction, such as opening of a fixed deposit, application of interest to a deposit, and closing of a deposit.

It is necessary that the inputs taken are complete in all respects so that there is no issue later on to carry out the necessary transactions. Let us take a banking application. For example, when opening a fixed deposit, we did not obtain the type of the depositor and hence we are not able to decide on the appropriate rate of interest; when opening the account we did not obtain the details as to which account the periodic interest should be credited to, because of which, during the interest run we are not able to credit the amount to the corresponding account. Not taking the complete inputs (at least the mandatory ones) can hold back the completion of a transaction and such mistakes should be avoided as it impacts the business adversely and puts the customers at great inconvenience. Hence, it is necessary that the completion of the inputs should be checked by the application and ensured.

Let us take an example of a hospital application. For example, while admitting a patient, the details of the allergies that the patient is susceptible to are not obtained. Consequently, the doctors will be in dilemma as to the application of any medicine which may have side effects. Think of a situation if the patient is unconscious and no close relative is around and because of this lack of information, the doctors do not know how to act.

Completeness of the inputs, particularly of all the critical master and transaction related data in case of banking applications, has to be ensured by the application. Similarly, completeness of all the critical inputs which have bearing on the integrity of the data also has to be ensured. The relevant checks have to be built in the applications to ensure this.

Ineffective database design or inadequate input validation can lead to errors related to completeness of inputs.

Correctness of the Inputs

Another important characteristic every application has to ensure and every software developer has to ensure is that the inputs are correct (i.e., accurate). Integrity of any database or data set depends upon the correctness of the inputs. The inputs have to be validated for accuracy. Some of the inputs can only vary between certain values. In applications like banking, medical devices, and aviation, accuracy of master data is critical.

Think of a situation where the cockpit navigation system of an airplane is guided by a map with wrong coordinates for an airport - imagine what can go wrong!! Think of a ventilator medical device which senses the flow of oxygen wrongly (e.g., when the oxygen is being supplied slowly, the gauge is showing the oxygen as being pumped fast) and subsequent to this observation, the oxygen input is further reduced, imagine what can happen to a patient whose breathing is already impaired!! Think of a medical device which takes the unit of medication as grams instead of milligrams - imagine what can go wrong if a patient is administered a drug which is required to be administered in milligrams instead of in grams! All these can lead to serious implications endangering lives.

All critical data needs to be validated to ensure that they are accurate. One of the ways to ensure the same is to have bound checks on the tolerance for accuracy, another way may be to re-verify or checking of the data through somebody else before the data is accepted. Exceptions have to be handled effectively. Tallying with the control totals is one of the mechanisms to ensure the accuracy. However, in this method there are possibilities of compensatory mistakes.

Inadequate input validation or bounds checking or checks with respect to acceptable values lead to errors related to accuracy or correctness of the inputs.

Completeness of Processing

Another important characteristic every application has to ensure and every software developer has to ensure is that the processing of the information is complete. Integrity of the database or data set also depends upon this. The processing has to be checked to ensure that it was complete. In applications like banking, medical devices, aviation and such other critical applications, this aspect can be well appreciated.

Take a banking example. You have initiated a transaction of transferring USD 2 million from one of your accounts in one bank to your other account in another bank. What if the processing ensures that USD 2 million is debited from your account from where you are transferring the amount and USD 2 million is not credited to your other account? If the situation was the other way round, that is, my other account got credited with USD 2 million but my account which was to be debited did not get debited, I may be happy but what would happen to the bank? What if an X-ray taken does not show the portion of the bone which is cancerous? What if a brain scan shows the cancerous tissue with a displacement of only half an inch? Possibly, good tissue may be removed instead of the malicious tissue. Again, implications of such errors can be many.

Completeness of processing has to be checked by every critical application and confirmed. If the processing is partially complete, then the partially completed portion has to be reversed and the entire transaction has to be redone or the other portion has to be completed to ensure the integrity of the data. Comparison of earlier data, like the total interest paid for the last quarter when compared with this quarter, would indicate the possible mistake if for example the deposits have reduced and the rate of interests have reduced but the total interest paid for the quarter has increased!! Some of the issues related to completeness of issues can happen if you were in the middle of the processing and the system crashed or because electricity went off, the processing abruptly stopped as we did not have UPS backup.

In a transaction oriented system, the check on the completion of all parts of the transactions is critical to ensure the completeness of the processing.

Correctness of Processing

Another important characteristic every application has to ensure and every software developer has to ensure is that the processing of the information is accurate. If this is not taken care of by the applications, again integrity of the data becomes questionable and data may become useless or not reliable.

What if your airplane calculates the distance to the next mountain wrongly? Imagine if it calculates the distance as 100 miles instead of the actual distance of 50 miles? What if you are travelling on such an airplane? What if the medical device pumping a medicine into the patient's body calculates the quantum of medicine wrongly? What if the banking application calculates the interest on your loan account wrongly at 109% whereas it was supposed to calculate the same at 9%?

The above type of mistakes in processing may happen because of the mistake in logic employed or may be on account of mistakes in an algorithm used or may be on account of wrong master data used, wrong application component used, or wrong menu used.

Correctness (i.e., accuracy) of processing has to be checked by every critical application for every critical processing and confirmed. If the processing is wrong, then the defects in the application have to be fixed appropriately.

Application weaknesses like SQL injection, Command Injection, Buffer Overflows, and Cross Site Scripting Attacks can severely impact the application data integrity if these are not taken care of while designing and developing the applications.

Completeness of the Updates

Like the completeness of the inputs, the completeness of the updates also has to be ensured by the applications. Updates are typically to the master data. Suppose the rates of interest on the fixed deposits have been reduced w.e.f. 1st of January 2014 but the same is not updated on the master data, the interest to the depositors will be paid at a higher rate of interest and consequently the bank will lose a substantial amount of money. Imagine that a patient was operated upon yesterday but his record was not updated with the fact of this operation, think of a possibility of his being taken again to the operation theatre for another operation. Of course, the operation may not happen as the wound and sutures are already there but think of the time and effort spent on scheduling, prepping, and so on.

Verification of all critical updates has to be provided through the application to ensure that no critical update is left out and the critical updates are complete in all respects and not partially.

Correctness of the Updates

Like the completeness of the updates, the correctness of the updates is also important. If a patient has to be operated upon the left knee and the update in the medical record of the patient has been made that the procedure is required for the right knee (e.g., knee replacement operation) by oversight or mistake, it is possible that the patient, very much under general anesthesia, may realize the blunder only after the operation is over. He may have to suffer doubly because he already has a problem on his left knee and now he is additionally operated on the right knee!!

Verification of all critical updates for correctness has to be provided through the application to ensure that no critical update is wrong and that the critical updates are accurate in all respects and not partially.

Preservation of the Integrity of the Data in Storage

This refers to the integrity of the data in the underlying database of any application or data under the custody of any application. It should not be possible for anybody to modify this data directly without the authorized provisions of the application. If anybody can modify the underlying data of any application, then the application and data loses its sanctity. For example, the deposit in the banking account of a particular person is USD 1,000,000. The same is modified, without authorization, directly through back end, by running an unauthorized script, to USD 10,000,000. These types of possibilities seriously impact the credibility of the applications.

There are various mechanisms used by the application developers to avoid such issues. Some of these are record checksums, database checksums, or the system is configured in such a way that data can only be modified through the normal application interfaces and not through backend or through any other script directly, regular integrity check runs throwing up the mistakes, strong logging mechanism which ensures that critical data is checked regularly with the corresponding audit logs / pre-image.

Preservation of the Integrity of the Data while in Transmission

We are now in the highly networked era. Applications may be distributed or databases may be distributed. Data may have to be transmitted from one system to another, may be across the oceans and across the continents, such as ecommerce transactions, and credit card transactions. Such data are prone for Man in the Middle Attacks or Man in the Browser Attacks, and similar attacks which can lead to unauthorized modification of the data being transmitted.

Hence, application security should consider the security of the data while in transmission and ensure secure transmission of critical data through mechanisms like encryption.

Importance of an Effective Application Design and Development Life Cycle

A strong, well-defined Software Application Design and Development Life Cycle is essential in any organization that develops critical applications like medical applications, nuclear plant control applications, missile control applications, gas / oil pipeline control applications, electricity control applications, banking applications, and rail network control applications.

Effective time has to be spent during the initial phases of the development life cycle: Requirements Gathering and Analysis and Architecture & Design to understand thoroughly the expectations or requirements of the application. These should also focus on non-functional requirements like processing time, response time, integrity requirements, availability requirements, scalability requirements, flexibility requirements, usability requirements, reliability requirements, and security requirements. Integrity of the data and security of the data are very crucial to be considered at each phase of the design and development life cycle. Sufficient consideration for all the applicable functional and non-functional requirements should be provided for at the design stage by according adequate thinking.

At each phase, strong risk assessment focus should be provided which should bring out what can go wrong. Based on the risk assessment results, necessary risk mitigation or risk control steps have to be built into the applications or into the procedures governing those applications like segregation of duties among the employees, and multiple checks. Some of these controls are possible to be built through the application itself, such as a transaction that is routed for second check before it is debited to the account.

Programming languages and the code itself may bring out security issues. Secure coding standards are emerging. Application architects, designers and developers are learning from various issues they have encountered, technical / technological loop holes or deficiencies they have observed, or those which were reported in the media, and so on. Deficiencies of the third party tools used, deficiencies of the underlying platforms including those of the operating system, and the database systems have to be considered.

Strong testing at various phases of development right from Unit Testing to System Testing to Integrated Testing has to be ensured. When changes are carried out to the existing products, Regression Testing has to be ensured. These tests complement each other and have to be done without fail. The test cases and the corresponding test data have to be well thought out and included as part of the Test Case Design. Similarly, testing should not only focus on what is expected to be done by the application but also what it should not do.

Unfortunately, “thinking” consumes (as per my knowledge of our brain’s working) lots of energy. Hence, as human beings, we hesitate to carry out adequate thinking. In order to ensure that the applications are effective i.e. they do only what is expected and not do what is not expected of them, adequate and sufficient thinking has to be carefully provided during each and every phase of the Application Design and Development life cycle. This pro-activeness on our part can lead to effective, strong, and robust applications which are capable of protecting the users / end-users and their interests.

Important Guidelines for Secure Design and Development

Any software design and development team typically adheres to the following generic guidelines, which are illustrated in Figure 6-2:

- Understand the Security Requirements of the application (functionality and data related) and document them as part of the Requirements Specifications Document.
- Ensure that the Security Requirements are considered during Architecture and Design
- Follow Secure Coding Standards
- Validate all the inputs including the boundary checks, check against allowed values, and format.
- Ensure strong login mechanisms (including the need for strong passwords)
- Ensure encrypted transmission of data (where the confidentiality of the data has to be maintained and integrity is of prime importance) – like in case of banking applications, medical software involving patient health and safety, aviation, nuclear safety, military systems and weaponry, satellite systems, and such other critical domains / systems
- Ensure periodic mandatory change of passwords
- Appropriate privileges / access rights to various processes of the applications and various roles – work on Least Privilege Principle
- Appropriate handling of errors (including customized error messages which do not give out unnecessary information)
- Appropriate exception handling mechanisms built into the application
- Appropriate configuration – All settings have to be appropriately made
- Use of vetted algorithms
- Counter checks to ensure complete and accurate processing of critical processing
- Eliminate all unwanted / unused functions and routines
- Ensure proper log out mechanisms
- Use secure protocols
- Ensure proper logging and auditing mechanisms
- Have strong configuration management during development
- Have strong testing
- Have strong control over software releases

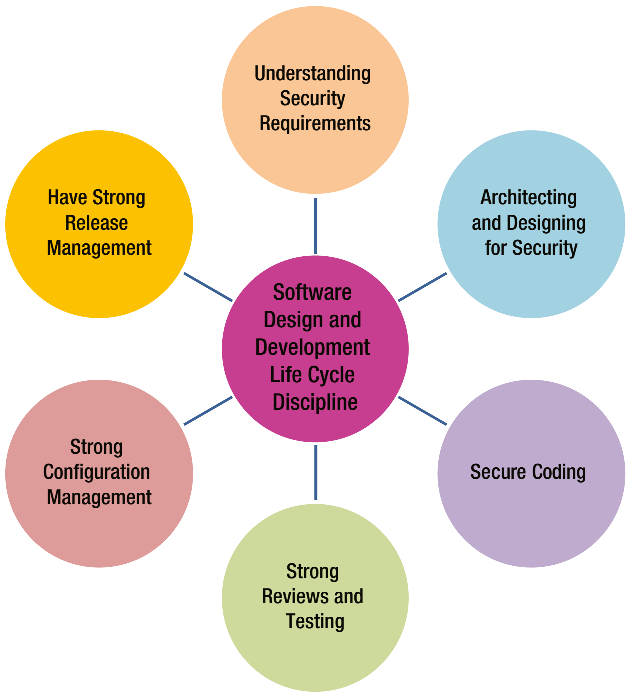
Understanding Security Requirements
Have Strong Release Management
Architecting and Designing for Security
Software Design and Development Life Cycle Discipline
Strong Configuration Management
Secure Coding
Strong Reviews and Testing

Figure 6-2. *Software Design & Development Life Cycle Discipline*

Some of the important secure coding practices that an application design and development team are required to follow are illustrated n Figure 6-3.

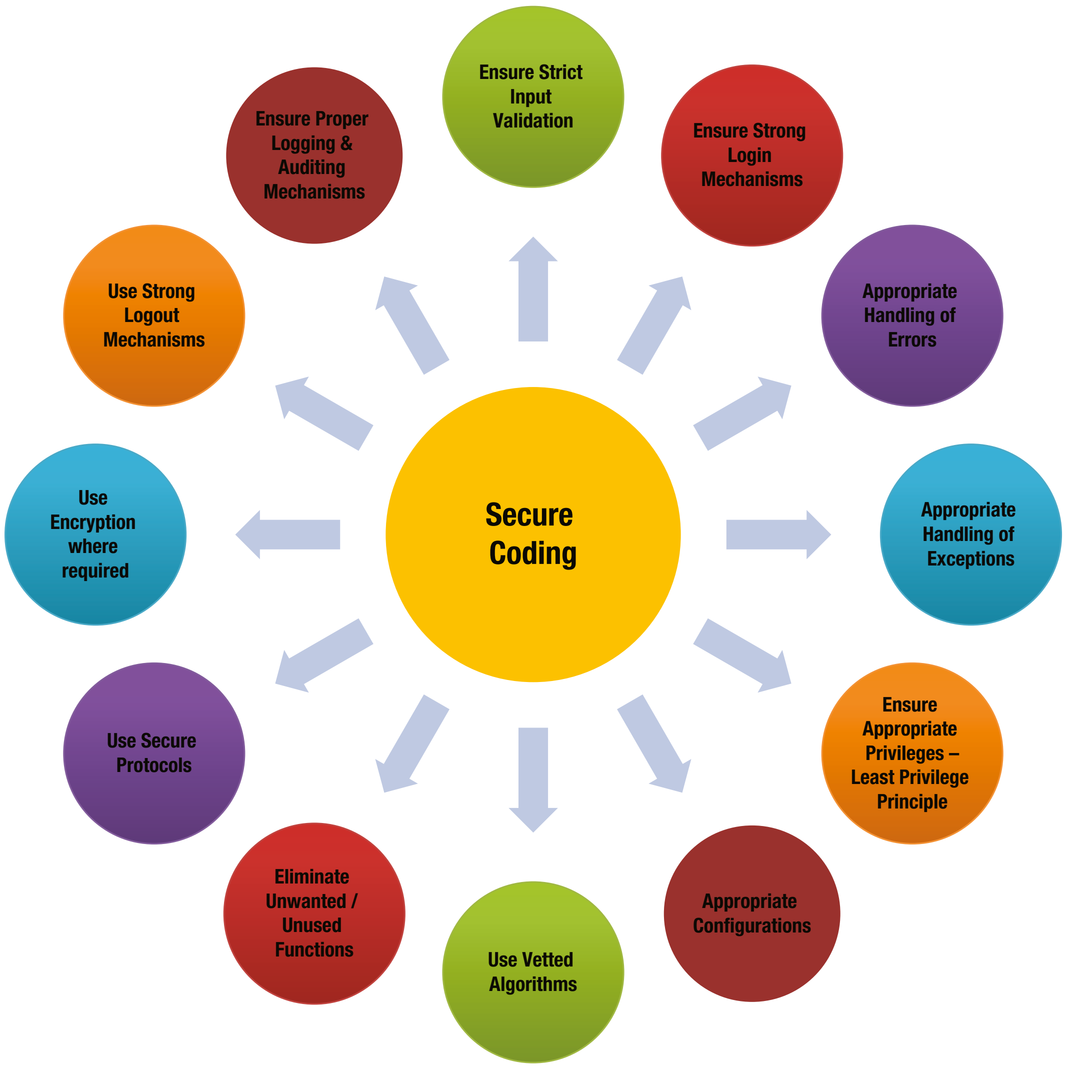
Secure Coding
Ensure Strict Input Validation
Ensure Strong Login Mechanisms
Appropriate Handling of Errors
Appropriate Handling of Exceptions
Ensure Appropriate Privileges – Least Privilege Principle
Appropriate Configurations
Use Vetted Algorithms
Eliminate Unwanted / Unused Functions
Use Secure Protocols
Use Encryption where required
Use Strong Logout Mechanisms
Ensure Proper Logging & Auditing Mechanisms

Figure 6-3. *Secure Coding Considerations*

Web Browsers, Web Servers, and Web Applications

Web Applications have become very popular because of their ease of operation and availability to a high number of users over the popular platform of World Wide Web or Internet.

Web Servers enable or facilitate the Web Applications by providing the requisite environment. Like any other three-tier-architecture application, Web Applications normally have a client tier (which is rendered through a Web Browser), a Business Layer which is facilitated through an Application Server, a Data Layer or the Backend which is facilitated through the database systems / servers. Web servers can be separate or a part of the Application server. Web Servers host websites or serve clients with requested web-pages. These web-pages may be static ones or dynamic ones created on the basis of a client's request. HTTP or HTTPS are the protocols typically used for communication between clients and Web Servers. A Web Server is typically a computer program running on a physical server.

Web Servers serve Web Applications. A Web Server, an Application Server and a Database Server should ideally be on separate hardware / physical servers for security reasons but may be on a single hardware / physical server in smaller organizations. Web Applications are software components which provide requisite functionality to the end users. Web Applications facilitate the clients to access the functionalities required by them and also facilitate access to the data lying in the database or data store. They provide for various functionalities like ecommerce, creating / updating of the blogs, providing feedback, carrying out banking transactions, ticketing etc.

The way a typical web-application works is illustrated in Figure 6-4.

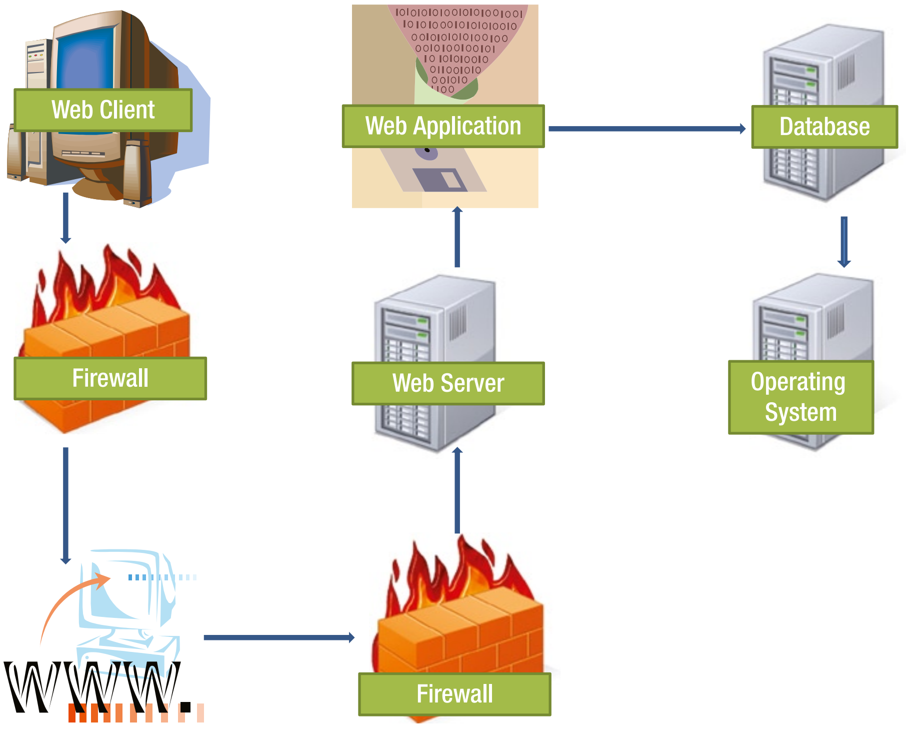

Figure 6-4. *How a Web Application Works*

Each of the three components: Web Browsers, Web Servers, and Web Applications, has its own vulnerabilities. Together, they also combine to create other vulnerabilities. The reasons for this are illustrated in Figure 6-5.

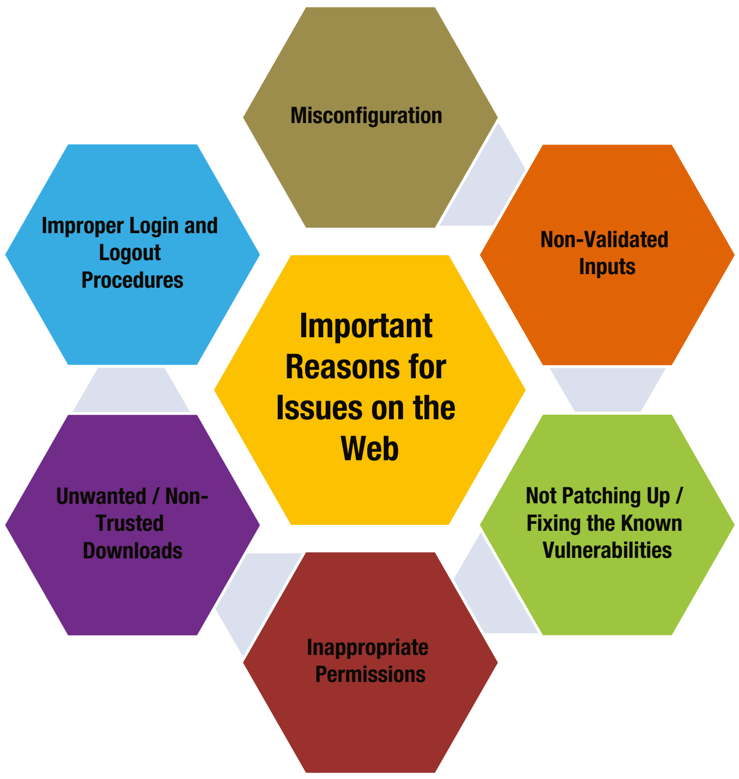
Misconfiguration
Improper Login and Logout Procedures
Non-Validated Inputs
Important Reasons for Issues on the Web
Unwanted / Non-Trusted Downloads
Not Patching Up / Fixing the Known Vulnerabilities
Inappropriate Permissions

Figure 6-5. *Important Reasons for Issues on the Web*

Vulnerabilities in Web Browsers

Web Browsers are well known and the most used in today's Internet based world. Some of the most used browsers are Microsoft's Internet Explorer, Google's Chrome, and Mozilla Firefox. However, these different web browsers are designed and coded differently, and thus have different vulnerabilities. Some are less used and some are extensively used. Less used ones may be found to have less vulnerabilities may be because the hackers have not found much value in making lots of effort in finding their vulnerabilities in order to exploit them.

Some of the typical vulnerabilities a Web Browser (may vary from one browser to another) is prone to are:

- Inappropriate Configuration
- Unnecessary or Untrusted Add-ons
- Malware (like scripts and executables) run on the Web Browser
- Not patching up or carrying out the security updates

Inappropriate Configuration

It is observed that in many cases, the configuration of the web browsers would have been carried out inappropriately. Pop-up may not have been set to be blocked, allowing the possibility of download of malware / spyware. All types of cookies, by default, would have been allowed whereas it may be risky to allow third party cookies by default, without validating them. Also, all types of scripts would have been allowed to be downloaded / run without any validation.

This is very risky as this may lead to the download of malware (without the user being even aware of it) which can impact the leakage of data held on the computer, crash of the applications, crash of the computer, or leakage of data being transmitted through the browser. Configurations of interest are security settings, privacy settings, and protocols allowed.

Unnecessary or Untrusted Add-ons

Usually, as seen, most of the users of the internet have a habit of downloading the add-ons without really verifying whether they are trustworthy. These add-ons may not be the trusted ones and may lead to malware infection of the web browser unless they are downloaded and enabled cautiously after verifying their usefulness and trusted status. The best way is to keep the unwanted add-ons disabled when you do not want them. This will also increase the performance of the web browser, which would have been reduced because of enabling too many add-ons.

Malware or Executables run on the Web Browser

Malware and other executables can be executed on the Web Browser. These can lead to theft of the data from the system on which the Browser runs (transfer of data from the system to the attacker), execute programs that can infect the system or crash the system or modify the integrity of the data on the system or lead to many such similar adverse impacts on the system.

No Patching up or Carrying out the Security Updates

From time to time, defects or security flaws have been observed in the web browsers. Some of these may be minor but some of these could also be major ones. It is necessary to protect ourselves from such security flaws. Everybody needs to ensure that updates to the browser, particularly security updates, are applied invariably. It is best to enable automatic updates in case of individuals as otherwise we may forget to update them manually. In case of organizations, they can analyze / test the possible impact of such updates on their infrastructure / various applications before carrying out such updates. However, evaluations, patches, and updates should be applied as early as possible if they are applicable to the organization and do not have any adverse impacts. Ideally such updates have to be ensured centrally in big organizations.

How to Overcome the Vulnerabilities of Web Browsers

The following precautions are some of the ways to overcome Web Browser vulnerabilities:

- Configure the Web Browser with appropriate settings – Consult experts if required
- Keep the Web Browser regularly updated with Updates and Security Patches
- Disable or Uninstall unnecessary or untrusted add-ons
- Understand the errors thrown up by the browser like Expired Certificate alerts and then act on them appropriately – If not understood, do not simply act on the same – Consult the experts
- Configure your Operating System properly – Change the default settings to appropriate settings
- Have a well known and reputed Anti-Virus installed on your Operating System

Vulnerabilities of Web Servers

The following web servers are typically used: Internet Information Services (IIS) Windows Server from Microsoft Corporation, Apache Web Server, Nginx Web Server, and Tomcat Web Server.

Some of the typical vulnerabilities of web servers include:

- Default Users and default permissions not changed
- Sample files / scripts not removed
- Default configuration not changed
- Directory permissions not set up properly
- Security loop-holes or defects in the web server software or underlying operating system not fixed / patched up

Other typical attacks possible on the web servers and web sites are:

- DNS attack which redirects users to another web server
- URL Poisoning which redirects users to another URL
- Man in the Middle Attacks
- Password Cracking
- Cookie Poisoning
- Compromise of underlying servers like Email Server and FTP Server
- Buffer Overflow attacks
- SQL Injection attacks

Some of these are self-explanatory and some others like Buffer Overflow attacks and SQL Injection attacks are explained in more detail in the context of Web Application Security.

Default Users and Default Permissions are not changed

Usually, Web Servers like Internet Information Services come with default users with default or no passwords and default permissions. It is necessary to look at the need of actual users and appropriate user IDs and strong passwords have to be set. Default permissions have to be modified according to the need.

Sample files and scripts are not removed

In order to enable the users to set up the Web Server effectively and enable requisite functionalities, Web Servers like IIS come with some sample files / scripts. These need to be removed as they may not be used by the experienced system administrators. If not cleaned up and allowed to lie on the Web Server they may be misused by hackers to attack the Web Server. Hence, all such unwanted sample files / scripts should be cleaned up by removing them from the Web Server.

Default Configuration is Not Changed

Configuration of the Web Server has to be done effectively. Default configurations have to be evaluated and based on the need of the organization and related web applications, these configurations have to be set appropriately. Some of the configurations to be set appropriately are: setting up of SSL Certificates, setting up of administrative functions, setting up of encryption functions, and setting up of debug functions. Unnecessary and unwanted configurations have to be modified appropriately.

File and Directory Permissions are not Set Properly

It is possible that the system administrators have not set the permissions to various files and / or directories on the Web Server effectively. Using such improper / inappropriate permissions, will enable hackers to traverse various directories on the Web Server, which they are normally not authorized to access. This type of attack is known as Directory Traversal Attack and this is mostly observed in old, unpatched Windows IIS Web Servers.

Security Loop-Holes or Defects in the Web Server Software or Underlying Operating System

Various loop holes or defects related to a web server and the underlying operating systems come to the fore and get published by those who find it or the vendors-commissioned organizations which track vulnerabilities. The vendors then come up with solutions or patches / updates to address the same. Up-to-date patching up of the web server and the underlying operating system assures us that the web server is secured to a reasonable extent.

How to Overcome the Web Server Vulnerabilities

The following precautions are some of the ways to overcome Web Server vulnerabilities:

- Change / Disable Default Users and Default Settings and Configure the Web Server appropriately – Disable serving malicious file types which are not required from the perspective of the Web Application
- Regularly patch the Web Server
- Unnecessary and unwanted sample files / scripts to be removed
- Patch up the security loop holes on the underlying Operating System
- Set up file / directory permissions appropriately – Use Least Privilege Method – Set up only the privileges as absolutely required – Evaluate privileges before providing them – Apply the Least Privilege Method even to the database side and to the processes
- Analyze the impact of the changes from a security perspective and carry them out appropriately – Do not jump in and perform the changes without impact analysis
- Ensure strong passwords for both the user accounts and for administrator accounts – Implement periodic mandatory change of password policy
- Ensure that only the need-based ports are open and only the need-based services are active – Disable services like FTP, SMTP, and so on if not needed
- Encrypt the traffic as necessary
- Ensure data files are kept out of the Web Server
- Monitor the Logs regularly
- Delete unnecessary file shares
- Disable Tracing and Debugging
- Check for the validity of the Certificates
- As much as possible, use a dedicated machine for Web Server, other servers like database services should be installed on separate physical servers

- No local logins on the Web Server machine should be allowed
- Server side session ID validation
- Scan the Web Server periodically through Vulnerability Scanners to identify and fix the vulnerabilities
- Perform boundary checking / validations on the inputs

Web Applications

As discussed earlier, web applications provide the requisite interface to the clients to carry out the necessary interactions with the concerned servers and access relevant data.

Some of the important vulnerabilities in web applications include (see Figure 6-6):

- SQL Injection Attacks
- Command Injection Attacks
- Buffer Overflow Attacks
- Cross-Site Scripting
- Cookie Poisoning
- Session Hijacking

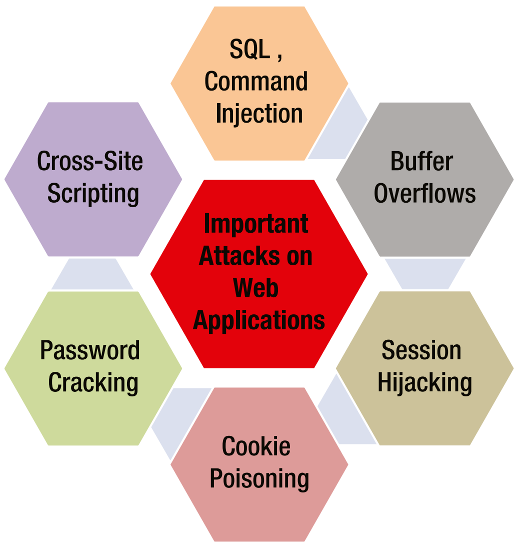

Figure 6-6. *Important Attacks on the Web Applications*

Most of these, like SQL Injection Attacks, Command Injection Attacks, Buffer Overflow Attacks, and Cross-Site Scripting happen because of non validated inputs. Inputs are processed by the application as they are fed, without further validation by the web application or the underlying database server.

SQL Injection Attacks

Any web application wherein the inputs are not validated before being processed further and where the database elements are not protected effectively by means of appropriate permissions, are likely to be targets of SQL injection attacks. These are possible when the user provided inputs on the web forms are used to create the SQL query (Select, Insert, etc.) and these inputs are not validated for the appropriateness before they are processed. This may happen through a login page or other input pages from the web forms. Inputs through a login page may allow the authentication to be bypassed and direct access to even the user-ids and passwords of others.

Many of the web applications use default error messages, which may make the users aware of the vulnerabilities possible in the concerned web form. All the error messages have to be customized so that they do not give out more information than necessary to the users or expose the internal vulnerabilities.

Most of the SQL injection methods fool the internal processing mechanism by defaulting the validation of input to be "true" so that the underlying SQL Query is fired even when the actual input is not the right one. Typically "……..' or 1=1- -" is the input used. Two hyphens (- -) stand for comments and makes the SQL statement beyond this point to be treated as a comment. Such SQL injections can throw up as a response, multiple rows from a table or an entire table.

Some examples of the uses of SQL commands are:

```
Blah';exec master..xp_cmdshell "dir c:\*.*  / s > c:\directorylist.txt"--

Blah';exec master..xp_cmdshell "ping 192.168.1.2"--
```

SQL Injection attacks allow the attacker to delete, alter database information, and to create / add new information to the database. This leads to the loss of integrity of the data within the database.

SQL Injection attacks are typically used to understand the database structure and database contents, getting data from the database, adding / modifying / dropping data from the table, and bypassing authentication, for executing commands remotely.

SQL Injection attacks can also be carried out using dynamic strings which are built and executed at a point in time.

Command Injection Attacks

Command Injection Attacks are carried out by the attackers by creating an input string which enables gaining shell access to the underlying web server. Some of the operators that are used to carry out this attack appear in Table 6-1[4].

Table 6-1. *Operators Used for Command Injection Attacks along with Purpose of Attack*

Operator	Purpose for which it is used by the Attacker
;	This is used to add execution of the additional unintended commands
<, >, >>	These are used to modify files or create new files
|	This pipe symbol is used to chain multiple commands
$, &&, ||	These symbols allow logical operations to be performed like 'AND' or 'OR' on the data before and after them

This type of vulnerability can also be used to inject code with malicious intent into the system files and also to execute scripts like HTML scripts.

Buffer Overflow Attacks

Again this attack becomes possible because of non-validated inputs. These happen, primarily, because of a lack of bound checks or a lack of validation in total. Buffer Overflow attack happens when a Web Application writes more data to a memory location than what it can hold, thus overwriting the adjacent memory space, leading to a buffer overflow. Here, the Web Application does not check the size or format of the input data before storing it in a memory variable.

Buffer Overflow Attacks allow the attacker to alter the sequence of process / program execution, control the process / program execution, and modify the internal variables leading to the execution of malicious code planted in the memory at a particular address. This overflow which overwrites the neighboring buffer area can lead to abrupt termination of the program being executed, incorrect outputs, and abrupt shutdown of the system. However, these types of attacks are normally intentionally crafted exploits where the return pointer is overwritten through buffer overflows to point to the malicious code inserted by the attacker and the execution of the same. The extent of the attack possible is dependent on the context in which the program attacked is running or the privileges under which the program is running.

Two types of Buffer Overflow Attacks can occur:

- Stack Based Buffer Overflows
- Heap Based Buffer Overflows

Both of these buffer overflows act like the holders of the variables until the corresponding program function needs them. Stack is a Last-In-First-Out method to refer to a memory address space which is used to hold the variables and to pass the arguments to the functions. Stack is a static location. Heap is again a memory address space, but is allocated dynamically by the program at runtime. Both are prone to Buffer Overflow Attacks.

In Stack Based Buffer Overflows, the buffer is overwritten by the overflow, which enables the attacker to overwrite the return pointer to point to the malicious code so that the malicious code is executed instead of the originally intended function / code.

The popular Stack Based Buffer Overflow Attack method is the “Smashing the Stack” attack. In this type of attack any input overflow will overwrite the other variables, base pointer, and return address. In the process the malicious code is also inserted by the attacker next to the return address. The attacker intelligently crafts this exploit in such a way that the overwritten return address points to the malicious code injected and leads to the execution of the same. For this attack to be successful it is necessary for the attacker to be able to predict the return address location on the stack as well as the correct return address. In order to enable this, the attacker normally uses multiple NOP (No Operation) instructions along with multiple times the predicted start address. Figure 6-7 illustrates the Stack Based Buffer Overflow Attack.

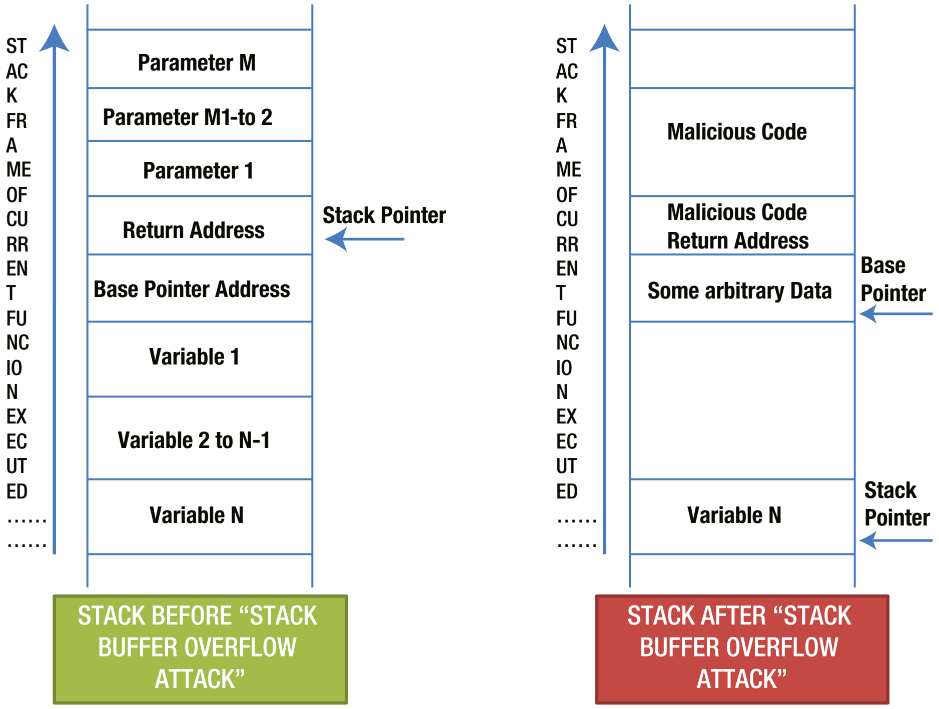
STACK FRAME OF CURRENT FUNCTION EXECUTED …… ……
Parameter M
Parameter M1-to 2
Parameter 1
Return Address
Base Pointer Address
Variable 1
Variable 2 to N-1
Variable N
Stack Pointer
STACK BEFORE "STACK BUFFER OVERFLOW ATTACK"
STACK FRAME OF CURRENT FUNCTION EXECUTED …… ……
Malicious Code
Malicious Code Return Address
Some arbitrary Data
Variable N
Base Pointer
Stack Pointer
STACK AFTER "STACK BUFFER OVERFLOW ATTACK"

Figure 6-7. *Stack Based Buffer Overflow Attack*

Usage of the non-executable stacks or usage of "canary value" are some of the mechanisms used to avoid Stack Based Buffer Overflow Attacks.

Normally, Heap Based Buffer Overflows, take large inputs which will overwrite the heap management information. They may also overwrite other dynamic variables. The Heap Based Overflows can lead to unknown side effects. In these types of attacks, the program can typically open a command prompt or stop execution of the program.

Heap Based Buffer Overflows are common in C and C++ programming languages. These are possible because the objects are loaded onto heap and such objects hold both data as well as the pointers. The data can be made to overflow the pointers, thus overwriting the address of the program statement being executed to the malicious code. Also, the input data can overwrite the neighboring memory locations on buffer overflow. Figure 6-8 illustrates the Heap Based Buffer Overflow Attack.

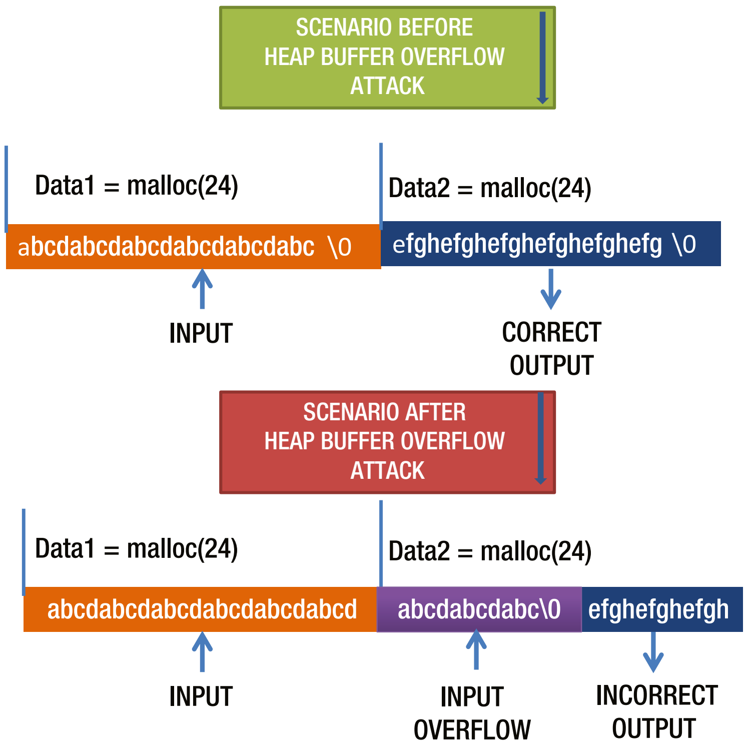

Figure 6-8. *Heap Based Buffer Overflow Attack*

Both the Stack Based Buffer Overflow Attack as well as the Heap Based Overflow Attacks are not easy and require elaborate study or proper prediction on the part of the attacker. The success for the attacker depends upon the right prediction.

Some of the C programming language functions like strcpy(), bcopy(), gets(), scanf(), sprint(), strcat() etc. do not validate the target buffer size and hence can lead to buffer overflows. No Operations (typically known as NOPs) are used in the Buffer Overflow Attacks to alter the sequence of execution of the program steps.

Cross-Site Scripting

Cross-Site Scripting again occurs mostly when the inputs are not validated. Attackers will embed the client side script in a Web Form in a field (e.g., “Comments” field) which is stored by the server and is provided to another unsuspecting user, in whose browser, the script embedded by the earlier user gets executed. This allows for easy embedding of malicious scripts by an attacker with the intention of getting it executed in the browser of other users who request such data and leads to the malicious results intended by the attacker. These types of attacks target the dynamically generated web pages. These attacks can also do other things like redirecting the users to other malicious servers, exploit the other user’s privileges, and session hijacking.

These may also be initiated by the attacker through a well-crafted malicious link in an e-mail, which initiates a request to the malicious server, which in turn, returns a page with malicious content which gets executed on the user’s web browser. These types of attacks can provide the user’s cookies to the attacker, who can use the same to transact on a genuine and trusted site.

Similarly, Cross-Site Request Forgery Attacks can execute malicious code on a genuine server (e.g., trusted server of a bank) by exploiting the weakness on the web page of the bank. This requires the user to visit the malicious site which downloads the malicious code when he is in session with the genuine site so that the malicious code is executed on the genuine server.

Scripting languages like VBScripts, JavaScripts, HTML, and ActiveX are used by the attackers to craft these types of attacks.

Cookie Poisoning

This type of attack is normally carried out through Cross-Site Scripting. However, this type of attack also can be carried out through other means. Another popular method is where the attacker initiates a phishing attack, where a genuine looking link is sent to the user while he is in transaction with a genuine site and is convinced to click on the link which will steal the cookie and transfer the same to the attacker. The attacker can then modify the contents of the cookie as required to fool the genuine site or to carry out unauthorized transactions on the genuine site.

Session Hijacking Attacks

Session Hijacking happens in the following ways:

- **Session Sidejacking**: A user is connected to a genuine server like that of a bank. There is a session established. The session is tracked by the server through the session IDs. Usually, the attacker sniffs the traffic and then cuts off the traffic between the user and the genuine server and takes over the session by predicting the next sequence number of the communication between the genuine user and the genuine server. This typically uses the weaknesses of the Transport Control Protocol where most of the authentication is carried out at the beginning of the establishment of the session.

- **Session Prediction**: In this case the attacker sniffs any victim's traffic to the site like a bank and collects various session IDs. He can monitor the traffic pertaining to multiple users also. He then studies these session IDs to understand how they are generated. If the simple techniques like user ID, date, and time of the day is used it is very easy for the attacker to predict the session ID for any user, date, and time and then use the predicted session id to access the site.

  In case the session ids are generated using complicated algorithms, then he can use the brute force mechanism to crack the session ID logic and once he has cracked the same use the same to generate a valid session ID and use it to attack the site.

- **Session Fixation**: In this case the attacker has a legitimate account with the server, such as the server of a particular bank the account holders of whom he want to dupe. Once he initiates the login process, he will get the session ID. This session ID is then issued to the browser of the victim's machine. The attacker ensures that this session ID is maintained valid by him. Once the genuine user connects to the server the browser of the user uses this fixed session ID. Once the user is authenticated to the server concerned using his login credentials, the attacker uses the session ID so fixed to connect to the server concerned and carry out fraudulent transactions.

  The session can be fixed also through an e-mail purported to be from the bank but actually sent by the attacker with the session ID known to him. This can be done by terminating the connection in between a valid session user and the bank and then by sending an e-mail to the user that restores the connection, the link given in the email ID can be used. The email through which normally the link is sent is designed in such a way as to make the victim believe that it is from the valid source, such as the bank itself. Session Fixation Attack is illustrated in Figure 6-9.

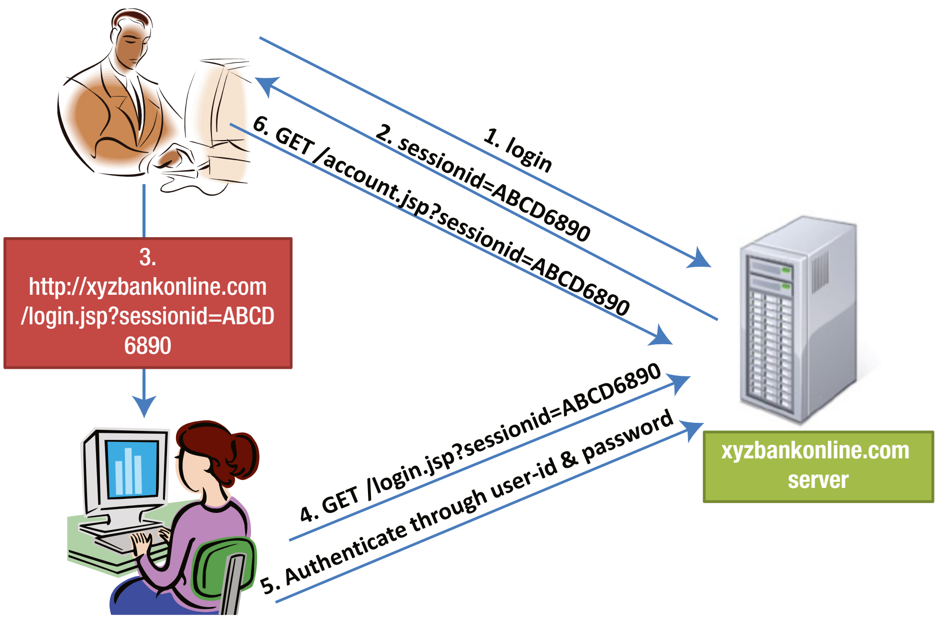
1. login
2. sessionid=ABCD6890
3. http://xyzbankonline.com /login.jsp?sessionid=ABCD 6890
4. GET /login.jsp?sessionid=ABCD6890
5. Authenticate through user-id & password
6. GET /account.jsp?sessionid=ABCD6890
xyzbankonline.com server

Figure 6-9. *Session Fixation Attack*

- **System Access**: In this case the attacker steals the cookie and thus steals the session key also. This happens when he has access to the victim's machine either directly or through the network. This can also be done by scanning the memory contents of the victim's machine. Once he steals the cookie he uses it to connect to the legitimate server.
- **Cross Site Scripting**: Here the user is conned to run malicious code on his computer which when executed steals the session key and sends it to the attacker. This attack can be done through an intelligently created link which the user believes in.

How to Overcome Web Application Vulnerabilities

The following precautions are some of the ways to prevent Web Application vulnerabilities:

- Invariably validate all the inputs including format checking, bounds checking, and acceptable values.
- Invariably configure Web Applications appropriately
- Regularly patch up all the servers including Web Server, Application Server, Database Server (as relevant and applicable)
- Do not save your Login credentials including passwords on the Web Browser
- Do not save unnecessary information on your Web Browser
- Log off immediately after the work on a Web Application is over – Do not keep the session open unnecessarily for long

- Use strong encryption keys and strong encryption where required - Do not store the encryption keys on the Web Server - Use Secure Socket Layer (SSL) and Digital Certificate for strong authentication
- Define appropriate access rights
- Ensure appropriate Log Out mechanisms in the Web Applications
- Ensure that the Certificate is Valid and has not expired
- Implement effective Cookie Management including Cookie time-out, do not store passwords in a Cookie, authenticate Cookies
- Carry out regular Vulnerability Scans and Penetration Testing to understand and fix the underlying vulnerabilities

Secure Socket Layer (SSL) Security and Digital Certificate

Secure Socket Layer (SSL) protocol and the Digital Certificate ensure that the information exchanged between the Web Server and Client Browser is secure.

Secure Socket Layer (SSL) protocol is used to secure the communications between the Web Server and the Client Browser and uses public key encryption to exchange symmetric encryption keys between the Web Server and the Client Browser. These exchanged symmetric encryption keys are used for encrypting the messages between the Web Server and the Client Browser. Digital Certificate is used for the authentication of Web Server to the Client Browser. Authentication here means that the Web Server confirms that it is legitimate / genuine to the Client Browser. These Digital Certificates are issued by the certificate authority to any organization only after validating the entity to which they are issuing the Digital Certificate. Certificate authorities are the trusted agencies. Hence, it is not possible for a bogus / non-existent company to get the Digital Certificate from a recognized certificate authority. The Digital Certificate clearly shows to which organization it is issued and the validity of the digital certificate.

The SSL protocol ensures the confidentiality of the data being exchanged as the encryption cannot be so easily cracked. This protocol ensures that the integrity of the data being exchanged is genuine, as decryption fails if any piece of the encrypted information is changed / modified. The Digital Certificate provided by the Web Server to the Client Browser and validated by the Client Browser assures that it is handshaking with the legitimate / genuine Web Server.

SSL is used to encrypt transactions carried out through protocols like HTTP and FTP. SSL is also implemented on all the important commercially available browsers. The SSL process is illustrated in Figure 6-10.

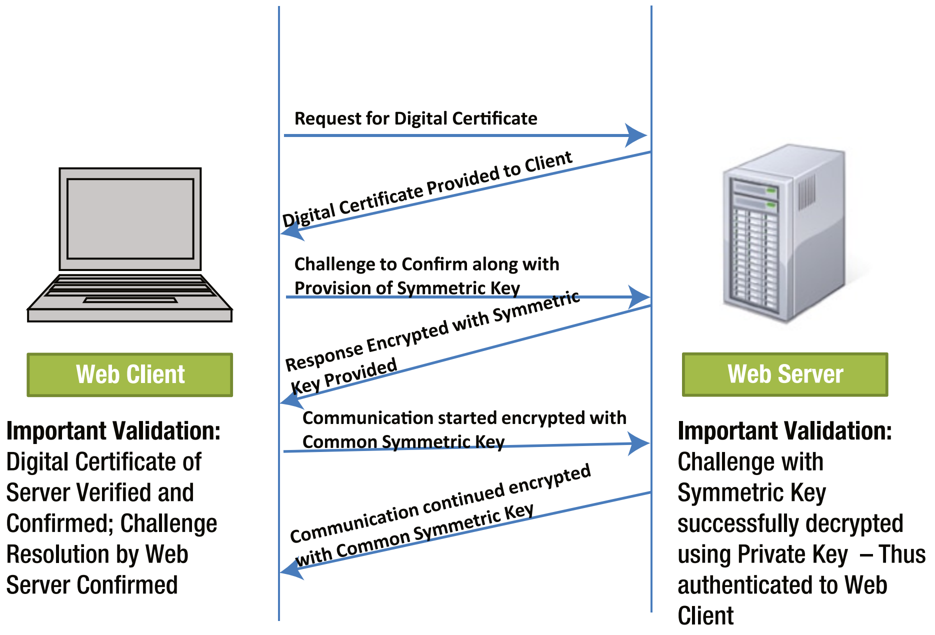
Request for Digital Certificate
Digital Certificate Provided to Client
Challenge to Confirm along with Provision of Symmetric Key
Response Encrypted with Symmetric Key Provided
Communication started encrypted with Common Symmetric Key
Communication continued encrypted with Common Symmetric Key
Web Client
Important Validation: Digital Certificate of Server Verified and Confirmed; Challenge Resolution by Web Server Confirmed
Web Server
Important Validation: Challenge with Symmetric Key successfully decrypted using Private Key – Thus authenticated to Web Client

Figure 6-10. *Secure Socket Layer Process*

The six steps in the process are:

Step 1: The Client Browser requests the Digital Certificate from the Web Server with which it has to communicate.

Step 2: The Web Server provides its certificate to the Client Browser.

Step 3: The Client Browser checks the certificate to ensure that it is from a trusted certificate authority and contains the Web Server's "public key". The Client Browser then sends a challenge to the Web Server to check that the Web Server has the "private key" corresponding to the "public key" contained in the certificate. The challenge includes the symmetric key which will be used to encrypt the SSL traffic. This challenge cannot be responded to by others because only the genuine Web Server has the "private key".

Step 4: The Web Server decrypts the challenge using its "private key". Then it encrypts its response using the symmetric key it has received. Owing to this exercise, the Client Browser gets assured that it is communicating with a genuine / legitimate Web Server. Now, both the Client Browser as well as the Web Server have the same symmetric key which they can use for further communications between them.

Step 5: The Client Browser sends its request to the Web Server encrypting the request with the common symmetric key.

Step 6: The Web Server responds to the Client request by encrypting the response with the common symmetric key.

The Web Browser, if configured properly, will provide warnings on the certificate in the following cases:

- Certificate is invalid or expired
- Certificate not signed by recognized certificate authority
- Name on the certificate does not match domain name of the server

Ideally, in such a case, the user should not accept the certificate. If the certificate is accepted, the user is susceptible to attack.

TLS (Transport Layer Security) protocol is an improvement of SSL.

Chapter Summary

- We introduced the relative fact that infrastructure related security and hardware security are easy to handle compared to the security aspects related to software (which includes the operating system, software, software applications, utilities / tools, etc.) which are more complex and difficult to handle.
- We examined software security, namely, primarily software application security and how the integrity of software application is very important and what are the essentials to ensure this integrity. In this context, we explored in detail the eight fundamental aspects, Completeness of Inputs, Correctness of Inputs, Completeness of Processing, Correctness of Processing, Completeness of Updates, Correctness of Updates, Preservation of the Integrity of Data in Storage, and Preservation of the Integrity of Data in Transmission. We discussed all of these with relevant examples from critical domains / fields like banking, aviation, and medical devices.
- We explored how adequate and appropriate attention to security requirements and potential risks during the Software Design and Development Life Cycle can substantially reduce the vulnerabilities in the software application developed. We also explored how secure design and coding contribute to this regard. We then looked into some of the important aspects which have to be considered as part of Secure Coding.
- We reviewed the basics of Web Browser, Web Server, and Web Applications and how they work with each other and the uses of each of these.
- We explored the vulnerabilities of Web Browsers, and looked into some of the practices which will reduce the Web Browser vulnerabilities or possibilities of attack on / through Web Browsers. In this context, we looked into the importance of appropriately and securely configuring the Web Browsers, importance of not having unnecessary add-ons, and how the regular patching up / updating of the Web Browser software will help.
- We elaborated on the vulnerabilities of Web Servers and looked into some of the practices which will reduce the Web Server vulnerabilities or possibilities of attack on Web Servers. In this context, we looked into the importance of modifying / removing default users / passwords, ensuring the removal of sample files / scripts, modification of the default configuration into appropriate configuration, fixing of the defects / loop holes in the Web Server software as well as the underlying Operating System.

- We discussed the vulnerabilities of Web Applications, and looked into some of the practices which will reduce Web Application vulnerabilities or possibilities of attack on Web Applications. In this context, we looked in detail into some of the important attacks like SQL Injection Attacks, Command Injection Attacks, Buffer Overflow Attacks, Cross-Site Scripting Attacks, Cookie Poisoning Attacks, and Session Hijacking Attacks. We also explored some of the important steps / actions we need to take to avoid becoming prey to Web Application attacks. We then explored the role of SSL and Digital Certificate in securing the communication between the Web Client and the Web Servers. We also provided a passing remark that Transport Layer Security (TLS) protocol has expanded upon the SSL concepts further, so that the persons interested can further study the same.