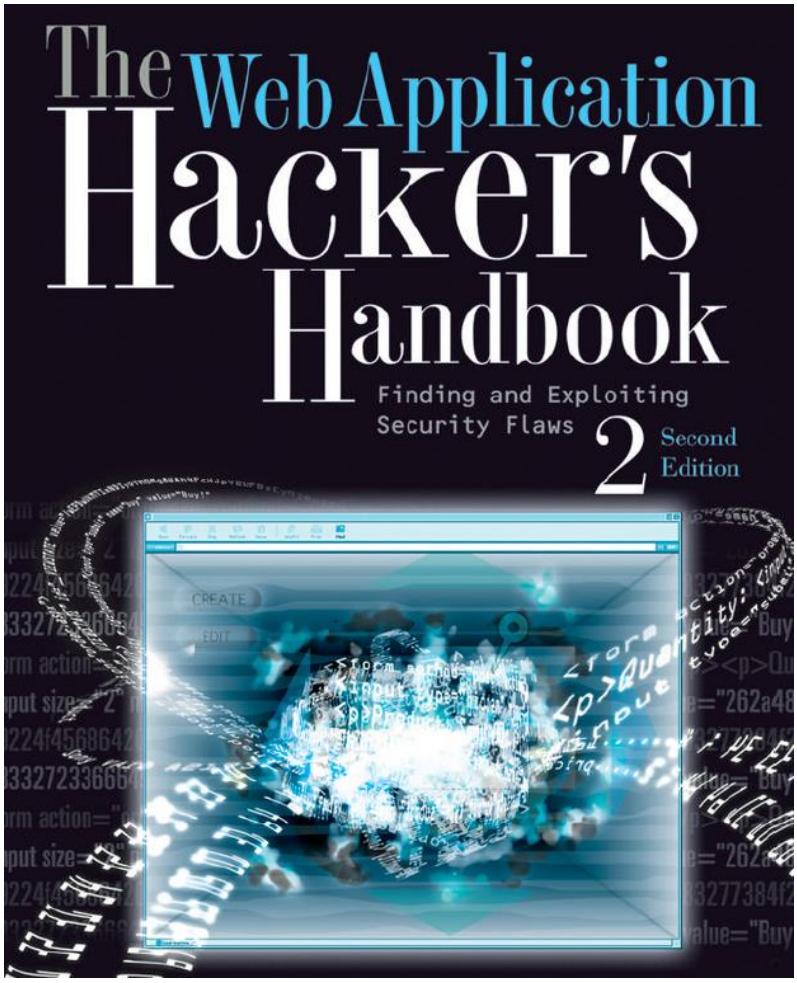



# INTRODUCTION

Areas to Focus - How to hack into a web application, steal sensitive data, and perform unauthorized Actions, vulnerabilities that web applications contain, detection of each type of vulnerability and how to exploit vulnerabilities to perform unauthorized actions, illustration about how different kinds of security flaw manifest themselves in today's web applications.

Chapter -1. Web Application (In)Security

The content presented to users is generated dynamically on the fly and is often tailored to each specific user. Much of the information processed is private and highly sensitive. Security, therefore, is a big issue.

The most serious attacks against web applications are those that expose sensitive data or gain unrestricted access to the back-end systems on which the application is running. High-profile compromises of this kind continue to occur frequently. For many organizations, however, any attack that causes system downtime is a critical event.

Application-level denial-of-service attacks can be used to achieve the same results as traditional resource exhaustion attacks against infrastructure. However, they are often used with more subtle techniques and objectives. They may be used to disrupt a particular user or service to gain a competitive edge against peers in the realms of financial trading, gaming, online bidding, and ticket reservations.

Since decades, compromises of prominent web applications have remained in the news. There is no sense that a corner has been turned and that these security problems are on the wane. By some measure, web application security is today the most significant battleground between attackers and those with computer resources and data to defend, and it is likely to remain so for the foreseeable future.

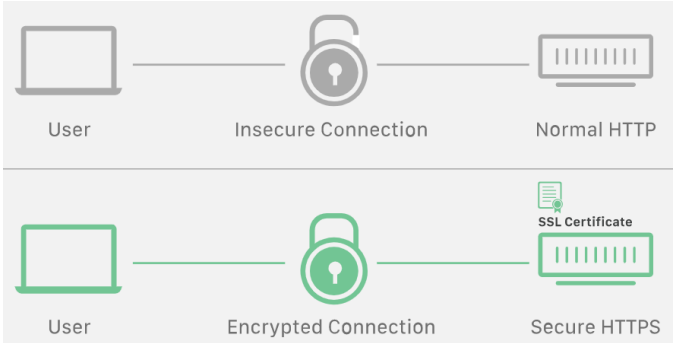
Security Issues

1. “This Site is Secure”

This is the basic, most basic building block or the 1st ingredient for the recipe of a tasty secure web application that a site must contain “SSL” (What in terms is said to be the SSL Certificate – The Domains that are most likely to start from https://and-so-on )

SSL - SSL, or Secure Sockets Layer, is an encryption-based Internet security protocol.

- SSL was first developed by Netscape in 1995 for the purpose of ensuring privacy, authentication, and data integrity in Internet communications.
- SSL is the predecessor to the modern TLS encryption used today.
- A website that implements SSL/TLS has "HTTPS" in its URL instead of "HTTP."




- How does SSL/TLS work?
- In order to provide a high degree of privacy, SSL encrypts data that is transmitted across the web. This means that anyone who tries to intercept this data will only see a garbled mix of characters that is nearly impossible to decrypt.
- SSL initiates an authentication process called a handshake between two communicating devices to ensure that both devices are really who they claim to be.

- SSL also digitally signs data in order to provide data integrity, verifying that the data is not tampered with before reaching its intended recipient.

Applications tested during 2007 and 2011 were found to be affected by some common categories of vulnerability:

**Broken authentication (62%)** — Vulnerability category that encompasses various defects within the application's login mechanism, which may enable an attacker to guess weak passwords, launch a brute-force attack, or bypass the login.

**Broken access controls (71%)** — Involves cases where the application fails to properly protect access to its data and functionality, potentially enabling an attacker to view other users' sensitive data held on the server or carry out privileged actions.

**SQL injection (32%)** — The vulnerability enabling an attacker to submit crafted input to interfere with the application's interaction with back-end databases. An attacker may be able to retrieve arbitrary data from the application, interfere with its logic, or execute commands on the database server itself.

**Cross-site scripting (94%)** — Vulnerability that enables an attacker to target other users of the application, potentially gaining access to their data, performing unauthorized actions on their behalf, or carrying out other attacks against them.

**Information leakage (78%)** — Involves cases where an application divulges sensitive information that is of use to an attacker in developing an assault against the application, through defective error handling or other behavior.

**Cross-site request forgery (92%)** — This flaw means that application users can be induced to perform unintended actions on the application within their user context and privilege level. The vulnerability allows a malicious web site visited by the victim user to interact with the application to perform actions that the user did not intend.

SSL in Web Application w.r.t Security

SSL is an excellent technology that protects the confidentiality and integrity of data in transit between the user's browser and the web server. It helps defend against eavesdroppers, and it can provide assurance to the user of the identity of the web server he is dealing with. But it does not stop attacks that directly target the server or client components of an application, as most successful attacks do. Specifically, it does not prevent any of the vulnerabilities just listed, or many others that can render an application critically exposed to attack. Regardless of whether they use SSL, most web applications still contain security flaws.

## The Core Security Problem - Users Can Submit Arbitrary Input

As with most distributed applications, web applications face a fundamental problem they must address to be secure. Because the client is outside of the application's control, users can submit arbitrary input to the server-side application.

The application must assume that all input is potentially malicious.

It takes steps to ensure that attackers cannot use crafted input to compromise the application by interfering with its logic and behavior, thus gaining unauthorized access to its data and functionality.

This core problem manifests itself in various ways:

(a) Users can interfere with any piece of data transmitted between the client and the server, including request parameters, cookies, and HTTP headers.

(b) Any security controls implemented on the client side, such as input validation checks, can be easily circumvented.

(c) Users can send requests in any sequence and can submit parameters at a different stage than the application expects, more than once, or not at all. Any assumption developers make about how users will interact with the application may be violated.

(d) Users are not restricted to using only a web browser to access the application. Numerous widely available tools operate alongside, or independently of, a browser to help attack web

applications. These tools can make requests that no browser would ordinarily make and can generate huge numbers of requests quickly to find and exploit problems.

The majority of attacks against web applications involve sending input to the server that is crafted to cause some event that was not expected or desired by the application's designer.

SSL does not stop attacker an attacker from submitting crafted input to the server. If the application uses SSL, this simply means that other users on the network cannot view or modify the data in transit. Because the attacker controls its end of the SSL tunnel, it can send anything it likes to the server through this tunnel. If any of the previously mentioned attacks are successful, the application is emphatically vulnerable, regardless of what its FAQ may tell.

NOTE - For an attacker targeting an organization, gaining access to the network or executing arbitrary commands on servers may not be what he wants to achieve. Often, and perhaps typically, what an attacker really wants is to perform some application-level action such as stealing personal information, transferring funds, or making cheap purchases. And the relocation of the security perimeter to the application layer may greatly assist an attacker in achieving these objectives.

Network administrators are familiar with the idea of preventing their users from visiting malicious web sites, and end users themselves are gradually becoming more aware of this threat. But the nature of web application vulnerabilities means that a vulnerable application may present no less of a threat to its users and their organization than a web site that is overtly malicious. Correspondingly, the new security perimeter imposes a duty of care on all application owners to protect their users from attacks against them delivered via the application.

# Chapter - 2. Core Defense Mechanisms

Security problem with web applications these days is that all user input is untrusted — gives rise to a number of security mechanisms that applications use to defend themselves against attack.
The defense mechanisms employed by web applications comprise the following
core elements:

- Handling user access to the application's data and functionality to prevent users from gaining unauthorized access
- Handling user input to the application's functions to prevent malformed input from causing undesirable behavior.
- Handling attackers to ensure that the application behaves appropriately when being directly targeted, taking suitable defensive and offensive measures to frustrate the attacker.
- Managing the application itself by enabling administrators to monitor its activities and configure its functionality.

Handling User Access:
The fundamentals to an application's overall security posture are

1. Authentication
2. Session Management
3. Access Control

Because of interdependencies, the overall security provided by these mechanisms is as strong as the weakest link in the chain. A defect in any single component may enable an attacker to gain unrestricted access to the application's functionality and data.

**Authentication:**

User involves establishing that the user is in fact who he claims to be. Without this facility, the application would need to treat all users as anonymous — the lowest possible level of trust. Despite Superficial simplicity, authentication mechanisms suffer from a wide range of defects in both Design and Implementation. When attacking a web application, investigation on a significant amount of attention to the various authentication-related functions should be taken care of. Surprisingly frequently, defects in authentication functionality enables gain of unauthorized access to sensitive data and functionality.

**Session Management:**

Session Management is a logical process for handing user to manage authenticated user's session. After successfully logging in to the application, the user accesses various pages and functions, making a series of HTTP requests from the browser. To enforce effective access control, the application needs a way to identify and process the series of requests that originate from each unique user.
Virtually all web applications meet this requirement by creating a session for each user and issuing the user a token that identifies the session. The session itself is a set of data structures held on the server that track the state of the user's interaction with the application.
Token - Token is a unique string that the application maps to the session.
When a user receives a token, the browser automatically submits it back to the server in each subsequent HTTP request, enabling the application to associate the request with that user. In terms of attack surface, the session management mechanism is highly dependent on the security of its tokens.
Majority of attacks against it seek to compromise the tokens issued to other users - If this is possible, Attacker can masquerade the victim user and application just as if user had actually authenticated as that user. A small number of applications dispense with the need for session tokens by
using other means of reidentifying users across multiple requests. If HTTP's built-in authentication mechanism is used, the browser automatically resubmits the user's credentials with each request, enabling the application to identify the user directly from these. In other cases, the application stores the state information on the client side rather than the server, usually in encrypted form to prevent tampering.

**Access Control:**
Access Control is the final step for the process of handling user access and is implemented by making and enforcing the correct decisions about whether each individual request should be permitted or denied.
The access control mechanism usually needs to implement some fine-grained logic, with different considerations being relevant to different areas of the application and different types of functionalities. Specific functions may implement transaction limits and other checks, all of which need to be properly enforced based on the user's identity. Because of the complex nature of typical access control requirements, this mechanism is a frequent source of security vulnerabilities that enable an attacker to gain unauthorized access to data and functionality. Developers often make flawed assumptions about how users will interact with the application and frequently make oversights by omitting access control checks from some application functions. Because of the prevalence of access control flaws, however, this effort is always a worthwhile investment when you are attacking a web application.

HANDLING USER INPUT
Variety of attacks against web applications involve submitting unexpected input, crafted to cause behavior that was not intended by application's designer. Correspondingly, a key requirement for an application's security defenses is that the application must handle user input in a safe manner. Input-based vulnerabilities can arise anywhere within an application's functionality, and in relation to practically every type of technology in common use. "Input validation" is often cited as the necessary defense against these attacks. No single protective mechanism can be employed everywhere, and defending against malicious input is often not as straightforward as it sounds.

Varieties of Input -
Typical web application processes user-supplied data in many different forms.
1. In many cases, an application may be able to impose very stringent validation checks on a specific item of input.
2. The application must tolerate a wider range of possible input.
3. An application may need to accept arbitrary input from users.
In addition to the various kinds of input that users enter using the browser interface, a typical application receives numerous items of data that began their life on the server and that are sent to the client so that the client can transmit them back to the server on subsequent requests. This includes items such as cookies and hidden form fields, which are not seen by ordinary users of the application but which an attacker can of course view and modify, for these cases, applications can often perform very specific validation of the data received.