Secure Application Design

Security considerations that should be part of the development cycle

1. Web applications
2. Client applications
3. Remote administration

After an application is written, it is deployed into an environment of some sort, where it remains for an extended period of time with only its original features to defend it from whatever threats, mistakes, or misuse it encounters.

Secure Development Lifecycle

A secure development lifecycle (SDL, or sometimes SSDL, for secure software development lifecycle) is essentially a development process that includes security practices and decision-making inputs.

Typically, an SDL contains three primary elements:

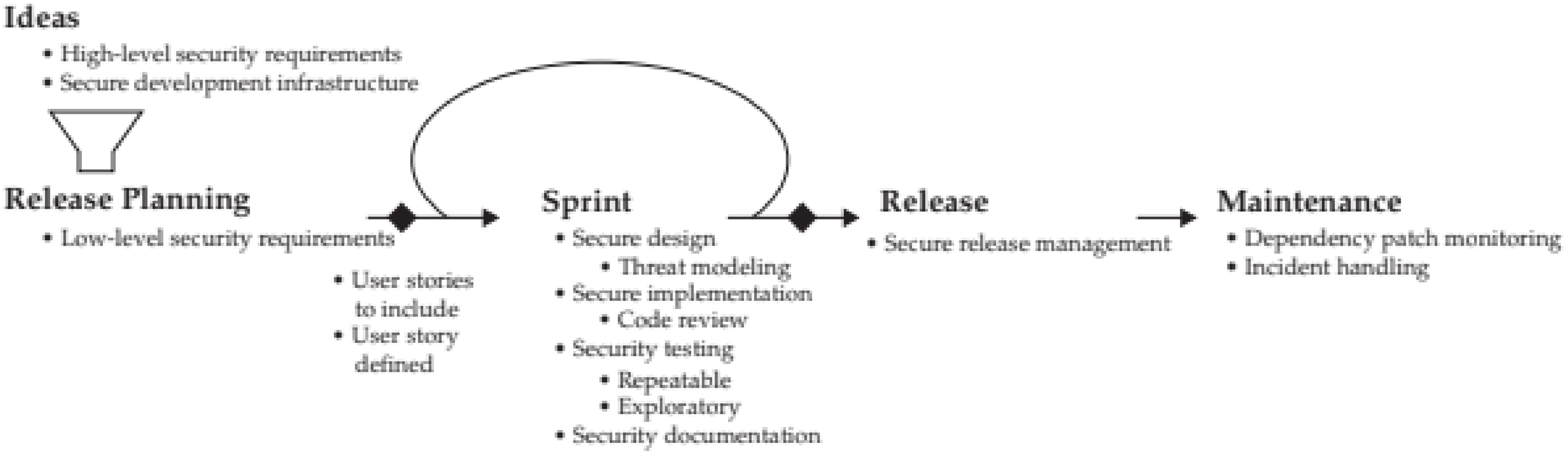
1. Security activities that don't exist at all in the original lifecycle; for instance, threat modeling

2. Security modifications to existing activities; for instance, adding security checks to existing peer reviews of code

Typically, an SDL contains three primary elements:

3. Security criteria that should affect existing decisions; for instance, the number of open high-severity security issues when a decision to ship is made

Secure development lifecycle in Agile



Application Security Practices

1. Security Training
2. Secure Development Infrastructure
3. Security Requirements
4. Secure Design

Application Security Practices

5. Threat Modeling

6. Secure Coding

7. Security Code Review

8. Security Testing

Application Security Practices

9. Security Documentation

10. Secure Release Management

11. Dependency Patch Monitoring

12. Product Security Incident Response

13. Decisions to Proceed

Web Application Security

Web application security, including the vulnerabilities attackers can exploit in insecure web applications in order to compromise a web server or deface a web site, and how developers can avoid introducing these vulnerabilities.

Web Application Security

- SQL injection
- Forms and scripts
- Cookies and session management
- General attacks

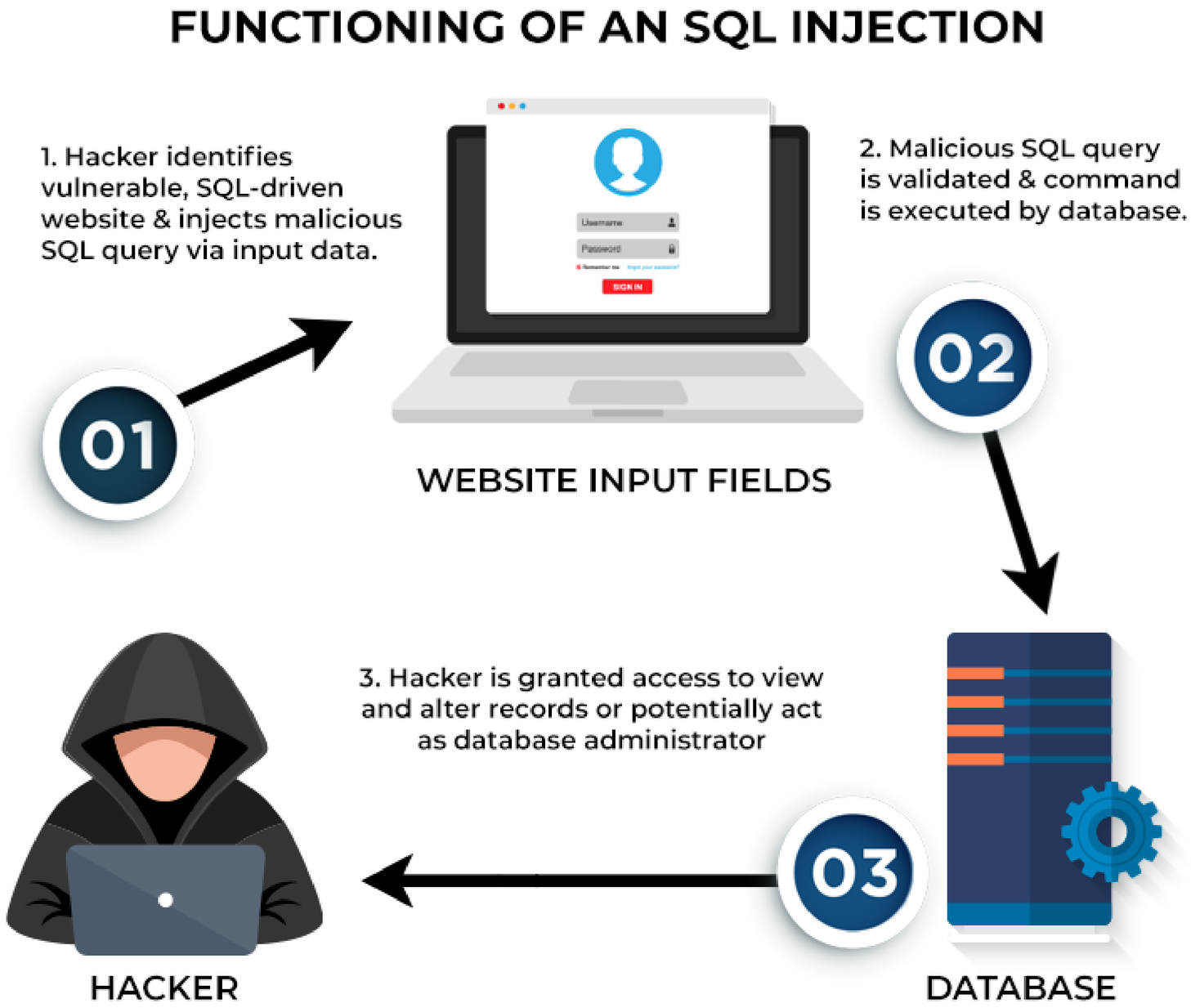
SQL Injection

SQL (Structured Query Language) is standardized by the American National Standards Institute (ANSI) and serves as a common language for communicating with databases.

Every database system adds some proprietary features to the basic ANSI SQL.

SQL injection is a technique to inject crafted SQL into user input fields that are part of web forms—it is mostly used to bypass custom logins to web sites.

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.



Simple Login Bypass

When SQL Injection Goes Bad

Procedure Invocations and SQL Administration

Simple Login Bypass

The most basic form of SQL injection is bypassing a login to a web site.

```
...
<form action="login.asp" method="post">
<p>Username:<input type=text name="username" /></p>
<p>Password:<input type=password name="password" /></p>
<p><input type=submit name="submit" value="login" /></p>
</form>
...
```

This page requests two pieces of information from the user (username and password), and it submits the information in the fields to login.asp. The login.asp file looks like this:

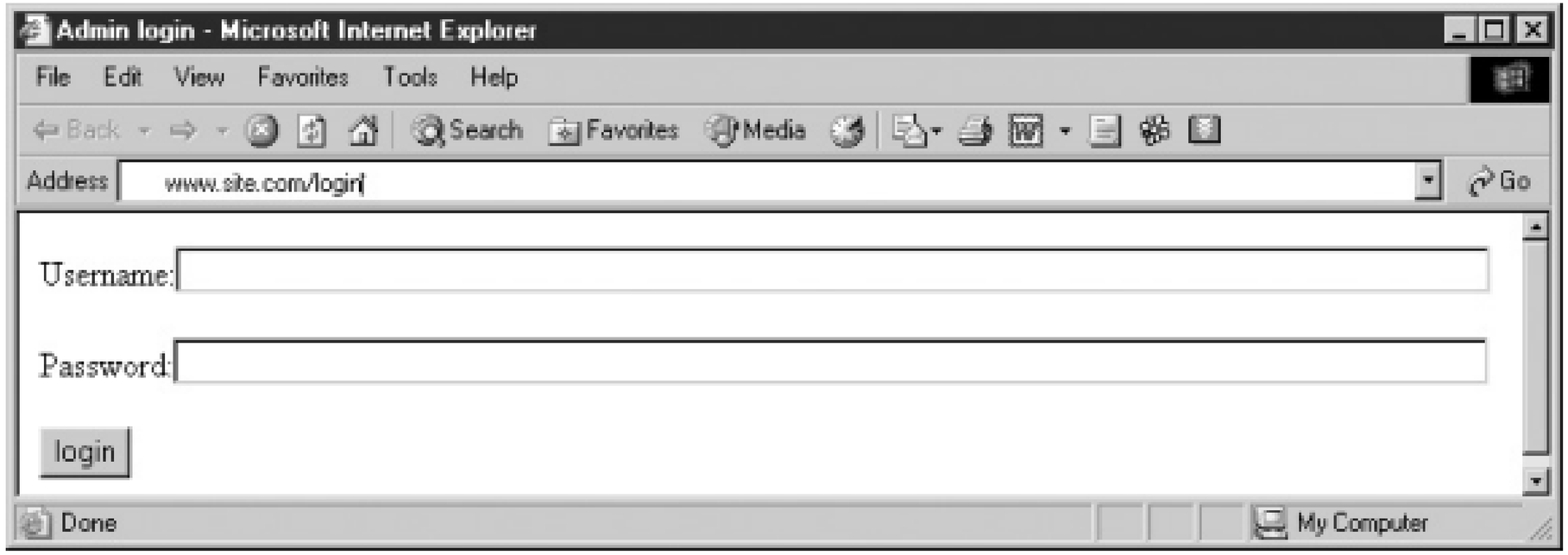


Figure 26-2 A typical login form for a web site

The following SQL statement is built when a user enters **admin** as the username and **somepassword** as the password (as shown in Figure 26-3):

```
select * from users where username='admin' and password='somepassword'
```

Let's go over the query:

- `select *` means "give me all the data"
- `from users` means "take it from the table called *users*"
- `where username='admin' and password='somepassword'` means "find a row where both the username is *admin* and the password is *somepassword*"

The username and password are placed inside the SQL string without any sanity checks.

(Sanity checks, known as “form field validation,” are performed to make sure user input doesn’t contain any characters an attacker could use to modify the SQL statement.)

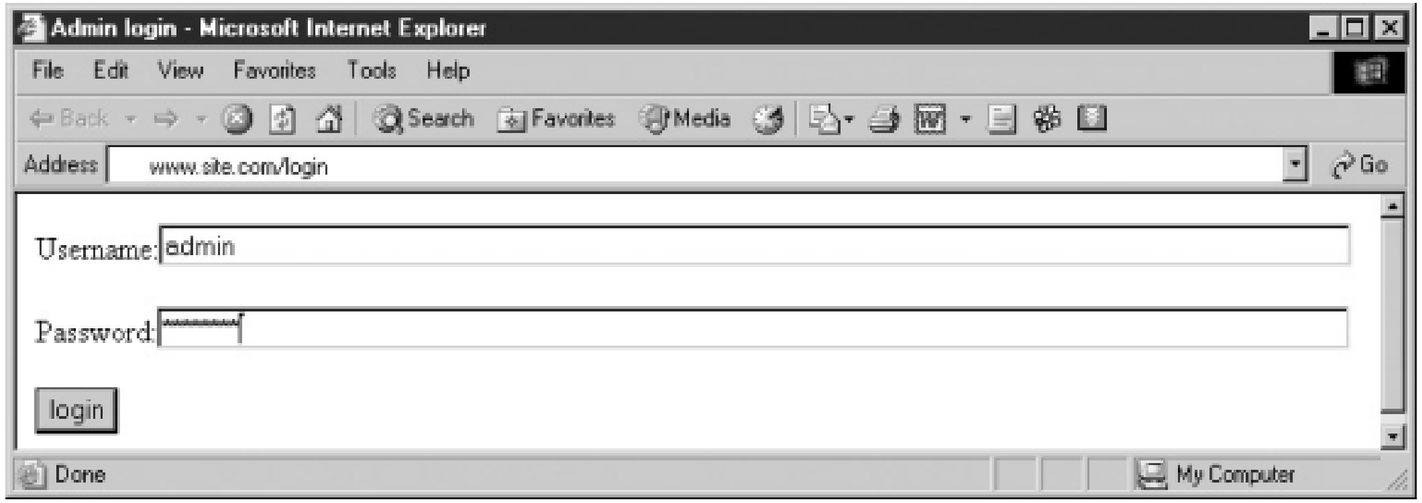


Figure 26-3 A user signing in using the login web form

This means that an attacker can inject custom code into the user input fields without being detected.

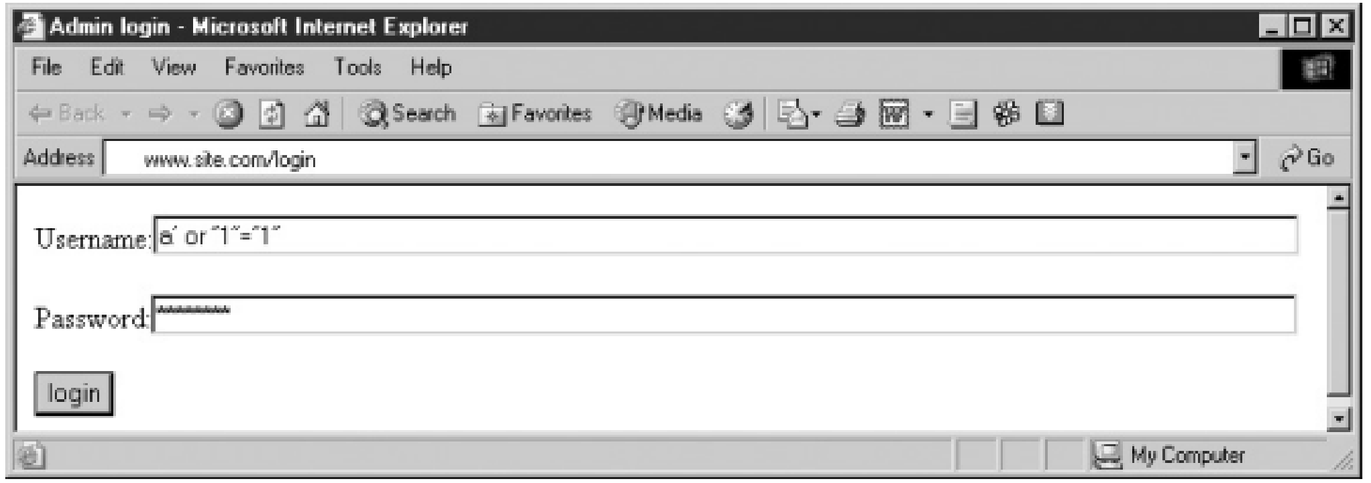


Figure 26-4 An attacker attacking the login web form with SQL injection

This attack was made possible because the programmer didn't filter the apostrophe (') inside the user input fields, which allowed the attacker to break the SQL syntax and enter custom code.

The following code solves this problem by filtering the apostrophes (every occurrence of ' in the user input is removed):

```
strLoginSQL="select * from users where username='" & Replace
 (Request.Form("username"),"'","") & "' and password='" & Replace
 (Request.Form("password"),"'","") & "'"
```

Solutions for SQL Injection

Developers and administrators can take a number of different steps in order to solve the SQL injection problem.

These are some solutions for developers:

- Filter all input fields for apostrophes (') to prevent unauthorized logins.
- Filter all input fields for SQL commands like `insert`, `select`, `union`, `delete`, and `exec` to prevent server manipulation. (Make sure you do this after filtering for the apostrophes.)
- Limit input field length (which will limit attackers' options), and validate the input length with server-side scripts.

- Use the option to filter “escape characters” (characters that can be used to inject SQL code, such as apostrophes) if the database offers that function.
- Place the database on a different computer than the web server. If the database is hacked, it’ll be harder for the attacker to reach the web server.
- Limit the user privileges of the server-side script. A common practice is to use the administrative user when logging in from the server-side script to the database, but this can allow an attacker to run database tasks (such as modifying tables or running stored procedures) that require the administrative user. Assign a user with minimal privileges for this purpose.
- Delete all unneeded extended stored procedures to limit attackers’ possibilities.
- Place the database in a separate container (behind a firewall), separated from the web container and application server.

Forms and Scripts

Forms are used to allow a user to enter input, but forms can also be used to manage sessions and to transfer crucial data within the session (such as a user or session identifier).

Attackers can exploit the data embedded inside forms and can trick the web application into either exposing information about another user or to charge a lower price in e-commerce applications.

Three methods of exploiting forms are these:

- Disabling client-side scripts
- Passing parameters in the URLs
- Passing parameters via hidden fields

Client-Side Scripts

Some developers use client-side scripts to validate input fields in various ways:

- Limit the size of the input fields
- Disallow certain characters (such as apostrophes)
- Perform other types of validation (these can be specific to each site)

By disabling client-side scripting (either JavaScript or VBScript), this validation can be easily bypassed. A developer should validate all fields at the server side.

Passing Parameters via URLs

A form has two methods of passing data: post and get. The post command sends the data in the content stream and the get command sends the data in the URL.

Attackers
can exploit the get command to send invalid or incorrect data, or to send malicious code.

Passing Data via Hidden Fields

The post method sends the data using the POST HTTP command. Unlike get, this method doesn't reveal the data in the URL, but it can be exploited rather easily as well.

Managing Session Information

Using GUIDs

Encrypting Data

Using GUIDs A globally unique identifier, or GUID, is a 128-bit randomly generated number that has 2128 possible values. GUIDs can be used as user identifiers by the web application programmer.

Cookies and Session Management

Web sessions are implemented differently by each server-side scripting technology, but in general they start when the user enters the web site, and they end when the user closes the browser or the session times out.

Sessions are used to track user activities, such as a user adding items to their shopping cart—the site keeps track of the items by using the session identifier.

Attackers can abuse both sessions and cookies, and this section will deal with the various risks:.

- Session theft
- Managing sessions by sending data to the user
- Web server cookie attacks
- Securing sessions

Client Application Security

Application security is mainly controlled by the developer of the application. The administrator
can tighten the security for some applications

Writing a secure application is difficult, because every aspect of the application, like the GUI, network connectivity, OS interaction, and sensitive data management, requires extensive security knowledge in order to secure it.

From the administrator's point of view, there are a number of security issues to keep in mind:

- Running privileges
- Administration
- Application updates
- Integration with OS security
- Adware and spyware
- Network access

If the application is exploited by attackers

Low privileges protect the computer from an embedded Trojan

Running Privileges

An administrator should strive to run an application with the fewest privileges possible

The user won't be able to save data in sensitive areas (such as areas belonging to the OS) or even access key network resources

Application Administration

Most applications offer some type of interface for administration (mostly for application configuration), and each administration method poses security risks that must be addressed, such as these:

INI/Conf Files

The most basic method of administrating an application is to control it via text-based files. To secure such an application, the administrator needs to limit access to the configuration files either by using built-in OS access management,

GUIs

Most applications have a GUI for administrating them. In addition to providing security at the GUI level, the administrator should provide security for the communications between the GUI and the application.

Web-Based Control

A popular way to allow application administration is via a web interface, which doesn't require a dedicated client and can be used from multiple platforms.

Application Updates

Keeping applications up to date with the latest security patches is one of the most important security measures that you can take.

- Manual updates
- Automatic updates
- Semi-automated updates
- Physical updates

Remote Administration Security

If an attacker manages to penetrate the administration facilities, other security measures can be compromised or bypassed.

- **Relocated servers** An administrator needs an interface to administer any relocated web servers (computers that belong to an organization but that are physically located at the ISP).
- **Outsourced services** Managing security products requires knowledge that some organizations don't possess, so they often outsource their entire security management to a firm specializing in that area. In order to save costs, that firm needs to manage all the security products through the Internet.
- **Physical distance** An administrator may need to manage a large number of computers in the organization. Some organizations span several buildings (or cities), and physically attending the computers can be a tedious and time-consuming task. Additionally, physical access may be limited to the actual data centers.

Remote Administration Using a Web Interface

Using a web interface to remotely administer an application or a computer has many advantages, but it also has its costs, and some advantages are also disadvantages.

These are some advantages of remote web administration:

- **Quick development time** Developing a web interface is faster than developing a GUI client, in terms of development, debugging, and deployment.
- **OS support** A web interface can be accessed from all the major OSs by using a browser (unless the developers used an OS-specific solution, like ActiveX, which only runs on Windows).
- **Accessibility** A web interface can be accessed from any location on the Internet. An administrator can administrate even if he's not in the office.
- **User learning curve** An administrator knows how to use a browser, so the learning curve for the administrator will be shorter.

Although remote web administration has some disadvantages,

- **Accessibility** Because web administration is accessible from anywhere on the Internet, it's also accessible to an attacker who may try to hack it.
- **Browser control** Because a browser controls the interface, an attacker doesn't need to deploy a specific product control GUI (which might be hard to come by).
- **Support** Web-based applications are typically easier to support and maintain.

HTTP Authentication Methods

Basic authentication

Digest authentication

Secure Sockets Layer (SSL)

Encrypted basic authentication

CAPTCHA