

Fundamentals of Web Development

Third Edition by Randy Connolly and Ricardo Hoar



Chapter 16

Security

In this chapter you will learn . . .

- About core security principles and practices
- Best practices for authentication systems and data storage
- About public key cryptography, SSL, and certificates
- How to proactively protect your site against common attacks

Security Principles

Security theory and practice will guide you in that never-ending quest to defend your data and systems, which you will see, touches all aspects of software development.

Not only is a malicious hacker on a tropical island a threat but so too is a sloppy programmer, a disgruntled manager, or a naive secretary. Moreover, threats are ever changing, and with each new counter measure, new threats emerge to supplant the old ones.

You must draw from those fields to learn many foundational security ideas. Later, you will apply these ideas to harden your system against malicious users and defend against programming errors.

Information Security

Information security is the holistic practice of protecting information from unauthorized users.

Information assurance ensures that data is not lost when issues do arise.

At the core of information security is the **CIA triad**: *confidentiality, integrity, and availability*

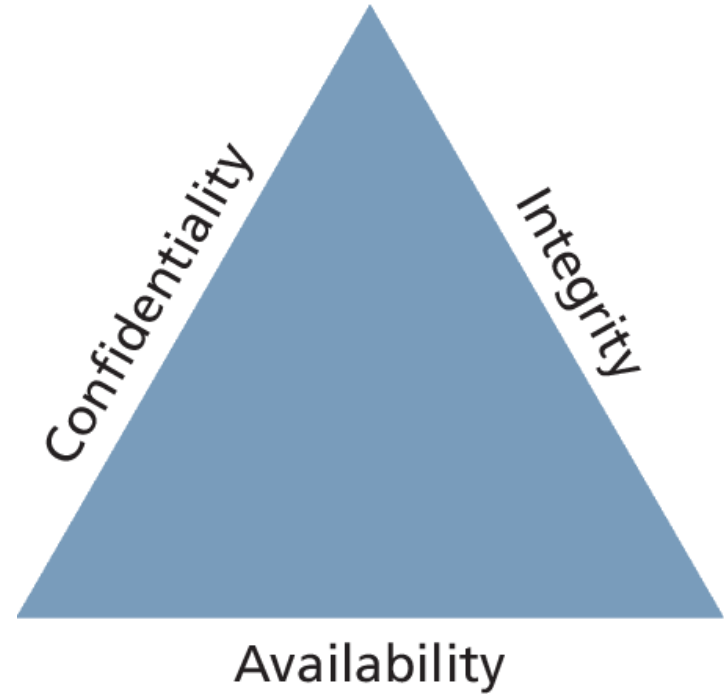
In addition to the triad, there are ISO standards ISO/IEC 27002-270037 that speak directly (and thoroughly) about security techniques and are routinely adopted by governments and corporations the world over.

CIA Triad

Confidentiality is the principle of maintaining privacy for the data you are storing, transmitting, and so forth

Integrity is the principle of ensuring that data is accurate and correct.

Availability is the principle of making information available to authorized people when needed.



Risk Assessment and Management

Risk assessment uses the concepts of actors, impacts, threats, and vulnerabilities to determine where to invest in defensive countermeasures.

- **Actors** refers to the people who are attempting to access your system. They can be categorized as internal, external, and partners.
- The **impact** of an attack depends on what systems were infiltrated and what data was stolen or lost. The impact relates back to the CIA triad since impact could be the loss of availability, confidentiality, and/or integrity
- A **threat** refers to a particular path that a hacker could use to exploit a vulnerability and gain unauthorized access to your system.
- **Vulnerabilities** are the security holes in your system.

STRIDE

Broadly, threats can be categorized using the **STRIDE** mnemonic, developed by Microsoft, which describes six areas of threat:

- **S**poofing—The attacker uses someone else's information to access the system.
- **T**ampering—The attacker modifies some data in nonauthorized ways.
- **R**epudiation—The attacker removes all trace of their attack so that they cannot be held accountable for other damages done.
- **I**nformation disclosure—The attacker accesses data they should not be able to.
- **D**enial of service—The attacker prevents real users from accessing the systems.
- **E**levation of privilege—The attacker increases their privileges on the system, thereby getting access to things they are not authorized to.

Assessing Risk

In risk assessment, you begin by identifying the **actors**, **vulnerabilities**, and **threats** to your information systems.

Table 16.1 illustrates the relationship between the probability of an attack and its impact on an organization.

The table weighs impact on the x scale and probability on the y scale.

		Impact (n ²)				
		Very low	Low	Medium	High	Very high
Probability	Very high	5	10	20	40	80
	High	4	8	16	32	64
	Medium	3	6	12	24	48
	Low	2	4	8	16	32
	Very low	1	2	4	8	16

Security Policy

- **Usage policy** defines what systems users are permitted to use, and under what situations.
- **Authentication policy** controls how users are granted access to the systems. These policies most often manifest as simple **password policies**
- **Legal policies** define a wide range of things including data retention and backup policies as well as accessibility requirements

Good policies aim to modify the behavior of internal actors, but will not stop foolish or malicious behavior by employees.

Business Continuity

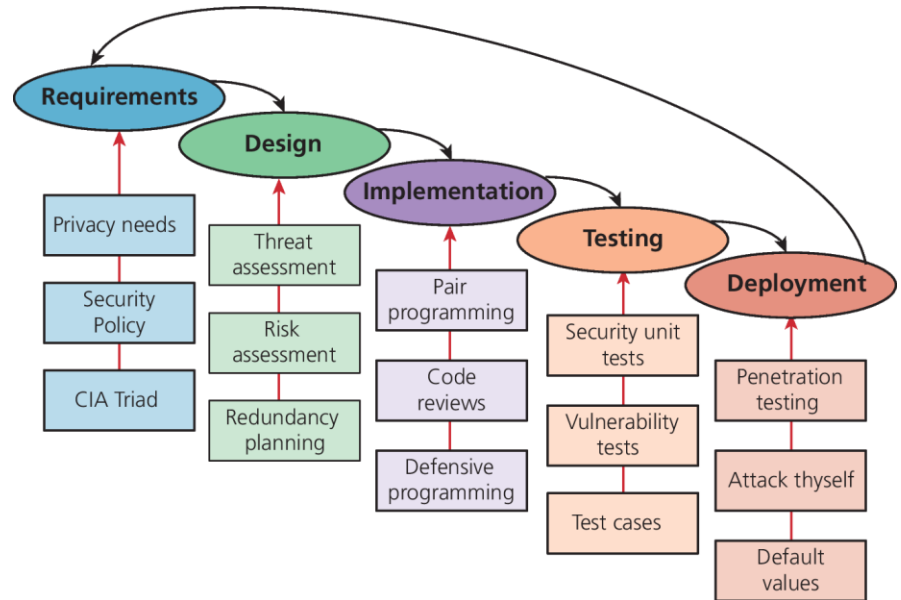
The best way to be prepared for the unexpected is to plan while times are good and thinking is clear. Some considerations include:

- **Admin Password Management**
- **Backups and Redundancy**
- **Geographic Redundancy**
- **Stage Mock Events**
- **Auditing**

Secure by Design

Secure by design is a software engineering principle that tries to make software better by acknowledging that there are malicious users out there and addressing it.

By continually distrusting user input (and even internal values) throughout the design and implementation phases, you will produce more secure software than if you didn't consider security at every stage.



Secure by Design (ii)

In a **code review** system, programmers must have their code peer-reviewed before committing it to the repository.

Unit testing is the practice of writing small programs to test your software as you develop it.

Pair programming is the technique where two programmers work together at the same time on one computer.

Security testing is the process of testing the system against scenarios that attempt to break the final system.

Secure by default aims to make the default settings of a software system

secure

Social Engineering

Social engineering is the broad term given to describe the manipulation of attitudes and behaviors of a populace, often through government or industrial propaganda and/or coercion.

No one would click a link in an email that said *click here to get a virus*, but they might click a link to *get your free vacation*.

Authentication Factors

To achieve both *confidentiality* and *integrity*, the user accessing the system must be who they purport to be.

Authentication is the process by which you decide that someone is who they say they are and therefore permitted to access the requested resources.

Authentication factors are the things you can ask someone for in an effort to validate that they are who they claim to be.

- Knowledge factors
- Ownership factors
- Inherence Factors

Single versus Multifactor Authentication

Single-factor authentication is the weakest and most common category of authentication system where you ask for only one of the three factors.

Multifactor authentication is where two distinct factors of authentication must pass before you are granted access.

To enhance authentication, one should use multiple factors rather than multiple instances of the same factor.

Approaches to Web Authentication

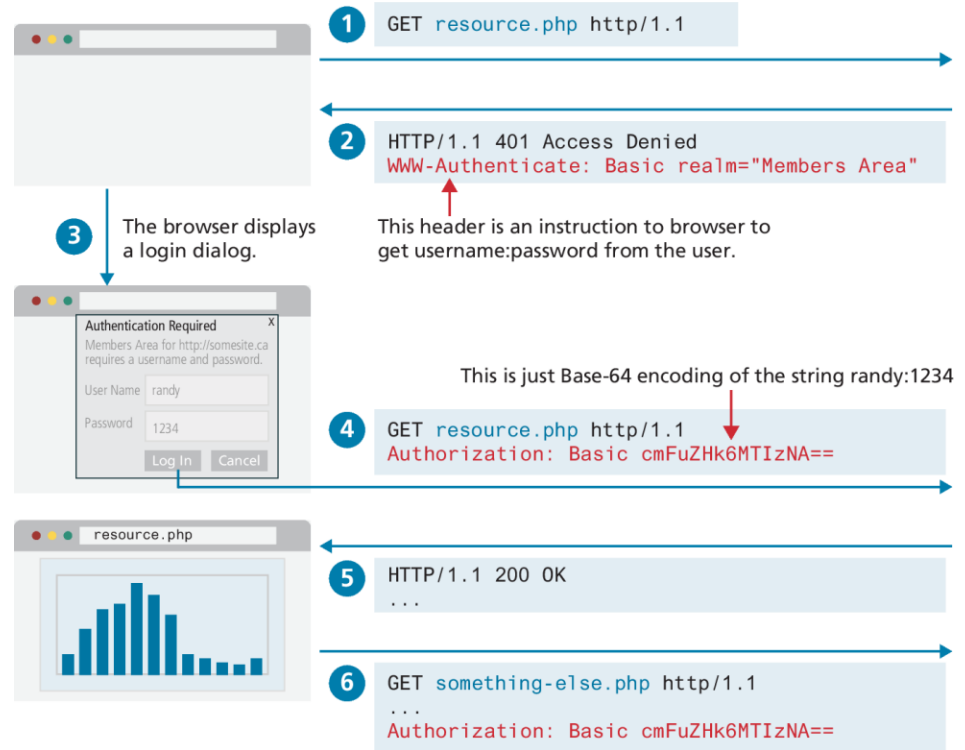
In web applications, there are four principle strategies used for authentication:

- Basic HTTP Authentication
- Form-Based Authentication
- Token HTTP Authentication
- Third-Party Authentication

Basic HTTP Authentication

HTTP Basic Authentication is a way for the server to indicate that a username and password is required to access a resource.

This approach is sometimes referred to as an example of challenge-response authentication, in that the server provides a “challenge” and the client has to *immediately* provide a response.

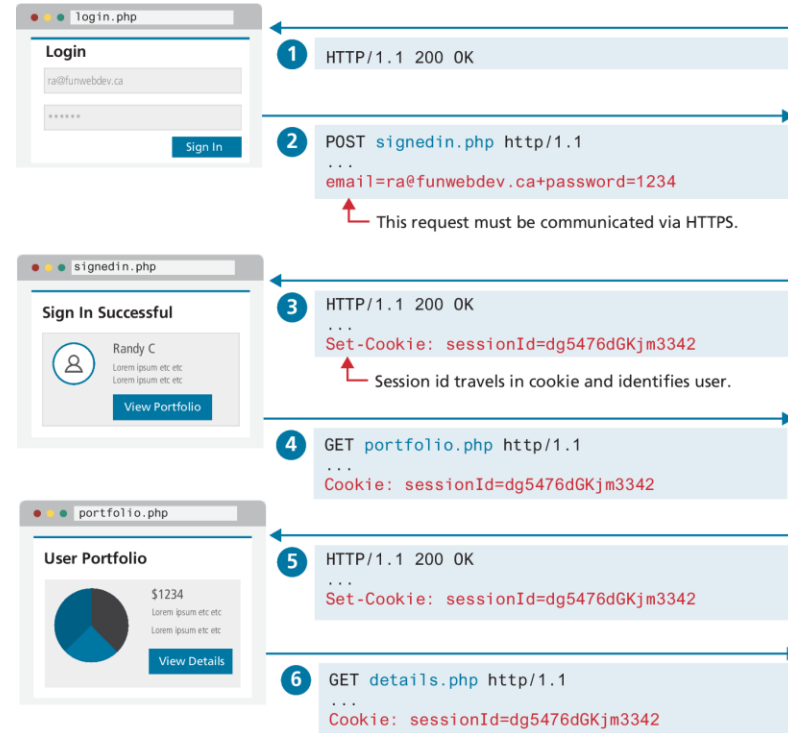


Form-Based Authentication

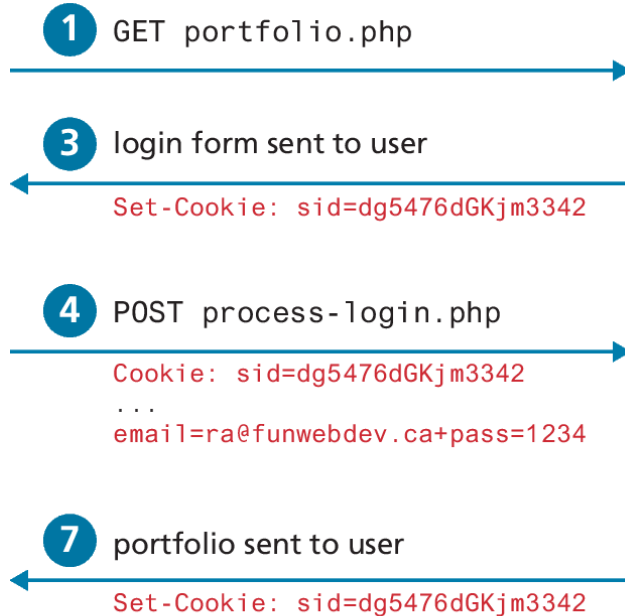
When secure communication is needed, websites generally use some form of **form-based authentication**

An HTML form is presented to the user, and the login credential information is sent via regular HTTP POST.

Form authentication keeps track of the user's login status. The example in the diagram is using a session cookie and must check credentials.



Managing HTTP Form login status



Server

- 2 Create session for this new user

Session ID	Logged-In?
...	
dg5476dGKjm3342	No

- 5 Verify credentials

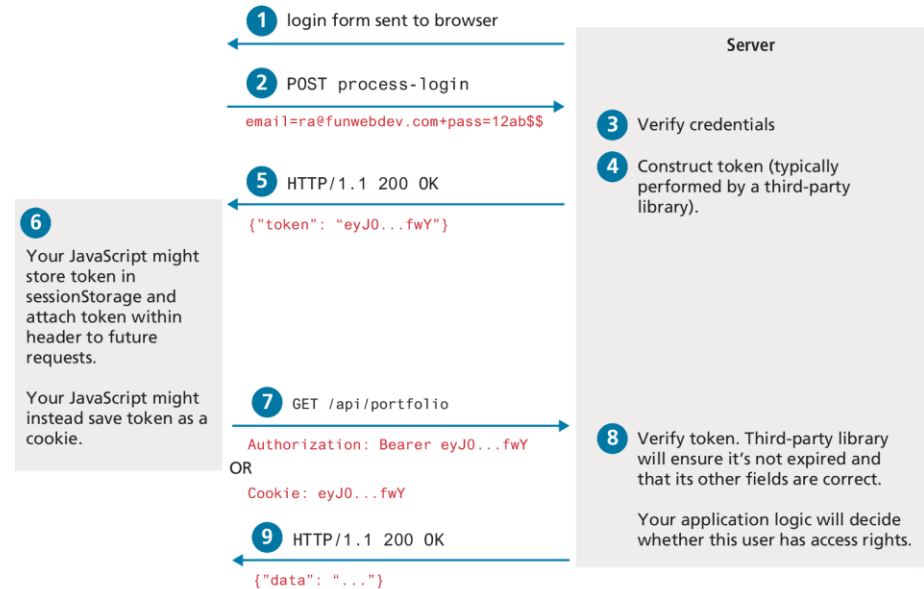
- 6 Change status in session state

Session ID	Logged-In?
...	
dg5476dGKjm3342	Yes

HTTP Token Authentication

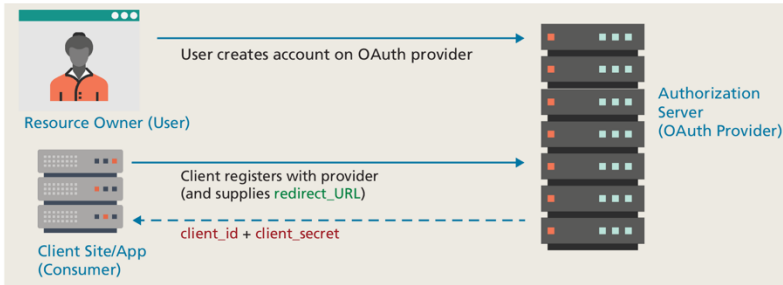
HTTP Token Authentication (also known more formally as **Bearer Authentication**) is a form of HTTP authentication that is commonly used in conjunction with form authentication, as well as with APIs and other services without a user interface

Token authentication provides a way to implement **stateless authentication**.

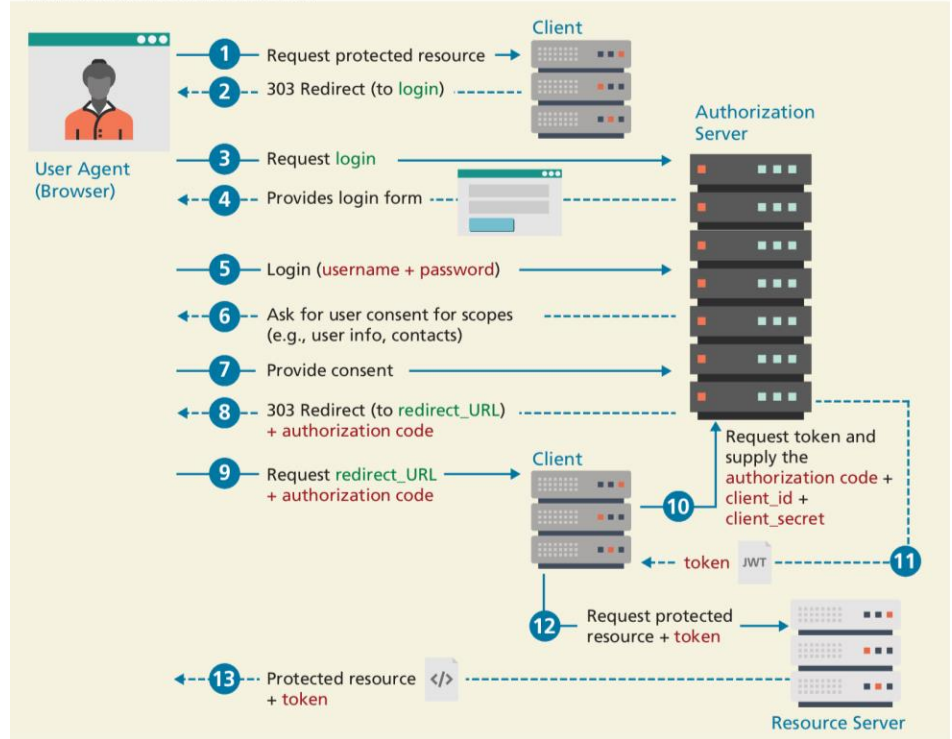


Third-Party Authentication

Prior to Authorization

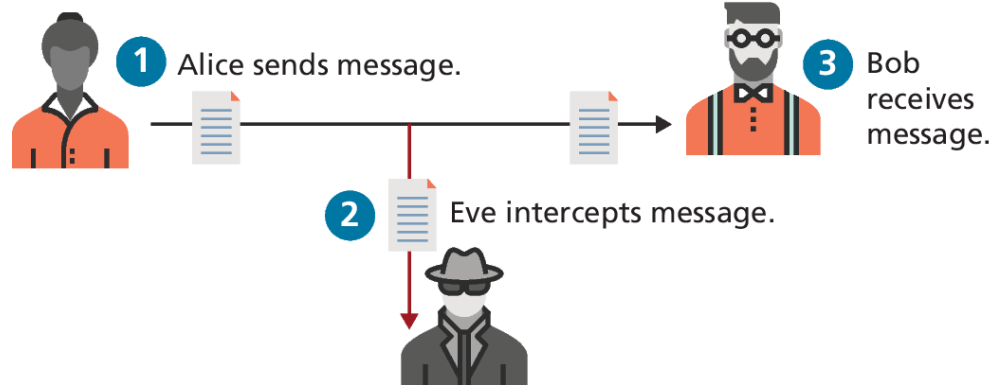


Authorization Code Grant Flow



Cryptography

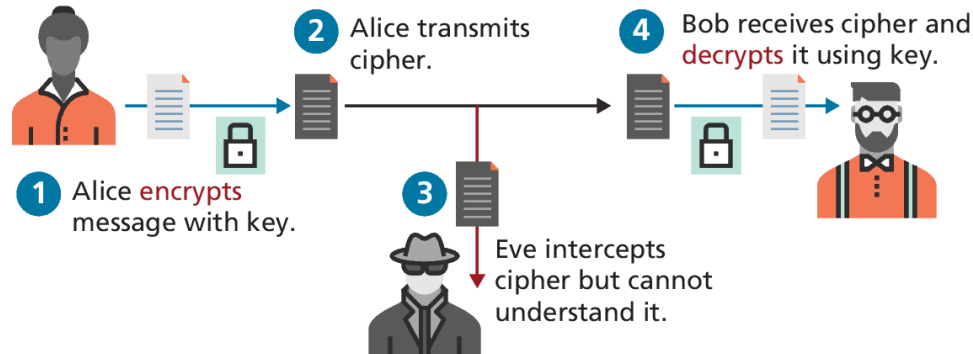
At a basic level we are trying to get a message from one actor (we will call her **Alice**), to another (**Bob**), without an eavesdropper (**Eve**) intercepting the message (as shown in Figure 16.9)



Cryptography (ii)

A **cipher** is a message that is scrambled so that it cannot easily be read, unless one has some secret knowledge. This secret is usually referred to as a **key**.

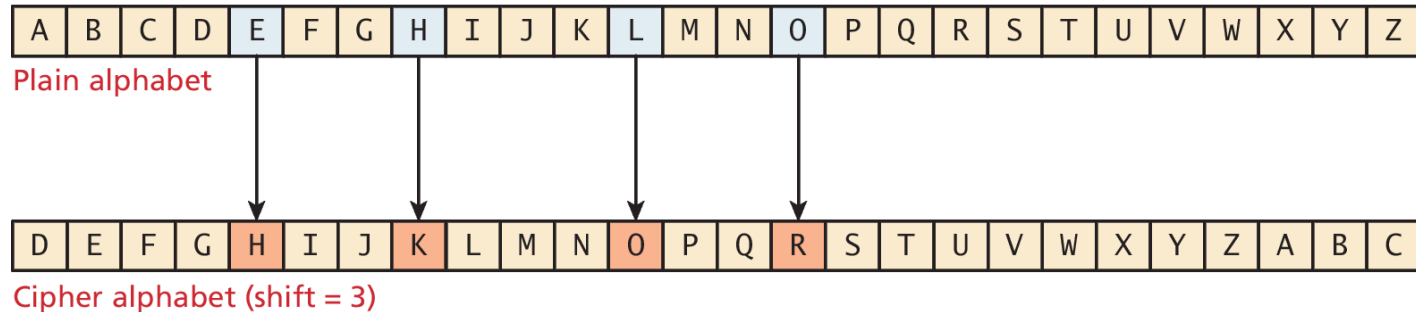
Alice encrypts the message (**encryption**) and Bob, the receiver, decrypts the message (**decryption**), both using their keys.



Substitution Ciphers

A **substitution cipher** is one where each character of the original message is replaced with another character according to the encryption algorithm and key.

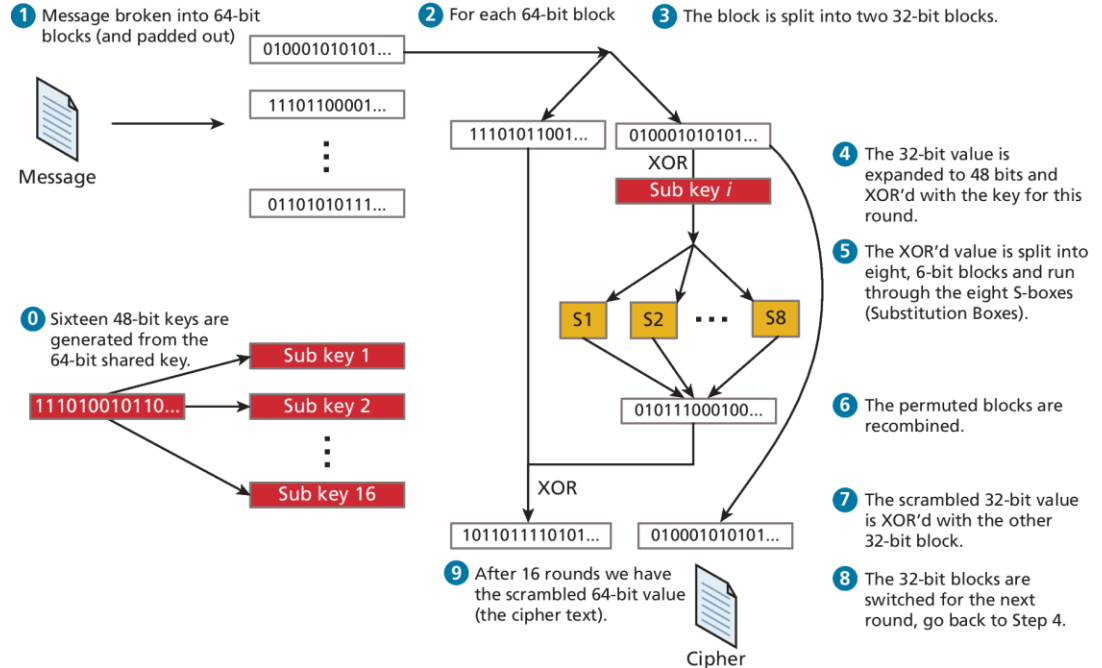
The Caesar cipher is a substitution cipher where every letter of a message is replaced with another letter, by shifting the alphabet over an agreed number (from 1 to 25). The message HELLO, for example, becomes KHOOR when a shift value of 3 is used



Modern Block Ciphers

Block ciphers encrypt and decrypt messages using an iterative replacing of a message with another scrambled message using 64 or 128 bits at a time (the block)

The Data Encryption Standard (DES) and its replacement, the Advanced Encryption Standard (AES) are two-block ciphers still used in web encryption today.



Public Key Cryptography

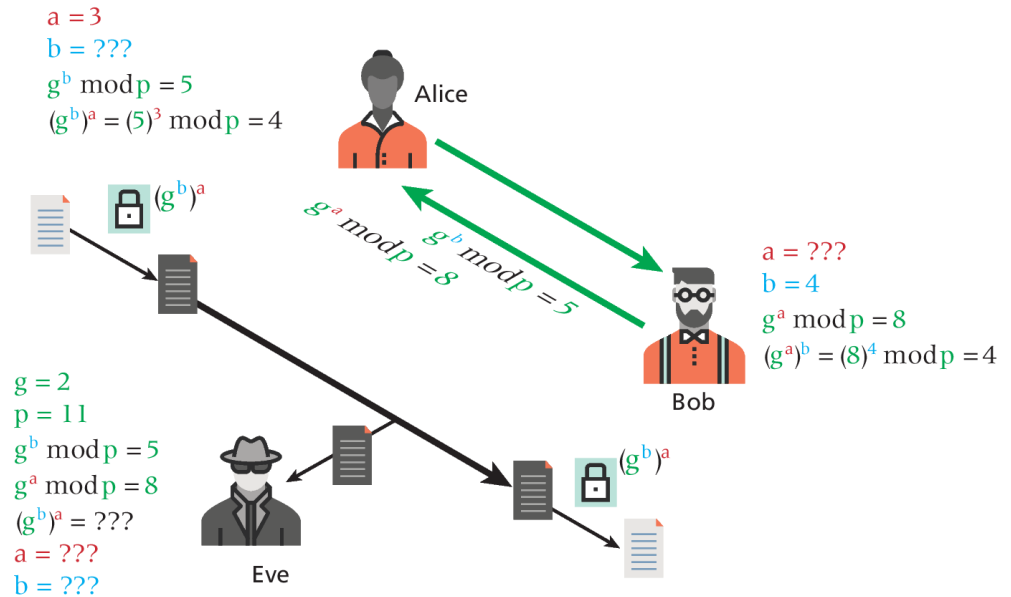
All of the ciphers we have covered thus far use the same key to encode and decode, so we call them **symmetric ciphers**. The problem is that we have to have a shared private key!

Public key cryptography (or **asymmetric cryptography**) solves the problem of the secret key by using two distinct keys: a public one, widely distributed and another one, kept private.

Diffie-Hellman key exchange are accessible to a wide swath of readers, and subsequent algorithms (like RSA) apply similar thinking but with more complicated mathematics.

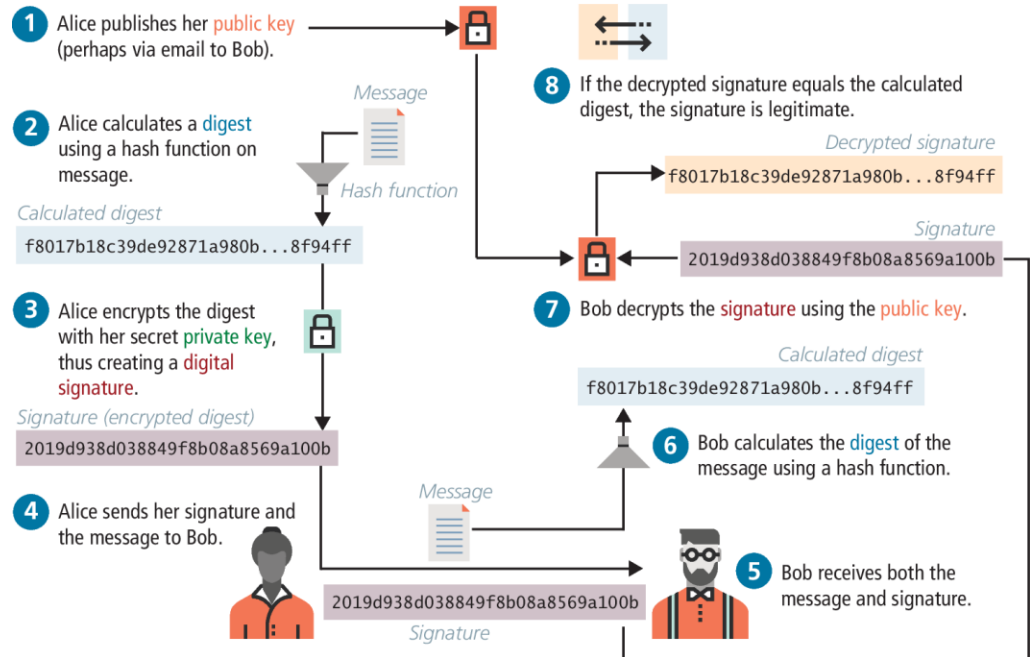
Diffie-Hellman Key Exchange

The essence of the key exchange is that this g^{ab} can be used as a *symmetric* key for encryption, but since only g^a and g^b are transmitted the symmetric key isn't intercepted.



Digital Signatures

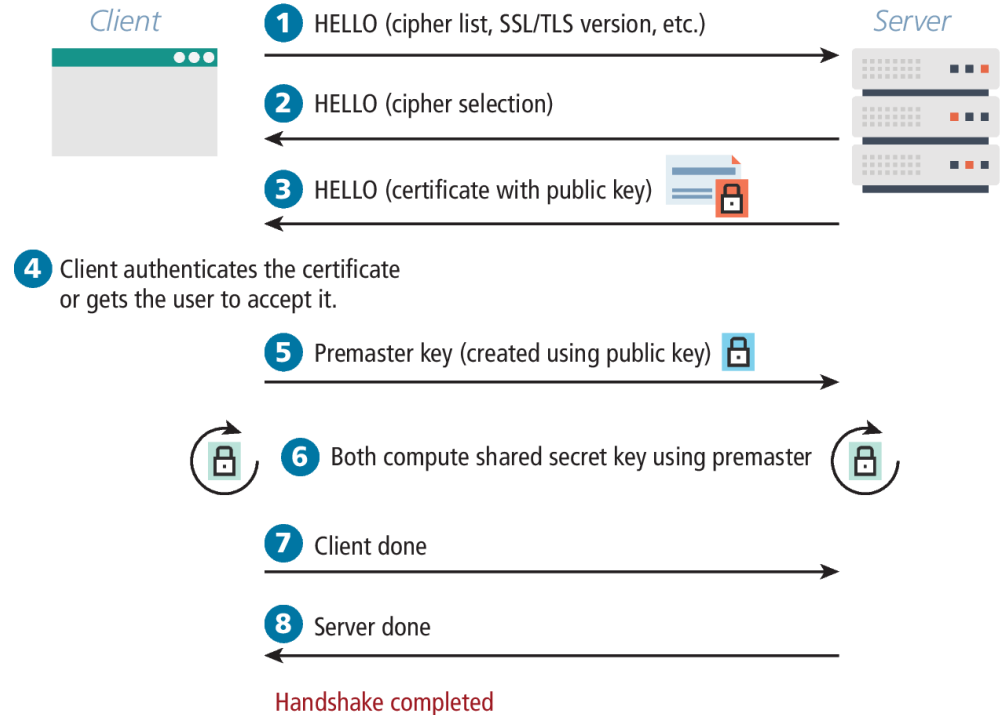
A **digital signature** is a mathematically secure way of validating that a particular digital document was created by the person claiming to create it (authenticity), was not modified in transit (integrity), and to prevent sender from denying that she or he had sent it (nonrepudiation).



SSL/TLS Handshake

The client initiates the handshake by sending the time, the version number, and a list of cipher suites its browser supports to the server.

The server, in response, sends back which of the client's ciphers it wants to use as well as a **certificate**, which includes a public key.

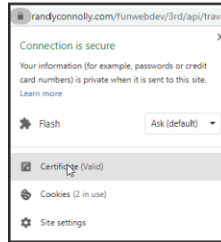


Certificates and Authorities

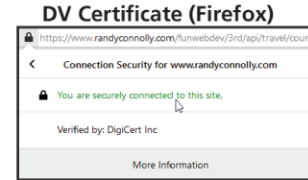
A **Certificate Authority** (CA) allows users to place their trust in the certificate since a trusted, independent third party signs it.

There are three types of SSL certificates that can be purchased:

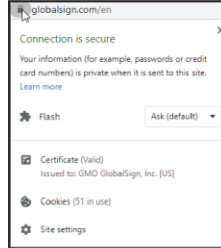
- **Domain-validated certificates**
- **Organization-validated certificates**
- **Extended-validation certificates**



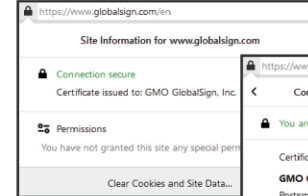
DV Certificate (Chrome)



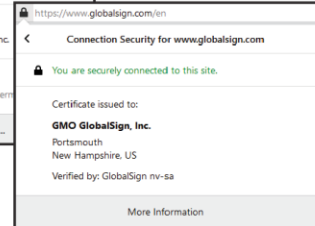
DV Certificate (Firefox)



OV/EV Certificate (Chrome)



OV/EV Certificate (Firefox)



Domain-Validated (DV) Certificates

This is the most affordable option. Most CAs will only verify the email listed in the whois registration database (see Chapter 2) via a confirmation link.

As a consequence, the process of obtaining the certificate is very fast.

Many CAs also offer wildcard certificates or even multi-domain certificates that allow an organization to secure a wider range of domains they own.

Organization-Validated (OV) Certificates

With these certificates, the CA takes additional steps to verify the identity of the organization seeking the certificate.

While it will perform the same domain verification as with domain-validated certificates, it also typically requests a variety of business documents, such as a government license, bank statement, or legal incorporation records. As a consequence, this type of certificate typically takes several days and is more expensive.

Extended-Validation (EV) Certificates

These are similar to the organization-validated certificates, but have even stricter requirements around the documentation that needs to be provided by the purchaser.

The rationale for choosing this option is similar to that of the OV: it's to improve the trust of the end user.

Migrating to HTTPS

Coordinating the migration of a website can be a complex endeavor involving multiple divisions of a company. In addition to business considerations, there are also some technical considerations in migrating to HTTPS.

- **Mixed Content**
- **Redirects from Old Site**
- **Preventing HTTP Access**

Mixed Content

In order to fully address a transition from HTTP to HTTPS, developers have to consider every place a HTTP reference exists in their code

Hardcoded links (which are bad style—and now we see why) should be replaced with relative links that easily transform according to the protocol being used. These links might include the following:

- Internal links within the site.
- External links to frameworks delivered through a CDN.
- Any links or references generated by server code that might include a hardcoded http.

Redirects from Old Site

Once you move your site over to HTTPS, there likely be links remaining from third-party sites to your former HTTP URLs and it's important that that such links still work.

To enable such behavior for every possible resource, both Apache and Nginx server provide mechanisms for redirecting HTTP requests for a resource to HTTPS requests.

For instance, in Apache, the following two lines will send a 301 code and the new link location on https.

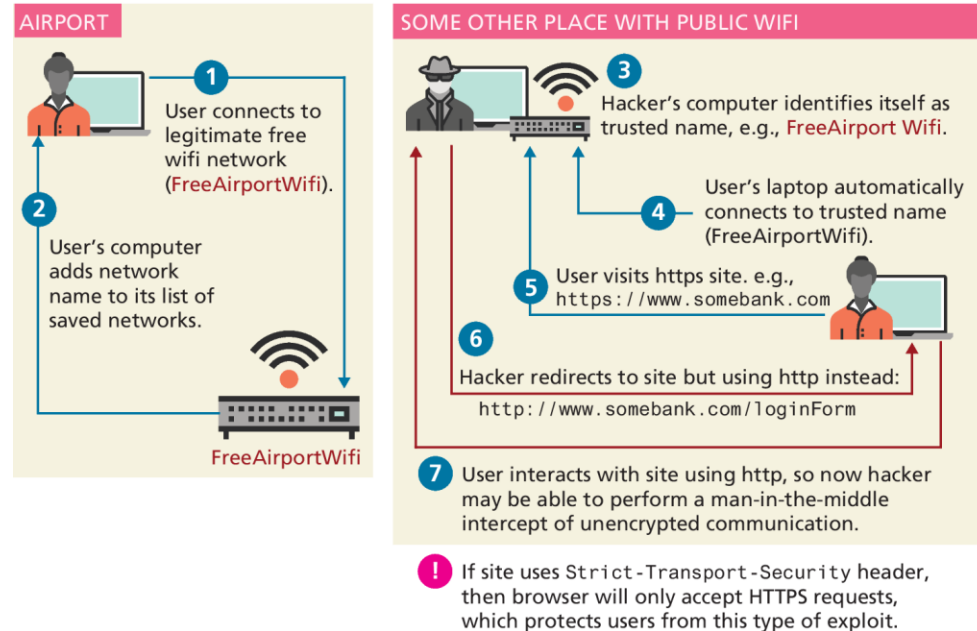
```
RewriteCond %{HTTPS} off
```

```
RewriteRule ^(.*)$ https://%{HTTP_HOST}%{REQUEST_URI} [L,R=301]
```

Preventing HTTP Access

Once your site has added HTTPS capabilities, it often makes sense to prevent users from accessing your site resources using HTTP.

In this example, the hacker will be able to redirect the user from HTTPS to HTTP, thereby having unencrypted access to the user's experience.



Credential Storage

When (not if) you are breached, what data will the attacker have access to?

SQL table structures that store the username and password in the table are bad form

UserID	Username	Password
1	ricardo	password
2	randy	password

```
function insertUser($username,$password){
    $pdo = new PDO(DBCONN_STRING,DBUSERNAME,DBPASS);
    $sql = "INSERT INTO Users (Username,Password)
           VALUES('?',?)";
    $smt = $pdo->prepare($sql);
    $smt->execute(array($username,$password));
}

function validateUser($username,$password){
    $pdo = new PDO(DBCONN_STRING,DBUSERNAME,DBPASS);
    $sql = "SELECT UserID FROM Users WHERE Username=? AND
           Password=?";
    $smt = $pdo->prepare($sql);
    $smt->execute(array(($username,$password)));
    if($smt->rowCount()){
        return true;
    }
    return false;
}
```

LISTING 16.1 First approach to storing passwords (very insecure)

Credential Storage – Hash Function

Instead of storing the password in plain text, a better approach is to store a **hash** of the data, so that the password is not discernable.

UserID	Username	Password
1	ricardo	5f4dcc3b5aa765d61d8327deb882cf99
2	randy	5f4dcc3b5aa765d61d8327deb882cf99

```
function insertUser($username,$password){
    $pdo = new PDO(DBCONN_STRING,DBUSERNAME,DBPASS);
    $sql = "INSERT INTO Users (Username>Password)
            VALUES('?',?)";
    $smt = $pdo->prepare($sql);
    $smt->execute(array($username,md5($password)));
}
function validateUser($username,$password){
    $pdo = new PDO(DBCONN_STRING,DBUSERNAME,DBPASS);
    $sql = "SELECT UserID FROM Users WHERE Username=? AND
            Password=?";
    $smt = $pdo->prepare($sql);
    $smt->execute(array($username,md5($password)));
    if($smt->rowCount()){
        return true;
    }
    return false;
}
```

LISTING 16.2 Second approach to storing passwords
(better but still insecure)

Rainbow Tables

Rainbow tables are massive generated tables for common words and phrases that allow anyone who has access to the digest to quickly look up the original password

As a consequence, storing the MD5 digest of just the password is *not* recommended.

Search in 17,172,461,530 decrypted hashes

Hash:

Decrypt Hash Results for: 34819d7beeabb9260a5c854bc85b3e44

Algorithm	Hash	Decrypted
md5	34819d7beeabb9260a5c854bc85b3e44	mypassword

md5 digest	original
3dbe00a167653a1aaee01d93e77e730e	aaaaaaaa
0e976d4541c8b231ec26e2c522e841aa	baaaaaaa
0b23c6524e8f4d91afc91b60c786931c	caaaaaaa
...	...
34819d7beeabb9260a5c854bc85b3e44	mypassword
...	...

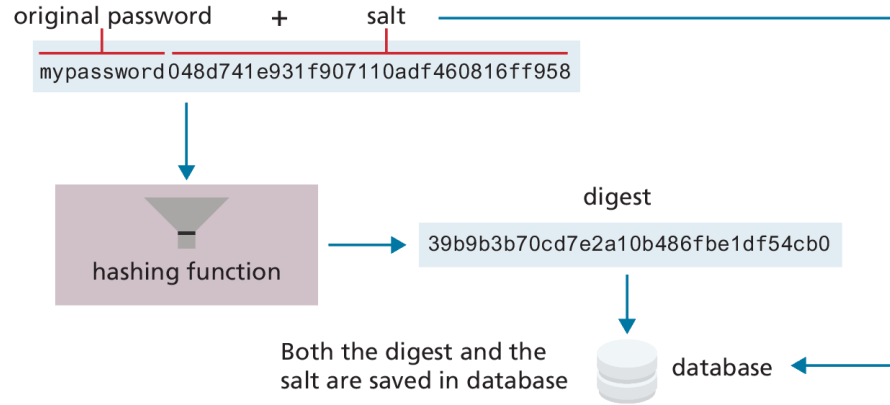
Rainbow tables can be used to quickly look up the plaintext input from common hashing functions.

Salting the Hash

The solution to the rainbow table problem is to add some unique noise to each password, thereby lengthening the password before it is hashed.

The technique of adding some noise to each password is called **salting** the password.

Slow hash, using bcrypt even better



Search in 17,172,403,021 decrypted hashes

Hash: 39b9b3b70cd7e2a10b486fbe1df54cb0

No hashes found for 39b9b3b70cd7e2a10b486fbe1df54cb0

By making the password long and complex, salting generally protects leaked passwords against rainbow table decryption.

Monitor Your Systems

System Monitors allow you to preconfigure a system to check in on all your sites and servers periodically. Nagios, for example, comes with a web interface that allows you to see the status and history of your devices, and sends out notifications by email per your preferences.

Automating intrusion detection reads the log files and detects failed login attempts, then uses a history to determine the originating IP addresses to automatically block it in future

Common Threat Vectors

- **Brute-Force Attacks**
- **SQL Injection**
- **Cross-Site Scripting (XSS)**
- **Cross-Site Request Forgery (CSRF)**
- **Insecure Direct Object Reference**
- **Denial of Service**
- **Security Misconfiguration**

Brute-Force Attacks

In this attack, an intruder simply tries repeatedly guessing a password. For instance, an automated script might try looping through words in the dictionary or use combinations of words, numbers, and symbols.

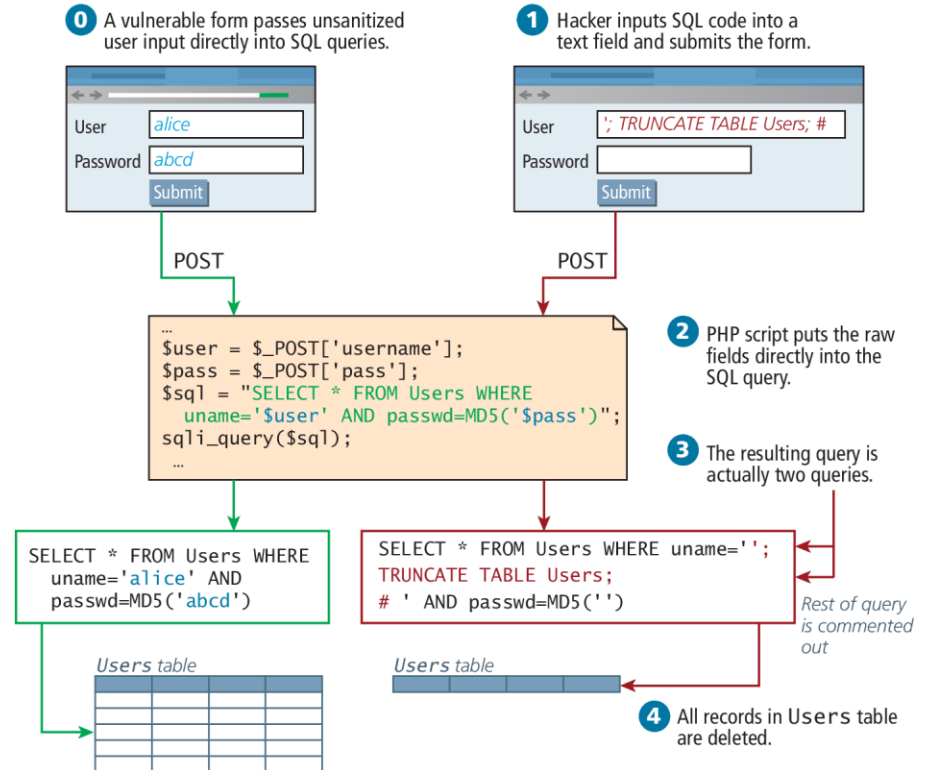
It is important to throttle login attempts. One approach is to lock a user account after some set number of incorrect guesses. Another approach is to simply add a time delay between login attempts.

Automating intrusion detection can monitor logs to detect and block these attacks.

SQL Injection

SQL injection is the attack technique of entering SQL commands into user input fields in order to make the database execute a malicious query.

There are two ways to protect against such attacks: sanitize user input, and apply the least privileges possible for the application's database user.



Defending against SQL Injection

To **sanitize** user before using it in a SQL query, you can apply sanitization functions and bind the variables in the query using parameters or prepared statements.

Despite the sanitization of user input, there is always a risk that users could somehow execute a SQL query they are not entitled to. A properly secured system only assigns users and applications the privileges they need to complete their work, but no more.

Cross-Site Scripting (XSS)

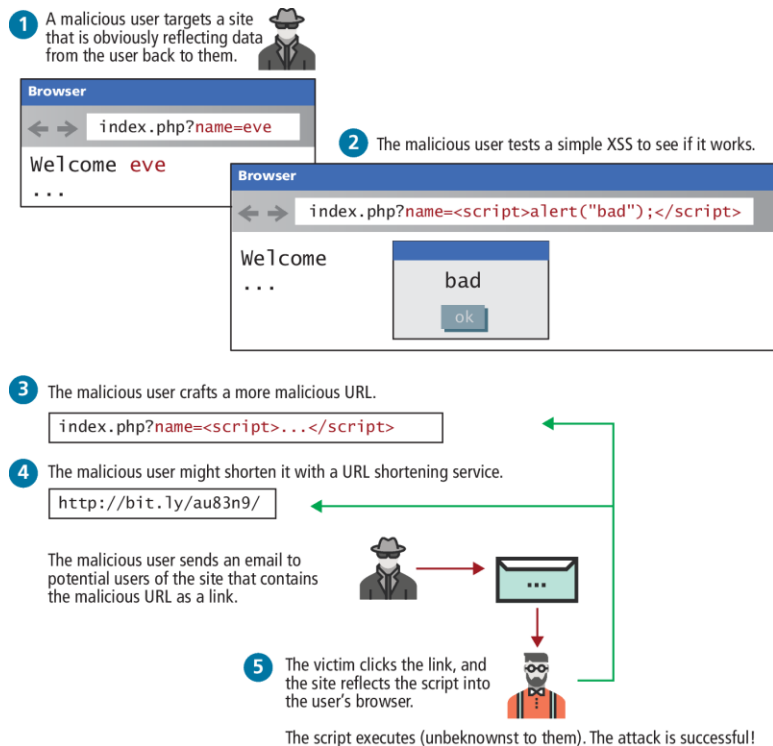
Cross-site scripting (XSS) refers to a type of attack in which a malicious script (JavaScript) is embedded into an otherwise trustworthy website.

In the original formulation for these type of attacks, a malicious user would get a script onto a page and that script would then send data to a malicious party, hosted at another domain (hence the **cross**, in XSS).

That problem has been partially addressed by modern browsers, which restricts script requests to the same domain. However, with at least 80 XSS attack vectors to get around those restrictions, it remains a serious problem.²⁰ There are two main categories of XSS vulnerability: **Reflected XSS** and **Stored XSS**.

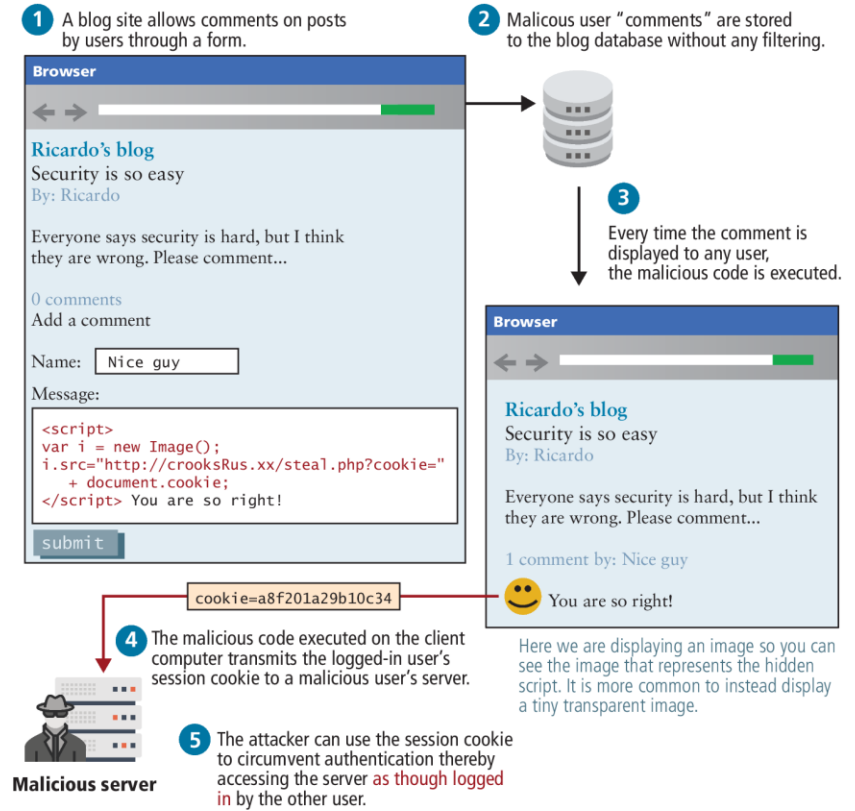
Reflected XSS

Reflected XSS (also known as nonpersistent XSS) are attacks that send malicious content to the server, so that in the server response, the malicious content is embedded.



Stored XSS

Stored XSS (also known as persistent XSS) is even more dangerous, because the attack can impact every user that visits the site.



Defending XSS Attack - Filtering

Sanitizing user input is crucial to preventing XSS attacks but attackers have gone far beyond using HTML script tags, and employ subtle tactics

- **Embedded attributes** use the attribute of a tag, rather than a `<script>` block
- **Hexadecimal/HTML encoding** embeds an escaped set of characters such as:
%3C%73%63%72%69%70%74%3E%61%6C%65%72%74%28%22%68%65%6C%6C%6F%22%29%3B%3C%2F%73%63%72%69%70%74%3E
instead of `<script>alert("hello");</script>`.

Most significant frameworks such as React or EJS provide built-in sanitization when outputting content. A library such as the opensource HTMLPurifier from <http://htmlpurifier.org/> allows you to easily remove a wide range of dangerous characters

Escape Dangerous Content

Even if malicious content makes its way into your database, there are still techniques to prevent an attack from being successful.

Escaping content is a great way to make sure that user content is never executed, even if a malicious script was uploaded. This technique relies on the fact that browsers don't execute escaped content as JavaScript, but rather interpret it as text. Ironically, it uses one of the techniques the hackers employ to get past filters.

That means even if the malicious script did get stored, you would escape it before sending it out to users, so they would receive the following:

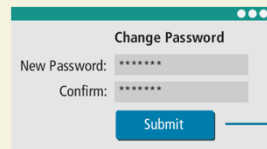
```
&lt;script>alert(&quot;hello&quot;);&lt;/script>
```

The browsers seeing the encoded characters would translate them back for display, but will not execute the script! Instead your code would appear on the page as text.

Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF) is a type of attack that forces users to execute actions on a website in which they are authenticated.

State-Changing Form



→ GET `somesite.ca/login`
← Set-Cookie: `sid=dg5476dGKjm3342`
→ GET `somesite.ca/passform`
Cookie: `sid=dg5476dGKjm3342`
←
→ POST `somesite.ca/changepass`
Cookie: `sid=dg5476dGKjm3342`
...
`new=pass123&confirm=pass123`

CSRF takes advantage of authentication cookies.

CSRF takes advantage of the discoverability of state-changing commands in a web application.

CSRF Attack

Web Email

From: `hackersRus.com`
Subject: `Your meeting this week`
Message: [View Calendar](#)

Form will auto-submit when it is displayed if JavaScript is enabled in web-based email client.

→ `<body onload="document.querySelector('#csrf').submit()">`
`<form action="somesite.ca/changepass"`
`method="post" id="csrf">`
Hidden form fields → `<input type="hidden" name="new" value=hackerPass>`
`<input type="hidden" name="confirm" value=hackerPass>`
`<input type="submit" value="View Calendar">`
`</form>`
`</body>`

If no JavaScript, then social engineering can be used to trick user into submitting form.

If user is still logged into `somesite.com` (i.e., authentication cookie is still valid), then user has changed their password without realized it.

POST `somesite.ca/changepass`
Cookie: `sid=dg5476dGKjm3342`
...
`new=hackerPass&confirm=hackerPass`

Defending against CSRF

Regardless of whether one uses tokens or cookies, the most common way to prevent CSRF attacks is to add a **one-time use CSRF token** to any state-changing form via a hidden field:

```
<input type="hidden" name="csrf-token" value="lR4Xbi...wX4WFoz" />
```

This value should be long, increment in an unpredictable way or contain a timestamp, and be generated with a static secret.

Each time the server serves the form, it should generate a new CSRF token and include it in the form. If a hacker tries to create a CSRF exploit by including the hidden field they see when they examine the form's HTML source, the exploit will fail because the server code will check and see that the increment value or timestamp in the attack form is incorrect.

Insecure Direct Object Reference

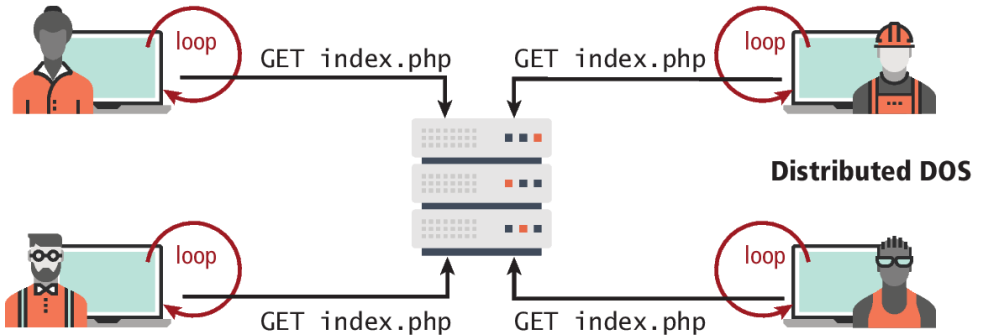
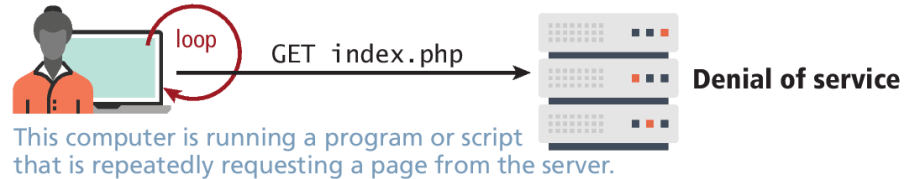
An **insecure direct object reference** is a fancy name for when some internal value or key of the application is exposed to the user, and attackers can then manipulate these internal keys to gain access to things they should not have access to.

For instance, if a user can determine that his or her uploaded photos are stored sequentially as **/images/99/1.jpg**, **/images/99/2.jpg**, . . . , they might try to access images of other users by requesting **/images/ 101/1.jpg**

Denial of Service

Denial of service attacks (DoS attacks) are attacks that aim to overload a server with illegitimate requests in order to prevent the site from responding to legitimate ones.

Distributed DoS Attack (DDoS) distribute requests from multiple machines



Each computer in this bot army is running the same program or script that is bombarding the server with requests. These users are probably unaware that this is happening.

Security Misconfiguration

The broad category of security misconfiguration captures the wide range of errors that can arise from an improperly configured server.

- Out-of-Date Software
- Open Mail Relays
- Virtual Open Mail Relay
- Arbitrary Program Execution

Key Terms

asymmetric cryptography auditing authentication authentication factors authentication policy authorization availability bearer authentication block ciphers Certificate Authority cipher CIA triad code review confidentiality	Content Security Policy cross-site request forgery (CSRF) cross-site scripting (XSS) cryptographic hash functions decryption denial of service attacks digest digital signature domain-validated certificates encryption extended-	validation certificates form-based authentication hash functions high-availability HTTP basic authentication HTTP Token Authentication Hypertext Transfer Protocol Secure (HTTPS) information assurance information security insecure direct	object reference integrity JWT (JSON Web Token) key legal policies logging man-in-the-middle attacks multifactor authentication open authorization (OAuth) open mail relay organization- validated certificates	pair programming password policies phishing scams principle of least privilege public key cryptography rainbow table reflected XSS salting secure by default secure by design Secure Sockets Layer security testing security theater self-signed certificates single-factor	authentication social engineering stateless authentication stored XSS SQL injection STRIDE substitution cipher symmetric ciphers threat Transport Layer Security (TLS) unit testing usage policy vulnerabilities
--	--	--	---	---	---

Copyright



This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.