Introduction to Security Risks

by Sophia

WHAT'S COVERED

In this lesson, you will review security concepts with regard to web development and our responsibility to protect our customers' and clients' data. You will learn about the security concerns with introducing a web form into a website and how to protect it from such vulnerabilities. Additionally, you will learn about how we can safely use web storage on client systems.

Specifically, this lesson will cover the following:

1. Review of Web Security Concepts

Thus far, you have learned a lot of varying and interconnected concepts related to web development and have been empowered to create dynamic and interactive websites. You have learned how to create web forms for receiving information from users and how to use the server-side scripting language PHP to receive, handle, and respond to the form's submission. At this point, it is important that you be made aware of the different security risks and security concepts with regard to web development as well as accepting and processing data from untrusted users.

Some of the following concepts and topics have been discussed previously throughout this course and will be reviewed here to provide more of a "big picture" with regard to web development security. Additionally, **risk management** and **risk assessment** concepts and practices will be discussed, which will provide you with the tools to ensure your development projects are designed with security and privacy in mind.

KEY CONCEPT

So, why is security so important as a web developer? The primary reason is to ensure the three aspects of **confidentiality**, **integrity**, and **availability**. This is referred to as the "**CIA triad**," which is a common model for providing security around applications and systems. The following explains each of the three aspects of the CIA triad and how it relates to being a web developer.

Confidentiality refers to maintaining privacy and preventing unauthorized users and systems from accessing information. This is a critical aspect of security in that we are obligated to protect the information collected from our users, whether they are users or customers. This is not just the right thing to do; organizations are legally required to provide protective measures to ensure the confidentiality of customer data. As developers, this means that we need to ensure our site and software use encryption and secure methods of communication and that we are aware of the vulnerabilities and protective measures of the technologies we use such as forms and server-side scripting languages. This also includes ensuring the proper access controls are in place, the default passwords are replaced, and the software and systems are patched and up to date.

Integrity refers to the validity and originality of information and data that is collected. Essentially, integrity means verifying and ensuring that data has not been maliciously or accidentally manipulated or destroyed by preventing unauthorized access and limiting permissions. This is done by limiting access to data to only individuals who are responsible for maintaining the data. This is less of a "security" concern for developers as it is a concern of data validity and accuracy. In short, this means making sure the proper access controls are in place and that only the correct data is collected and stored. Data validation and access controls are our primary tools as web developers to ensure the integrity of the data we collect and store.

Accessibility refers to the continuous ability to access the resource. In this case, the "access" in accessibility pertains to all legitimate user and company operations. The organization needs access to its data in order to carry out business operations. However, if a webserver is left vulnerable to an attack due to a lack of network defenses or a lack of defensive programming, the server could be compromised and brought down. This can be detrimental to the organization, its reputation, and more. Web developers can help by providing proper validation and sanitization techniques to avoid injection-type attacks.

TERMS TO KNOW

Risk Management
The continuing process of identifying, analyzing, evaluating, and treating exposure to loss and monitoring the risk controls and financial resources required to mitigate any impact on the organization.

Risk Assessment
The process of identifying potential hazards and analyzing what could happen if the hazard or disaster occurred.

Confidentiality
The state of keeping information private and preventing access to those who are not authorized to view the information.

Integrity

In computer science, the state of being authentic and unaltered.

Availability
The state of being on and accessible to users.

CIA Triad
A common model that forms the basis for the development of security systems and includes the facets of confidentiality, integrity, and access.

1a. Encryption

Encryption is one of the first lines of defense in protecting the confidentiality of our customer and internal data. This is also one of the easier security measures to deploy. To employ encryption on a website, you will need to purchase and install a SSL (Secure Socket Layer) certificate to your webserver. Most hosted webserver providers make this process very easy. Once installed, all traffic to and from the webserver will be encrypted. If the certificate is not working or not being used for any reason, the user will be informed of this and their browser will recommend that they do not continue to the site for their own safety.

What happens when there is no SSL certificate?

Any data transmitted between the user and the webserver is communicated across the untrusted Internet without any encryption. In other words, the data travels across the Internet as plain text, and anyone who may be looking will be able to see everything. This problem comes with a whole slew of other issues. Not only will sensitive customer data be visible to the Internet, so will credentials and login information, which could lead to a breach and thus a data leak, data destruction, loss of access, vandalism, and so on.

However, even with an SSL certificate installed and in use, there are still ways that user information could be sent without encryption. The best example of this is when we use the GET method on web forms or AJAX requests. Remember that when data is transmitted using GET, the data is appended to the URL contained within the request and the URL of an HTTP request is never encrypted.

Just remember to use the POST method when creating a web form or programming an AJAX request whenever any potentially sensitive information is involved.

EXAMPLE Using the POST method and setting request headers

```
const xhr = new XMLHttpRequest();
xhr.open("POST", "/scripts/handleRequest.php");

xhr.setRequestHeader( "content-type", "application/x-www-form-urlencoded" );

xhr.onreadystatechange = () =>
{
  if(xhr.readyState == 4 && xhr.status == 200)
  {
```

```
    //insert xhr.responseText or xhr.responseXML into the DOM
  }
  else
  {
    //report or log the error based on status and readyState values.
  }
};
xhr.send("user=jDoe&pass=123456");
```

EXAMPLE Using the POST method with the web form

```
<form action="formHandler.php" method="POST">
        …
</form>
```

1b. Security Policies

Organizations can and should employ a variety of security-focused policies. While the range of policies is wide, security audits are one policy that web developers could be involved in. Web developers who have control over any access permissions or access control lists should periodically audit those permissions to ensure they have not changed and that they are working as intended. This process should include reviewing and testing account permissions and restrictions.

Furthermore, security logs should also be audited for unusual activity. A large number of failed login attempts could point to a potential breach attempt. Login attempts at unusual times of the day may also indicate a breach attempt. It is also important to review the logs of other systems that you may be responsible for, including database logs, network logs, and other system logs for unusual or concerning activities as they could point to a potentially costly incident.

Another policy or process that may be important in ensuring the security of a website and possibly web applications for the benefit of the organization is **risk analysis**. This is not often the responsibility of the web developer, but it is an important aspect to be aware of as an IT professional. Risk analysis is the continuous process of examining possible issues that could negatively impact key business operations or initiatives. In technology, we need to conduct risk analysis to determine if mitigating factors are cost effective compared to the potential loss we might incur should we choose to ignore the risk. The idea is that we measure the cost of fixing or closing a vulnerability and avoiding the risk versus the cost of recovering from such a successful attack.

IN CONTEXT

Let’s take a closer look at risk analysis. Imagine we have a webserver that only hosts static information to the public. It has no forms and no database attached. Should this webserver be attacked and brought down, it would not take long or incur any additional costs to reload the server from a backup image and get it back online. The only cost associated with this would be the time put in by the IT professional and the backup solution already used throughout the organization. However, to avoid

such an attack, it would cost the organization anywhere from $500 to $5,000 to purchase, provision, and maintain a security appliance just for that webserver. This type of scenario may dictate that it is more cost effective to accept the risk than to prevent it.

On the flip side, if we have a financial database within an organization, should that database be compromised, it could cost the organization a lot more than a couple of thousands of dollars. In potentially terminal cases such as this, where the organization stands to lose a lot more, it would be cost effective (and mission critical) to put in place the necessary protections to avoid such a catastrophic event or disruption of operations.

Another important aspect and use case for risk analysis is the introduction of new features within a website or web application. Businesses do this when considering offering a new service or product to examine the potential impact (positive and negative) of such a decision. As web developers, we need to also employ this same approach whenever considering adding a new page, tool, or feature to a website or web application. Does the new feature introduce significant risks to the organization or its technology assets and, if so, does it incur additional costs to avoid or recover from the potential risk? Performing our own due diligence can help us inform leadership and ourselves of the possible risks as well as the costs associated with the risk and its mitigation.

Risk Analysis
The process of identifying and analyzing potential issues that could negatively impact business operations, projects, or assets.

2. Web Form Security

Web forms are relatively easy to implement and yet are a potential source of vulnerabilities for a website and the server it is hosted on. The concern for web forms extends beyond the use of POST over GET. Web forms are subject to different injection attacks in which malicious code is inserted into a web form's text field, and there is the potential that the malicious code is executed by the server. An example of this would be placing an SQL query designed to retrieve the usernames of all users within the database. When the malicious user clicks "Submit," if there are no protections in place, the code makes its way to the database, which in turn executes the command placed within the form. The result could be the sensitive data being displayed on the screen of the malicious user.

This is where form **validation** and **sanitization** are critical. Validation simply means to ensure that the data entered into a form's field is actually the appropriate data. When the form asks for a first name, the only thing contained within that field should be plain text. When the field asks for a phone number, the only characters in the field should be numbers and possibly the plus sign for international numbers, dashes between the sets of numbers, and parentheses for area codes. Anything else should prevent the form from submitting or should be rejected by the server's form handler script.

Sanitization refers to checking form data entries for unusual characters and character patterns that resemble sequences of code. The primary goal of sanitization is to ensure that no “code” of any kind is present in any of the form data. While form field validation helps with this, form field validation takes place on the client’s computer and is ultimately under the control of the user and could be bypassed in order to submit malicious code within the form. This is where our server-side scripts can further enforce validation but also sanitize any data that makes it through by changing actual characters, such as < and >, to their character entity code equivalent of > and <. The result is that should malicious code make its way to the webserver, it would effectively render the code unrecognizable to the targeted system.

TERMS TO KNOW

Validation
The process of ensuring that only the correct type of data is entered into the correct field.

Sanitization
The process of detecting potentially malicious code placed inside of a form field and rendering it ineffective in order to prevent XSS attacks.

2a. Validation

Form validation is the first step to preventing attacks through a website by ensuring only the correct data is entered into a form’s fields. This requires developers to employ the use of the correct form field types and include any and all applicable HTML attributes to limit what the user can and cannot enter into a form field. Additionally, we can perform additional validation checks using JavaScript. When the user clicks “Submit,” the button’s onClick event can be redirected to a JavaScript function, which can manually perform additional validation checks on the form’s data prior to allowing the form to be submitted. This is also the first opportunity that we have to sanitize the data using JavaScript. Sanitization processes will be discussed in the next section.

The issue with relying solely on client-side validation is, as we discussed earlier, that these controls are under the influence of the user and their system. When HTML, CSS, and JavaScript make their way to the client’s system, the client has the ability to view and manipulate any JavaScript code present on the page. The result is that they can manipulate and disable any of the JavaScript code and bypass any validation techniques. This is why additional sanitization should be considered on the server, prior to sending the data to a database of another targeted system.

2b. Sanitization

Sanitization, as discussed previously, is the process of rendering potentially malicious code illegible to targeted systems without destroying the original data (for cases where the data is legitimate). Sanitization is possible on the client side using JavaScript and is a recommended approach to sanitizing form data. In fact, there are libraries available that can be added to a site and utilized to perform XSS sanitization and validation. **DOMPurify** is one such library that uses only native features and functions already built into all modern browsers.

We can deploy DOMPurify or other sanitization methods and techniques using JavaScript by intercepting the form submission using a JavaScript function. The function performs basic validations per the form field’s attributes and types, but it also gives us the opportunity to perform additional operations on the form data before we allow the form to submit.

EXAMPLE Sanitizing a form using the DOMPurify library

```
<!DOCTYPE html>
<html>
<head>
  <script type="text/javascript" src="purify.min.js"></script>
  <script>
    //intercept form submission event handler.
    //Cannot attach a handler until after form has been created.
    //We will call this later.
    function interceptForm()
{
      var form = document.getElementById("myForm");
      if (form.attachEvent)
         { form.attachEvent("submit", processForm); }
      else
         { form.addEventListener("submit",  processForm); }
    }

    //New form submission handler.
    function processForm(e)
    {
            alert("validating form");
      if (e.preventDefault)
         e.preventDefault();

      //This is where we can validate and sanitize the input.
      let fields = document.getElementsByTagName('input');
      let dirty = fields[0].value;
      let clean = DOMPurify.sanitize(dirty);
            alert("Dirty: " + dirty + "\nClean: " + clean);
      fields[0].value = clean;

      //Manually submit the form after sanitization.
      document.getElementById("myForm").submit();
    }

  </script>
</head>
<body>
  <form action="validfhandler.php" method="POST" id="myForm">
    <input type="text" name="fullName" />
```

```
    <input type="submit" value="Submit">
  </form>
  <script>
    //Calling interceptForm function now that the form is ready.
    interceptForm();
</script>
</body>
</html>
```

Let's break down the above sanitization example. First, we simply download the project repo from github.com and then copy the purify.min.js into the site's directory.

Then, we link to it using a <script> tag. Next, we set up a function that will intercept the submit function of the form and trigger a custom function instead. The function redirects the "submit" event handler to our custom function.

The custom function first negates the "Submit" button's default behavior (submitting the form) by calling e.preventDefault();, wherein `e` is the button that was just clicked and triggered the function, and thus we call the `preventDefault()` function, which prevents the form from automatically submitting.

Next, the custom function gets the input fields from the form using document.getElementsByTagName("input"), which returns an array of the input fields. We then capture the value in the field, store it in the "dirty" variable, pass it into the `DOMPurify.santize(dirty)` function as the only argument, and store the returned sanitized value back into the "clean" variable. We then swap the cleaned value back into the form field that we just extracted it from. Note that this could be done using a loop when there are more fields to sanitize.

The final step is to submit the form by calling the form's submit() function. Note that this example illustrates the sanitization of the form data by passing the dirty and clean variables into an alert(). If you enter a script command as in a XSS attack, it will look like this:
John <script>alert("die");</script>Doe.

Any of the offending content is removed, leaving only the legitimate data behind:
`John Doe.`

TERM TO KNOW

DOMPurify
A free JavaScript API that utilizes DOM-only features to help prevent XSS attacks through sanitization.

WATCH

View the following video for more on sanitizing input.

3. Web Storage Security

Another security concern worth mentioning is the use of web storage technology. Recall that web storage comes in two forms: local and session. Data stored in the sessionStorage object remains in the browser until the browser tab or window is closed. Local storage remains on the client's system until it is removed by clearing the browser cache. However, one thing to keep in mind is that neither of these storage options is secure and should not be used to store sensitive information. Even when the sensitive information belongs to the user themselves, we should avoid storing their information using unsecure methods as their system could be compromised.

Instead, sensitive information and assets that need to be stored on the user's system should be stored in a secured cookie with either no expiration (expires when the tab or window is closed) or a relatively short expiration time.

Secure cookies using the "Secure" and "HttpOnly" options are almost always done on the server side either as part of the server-side application code or as part of the webserver's configuration. The implementation is different depending on what webserver software you are using (IIS, Apache, Tomcat, etc.). However, the idea is to set the HttpOnly flag as well as the Secure flag. HttpOnly ensures that the only way cookies can be accessed or transmitted is via HTTP. This prevents cross-site scripting attacks by blocking JavaScript from accessing the cookie set by the server. The secure flag also prevents data from being leaked by requiring a secure connection through HTTPS in order for the cookie to be created and sent to the client's system. This prevents any cookie from being transmitted across an unencrypted connection.

Another data asset that is often stored on the local system for use with the webserver or web application is the **authentication token**. This is a digital value that indicates an authenticated user and is used to validate the user's activities. Someone without a valid authentication token would be denied access and likely taken to the site's login page. Placing these tokens on unsecure storage risks the token being stolen and used by an unauthorized user. Using secure cookies is a good way to protect these tokens, but we can reduce the risk associated with a stolen token by requiring authentication tokens to expire and the user to be authenticated again. Avoid long expiration times for authentication tokens.

TERM TO KNOW

Authentication Token
A digital file that serves as proof of authentication and can be used to access one or more systems without having to reauthenticate each time.

MAKE THE CONNECTION

Congratulations on finishing the course content! You have gained the necessary knowledge and skills to design and build a website that meets client needs. Now it is time to demonstrate what you have learned by completing the course Touchstone assignment.

SUMMARY

In this lesson, you were introduced to the CIA triad and how it impacts what we do as developers. Additionally, you reviewed the web security concepts of **encryption** and **security policies**, as well as other security-related concepts, such as access controls that you may have some level of responsibility

over. Additionally, you learned about the concept of **web form security** that uses input **validation** as well as how to **sanitize** input data to protect against cross-site scripting attacks. Lastly, you learned about **web storage security** concerns and how you should use secured cookies to store sensitive information on the client's system.

Source: This Tutorial has been adapted from "The Missing Link: An Introduction to Web Development and Programming " by Michael Mendez. Access for free at https://open.umn.edu/opentextbooks/textbooks/the-missing-link-an-introduction-to-web-development-and-programming. License: Creative Commons attribution: CC BY-NC-SA.

TERMS TO KNOW

Authentication Token
A digital file that serves as proof of authentication and can be used to access one or more systems without having to reauthenticate each time.

Availability
The state of being on and accessible to users.

CIA Triad
A common model that forms the basis for the development of security systems and includes the facets of confidentiality, integrity, and access.

Confidentiality
The state of keeping information private and preventing access to those who are not authorized to view the information.

DOMPurify
A free JavaScript API that utilizes DOM-only features to help prevent XSS attacks through sanitization.

Integrity
In computer science, the state of being authentic and unaltered.

Risk Analysis
The process of identifying and analyzing potential issues that could negatively impact business operations, projects, or assets.

Risk Assessment
The process of identifying potential hazards and analyzing what could happen if the hazard or disaster occurred.

Risk Management
The continuing process of identifying, analyzing, evaluating, and treating exposure to loss and monitoring the risk controls and financial resources required to mitigate any impact on the organization.

Sanitization

The process of detecting potentially malicious code placed inside of a form field and rendering it ineffective in order to prevent XSS attacks.

Validation

The process of ensuring that only the correct type of data is entered into the correct field.