OWASP Top 10

Doc Writer

Version 1.0.0-SNAPSHOT, 2014-09-09

Table of Contents

This is a user manual for an example project.

What is PCI DSS and why does it matter?

- PCI DSS policies were developed to strengthen transaction security.
- The Payment Card Industry Data Security Standards (PCI DSS) is a set of policies and procedures created with a two-fold objective: to strengthen credit, debit, and cash card transaction security and to protect cardholders against card fraud and theft of their personal information.
- The PCI Data Security Standard is administered by the Payment Card Industry Security Standards Council (PCI SSC) which is comprised of five founding global payment brands:
  1. Visa Inc.
  2. MasterCard
  3. American Express
  4. Discover Financial Services
  5. JCB International

Complying With PCI DSS

- The PCI DSS applies to organizations of any size that accept, transmit, or store any cardholder data. While organizations' degrees or levels of compliance vary depending on the size of their transactions, they generally need to comply with the same requirements.
- The PCI DSS specifies six major requirements:

Requirement #1: Build and Maintain a Secure Network

- A secure network ensures that every transaction is safely conducted.
- This requirement involves the installation and maintenance of firewalls without causing any inconveniences to cardholders and vendors.
- Firewalls minimize instances of hacking and illegal eavesdropping or surveillance of an organization's network.
- This requirement also includes changing vendor-supplied default passwords for systems and security modules.
- The PCI DSS also requires enterprises to enable their customers to easily change their user passwords, if any.

Requirement #2: Protect Cardholder Data

- The PCI DSS requires enterprises to protect cardholder information wherever it is stored.
- Repositories keeping vital information such as dates of birth, credit card numbers, mailing addresses, and other personally identifiable information should be kept secure against hackers

and malicious software.

- The PCI DSS requires the use of encryption especially when cardholder data is shared through open and public networks. This requirement is critical in all credit card transactions, but more especially in e-commerce transactions.

Requirement #3: Maintain a Vulnerability Management Program

- This requirement requires enterprises to ensure that the security of their systems and software are up to date to protect themselves and their customers' information from emerging threats.
- This requirement includes using and regularly updating anti-virus, anti-spyware, and other anti-malware software.
- This also includes ensuring that software is free from bugs that can exploit vulnerabilities in the enterprises' systems. Strengthening the security of systems and applications also involves downloading and installing the latest patches for operating systems, which provide the backbone for applications.

Requirement #4: Implement Strong Access Control Measures

- The PCI DSS necessitates strong internal controls especially when it comes to access to systems and customer information.
- First, enterprises should not require cardholders to provide additional personal information that is not required to process business transactions.
- Second, enterprises need to internally restrict access to cardholder data only to those who are processing transactions.
- Third, each person with computer access to the systems must be assigned a unique ID or confidential identification number.
- Finally, physical access to cardholder data must also be restricted by enterprises.
- This includes the use of secure locations in keeping paper documents containing cardholders' personal information, proper disposal of documents, limiting duplication of documents, and even the utilization of locks and chains in securing offices where documents are kept.

Requirement #5: Regularly Monitor and Test Networks

- Meeting PCI DSS compliance requirements does not stop at establishing safeguards and security procedures.
- Enterprises also have to continually monitor and test their networks and systems, as well as track and monitor access to cardholder data and network resources to identify risk areas.
- Understanding the flow of information or cardholder data also prevents potential leakages in the systems. Frequently stress-testing security systems and procedures will also ensure that they can handle malicious attacks of varying degrees of complexity.

Requirement #6: Maintain an Information Security Policy

- Finally, enterprises must formalize their security measures into enterprise-wide policies.
- The PCI DSS requires enterprises to maintain policies for their information security that employees, contractors, and all partners need to follow.

Best Practices for PCI DSS Compliance

While PCI DSS compliance can seem overwhelming, the PCI Security Standards Council has integrated the best practices of enterprises and security experts from around the world, compiling them into eight straightforward tips for easier implementation: * Purchase and utilize only approved PIN entry devices at points of sale (POS). * Purchase and utilize only validated payment software at POS and/or e-commerce shopping cart. * Avoid storing sensitive cardholder data and/or PII in computers or on paper documents. * Install firewalls on networks, personal computers, and devices. * Password-protect and encrypt wireless routers and LAN. * Regularly change and use strong passwords on hardware and software. * Frequently check PIN entry devices to ensure protection from skimming devices and malicious software. * Create formal policies and procedures and train employees about information security and cardholder data protection measures.

The Benefits of Complying With PCI DSS

Aside from avoiding fines and penalties that can add up to $500,000, complying with the PCI DSS also provides business benefits: === Minimizing Security Risks * Compliance with the PCI DSS is not just putting a tick mark on every checkbox in a compliance checklist. * The PCI DSS is proven to optimize security to prevent attacks against enterprises and protect customer data. * According to a report published by Verizon, companies that have experienced security breaches were less likely to be compliant with PCI DSS.

Increasing Customer Confidence

- Customers are becoming more and more aware of the security risks associated with giving away their credit card information to merchants.
- While they don’t fully understand what it takes to become PCI DSS compliant, they look for PCI compliance to know that merchants are following best practices in protecting their information.
- In fact, according to the same Verizon report, 69% of consumers are less likely to do business with an organization that has suffered from a security breach.

Adhering to Generally Accepted Standards

- Many enterprises still do not know where to begin when it comes to credit card security.
- The PCI DSS is a set of standards accepted globally.
- It provides enterprises with a baseline that they can easily comply with and implement

throughout their entire organization, regardless of where they are operating in the world.

Complying With EU GDPR

- The European Union's General Data Protection Regulation (EU GDPR) will be in full effect by the 25th of May 2018.
- This will lead to stricter regulations for protecting the personally identifiable information (PII) of EU citizens, whether they are in the EU or not.
- The PCI DSS is designed to safeguard PII regardless of citizenship.
- Its standards and regulations are aligned with those of the GDPR.
- While it is not the complete answer to GDPR compliance, adherence to the PCI DSS is considered a good first step in preparing for the new set of regulations.

Reduce Your PCI DSS Compliance Burden With Tokenization

- Even with best practices and quick implementation guides for PCI DSS compliance, many enterprises still find the compliance procedures burdensome and time-consuming.
- Most merchants, especially e-commerce companies, have to redirect time, money, and resources from their strategic objectives in order to comply with PCI regulations.
- Tokenization enables enterprises to replace sensitive data such as personally identifiable information with surrogate data, or tokens.
- Instead of using cardholder data to process transactions and storing them in repositories, enterprises can replace sensitive PCI and PII data throughout their applications with randomly generated tokens, thus removing the points of compromise.

Cross Site Scripting

Reflected

- Unlike Persistent XSS, with Reflected Cross-Site Scripting (XSS) attacker-supplied script code is never stored within the application itself.
  - Instead the attacker crafts a malicious request to the application to illicit a single HTTP response by the application that contains the attackers's supplied script code.
  - Successful attacks require victim users to open a maliciously crafted link (which is very easy to do).
- To help defend against Reflected Cross Site Scripting (XSS) attacks it is important to ensure that user-supplied data is output encoded before being served by the application to other users.
  - In essence output encoding effectively works by escaping user-supplied data immediately before it is served to users of the application.

- By correctly escaping the data, when it is served to the user for display in their browser, the browser does not interpret it as code and instead interprets it as data, thus ensuring it does not get executed.
- For example the string: `<script>` is converted to: `&lt;script&gt;` when properly escaped and simply render as text in the user's browser window, rather than being interpreted as code.

Example

- In order to generate the search results web page, TradeSEARCH developers have used the JSP Standard Templating Library (JSTL) which provides standard actions and methods for formatting and rendering HTML pages.
  - When rendering a user's search result web page, the c:when conditional tag is called to check if any search results were returned by the server, which are then formatted and rendered by searchResults.jsp
  - However, should no matching results be found for a specified keyword, the c:otherwise conditional tag will render the HTML markup on line 9 and 10 that renders a No results found for: message followed by the user supplied search phrase.
  - To render the No results found for: error message followed by our search phrase string, `request.getParameter()` method is called to extract the search parameter from the URL which then gets directly rendered as a JSP expression.
  - e.g. For projects?search=sometext the JSP expression `request.getParameter("search")` will yield sometext which is finally rendered in the user's browser.
  - Unfortunately, JSP's expression language does not escape expression values, so if the search parameter contained HTML formatted data, Java's EL expression will simply render this string without escaping or encoding it first.
  - Finally when the No results found for: message is displayed, the search parameters HTML data string will also get rendered, thereby allowing code injection in the users browser context.
- The most effective method to protect against XSS attacks is by using JSTL's c:out tag or `fn:escapeXml()` EL function when displaying user-controlled input. These tags will automatically escape and encode HTML characters within the rendered HTML including < , > , " , ' and & thereby preventing injection of potentially malicious HTML code.

Summary

- In this module we learned how Reflected Cross Site Scripting (XSS) attacks work.
- We also learned that Reflected XSS and Non-Persistent XSS are two different terms for the same thing.
- Furthermore, we understood that to protect against XSS, Output Encoding needs to take place whenever user-supplied data is used by the application when building responses to other users.
- Finally, we learned that the `<c:out>` function escapes HTML characters to help prevent against XSS.

Persistent (Stored) XSS

- Persistent Cross-Site Scripting (XSS) is an application vulnerability whereby a malicious user tricks a web application into storing attacker-supplied script code which is the later served to unsuspecting user(s) of the application.
  - The attacker-supplied script code runs on the client-side system of other end user(s) of the application.
  - This type of vulnerability is widespread and affects web applications that utilize (unvalidated) user-supplied input to generate (unencoded) application output, that is served to users.
- To help defend against Stored Cross Site Scripting (XSS) attacks it is important to ensure that user-supplied data is output encoded before being served by the application to other users.
  - In essence output encoding effectively works by escaping user-supplied data immediately before it is served to users of the application.
  - By correctly escaping the data, when it is served to the user for display in their browser, the browser does not interpret it as code and instead interprets it as data, thus ensuring it does not get executed.
  - For example the string: `<script>` is converted to: `&lt;script&gt;` when properly escaped and is simply rendered as text in the user's browser window, rather than being interpreted as code.
- The most effective method to protect against XSS attacks is by using JSTL's c:out tag or `fn:escapeXml()` EL function when displaying user-controlled input. These tags will automatically escape and encode HTML characters within the rendered HTML including < , > , " , ' and & thereby preventing injection of potentially malicious HTML code.

Summary

- In this module we learned how Persistent Cross Site Scripting (XSS) attacks work.
- We also learned that Persistent XSS and Stored XSS are two different terms for the same thing.
- Furthermore, we understood that to protect against XSS, Output Encoding needs to take place whenever user-supplied data is used by the application when building responses to other users.
- Finally, we learned that the `<c:out>` function escapes HTML characters to help prevent against XSS.

DOM XSS

- Document Object Model (DOM) Based XSS is a type of XSS attack wherein the attacker's payload is executed as a result of modifying the DOM "environment" in the victim's browser used by the original client side script, so that the client side code runs in an "unexpected" manner.
- That is, the page itself (the HTTP response that is) does not change, but the client side code contained in the page executes differently due to the malicious modifications that have occurred in the DOM environment.

- To defend against Cross Site Scripting attacks within the application user's Browser Document Object Model (DOM) environment a defense-in-depth approach is required, combining a number of security best practices.
  - Note You should recall that for Stored XSS and Reflected XSS injection takes place server side, rather than client browser side.
  - Whereas with DOM XSS, the attack is injected into the Browser DOM, this adds additional complexity and makes it very difficult to prevent and highly context specific, because an attacker can inject HTML, HTML Attributes, CSS as well as URLs.
  - As a general set of principles the application should first HTML encode and then Javascript encode any user supplied data that is returned to the client.
  - Due to the very large attack surface developers are strongly encouraged to review areas of code that are potentially susceptible to DOM XSS, including but not limited to:
    - `window.name`
    - `document.referrer`
    - `document.URL`
    - `document.documentURI`
    - `location`
    - `location.href`
    - `location.search`
    - `location.hash`
    - `eval`
    - `setTimeout`
    - `setInterval`
    - `document.write`
    - `document.writeIn`
    - `innerHTML`
    - `outerHTML`

Example

- In our modified code example, an additional check is introduced which performs input validation against the name string variable.
- To accomplish this, we make use of javascript's `match()` method to run a regular expression search on name variable, identifying non alphanumeric characters e.g. # , < , > , " , ' , & .
- Should any non alphanumeric characters be encountered, the if check will fail and invoke the "Security error" warning, thus preventing malicious javascript or HTML characters from being passed to the `document.write()` method.

Summary

- In this module we learned how Document Object Model (DOM) Cross Site Scripting (XSS) attacks work, or DOM XSS for short.

- We also learned that writing code more securely to protect against DOM XSS is much trickier than Persistent XSS and Stored XSS, due to the much larger attack surface with DOM XSS.
- We also learned as a general rule an application should first HTML encode, and then Javascript encode user-supplied (or otherwise untrusted data) to improve an application's security posture against DOM XSS.
- Finally, we learned that as developers there are a number of areas of code that are potentially susceptible to DOM XSS and extra consideration should be made for XSS DOM vulnerabilities in these areas (innerHTML , outerHTML , eval , document.write etc).

Broken Authentication and Session Management

Insecure TLS Validation

Unvalidated Redirects and Forwards

Cross Site Request Forgery

Sensitive Data Exposure

Security Misconfiguration

Insecure Direct Object References

Injection

Missing Function Level Access Control

Source Code

Java code from project

```
public boolean contains(String haystack, String needle) {
    return haystack.contains(needle);
}
```

Attributes

Built-in

asciidoctor-version

1.5.5

safe-mode-name

unsafe

docdir

/Users/mateusz/Scripts/pci/owasp-java/src/docs/asciidoc

docfile

/Users/mateusz/Scripts/pci/owasp-java/src/docs/asciidoc/OWASP-Java.adoc

imagesdir

./images

Custom

project-version

1.0.0-SNAPSHOT

sourcedir

/Users/mateusz/Scripts/pci/owasp-java/src/main/java

endpoint-url

http://example.org

Includes

include::subdir/_b.adoc[]

content from *src/docs/asciidoc/subdir/_b.adoc*.

include::_c.adoc[]

content from *src/docs/asciidoc/subdir/c.adoc*.

Includes can be tricky!

build.gradle

```
buildscript {
    dependencies {
        classpath 'org.asciidoctor:asciidoctorj-pdf:1.5.0-alpha.15'
    }
}

plugins {
    id 'org.asciidoctor.convert' version '1.5.8.1'
}

apply plugin: 'java'

version = '1.0.0-SNAPSHOT'

asciidoctorj {
    version = '1.5.5'
}

asciidoctor {
    backends 'pdf'
    attributes \
        'build-gradle': file('build.gradle'),
        'sourcedir': project.sourceSets.main.java.srcDirs[0],
        'endpoint-url': 'http://example.org',
        'source-highlighter': 'coderay',
        'imagesdir': './images',
        'toc': 'left',
        'icons': 'font',
        'setanchors': '',
        'idprefix': '',
        'idseparator': '-',
        'docinfo1': ''
}
```