BAHIR DAR UNIVERSITY

**Bahir Dar Institute of Technology (BiT)**

**Faculty of Computing**

**Department of Software Engineering**

**Fundamental Of Software Security**

**Secure programming techniques**

<- \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ->

**Documented by: -**

1. **Abrham Desalegn..............1307050**
2. **Abiy Shiferaw....................1306078**
3. **Jemal Workie.....................1307712**
4. **Smachew Gedefaw............1308736**
5. **Solomon Muhye.................1309375**
6. **Yordanos Ayenew..............1311662**

**Submitted to: Instructor Kelemework**

**Submit date: January, 2022**

Contents

[Introduction 3](#_Toc123602432)

[Question #1 (conceptual) 3](#_Toc123602433)

[1.1 Trade-offs of Storing Passwords in a Relational Database 3](#_Toc123602434)

[1.2 Additional Input Validation for Usernames and Passwords in a Database 3](#_Toc123602435)

[Question #2 (conceptual) 3](#_Toc123602436)

[2.1 Question explanation 3](#_Toc123602437)

[2.2 HTML filter 3](#_Toc123602438)

[What is HTML filter? 3](#_Toc123602439)

[Pros and Cons of using HTML filter 4](#_Toc123602440)

[How does HTML filter actually work? 4](#_Toc123602441)

[2.3 The required HTML filter 6](#_Toc123602442)

[Some explanation about the code 6](#_Toc123602443)

[HTML filter code 7](#_Toc123602444)

[Question #3 (programming) 8](#_Toc123602445)

[Conclusion 8](#_Toc123602446)

[References 8](#_Toc123602447)

# Introduction

# Question #1 (conceptual)

In Chapter 9, the password manager stored passwords in a file. What would be some of the trade-offs involved in storing the passwords in a relational database instead of in a file? What types of additional input validation might need to be done on usernames and passwords if they are to be stored in a database?

## Trade-offs of Storing Passwords in a Relational Database

## Additional Input Validation for Usernames and Passwords in a Database

# Question #2 (conceptual)

Write an HTML filter that, given an arbitrary HTML document, produces an HTML document that will not result in the execution of script if loaded into a user’s browser, but leaves “basic markup” (fonts, formatting, etc.) intact. Consider the possibility that the input document is not well-formed HTML, and also consider browser-specific features.

## 2.1 Question explanation

Writing an HTML filter that can take an arbitrary HTML document as input and produce a safe HTML document for a user's browser is a difficult task due to the complexity of HTML and the many different kinds of browsers in use. As a result, this project requires a deep understanding of both HTML and browser-specific features.   
 The goal this question is to create an HTML filter that takes in any valid or invalid HTML document as input and produces an output that is safe for loading into any browser. The output should preserve all “basic markup” (fonts, formatting, etc.) while preventing execution of potentially dangerous script.

## 2.2 HTML filter

### What is HTML filter?

***HTML filtering*** is the process of removing malicious HTML code from a web page or email in order to prevent malicious code from being executed by the browser. This helps protect websites and users from becoming victims of malicious attacks such as cross-site scripting and SQL injection. It can be written with some different languages like PHP, CSS, Python, VBScript, Perl, ASP classic, … etc., but JavaScript is usually the go-to programming language for creating HTML filters. Tools such as jQuery, AngularJS, React, and Vue can also be used to build HTML filters.

Examples: -

\* *Email Spam Filter*: This HTML filter examines emails for certain keywords or content which are deemed as suspicious and then removes them from the email before it is sent to its intended recipient.

\* *Pornography Filter*: This HTML filter is used to strip out explicit images and videos found on websites, preventing users from accessing them.

\* *Malware Filter*: This type of HTML filter uses heuristics analysis of websites’ code in order to detect and block malicious software or threats.

### Pros and Cons of using HTML filter

Pros: -

* HTML filters help improve the security of websites and online applications by preventing users from submitting certain data that could harm the system.
* They can also reduce the amount of spam messages and other inappropriate content submitted.
* HTML filters can help protect website owners from malicious attackers.
* They can be used to help ensure copyright compliance, making sure that only authorized content appears on a website or application.
* HTML filters allow for flexibility in terms of what types of content are allowed or disallowed on a website.

Cons: -

* HTML filtering rules can sometimes be too restrictive, blocking out legitimate content which could have been beneficial to the site or application.
* Unfamiliarity with the specific filter settings may lead to the mistaken inclusion or exclusion of important elements during content screening.
* False positives (attribute blocks which should have been allowed through but were blocked) may occur during filtering, resulting in unnecessary frustration on part of both webmasters and users attempting to access such websites or applications.

### How does HTML filter actually work?

There are two main strategies for achieving this process of filtering HTML document: ***blocking certain elements*** or ***validating tag structure***.

One strategy for creating this sort of filter is to simply block certain types of elements from the input document that could be potentially dangerous. Tags such as <script> (in our case only this one) and <iframe> should be removed from the output altogether if present in the input as they represent code that can be executed once loaded into the user's browser. Additionally, attributes on existing tags such as **src="script.js"** should also be removed from tags or replaced with safer versions. Similarly, techniques used by modern browsers like inline JavaScript event handlers should be blocked due to their potential danger when executed in a user's browser environment. Allowing these sorts of features could leave end-users vulnerable to cross-site scripting attacks and malicious code running on their computers. By completely blocking these elements and attributes, we can ensure safety when our filtered output is loaded by a user's browser.

An alternative approach would be validating tag structure rather than just blocking all potentially dangerous tags outright, ensuring that our filtered output remains properly formatted and functioning as expected despite being stripped free of dangerous elements/attributes while preserving basic formatting options like font size/style/color etc... A validation step may check for nested unpaired tags, illegal content nesting rules like JavaScript inside html tag ( <html> … here … </html>), unbalanced quotes in parameter values or even other edge cases like duplicate attributes applied too frequently within elements etc., before allowing it through our filter system; such errors often lead to unexpected behaviors when rendered by various web browsers so proper validation here will prove beneficial especially when dealing with arbitrarily formed HTML documents which might not adhere strictly to common best practices at times.

Using either one or both of these methods, we can easily create a filter that takes in an arbitrary HTML document as input and produces a safely usable (and correctly structured) version meant exclusively for loading into any kind of browser while preserving “*basic markup*” options intact. By taking advantage of powerful tools such as W3C’s Html5Lib library (which provides Python based parsing & serialization) we can implement these strategies in an efficient manner allowing us to efficiently create powerful filters with minimal effort required from the developer's side All this adds up towards providing users safe & consistent browsing experiences no matter where they access content from online regardless if its through native web applications or mobile browsers expecting topnotch user privacy & security guarantees in all scenarios going forward without having them worry about potentially malicious pieces.

## 2.3 The required HTML filter

### Some explanation about the code

The function uses a list of tags and attributes to determine which tags are safe to use. Safe tags[[1]](#footnote-1) are those that don't execute script; unsafe tags are those that do. (It's important to note that some browsers will still execute unsafe tags even if they're not visible on screen.) The function then uses this information to remove any unsafe tags from the input document, leaving only safe tags behind. It also leaves all basic markup intact (fonts, formatting, etc.), since browsers may use this information to render content correctly even without executing script.

The following HTML filter will strip all script from a given HTML document and leave the basic markup intact. The first step is to look at the HTML document's DOCTYPE. If it contains "XHTML", we can safely assume that the document is well-formed, and we can be sure that our filter will not break the page. If it does not contain "XHTML", then we must examine the tag names in the document and remove any that are used for scripting or styling (such as <script> or <style>). Finally, if there are any tags left over—f.nor instance, if there were some tags in the DOCTYPE that were not in use—we should remove them as well.This is a filter that is meant to be used in a web application that allows users to upload HTML documents. The goal of this filter is to make sure that the document will not execute any script, but leave all of the basic markup intact. This filter will take an HTML document and turn it into a safe version of itself, which means: - All scripts have been removed - All <script> tags have been removed - All <style> tags have been removed - All <iframe> tags have been removed.The following HTML filter will remove all script tags and everything between them, while leaving basic markup intact: " Strip out all script tags and everything between them

### HTML filter code

<script>

      (*function* () {

*let* createdElement = document.createElement("script");

        createdElement.type = "text/javascript";

        createdElement.async = true;

*let* scriptValue = document.getElementsByTagName("script")[0];

        scriptValue.parentNode.insertBefore(createdElement, scriptValue);

      })();

    </script>

*<!--*

*If there are script tags in the attributes of other tags, strip out only the tags themselves and leave the rest of the attribute's value intact -->*

*<!-- Make sure to leave basic markup (fonts, formatting, etc.) intact*

*-->*

    <div *class*="textArea">

      <h1></h1>

      <p>This is some basic text.</p>

    </div>

*<!--*

*This is a basic HTML filter that will remove all script tags from an HTML document. It will also remove any event attributes (onclick, onmouseover, etc.) from tags to prevent any script from being executed.*

*-->*

    <script>

*//remove all script tags*

      $("script").remove();

*//remove all event attributes*

      $("\*").removeAttr("onclick onmouseover onload");

    </script>

# Question #3 (programming)

Implement HTTP digest authorization. Use a password file with salts, and reuse the BasicAuthWebServer from Chapter 9. Implement a program that allows you to add and delete passwords to and from the password file.

# Conclusion

# References

1. [↑](#footnote-ref-1)