

|  |  |  |
| --- | --- | --- |
|  | |  |
|  | | Engineering Design Document |
|  | |  |
|  | Cross-site scripting detector  **Version: 1.0**  **Last Revised:** January 15, 2011  **Author: Maksym Sukhovarov** | |

Table of Contents

[1 System Overview 4](#_Toc272235325)

[2 Assumptions and Dependencies 4](#_Toc272235326)

[3 Functional Design 4](#_Toc272235327)

[4 Technical Design 4](#_Toc272235328)

[4.1 Class diagram and description 4](#_Toc272235329)

[4.2 Interaction of components 12](#_Toc272235330)

[4.3 High-Level Design / Tasks 15](#_Toc272235331)

[4.4 Database Schema Changes 16](#_Toc272235332)

[5 Error list 16](#_Toc272235333)

[6 Outstanding Issues 16](#_Toc272235334)

Document Version History

|  |  |  |  |
| --- | --- | --- | --- |
| **Author** | **Revision No.** | **Date** | **Description of Change** |
| **Maksym Sukhovarov** | 1.0 | 09/30/2010 | First revision |
| **Maksym Sukhovarov** | 1.1 | 15/01/2011 | Update references |
|  |  |  |  |
|  |  |  |  |

Functional Design Review

| **Name** | **Title** | **Date Reviewed** | **Date Approved** |
| --- | --- | --- | --- |
| **Kyle Quest** | Architect |  |  |
| **Anatoliy Lokshin** | Development lead |  |  |
| **Yulia Kuchmai** | QA Representative |  |  |
|  |  |  |  |

Technical Design Review

| **Name** | **Title** | **Date Reviewed** | **Date Approved** |
| --- | --- | --- | --- |
| **Kyle Quest** | Architect |  |  |
|  |  |  |  |
|  |  |  |  |

Engineering Task Worklist Review

| **Name** | **Title** | **Date Reviewed** | **Date Approved** |
| --- | --- | --- | --- |
| **Anatoliy Lokshin** | Development lead |  |  |
|  |  |  |  |
|  |  |  |  |

References

|  |  |  |
| --- | --- | --- |
| **Document Name** | **Author** | **Location** |
| Cross-site cripting detector requirenmet document | Maksym Sukhovarov | [svn://qaautoserv/SecurityFramework/branches/SF1.0-Development/Docs/Requirements/Cross-site scripting detector requirements.docx](svn://qaautoserv/SecurityFramework/branches/SF1.0-Development/Docs/Requirements/Cross-site%20scripting%20detector%20requirements.docx) |
| MetraTech Security Framework Specification | Kyle Quest | [svn://qaautoserv/SecurityFramework/branches/SF1.0-Development/Docs/ MtSecurityFramework.doc](svn://qaautoserv/SecurityFramework/branches/SF1.0-Development/Docs/%20MtSecurityFramework.doc) |
| Subsystem Interfaces of Security Framework | Maksym Sukhovarov | [svn://qaautoserv/SecurityFramework/branches/SF1.0-Development/Docs/Architecture Design/Subsystem Interfaces of Security Framework.docx](svn://qaautoserv/SecurityFramework/branches/SF1.0-Development/Docs/Architecture%20Design/Subsystem%20Interfaces%20of%20Security%20Framework.docx) |
| Processor subsystem design | Viktor Grytsay | [svn://qaautoserv/SecurityFramework/branches/SF1.0-Development/Docs/Architecture Design/Processor subsystem design.docx](svn://qaautoserv/SecurityFramework/branches/SF1.0-Development/Docs/Architecture%20Design/Processor%20subsystem%20design.docx) |

Version Configuration

|  |  |  |
| --- | --- | --- |
| **Name** | **Version** | **Additional Comments** |
| MetraNet | 6.05 |  |
|  |  |  |

# System Overview

Cross-site scripting (XSS) detector should be corresponding the requirements (see document “RD Cross-site scripting detector.docx”). The main purpose of the detector is a defenithion XSS attack.

# Assumptions and Dependencies

Security Monitor depends on the configuration loader, detector subsystem, decoding engines and on processor subsystem.

# Functional Design

N/A

# Technical Design

Input string can contains an escape sequence and some encoding simblos. As a rule, to determine which of the algorithms for encoding was used can be by indirect attribute. Thus, other engines must to be included in processing of input string. It may be some unescaping and encoding engines. What are engines should be to includes to processing must be defined in a configuration file. Using conditional jumps processing of one engine to another engine we may to define complex sequences of processing.

And it can be realized in Processor subsystem.

## XSS detection algorithm

Approximate algorithm for identification of XSS attack see figure 1.



Figure 1. XSS Detectection Algorithm

Algorithm has a following steps:

1. Beginning of the algorithm
2. **Are all symbols encoded? (Unicode, Base64 …)** – check the input string. If all characters are encoded, it is believed that the XSS attack was. For example:

* ”*PHNjcmlwdD4KYWxlcnQoJ1hTUycpOwo8L3NjcmlwdD4=*” – **base64 encodng**;
* “*&#60;&#115;&#99;&#114;&#105;&#112;&#116;&#62;&#97;&#108;&#101;&#114;&#116;&#40;&#39;&#1055;&#1088;&#1080;&#1074;&#1077;&#1090;&#39;&#41;&#59;&#60;&#47;&#115;&#99;&#114;&#105;&#112;&#116;&#62*” – **Unicode encoding**;

If not all symbols encoded then go to the step 3.

If only all symbols was encoded then takes a decision that it is a XSS attack and go to the and of algorithm.

1. **Do contain are malicious expressions?** - Input string checks on the malicious expressions. If a string contains malicious symbols , it is possible that the XSS attack was. If not go to the step 4.
2. **Normalization** – unescape or encoding input string. Depending on which the characters were found in the input string will be depended on which decoder algorithm need to use:

* **‘%’ and hex code (2 digits) ‘%u’ and hex code (4 digets) and not** **contains %2B code** – Possibly, should be used JavaScript unescape. For example if input string is “+XSS привет/. <script+ >”, then encoding string is *“+XSS%20%u043F%u0440%u0438%u0432%u0435%u0442/.%20%3Cscript+%20%3E*”
* **‘%’ and hex code (2 digits) and sometimes contains %2B code** – Possibly, should be used URL decoding. For example if input string is “+XSS привет/. <script+ >”, then encoding string is *“%2BXSS+%D0%BF%D1%80%D0%B8%D0%B2%D0%B5%D1%82%2F.+%3Cscript%2B+%3E*”
* **‘*&lt;*’ or ‘*&gt;*’** – Possibly, should be used HTML decoding. For example if input string is “+XSS привет/. <script+ >”, then encoding string is “+XSS привет/. &lt;script+ &gt;”

After encoding go to the step 5.

1. **Comparison of original string to decoded string** – If strings are equal that go to the end (step 6) – XSS attack wasn’t found. If strings are not equale that go to the step 3.
2. Ending of the algorithm

## Class diagram and description

A high-level class diagram for the XSS Detector is shown on the Figure 2.

**IEnngine** base interface for engine is describe in [“Subsystem Interfaces of Security Framework”](#refSubsystemInterfsceSecFramework).

**Processor** subsystem was described in [“Processor subsystem engineering design document”](#refProcessorSubsystemDesign).

XSS detector consists from two functionality elements:

1. Decode input string – it is following engines:
   * + **HtmDecoder**;
     + **UrlDecoder**;
     + **EscapeDecoder**;
     + **UnocodeDecoder**;
     + **Base64Decoder**;
2. Search the malicious expressions – it is **SimpleXssDetector**.

This functionality lets implement the requirements which has been described in the requirements document. And rules let a call that or other engines for decoding makes it easy to customize the processing of the input string.

Decoding process solves the obfuscation code problem such as:

* URL encoding;
* Encoding binary data Base64 group;
* HTML encoding;
* Javascript escaping;
* Unicode encoding;

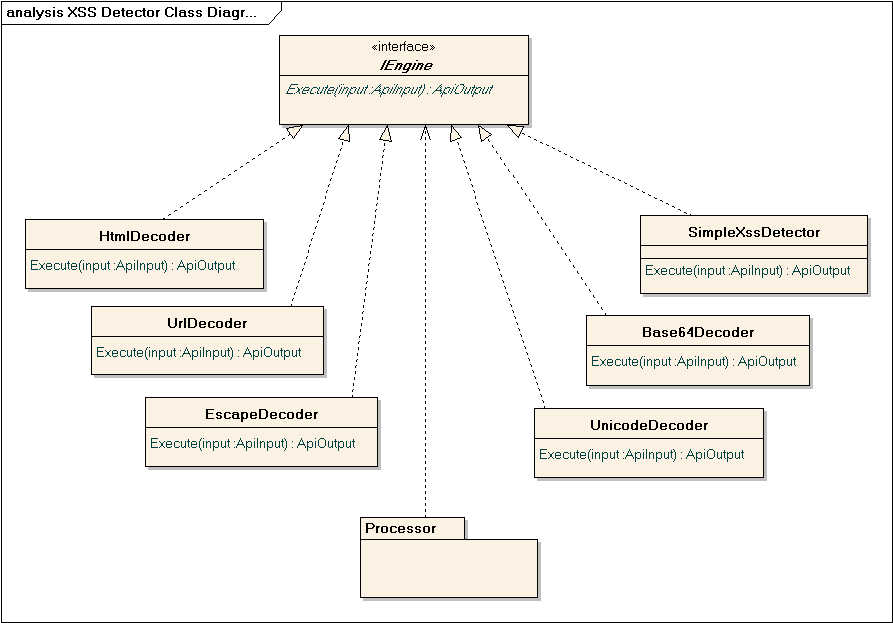


Figure 2. XSS Detector class diagram

The remaining tasks is:

1) Java script code detection and global functions detection;

2) XML DOM Elements;

3) HTML-tags detection;

4) VB script code detection;

5) Signs detection.

With these tasks should be solved an engine – it is **SimpleXssDetector** is shown on the Figure 3.

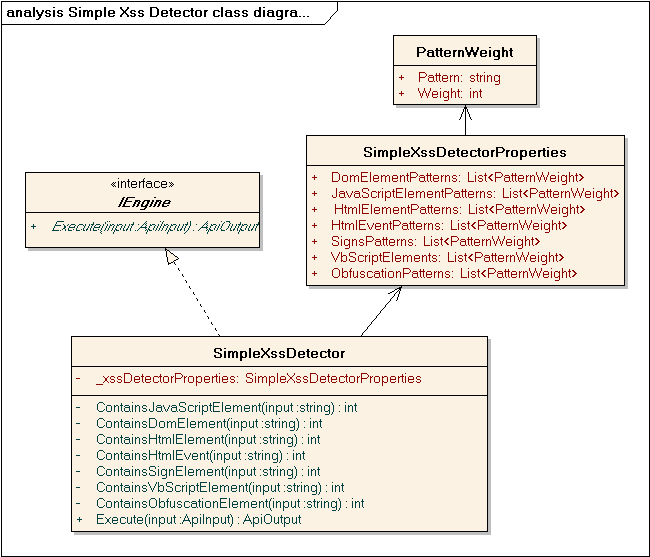


Figure 3. Simple XSS Detector class diagram

**PatternWeight** class used as container for string pattern and weight of this pattern.

All string paterns should be strored in to **SimpleXssDetectorProperties** class**.**

|  |  |
| --- | --- |
| **Member** | **Description** |
| JavaScriptElementPatterns: List<PatternWeight> | Contains JavaScript array of **PatternWeight** (string andweightof this string). |
| DomElementPatterns: List<PatternWeight> | Contains DOM array of **PatternWeight** (string andweightof this string). |
| HtmlElementPatterns: List<PatternWeight> | Contains HTML array of **PatternWeight** (string andweightof this string). |
| HtmlEventPatterns: List<PatternWeight> | Contains HTML Event array of **PatternWeight** (string andweightof this string). |
| SignsPatterns: List<PatternWeight> | Contains Signs array of **PatternWeight** (string andweightof this string). |
| VbScriptPatterns: List<PatternWeight> | Contains VbScript array of **PatternWeight** (string andweightof this string). |
| ObfuscationPatterns: List<PatternWeight> | Contains obfuscation array of **PatternWeight** (string andweightof this string). |

**SimpleXssDetectorProperties** should be serializeble from config file.

**IEngine** – it is a base interface for any engines;

**SimpleXssDetector** class – it is engine which detect XSS attack.

|  |  |
| --- | --- |
| **Member** | **Description** |
| \_xssDetectorProperties: SimpleXssDetectorProperties | Contains **SimpleXssDetectorProperties** which was serialized from config file (private property) |
| ContainsJavaScriptElement(input: string) : int | Check imput string. If JavaScript elements was not find the return 0. Otherwise it returns the sum of weights was detected patterns. (private method) |
| ContainsDomElement(input: string) : int | Check imput string. If DOM elements was not find the return 0. Otherwise it returns the sum of weights was detected patterns. (private method) |
| ContainsHtmlElement(input: string) : int | Check imput string. If HTML elements was not find the return 0. Otherwise it returns the sum of weights was detected patterns. (private method) |
| ContainsHtmlEvent(input: string) : int | Check imput string. If HTML events was not find the return 0. Otherwise it returns the sum of weights was detected patterns. (private method) |
| ContainsSignElement(input: string) : int | Check imput string. If sign elements was not find the return 0. Otherwise it returns the sum of weights was detected patterns. (private method) |
| ContainsVbScriptElement(input: string) : int | Check imput string. If VbScript elements was not find the return 0. Otherwise it returns the sum of weights was detected patterns. (private method) |
| ContainsObfuscationElement(input: string) : int | Check imput string. If obfuscation elements was not find the return 0. Otherwise it returns the sum of weights was detected patterns. (private method) |
| Execute (input: ApiInput): ApiOutput | Execute engine. Execute all ContainsXXX methods. If the total weight becomes equal to more than 100, then in the input string will be contained the malicious expression and will be thrown an exception. |

After bebuging and testing will be formed rules, which with high probability will be determined a malicious string sequence.

# Error list

The following table shows a listing of errors that can occur.

|  |  |  |  |
| --- | --- | --- | --- |
| **Error Code** | **Error Message** | **Description** | **Area** |
|  |  |  |  |
|  |  |  |  |

# Outstanding Issues

List all open issues regarding this document.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Date raised** | **Description and Resolution** | **Page/ Section** | **Raised by** | **Allocated to** | **Status** |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |