|  |  |
| --- | --- |
| January 20, 2011 |  |
|  | Security Framework |
| Version 1.0 |  |
| Cross-site scripting detector user’s guide | |



Table of Contents

1 Introduction 4

2 XSS detection algorithm 4

3 XSS detector configuration 7

3.1.1 First step of XSS algorithm (“Begin”) 7

3.1.2 Second step of XSS algorithm (“Do not all symbols in Unicode encoding?”) 7

3.1.3 Third step of XSS algorithm (“Do not all symbols in Base64 encoding?”) 8

3.1.4 Forth step of XSS algorithm (“Do not contain malicious expressions?”) 8

3.1.5 Fifth step of XSS algorithm (“Try to decode HTML encoding”) 9

3.1.6 Sixth step of XSS algorithm (“Try to sanitize Base64 encoding”) 10

3.1.7 Seventh step of XSS algorithm (“Try to decode URL encoding”) 10

3.1.8 Eighth step of XSS algorithm (“Try to decode JavaScript encoding”) 11

3.1.9 Eighth step of XSS algorithm (“Try to decode JavaScript encoding”) 12

3.1.10 Tenth step of XSS algorithm (“End”) 13

3.2 General API 14

3.3 API Examples 15

3.4 Appendix A – XSS detector configuration (full version) 16

References

|  |  |  |
| --- | --- | --- |
| **Document Name** | **Author** | **Location** |
| **­­­Cross-site scripting detector design** | MaksymSukhovarov | [svn://qaautoserv/SecurityFramework/branches/SF1.0-Development/Docs/Architecture Design/Cross-site scripting detector design.docx](svn://qaautoserv/SecurityFramework/branches/SF1.0-Development/Docs/Architecture%20Design/Cross-site%20scripting%20detector%20design.docx) |
| **Decoder user guide** | Anatoliy Lokshin | [svn://qaautoserv/SecurityFramework/branches/SF1.0-Development/Docs/Manuals/Decoders user guide.docx](svn://qaautoserv/SecurityFramework/branches/SF1.0-Development/Docs/Manuals/Decoders%20user%20guide.docx) |
| **Processoruser guide** | VictorGrytsay | [svn://qaautoserv/SecurityFramework/branches/SF1.0-Development/Docs/Manuals/Processor user guide.docx](svn://qaautoserv/SecurityFramework/branches/SF1.0-Development/Docs/Manuals/Processor%20user%20guide.docx) |
|  |  |  |

# Introduction

Cross-site Scripting (XSS) has emerged to one of the most prevalent type of security vulnerabilities. While the reason for the vulnerability primarily lies on the server-side, the actual exploitation is within the victim’s web browser on the client-side. Therefore, MetraTechSecurity Framework contains the XSS detector which is capable to prevent passing unsafe data to the application.It has enough high probability to define dangerous queries.

# XSS detection algorithm

Accordingly [“Cross-site scripting detector design”](#refXSSDetectorDesign) document the detection algorithm consists of deferent decoders and a simple XSS detector. Algorithm in this document was optimized (seeFigure1 – “XSS detector algorithm”).

The XSS detector algorithm consists of following steps:

1. **Begin** - the algorithm beginning;
2. **Do not all symbols in Unicode encoding?** - checks the input string. If all characters are encoded, it is believed thatthereis the XSS attack and go to end of algorithm (step 10). Else go to next step;
3. **Do not all symbols in Base64 encoding?** - checks the input string. If all characters are encoded, it is believed that there is the XSS attack and go to end of algorithm (step 10)Else go to next step;
4. **Do not contain malicious expressions? -** checks the input string on the malicious expressions. If the string contains malicious symbols, it is possible that there is the XSS attack and go to end of algorithm (step 10). Else go to next step;
5. **Try to decode HTML encoding\*[[1]](#footnote-1)**– trying to find the HTML encoding. If it was found then try to decode and if input and output strings are not equal then go to step 4 else go to next step;
6. **Try to sanitize BASE64 encoding\*\*[[2]](#footnote-2)**– trying to find the BASE64 encoding. If it was found then try to decode and if input and output strings are not equal then go to step 4 else go to next step;
7. **Try to decode URL encoding\*1**– trying to find the URL encoding. If it was found then try to decode and if input and output strings are not equal then go to step 4 else go to next step;
8. **Try to decode JavaScript encoding\*1**– trying to find the JavaScript encoding. If it was found then try to decode and if input and output strings are not equal then go to step 4 else go to next step;
9. **Try to decode VBScript encoding\*1**– trying to find the VBScript encoding. If it was found then try to decode and if input and output strings are not equal then go to step 4 else go to next step;
10. **End**– the end of algorithm.



Figure 1 – “XSS detector algorithm”

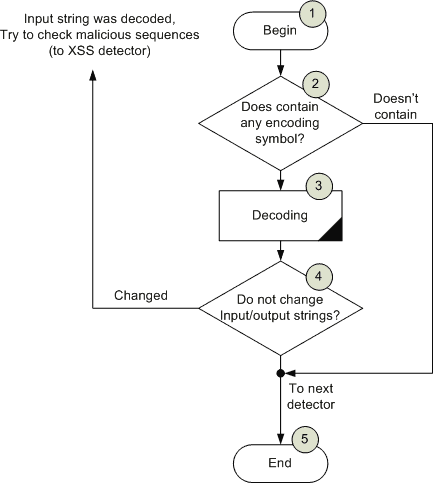


Figure 2 – “Detailed description of decoders”

The basic principle of the algorithm is to search for malicious expressions (it is simple XSS detector). When the encoded sequences are detected it tries to decode them. The decoded sequence is sent into simple XSS detector.

# XSS detector configuration

The XSS detector algorithm can be implementedby processor’s rules.Followingprocessor rules (see Chapter “Rules” in document)could be used for this implementation:

* + Sequence rule;
  + Switch rule;
  + Extension switch rule;

### First step of XSS algorithm (“Begin”)

The configuration of XSS detector represented in MtSfSsProcessor.xml file with “Processor.XssDetector” engine Id. And it can be represented as first step of XSS algorithm (see XSS detection algorithm chapter)

<EngineId="Processor.XssDetector"MaxExecution="100"Category="Xss"IsDefault="true"

IdFirstRule="IsAllSimbolsUnicodeEncoding"RealType="MetraTech.SecurityFramework.ProcessorEngine">

<Rules>

. . . . . . . . . . .

</Rules>

</Engine>

Attribute description:

* + MaxExecution**="100"** – maximum executions count for rules for this engine is 100;
  + Category="Xss" – the category of processor is XSS;
  + IsDefault="true" – this engine is default for XSS processor category;
  + IdFirstRule="IsAllSymbolsUnicodeEncoding" – the starting rule id for this engine is IsAllSimbolsUnicodeEncoding;
  + RealType="MetraTech.SecurityFramework.ProcessorEngine" – it’s the type of processor engine.

**Rules** tag defines the rules for this engine. The rules number for this engine is 14. Rile description see below.

### Second step of XSS algorithm (“Do not all symbols are in Unicode encoding?”)

The second step of XSS algorithm(“**Do not all symbols are in Unicode encoding?**”)is represented by two rules:

1. Rule with id= IsAllSymbolsUnicodeEncoding;
2. Rule with id= ThrowExceptionFoundUnicodRule;

<RuleId="IsAllSymbolsUnicodeEncoding"DefaultIdRule="TryToFindBase64Rule"IdExceptionRule="StopRule"

RealType="MetraTech.SecurityFramework.SwitchRule">

<Cases>

<CaseCompareValue="(?i:^(?:&amp;#0\*[0-9ABCDEF]{2,5};?|&amp;#x0\*[0-9ABCDEF]{2,5};?|%[0-9ABCDEF]{2}|\\x[0-9ABCDEF]{2}|\\u[0-9ABCDEF]{4}|%u[0-9ABCDEF]{4}){10,}$)"

OperationType="RegexIsMatch"IdNextRule="ThrowExceptionFoundUnicodRule" />

</Cases>

</Rule>

<!-- throw exception - Unicode encoding was found -->

<RuleId="ThrowExceptionFoundUnicodRule"Subsystem="Detector"

IdEngine="Xss.DetectUnicodeThrowException"IdNextRule="StopRule"IdExceptionRule="StopRule"/>

The IsAllSymbolsUnicodeEncoding is a switch rule with one case. The case tag contains regular expression (CompareValue attribute) which is compared to the input string. If the input string matches (OperationType attribute) then go to the ThrowExceptionFoundUnicodRule rule (IdNextRule attribute in case tag).Otherwise go to next rule which id is TryToFindBase64Rule(DefaultIdRule attribute in rule tag).

The ThrowExceptionFoundUnicodeRule is a sequence rule which executes engine from Detector subsystem (Subsystem attribute)andengine id Xss.DetectUnicodeThrowException (IdEngineattribute). If process switch to this rule so the DetectorInputDataExceptionwillbethrown (The XSS attack was found). This engine can be found in MtSfSsDetector.xml configuration file.

### Third step of XSS algorithm (“Do not all symbols are in Base64 encoding?”)

The third step of XSS algorithm (“**Do not all symbols are in Base64 encoding?**”) is represented by one rule with id isTryToFindBase64Rule.

<RuleId="TryToFindBase64Rule"Subsystem="Detector"IdEngine="Xss.DetectBase64"IdNextRule="XssDetectorRule"IdExceptionRule="StopRule"/>

The TryToFindBase64Ruleis a sequence rule which contains engine from Detector subsystem (Subsystem attribute) and engine id Xss.DetectBase64 (IdEngineattribute). This engine is trying to find a BASE64 sequence. Minimum number of characters which will be searchedfor content Base64 sequence in the input string is specified in theMtSfSsDetector.xml configuration file.

If input string is a Base64 sequence thentheDetectorInputDataExceptionwillbetrown(The XSS attack was found) otherwise go to next rule which idisXssDetectorRule (IdNextRule tag).

### Forth step of XSS algorithm (“Do not contain malicious expressions?”)

The forth step of XSS algorithm (“**Do not contain malicious expressions?**”) represented by one rule with id is XssDetectorRule.

<RuleId="XssDetectorRule"MaxExecution="21"Subsystem="Detector"IdEngine="Xss.V2"IdNextRule="TryNormalization.Html"IdExceptionRule="StopRule"/>

The XssDetectorRule is a sequence rule which contains engine from Detector subsystem (Subsystem attribute) and engine id Xss.V2 (IdEngineattribute). It is the main engine in the algorithm.This engine trying to find a malicious code(JavaScript, VBScript, dangerous HTML tags and so on).

This engine is based on regular expressions. All regular expressions are defined in the configuration file (MtSfSsDetector.xml) and functionally divided into seven sections:

* + The regular expression patterns for search JavaScript malicious codes (JavaScript syntax found);
  + The regular expression patterns for search dangerous DOM elements (document, window, etc.);
  + The regular expression patterns for search dangerous HTML tags (<script>, <object>, etc.);
  + The regular expression patterns for search any HTML events (OnLoad, OnUnload, etc.);
  + The regular expression patterns for search signs;
  + The regular expression patterns for search VBScript malicious codes (VBScript syntax found);
  + The regular expression patterns for search obfuscations expressions (JavaScript bitwise operator, etc.);

If input string contains a malicious code then the DetectorInputDataException will be thrown (The XSS attack was found) else go to next rule which id is TryNormalization.Html (IdNextRuleattribute).

### Fifth step of XSS algorithm (“Try to decode HTML encoding”)

The fifth step of XSS algorithm (“**Try to decode HTML encoding**”) is represented by two rules:

1. Rule with id = TryNormalization.Html – it is functionality ofthe step 2 in ;
2. Rule with id = HtmlUnescape – it is functionality of the step 3 and step 4 in ;

<RuleId="TryNormalization.Html"DefaultIdRule="TryNormalization.Base64"IdExceptionRule="StopRule"

RealType="MetraTech.SecurityFramework.SwitchRule">

<Cases>

<CaseCompareValue="(?i:(?:&amp;[A-Za-z]+?;?)|(?:(?:&amp;#x?)0\*?(?:\d|[ABCDEF]){1,5};?))"OperationType="RegexIsMatch"

IdNextRule="HtmlUnescape" />

</Cases>

</Rule>

<RuleId="HtmlUnescape"Subsystem="Decoder"IdEngine="Html.V1"DefaultIdRule="XssDetectorRule"IdExceptionRule="StopRule"

RealType="MetraTech.SecurityFramework.SwitchRuleEx">

<Cases>

<CaseCompareValue=""OperationType="IsInputOutputEqual"IdNextRule="TryNormalization.Base64"/>

</Cases>

</Rule>

The TryNormalization.Htmlis a switch rule with one case. The case tag contains regular expression (CompareValue attribute) which is compared to the input string. If the input string is matched (OperationType attribute)it is considered that found with escaped sequences that can be decoded so go to the HtmlUnescape (IdNextRule attribute in case tag)rule otherwise go to next rule which id is TryNormalization.Base64(DefaultIdRule attribute in rule tag).

The HtmlUnescapeis a switch rule which contains engine from Decoder subsystem (Subsystem attribute) and engine id is Html.V1 (IdEngineattribute). The engine configuration is defined in MtSfDecoder.xml configuration file. More about HTML decoder you can see in [“Decoder user guide”](#refDecoderUserGuide) document.

The result of decoding is compared with the input string (see OperationType attribute in case tag). And if these strings are equal then go to the TryNormalization.Base64 rule (IdNextRule attribute in case tag) otherwise, it returns toXssDetectorRule (DefaultIdRule attribute in Rule tag) to checking for dangerous sequences (this is XSS detector engine).

### Sixthstep of XSS algorithm (“Try to sanitize Base64 encoding”)

The sixth step of XSS algorithm (“**Try to sanitize Base64 encoding**”) is represented by one rule with id = TryNormalization.Base64 – it is functionality of the step 3 and step 4 in ;

<RuleId="TryNormalization.Base64"Subsystem="Sanitizer"IdEngine="Base64Sanitizer.V1"DefaultIdRule="XssDetectorRule"

IdExceptionRule="StopRule"RealType="MetraTech.SecurityFramework.SwitchRuleEx">

<Cases>

<!-- Are input and output value equal then go to STOP-->

<CaseCompareValue=""OperationType="IsInputOutputEqual"IdNextRule="TryNormalization.Url"/>

</Cases>

</Rule>

The TryNormalization.Base64 is a switch rule which contains engine from Sanitizer subsystem (Subsystem attribute) and engine id is Base64Sanitizer.V1 (IdEngineattribute). This engine is trying to find the sequence of the form ". . . base64, . . . (Base64 sequence) . . " , for example data: text / html; base64, PHNjcmlwdD5hbGVydChkb2N1bWVudC5jb29raWUpPC9zY3JpcHQ+

Also this engine used the composite Base64 decoder with id = Base64.ComplexDecoder which trying to decode different versions of Base64 decoding (for more detail see [“Decoder user guide”](#refDecoderUserGuide) document).

The engine configuration defined in MtSfSanitizer.xml configuration file.Configuration for this engine is very simple and no interest.

The result of decoding is compared with the input strings (see OperationType attribute in case tag). And if these strings are equal then go to the TryNormalization.Url rule (IdNextRule attribute in case tag) otherwise, it returns toXssDetectorRule (DefaultIdRule attribute in Rule tag) to checking for dangerous sequences (this is XSS detector engine).

### Seventh step of XSS algorithm (“Try to decode URL encoding”)

The seventh step of XSS algorithm (“**Try to decode URL encoding**”) is represented by two rules:

1. Rule with id = TryNormalization.Url – it is functionality of the step 2 in ;
2. Rule with id = UrlUnescape – it is functionality of the step 3 and step 4 in ;

<RuleId="TryNormalization.Url"DefaultIdRule="TryNormalization.JavaScript"IdExceptionRule="StopRule"

RealType="MetraTech.SecurityFramework.SwitchRule">

<Cases>

<CaseCompareValue="(?i:(?:(?:%|%u|\\x)(?:2B|\d|[ABCDEF]){2,}))"OperationType="RegexIsMatch"

IdNextRule="UrlUnescape" />

</Cases>

</Rule>

<RuleId="UrlUnescape"Subsystem="Decoder"IdEngine="Url.V1"DefaultIdRule="XssDetectorRule"IdExceptionRule="StopRule"

RealType="MetraTech.SecurityFramework.SwitchRuleEx">

<Cases>

<CaseCompareValue=""OperationType="IsInputOutputEqual"IdNextRule="TryNormalization.JavaScript"/>

</Cases>

</Rule>

The TryNormalization.Url is a switch rule with one case. The case tag contains regular expression (CompareValue attribute) which is compared to the input string. If the input string is matched (OperationType attribute)it is considered that found with escaped sequences that can be decoded so go to the UrlUnescaperule(IdNextRule attribute in case tag) else go to next rule which id is TryNormalization.JavaScript(DefaultIdRule attribute in rule tag).

The UrlUnescape is a switch rule which contains engine from Decoder subsystem (Subsystem attribute) and engine id is Url.V1 (IdEngineattribute). The engine configuration defined in MtSfDecoder.xml configuration file. More about URL decoder you can see in [“Decoder user guide”](#refDecoderUserGuide) document.

The result of decoding is compared with the input strings (see OperationType attribute in case tag). And if these strings are equal then go to the TryNormalization.JavaScript rule (IdNextRule attribute in case tag) otherwise, it returns toXssDetectorRule (DefaultIdRule attribute in Rule tag) to checking for dangerous sequences (this is XSS detector engine).

### Eighth step of XSS algorithm (“Try to decode JavaScript encoding”)

The eighth step of XSS algorithm (“**Try to decode JavaScript encoding**”) represented by two rules:

1. Rule with id = TryNormalization.JavaScript – it is functionality of the step 2 in ;
2. Rule with id = JavaScriptUnescape – it is functionality of the step 3 and step 4 in ;

<RuleId="TryNormalization.JavaScript"DefaultIdRule="TryNormalization.VbScript"IdExceptionRule="StopRule" RealType="MetraTech.SecurityFramework.SwitchRule">

<Cases>

<CaseCompareValue="(?i:(?:\\[0-7]{1,3})|(?:(?:%|\\x|%u)(?:20|\d|[ABCDEF]){2,4}))"OperationType="RegexIsMatch"

IdNextRule="JavaScriptlUnescape" />

</Cases>

</Rule>

<RuleId="JavaScriptlUnescape"Subsystem="Decoder"IdEngine="JavaScript.V1"DefaultIdRule="XssDetectorRule"

IdExceptionRule="StopRule"RealType="MetraTech.SecurityFramework.SwitchRuleEx">

<Cases>

<!--Are input and output value equal then go to STOP-->

<CaseCompareValue=""OperationType="IsInputOutputEqual"IdNextRule="TryNormalization.VbScript"/>

</Cases>

</Rule>

The TryNormalization.JavaScript is a switch rule with one case. The case tag contains regular expression (CompareValue attribute) which is compared to the input string. If the input string is match (OperationType attribute)it is considered that found with escaped sequences that can be decoded so go to the JavaScriptUnescaperule (IdNextRule attribute in case tag) else go to next rule which id is TryNormalization.VbScript (DefaultIdRule attribute in rule tag).

The JavaScriptUnescape is a switch rule which contains engine from Decoder subsystem (Subsystem tag) and engine id is JavaScript.V1 (IdEngineattribute). The engine configuration defined in MtSfDecoder.xml configuration file. More about URL decoder you can see in [“Decoder user guide”](#refDecoderUserGuide) document.

The result of decoding is compared with the input strings (see OperationType attribute in case tag). And if these strings are equal then go to the TryNormalization.VbScript rule (IdNextRule attribute in case tag) otherwise, it returns toXssDetectorRule (DefaultIdRule attribute in Rule tag) to checking for dangerous sequences (this is XSS detector engine).

### Ninth step of XSS algorithm (“Try to decode JavaScript encoding”)

The ninth step of XSS algorithm (“**Try to decode JavaScript encoding**”) is represented by two rules:

1. Rule with id = TryNormalization.VbScript – it is functionality of the step 2 in ;
2. Rule with id = VbScriptUnescape – it is functionality of the step 3 and step 4 in ;

<RuleId="TryNormalization.VbScript"DefaultIdRule="StopRule"IdExceptionRule="StopRule"

RealType="MetraTech.SecurityFramework.SwitchRule">

<Cases>

<CaseCompareValue="(?i:(?:chrw?\(\s\*(?:&amp;h(?:[0-9ABCDEF]){1,4}|\d{1,5})\s\*\)))"OperationType="RegexIsMatch"

IdNextRule="VbScriptUnescape" />

</Cases>

</Rule>

<RuleId="VbScriptUnescape"Subsystem="Decoder"IdEngine="VbScript.V1"DefaultIdRule="XssDetectorRule"

IdExceptionRule="StopRule"RealType="MetraTech.SecurityFramework.SwitchRuleEx">

<Cases>

<!-- Are input and output value equal then go to STOP-->

<CaseCompareValue=""OperationType="IsInputOutputEqual"IdNextRule="StopRule"/>

</Cases>

</Rule>

The TryNormalization.VbScript is a switch rule with one case. The case tag contains regular expression (CompareValue attribute) which is compared to the input string. If the input string is matched (OperationType attribute)it is considered that found with escaped sequences that can be decoded so go to the VbScriptUnescaperule(IdNextRule attribute in case tag) else go to next rule which id is StopRule (DefaultIdRule attribute in rule tag).

The VbScriptUnescape is a switch rule which contains engine from Decoder subsystem (Subsystem attribute) and engine id is VbScript.V1 (IdEngineattribute). The engine configuration is defined in MtSfDecoder.xml configuration file. More about URL decoder you can see in [“Decoder user guide”](#refDecoderUserGuide) document.

The result of decoding is compared with the input strings (see OperationType attribute in case tag). And if these strings are equal then go to the StopRule rule (IdNextRule attribute in case tag) otherwise, it returns toXssDetectorRule (DefaultIdRule attribute in Rule tag) to checking for dangerous sequences (this is XSS detector engine).

### Tenth step of XSS algorithm (“End”)

The tenth step of XSS algorithm (“**End**”) is represented by one rule with id = StopRule.

When the process reaches this step, the algorithm terminates.

Also process will go to this rule if exception occurs.

<RuleId="StopRule"RealType="MetraTech.SecurityFramework.StopRule"/>

</Rules>

</Engine>

The full configuration with comment you can see in Appendix A – XSS detector configuration (full version)

## General API

XSS detector is accessible through the Processor property of the MetraTech.SecurityFramework.SecurityKernel class

The following methods are defined in the API:

| Method | Description |
| --- | --- |
| Execute(engineId: string, input: ApiInput): ApiOutput | Invokes a processor engine with the specified ID.  Returns a processor result.  Throws the SubsystemInputParamException when an engine with the specified ID was not found. |
| ExecuteDefaultByCategory(categoryName: string, input: ApiInput): ApiOutput | Invokes a default processor of the specified category.  Returns a processor result.  Throws the SubsystemInputParamException when a category with the specified name was not found or a default engine was not specified for the category. |
| GetEngine(engineId: string): IEngine | Retrieves an engine the specified ID.  Throws the SubsystemInputParamException when an engine with the specified ID was not found. |
| GetDefaultEngine(categoryName: string): IEngine | Retrieves a default engine for a category with the specified name.  Returns null when there is no default engine specified for the category.  Throws the SubsystemInputParamException when a category with the specified name was not found. |
| GetEnginesForCategory(categoryName: string): IEngine[] | Retrieves all engines registered for the category with the specified name. |

## API Examples

usingMetraTech.SecurityFramework;

…

stringinput = "\\u0060\\u0115\\u0099\\u0114\\u0105\\u0112\\u0116\\u0062\\u0116\\u0062";

stringactual = SecurityKernel.Processor.Api.Execute("Html.V1", input);//Exception occurs

…

stringinput = "&#x60;&#x115;&#x99;&#x60;&#x115;&#x99;&#x60;&#x115;&#x99;&#x99;";

stringactual =

SecurityKernel.Processor.Api.ExecuteDefaultByCategory(ProcessorEngineCategory.**Xss**.ToString(),

input);//Exception occurs

## ­Appendix A – XSS detector configuration (full version)

<!-- BEGIN: Rules for XSS detector-->

<EngineId="Processor.XssDetector"MaxExecution="100"Category="Xss"IsDefault="true"IdFirstRule="IsAllSymbolsUnicodeEncoding"RealType="MetraTech.SecurityFramework.ProcessorEngine">

<Rules>

<!-- BEGIN (step 2): Are all symbols encoded? (Unicode, Base64 …) – check input string. If all characters are encoded, it is believed that the XSS attack was. -->

<RuleId="IsAllSymbolsUnicodeEncoding"DefaultIdRule="TryToFindBase64Rule"IdExceptionRule="StopRule"RealType="MetraTech.SecurityFramework.SwitchRule">

<Cases>

<CaseCompareValue="(?i:^(?:&amp;#0\*[0-9ABCDEF]{2,5};?|&amp;#x0\*[0-9ABCDEF]{2,5};?|%[0-9ABCDEF]{2}|\\x[0-9ABCDEF]{2}|\\u[0-9ABCDEF]{4}|%u[0-9ABCDEF]{4}){10,}$)"OperationType="RegexIsMatch"IdNextRule="ThrowExceptionFoundUnicodRule" />

</Cases>

</Rule>

<!-- throw exception - Unicode encoding was found -->

<RuleId="ThrowExceptionFoundUnicodRule"Subsystem="Detector"IdEngine="Xss.DetectUnicodeThrowException"IdNextRule="StopRule"IdExceptionRule="StopRule"/>

<!-- try to find Base64 encoding -->

<RuleId="TryToFindBase64Rule"Subsystem="Detector"IdEngine="Xss.DetectBase64"IdNextRule="XssDetectorRule"IdExceptionRule="StopRule"/>

<!-- END : Are all symbols encoded? (Unicode, Base64 …) – check input string. If all characters are encoded, it is believed that the XSS attack was. -->

<!-- (step 3): Detect XSS injections engine -->

<RuleId="XssDetectorRule"MaxExecution="21"Subsystem="Detector"IdEngine="Xss.V2"IdNextRule="TryNormalization.Html"IdExceptionRule="StopRule"/>

<!-- BEGIN (step 4 and 5):

1) Normalization – unescape or encoding input string. Unescape sequence: HTML, JavaScript, URL

2) Compare nput and output sequences -->

<!-- HTML escaping sequence-->

<RuleId="TryNormalization.Html"DefaultIdRule="TryNormalization.Base64"IdExceptionRule="StopRule"RealType="MetraTech.SecurityFramework.SwitchRule">

<Cases>

<CaseCompareValue="(?i:(?:&amp;[A-Za-z]+?;?)|(?:(?:&amp;#x?)0\*?(?:\d|[ABCDEF]){1,5};?))"OperationType="RegexIsMatch"IdNextRule="HtmlUnescape" />

</Cases>

</Rule>

<RuleId="HtmlUnescape"Subsystem="Decoder"IdEngine="Html.V1"DefaultIdRule="XssDetectorRule"IdExceptionRule="StopRule"RealType="MetraTech.SecurityFramework.SwitchRuleEx">

<Cases>

<CaseCompareValue=""OperationType="IsInputOutputEqual"IdNextRule="TryNormalization.Base64"/>

</Cases>

</Rule>

<!-- Base64 Sanitizer -->

<RuleId="TryNormalization.Base64"Subsystem="Sanitizer"IdEngine="Base64Sanitizer.V1"DefaultIdRule="XssDetectorRule"IdExceptionRule="StopRule"RealType="MetraTech.SecurityFramework.SwitchRuleEx">

<Cases>

<!-- Are input and output value equal then go to STOP-->

<CaseCompareValue=""OperationType="IsInputOutputEqual"IdNextRule="TryNormalization.Url"/>

</Cases>

</Rule>

<!-- URL escaping sequence-->

<RuleId="TryNormalization.Url"DefaultIdRule="TryNormalization.JavaScript"IdExceptionRule="StopRule"RealType="MetraTech.SecurityFramework.SwitchRule">

<Cases>

<CaseCompareValue="(?i:(?:(?:%|%u|\\x)(?:2B|\d|[ABCDEF]){2,}))"OperationType="RegexIsMatch"IdNextRule="UrlUnescape" />

</Cases>

</Rule>

<RuleId="UrlUnescape"Subsystem="Decoder"IdEngine="Url.V1"DefaultIdRule="XssDetectorRule"IdExceptionRule="StopRule"RealType="MetraTech.SecurityFramework.SwitchRuleEx">

<Cases>

<CaseCompareValue=""OperationType="IsInputOutputEqual"IdNextRule="TryNormalization.JavaScript"/>

</Cases>

</Rule>

<!--JavaScrip escaping sequence-->

<RuleId="TryNormalization.JavaScript"DefaultIdRule="TryNormalization.VbScript"IdExceptionRule="StopRule"RealType="MetraTech.SecurityFramework.SwitchRule">

<Cases>

<CaseCompareValue="(?i:(?:\\[0-7]{1,3})|(?:(?:%|\\x|%u)(?:20|\d|[ABCDEF]){2,4}))"OperationType="RegexIsMatch"IdNextRule="JavaScriptlUnescape" />

</Cases>

</Rule>

<RuleId="JavaScriptlUnescape"Subsystem="Decoder"IdEngine="JavaScript.V1"DefaultIdRule="XssDetectorRule"IdExceptionRule="StopRule"RealType="MetraTech.SecurityFramework.SwitchRuleEx">

<Cases>

<!-- Are input and output value equal then go to STOP-->

<CaseCompareValue=""OperationType="IsInputOutputEqual"IdNextRule="TryNormalization.VbScript"/>

</Cases>

</Rule>

<!-- VBScript escaping sequence-->

<RuleId="TryNormalization.VbScript"DefaultIdRule="StopRule"IdExceptionRule="StopRule"RealType="MetraTech.SecurityFramework.SwitchRule">

<Cases>

<CaseCompareValue="(?i:(?:chrw?\(\s\*(?:&amp;h(?:[0-9ABCDEF]){1,4}|\d{1,5})\s\*\)))"OperationType="RegexIsMatch"IdNextRule="VbScriptUnescape" />

</Cases>

</Rule>

<RuleId="VbScriptUnescape"Subsystem="Decoder"IdEngine="VbScript.V1"DefaultIdRule="XssDetectorRule"IdExceptionRule="StopRule"RealType="MetraTech.SecurityFramework.SwitchRuleEx">

<Cases>

<!-- Are input and output value equal then go to STOP-->

<CaseCompareValue=""OperationType="IsInputOutputEqual"IdNextRule="StopRule"/>

</Cases>

</Rule>

<!-- END (step 4 and 5): Normalization – unescape or encoding input string. Unescape sequence: HTML, JavaScript, URL-->

<!-- (step 6): STOP algorithm-->

<RuleId="StopRule"RealType="MetraTech.SecurityFramework.StopRule"/>

</Rules>

</Engine>

<!-- END: Rules for XSS detector-->

1. It’s composite operation with contain of three steps (see Figure 2 – “Detailed description of decoders”) [↑](#footnote-ref-1)
2. It’s composite operation with contain of three steps (see Figure 2 – “Detailed description of decoders” ) But 2 and 3 steps are composite to one and using to as engine. [↑](#footnote-ref-2)