Secure Coding.
Practical steps to defend your web apps.

A Hands-on XML External Entity Vulnerability Training Module

GIAC GCIA Gold Certification

Author: Carrie Roberts, clr2of8@gmail.com
Advisor: Rich Graves

Accepted: November 1, 2013

Abstract

Many web applications that accept and respond to XML requests are vulnerable to XML External Entity (XXE) attacks due to default XML parser settings. This vulnerability can be exploited to read arbitrary files from the server, including sensitive files such as the application configuration files. This paper provides detailed instructions for building a vulnerable web application using the standard XML parser that comes with the Java development kit. A virtual machine image of the complete system is also provided, allowing experimentation and visualization of the vulnerability. The virtual machine image can be used to provide engaging, hands-on XXE training for developers and intrusion analysts. Exploitation tools and techniques for reading the applications sensitive configuration file are demonstrated. A simple method for removing the vulnerability is reviewed. Finally network intrusion analysis is performed to discover how the vulnerability was exploited and what sensitive information was exposed.

1. Introduction

Web based attacks are on the rise, and the most exploited vulnerabilities are often not the newest (Symantec Corporation, 2013). One such vulnerability that has been around for many years is XML external entity injection or XXE. This vulnerability can affect web applications that exchange information using XML messages. The intriguing thing about this vulnerability is that many popular XML parsers are vulnerable by default (XML External Entity (XXE) Processing, 2013). This means that unless a developer takes deliberate steps to remove the vulnerability, the web application may be vulnerable.

The XXE vulnerability can be exploited to expose sensitive information and perform denial of service (DOS) attacks against the web application server (HPSR Threat Intelligence Briefing Companion Report to Episode 6, 2013). For example this vulnerability can be used to read arbitrary files from the server, including sensitive files such as the application configuration files. Port scanning of the web application and other systems on the same intranet is also possible (Vorontsov, 2012).

This vulnerability is an important one to understand because it exists by default for many popular XML parsers (XML External Entity (XXE) Processing, 2013). This report explains what XML external entities are and how they are used to attack systems. It provides instructions on how to create a sample vulnerable web application. This helps to understand how easily such an application can be developed. Instructions for downloading a virtual image of the sample web application are also included. Tools and techniques for exploitation of the vulnerability are given. Lastly, methods for detecting an analyzing this attack on a system is discussed and demonstrated. This provides a hands-on environment for developers and analysts to fully understand this vulnerability and its affects.

With this comprehensive understanding of XML external entity injection, developers, analysts and administrators will be prepared to detect and defend against this important web application vulnerability.

Carrie Roberts, clr2of8@gmail.com

2. XML External Entities – The Feature

Extensible Markup Language (XML) is a feature rich and widely used information exchange format and standard. The standard allows for defining the structure of the XML using a Document Type Declaration, or DTD. The DTD provides a mechanism for defining entities whose values can be substituted into the XML document contents. This is helpful when the entity value is used multiple times.

The XML in Figure 1 contains a DTD with an entity called "orientation" defined. The value of the "orientation" entity is set to "some orientation." Following the DTD are the XML elements which contain a "url" tag and an "orientation" tag. The value of the "orientation" element is "&orientation;" which is the name of the entity with an ampersand (&) symbol before and a semi-colon (;) after. This instructs the XML parser to replace "&orientation;" with the entity value defined in the DTD, namely "some orientation."

```
<!DOCTYPE convert[
<!ENTITY orientation "some orientation">
]>
<convert>
        <url>www.something.com</url>
        <orientation>&orientation;</orientation>
</convert>
```

DTD

Document Content

Figure 1. General Parsed Internal Entity

The full description of this example is a "General Parsed Internal Entity". General Entity means that the entities are for use within the document content (Extensible Markup Language (XML) 1.0, 2008). The term "Parsed" means that the XML parser will evaluate it according to the XML standard instead of just passing it through to the application as with an unparsed entity. An internal entity means that the value of the entity can be determined by the XML parser without accessing an external resource.

The XML parser will deliver the XML shown in Figure 2 after parsing of the general parsed internal entity. Notice that the DTD has been removed and that the orientation element has a value of "some orientation" as was defined in the DTD.

Carrie Roberts, clr2of8@gmail.com

```
<convert>
        <url>www.something.com</url>
        <orientation>some orientation</orientation>
</convert>
```

Figure 2. Document Content after Parsing of General Parsed Internal Entity

In addition to internal entities, there are external entities. External entities reference something outside of the parser itself in order to determine the value of the entity. One type of external entity is declared by using the "SYSTEM" keyword and passing a URI from which to fetch the value.

```
<!DOCTYPE convert[
<!ENTITY orientation SYSTEM "file:///etc/converter/converter.conf">
]>
<convert>
        <url>www.something.com</url>
        <orientation>&orientation;</orientation>
</convert>
```

DTD

Document Content

Figure 3. General Parsed External Entity

An example using a general parsed external entity is shown in Figure 3. The use of the "SYSTEM" keyword designates that the entity value should be read from the URI that follows. In this case the URI is a file located at "/etc/converter/converter.conf" but other URI's such as a web page URL can be specified as well. The resulting XML after being parsed would contain the contents of the converter.conf file in the "orientation" element.

For brevity in this paper, the terms "internal entity" and "external entity" will be used when referring to a general parsed internal entity and general parsed external entity, respectively. The use of entities is powerful and many XML parsers allow them by default, including most Java XML parsers (XML External Entity (XXE) Processing, 2013). Unfortunately this introduces multiple security issues including disclosure of sensitive information and denial of service.

3. XML External Entities – The Vulnerability

The DOCTYPE declaration with the SYSTEM keyword causes an XML parser to read data from a URI and permits it to be substituted into the document content. This document content can then be viewed by users of the application in various ways, including through error messages.

The example user input shown in Figure 3 includes a file URI pointing to the converter.conf file in the /etc directory. The contents of that file will be substituted into the XML document as the orientation parameter. Configuration files typically contain sensitive information such as database passwords. Consider a scenario in which the application will only accept a value of portrait or landscape for the orientation parameter. In this case, the application may return an error message stating that the orientation parameter is invalid and reflect the value submitted back to the user as a helpful error message. The contents of the converter.conf file are returned because the parser has fetched the file contents and substituted it for the value of the orientation parameter. Even without an error message, if the application lets users read data that has been submitted, the sensitive data will be exposed. For example, if the application allows submission of contact information and then lets user read that contact information.

Any file that is readable by the user under which the application is running can be read this way, including the sensitive user information found in /etc/passwd. Note that since the password hashes are stored in a file that is typically not world readable that retrieving the contents of this file (/etc/shadow) is likely not possible. The exceptions are if the application is running as the root user or if the shadow file or backups of the shadow file are readable by the user under which the application is running.

Another way that an XXE vulnerability can be used to reveal sensitive information is by pivoting to other systems in the same intranet where the application is running. An example scenario consists of another internal server hosting a site such as “internal.mycompany.com” that contains non-public information that should only be accessible by someone on the internal network. If the XXE vulnerable server is hosted inside this same intranet, the contents of “internal.mycompany.com” can be successfully fetch by the XML parser and returned to an outside user.

It is also common that developers provide functionality that is only accessible via a call to "localhost" such that no public access is given. A URI of "http://localhost/my_app/configs" may be implemented to let a developer read application configuration settings when ssh'ed into the server itself. If this URI is passes in a DOCTYPE declaration it can be successfully retrieved by the XML parser which runs on localhost and the contents of that site can be erroneously and unexpectedly returned to a user of a public API.

A denial of service (DOS) attack is also possible with an XXE vulnerability. This can be exploited by causing the parser to reading content from a continuous (non-ending) stream such as local devices like /dev/zero and /dev/random. The stream at /dev/zero will return as many nulls as are read from it and /dev/random will do likewise, returning random numbers. The parser will attempt to read this file until it reaches the end of the stream, which it never will. A few such requests will tie up all available threads for processing and result in a successful denial of service attack on the server. Although several references make note of the use of /dev/zero (Herzog, 2010) and /dev/random (XML External Entity (XXE) Processing, 2013), this particular DOS attack does not work against the openjdk parser included in the sample application described in this paper. When these DOS attacks are attempted on this parser an exception is thrown due to invalid bytes being read from the stream, thus circumventing the DOS attack.

Taking a look at a real world scenario using a sample vulnerable web application helps to clarify these vulnerabilities. The sample application will be described below. The steps taken to create the application are also included, showing how easily such an application can be developed.

4. A Sample Vulnerable Web Application

A web application that parses user supplied XML is susceptible to an XXE attack if it allows external general entities. To demonstrate this, a sample application that provides a public interface from converting HTML to PDF is used. Instructions for creating the application are provided as well as a download of a virtual machine image that contains the vulnerable web application. The virtual machine image also includes a variety of tools that can be used to perform the exploit.

Figure 4 shows the home page for the sample web application. It explains that this application is used to convert HTML documents to PDF documents and it provides an application programming interface (API) for doing so.

Figure 4. Home Page for the HTML to PDF Sample Web Application

Clicking on the "our conversion API" link describes how this application accepts and responds to HTML POST requests with XML payload. Figure 5, Figure 6 and Figure 7 show the information provided on the API page.

Figure 5. XML Payload for the POST Request

Figure 5 explains the XML format that should be sent to the application to request a conversion. The request should us an HTTP method of POST. The "url" element specifies the HTML page to be converted and the" orientation" element specifies whether the page orientation of the PDF should be landscape or portrait.

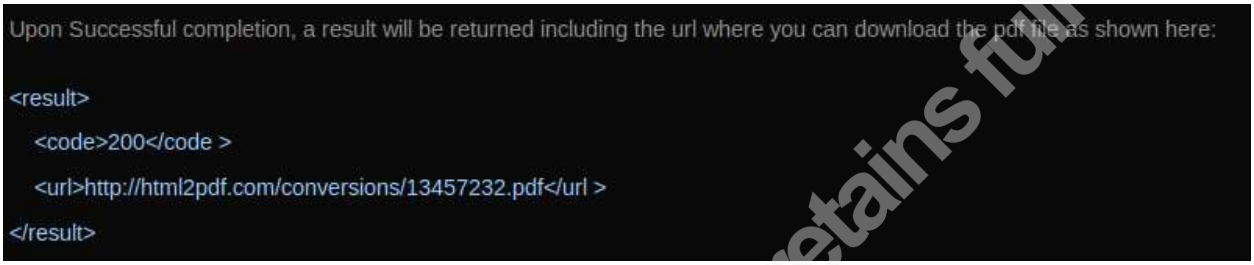

Figure 6. XML Response to API Request

If the request was properly formatted, an XML response similar to the one shown in Figure 6 is returned. It contains a status code and the URL where the converted PDF can be downloaded. Note that for this sample application this is not a functional URL and is only returned as an example.

Figure 7. XML Error Response

In the event that an error results, an XML message is returned. Figure 7 gives a description of the XML message that is returned upon error. It includes an error code and a description of the error. For example, if the request specified an orientation of "panorama", then an error with a description of "Invalid orientation parameter: panorama" is returned.

The requirement for an XXE vulnerability is that the application parses XML and external general entities. In addition, it must in some way reflect part of the XML data back to a user. In the case of the sample application this is done through a descriptive error message when an

invalid orientation is specified. There are others way that applications may reflect the information back to a user as well. For example, user comments may be submitted to an online forum through an API that defines the comment in an XML payload. When any user views the forum, they will see what was submitted as the "comment" portion of the XML payload.

The sample application also reads a configuration file located at "/etc/converter/converter.conf" at startup which contains sensitive database information. This file will be the focus of the exploit described in a later section.

Instructions are included in the following section for creating the pieces of this web application that are critical for demonstrating the vulnerability. It does not include the code for displaying the nicely formatted pages shown in Figure 4 and beyond because these are only informational and are not actually part of the underlying vulnerability. However, these pages are functional on the virtual machine image containing the sample application which is available for download.

5. Create an XXE Vulnerable Web Application

Instructions for creating the sample vulnerable web application are included here. It shows how easily it is to develop a vulnerable web application simply by using default settings. Alternatively, a virtual machine image of the sample web application can be downloaded instead of creating it manually. The manually created version described here only contains the bare application needed to demonstrate the vulnerability, whereas the virtual machine image contains other helpful information and exploitation tools. Follow the instructions below to manually create the sample web application.

5.1. Pre-requisites

The vulnerable web application is built on a virtual machine image of Ubuntu 13.04 Desktop. VMware Player is the virtualization software used and must be installed first. Use VMware Player to create a new virtual machine and install Ubuntu 13.04 Desktop on it. VMware Player can be downloaded from www.vmware.com and Ubuntu can be downloaded from www.ubuntu.com. Steps for completing these pre-requisites are readily available on the internet and are not included here.

5.2. Install the Tools Needed for Development

To build the vulnerable web application, Groovy and Grails will be used. Groovy is a dynamically typed language which runs on the Java platform. Grails is a web application framework that uses Groovy to enable quick and easy web application development. An easy way to install both Groovy and Grails is to use the Groovy Environment Manager, or GVM. The first step will be to install the Java Development Kit (JDK) since Groovy utilizes it and it includes the XML parser. A helper application, cURL, will be used to download the GVM. After executing the following commands in a terminal window on the Ubuntu virtual machine the system will be prepared with the needed tools to develop the vulnerable web application.

```
sudo apt-get install openjdk-6-jdk
sudo apt-get install curl
curl -s get.gvmtool.net | bash
```
note: open a new terminal as directed
```
gvm install grails 2.3.0
```

5.3. Develop the Web Application

With Grails installed, the system is prepared for writing the code for the web application. The following commands will created an application called "converter" and run it.

```
grails create-app converter
cd converter
grails run-app
```

The "run-app" command will cause the source code to be compiled, after the compilation is complete the output will indicate that the server is running and can be viewed by visiting http://localhost:8080/converter. Opening this link in a web browser will display the default index page containing information about Grails.

To demonstrate the XXE vulnerability the web application will be coded to provide a public API that accepts and responds to XML messages. A controller called "convert" will be used as the access point for the API. The create-controller command will create the controller, called "Convert" in this case. Open a new terminal in order to leave the Grails app running in the previous one and enter the following command to create the "Convert" controller.

grails create-controller Convert

The application is now prepared to accept and respond to http requests at http://localhost:8080/converter/convert. Accessing this URL in the browser will yield a "404" not found error until additional code is in place. The XML parsing code can now be put in place by editing the default "index" action inside of the ConvertController to look like the following.

```
def index() {
  if (request.method == "POST") {
    def parser = new XmlSlurper()
    def orientation = parser.parseText(request.reader.text).orientation.text()
    if (orientation.equalsIgnoreCase("portrait") | orientation.equalsIgnoreCase("landscape")){
      response.status = 200
      render "<result>\n  <code>200</code>\n  " +
             "<url>http://html2pdf.com/conversions/13457232.pdf</url>\n</result>"
    }
    else{
      response.status = 400
      render "<result>\n  <code>400</code>\n  <description>Invalid orientation parameter: " +
             orientation + "</description>\n</result>"
    }
  }
  else { //it's not a POST request
     render "Default Page"
  }
}
```

Also add "import groovy.util.XmlSlurper" to the top of the ConvertController.groovy file immediately after the first line that says "package converter."

It may be necessary to stop and restart the application after adding this code. This can be done by pressing ctrl+C in the terminal window where the "grails run-app" command was entered, and then reentering "grails run-app" to start it again.

With this code in place, if the request comes in as an http POST request the server processes the XML, otherwise the text "Default Page" is returned and displayed in the browser. If it is a POST request, a parser is instantiated from the XmlSlurper class. The XmlSlurper provides access to the underlying XML parser that comes bundled with the Java Development Kit. This is the key element related to the XXE vulnerability that is explored in this paper. Next, the "orientation" element is parsed from the XML. If it has a value of "portrait" or "landscape" (case insensitive), then a result indicating success is returned. Otherwise, an error code and descriptive message is returned stating that the "orientation" value is invalid. This is all the code that is needed to

demonstrate the XXE vulnerability. Notice that nothing special was done to create the vulnerability and that a parser with default settings is used.

Lastly, create the application configuration file that will be the target of the exploit. Name the configuration file "converter.conf" and place it in a directory also named "converter" inside of the "/etc" directory. Edit the resulting "/etc/converter/converter.conf" file to contain the sensitive database information shown in Figure 8.

```
# Database configuration
export DATABASE_HOST=apiconverter.czs893zdu2v5.us-east-1.rds.amazonaws.com
export DATABASE_USER=html2pdfroot
export DATABASE_PASSWORD=ThisIzTheePwd4Rdb
```

Figure 8. Contents of Sample Web Application Configuration File

This completes the steps for manually creating the basic vulnerable web application. In addition you can download a virtual machine image containing the application including exploitation tools as described in the following section.

6. Download an XXE Vulnerable Web Application

A virtual machine image of the vulnerable web application can be downloaded from sourceforge at: https://sourceforge.net/projects/xmlexternalentitytraining. The vulnerable web application described in the previous section is included as well as the tools used in the next section to perform the exploit. Unzip the downloaded file after download is complete.

The virtual machine image contains the basic code that was manually created in the last section. It also contains the enhanced user interface shown in Figure 4. The three tools for exploitation as described in the next section are also included.

VMware player is required for running the virtual machine image. It is freely downloadable from www.wmware.com for many different operating systems. Once VMware player is installed, select "Open a Virtual Machine" from the menu. Browse to the unzipped download and select the file called "XML_Vulnerability.vmx" and choose "Open." With the "XML_Vulnerability" virtual machine selected, choose "Play virtual machine" to start the virtual

machine. Select "I copied it" if a warning dialog appears inquiring as to whether the virtual machine was moved or copied.

The username and password for the virtual machine is listed in Figure 9. Log in to the virtual machine by entering "xmlvulnpwd" as the password for the "XML Vuln" user. If the desktop of the virtual machine is not completely filling up the VMware player window, simply resize the window and then maximize it again.

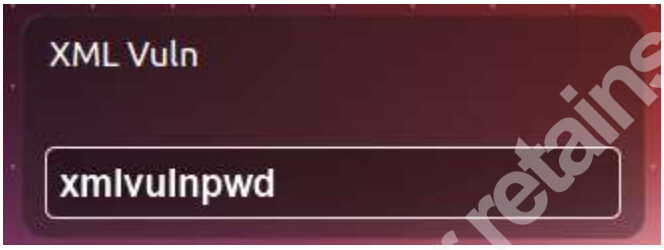

Figure 9. Username and Password for Virtual Machine Image

With the sample application in place either through manual creation or download, the actual XXE vulnerability can be exploited. Tools and techniques for doing this are included in the following section.

7. Exploit the Vulnerable Web Application

Many different tools can be used to exploit the sample web application. The web application itself must be running before the exploits are attempted. Open a terminal window and enter the following commands to start the web application.

```
cd converter
grails run-app
```

Figure 10 shows the launcher menu of the virtual machine that can be used to easily access the tools needed for exploitation.

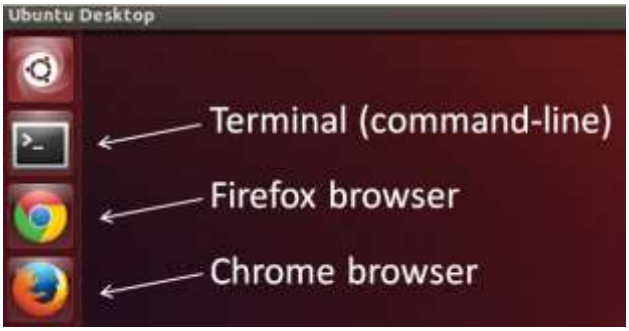

Figure 10. Ubuntu Desktop Launcher Inside Downloaded Virtual Machine

Three different tools for exploiting the vulnerability are reviewed. The first is the Chrome web browser with the Advanced Rest Client extension. The second is the Firefox web browser with the RESTclient Add-on. The third tool is the cURL program which is run on the command line. The exploitation payload shown in Figure 3 can be sent to the vulnerable web application using any of these. The use of the Chrome web browser will be the first exploitation tool discussed.

7.1. Exploit with Google Chrome Advanced Rest Client

Start the Chrome web browser by clicking on the icon in the launcher menu as shown in Figure 10. Two tabs will be started. The first is the home page for the vulnerable web application and the second is the advanced rest client extension which has already been installed.

From the "Advanced Rest Client Application" tab enter the target url of the vulnerable web application (http://localhost:8080/converter/convert). Select the radio button next to "POST" to send the payload as an HTTP POST request as is typical for this kind of request. On the "Payload" tab enter the XML payload shown in Figure 3. Set the "Content-Type" to "application/xml" and then click "send" to send the crafted HTTP request to the vulnerable web application.

The response from the web application will be displayed. Figure 11 shows the response which includes the sensitive database configuration information found in the /etc/converter/converter.conf file. This information could be used to further compromise the application by directly logging into the database.

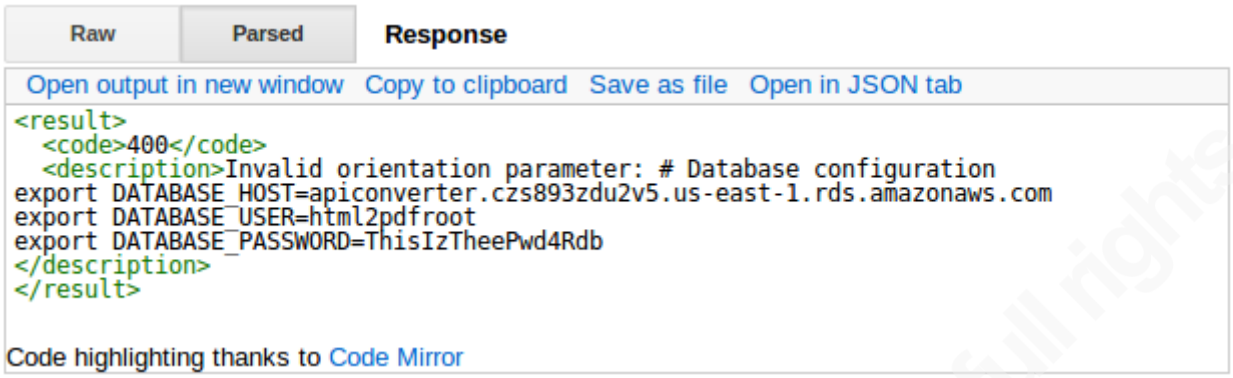

Figure 11. XML Server Response Containing Sensitive Configuration File Data (Chrome)

Another method of exploiting the application uses the Firefox browser.

7.2. Exploit with Firefox RESTClient Add-on

The Firefox with the RESTClient Add-on can also be used to perform the same exploit. Start Firefox by clicking on the link in the launcher as shown in Figure 10. Two tabs are already open. One shows the home page of the vulnerable web application and the other shows the RESTClient Add-on.

From the RESTClient tab, set the method to "POST" and enter the target URL of http://localhost:8080/converter/convert. Place the XML payload shown in Figure 3 in the "Body" window and click the red "SEND" button. The web application response will be returned. Click on the "Response Body (Highlight)" tab to see the results of the exploitation as shown in Figure 12.

Carrie Roberts, clr2of8@gmail.com

Figure 12. XML Server Response Containing Sensitive Configuration File Data (Firefox)

The results show that the sensitive database information has been accessed in a way that was not intended.

Alternatively, a command line tool called cURL can be used to exploit the web application.

7.3. Exploit with cURL on the Command-line

cURL is a command line tool that can be used to craft and send specific HTTP requests. It can be run on both Linux and Windows systems. It has already been installed for use on the virtual machine download of the sample vulnerable web application. Exploiting the vulnerable web application using cURL is as simple as entering the XML payload into a file and executing the following command in a Terminal window.

```
curl -H "Content-Type: text/xml" --data-binary @xml http://localhost:8080/converter/convert
```

The Terminal window can be started by clicking on the icon as shown in Figure 10. The curl command sets the HTTP header content type, reads the XML payload from a file called "xml" and sends the request to the vulnerable web application running on localhost port 8080. In this example the file named "xml" contains the exploitation payload and must be in the same directory where the curl command is run.

The response shown in Figure 13 shows the content of the sensitive database configuration file.

```
xmlvuln@ubuntu:~$ curl -H "Content-Type: text/xml" --data-binary @xml http://l
ocalhost:8080/converter/convert
<result>
  <code>400</code>
  <description>Invalid orientation parameter: # Database configuration
export DATABASE_HOST=apiconverter.czs893zdu2v5.us-east-1.rds.amazonaws.com
export DATABASE_USER=html2pdfroot
export DATABASE_PASSWORD=ThisIzTheePwd4Rdb
</description>
</result>xmlvuln@ubuntu:~$
```

Figure 13. XML Server Response Containing Sensitive Configuration File Data (cURL)

With this exploit, an unauthorized user can read arbitrary files on the web application server which is clearly a security concern. Fortunately, removing this vulnerability is very easy, as explained in the next section.

8. Removing the XXE Vulnerability

Many XML parsing libraries, including the one that comes with the Java Development Kit, allow external entities by default. Removing the vulnerability is as simple as configuring the parser to not allow external entities to be defined.

The exact methods for configuring the parsers vary. Methods for several parsers are included on the XML external processing OWASP site (XML External Entity (XXE) Processing, 2013). Instructions are included here for the parser that comes with the Java Development Kit (openjdk version 6) which was used in the sample application.

The most robust method to protect against XXE is to configure the applications XML parser to not allow DOCTYPE declarations. This is done by setting the parsers "disallow-doctype-decl" parameter to true as shown in Figure 14. With this set, an exception occurs if the input contains a DOCTYPE declaration and parsing stops, preventing the vulnerability from exposing sensitive information.

```
parser.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true)
```

Figure 14. Parser Setting to Disallow DOCTYPE Declaration

Alternatively, if DOCTYPE declarations are needed for other purposes, the setting shown in Figure 15 will allow DOCTYPE declarations but not external general entities to be placed into the document content. Remember that the SYSTEM keyword, as shown in Figure 3, defines an entity to be external. If such a reference is made to an external general entity the parser will not substitute the URI content into the XML document. For example, with the external-general-entities parameter set to false in the sample application, the orientation parameter would remain blank even though it contains a reference to the contents of the converter.conf file.

```
parser.setFeature("http://xml.org/sax/features/external-general-entities", false)
```

Figure 15. Parser Setting to Stop Parser from Including External General Entities

Experimentation can be done with both of these settings by modifying the ConvertController.groovy application file to contain one of these settings. The setting should appear directly after the "def parser = new XmlSlurper()" line.

The "disallow-doctype-decl" is the preferred setting because it also protects against XML entity expansion (XEE). XEE is when one entity refers to another entity such that the number of entities is greatly expanded.

```
<!DOCTYPE convert[
<!ENTITY localfile SYSTEM 'file:///etc/converter/converter.conf'>
<!ENTITY localfile2
  "&localfile;&localfile;&localfile;&localfile;&localfile;&localfile;&localfile;&localfile;
  &localfile;&localfile;">
<!ENTITY localfile3
  "&localfile2;&localfile2;&localfile2;&localfile2;&localfile2;&localfile2;&localfile2;&
  localfile2;&localfile2;&localfile2;">
<!ENTITY localfile4
  "&localfile3;&localfile3;&localfile3;&localfile3;&localfile3;&localfile3;&localfile3;&
  localfile3;&localfile3;&localfile3;">
<!ENTITY localfile5
  "&localfile4;&localfile4;&localfile4;&localfile4;&localfile4;&localfile4;&localfile4;&
  localfile4;&localfile4;&localfile4;">
<!ENTITY localfile6
  "&localfile5;&localfile5;&localfile5;&localfile5;&localfile5;&localfile5;&localfile5;&
  localfile5;&localfile5;&localfile5;">
<!ENTITY localfile7
  "&localfile6;&localfile6;&localfile6;&localfile6;&localfile6;&localfile6;&localfile6;&
  localfile6;&localfile6;&localfile6;">
<!ENTITY localfile8
  "&localfile7;&localfile7;&localfile7;&localfile7;&localfile7;&localfile7;&localfile7;&
  localfile7;&localfile7;&localfile7;">
]>
<convert>
<url>www.something.com</url >
<orientation>&localfile8;</orientation >
</convert>
```

Figure 16. Payload for an External Entity Expansion (XEE) Attack

Figure 16 shows an example payload of an XEE attack where one entity refers to another which refers to another. When the parser attempts to expand this simple document it becomes very

large. Submitting this payload to the sample application gives a parser error stating that more than 64,000 entity expansions were encountered. This is the default XML parser limit put in place to protect against XEE abuse. Now if the setting in Figure 14 is applied to not allow DOCTYPE declarations, the expansion does not occur. On the other hand, if the setting shown in Figure 17 is used, a stack overflow error occurs when receiving the XEE payload. For this reason, the first setting shown in Figure 14 is preferred.

9. Network Analysis of an XXE Attack

The seriousness of this vulnerability becomes evident after experimenting with the sample vulnerable web application. The vulnerability is prevalent because many parsers are vulnerable by default. It takes deliberate action by the developer to remove this vulnerability. In larger organizations it is not likely that the person maintaining and operating servers is also the developer of the application. So what can an operational person do to detect and alert on an XXE attempt or to determine how a compromise happened?

An intrusion detection system (IDS) can be used to detect the critical pieces of this attack. For example, the string "<!DOCTYPE" can be entered as an alerting rule on the IDS. Alternatively, an intrusion prevention system can be used to actively block such traffic. Note that for the sample application, the use of typical filter evasion techniques like case change, spacing variations, alternate character encoding and insertion of special characters will not result in a successful attack so searching for this exact string is acceptable.

If the application or the web server logs the XML body of incoming requests then log analysis tools can be used as well. Splunk is a good tool for analyzing, visualizing and alerting on events found in logs. The installation and use of Splunk for such purposes is thoroughly discussed in the paper "Discovering Security Events of Interest Using Splunk" (Roberts, 2013).

Analysis of this attack can also be done after the compromise has occurred if network traffic has been captured. Tcpdump will be used on the sample web application to capture network traffic for analysis. From a terminal window, enter the following command to capture traffic on the loopback interface and write it to a file called cap1.pcap:

.

Enter the password of "xmlvulnpwd" when prompted. Now use one the exploit techniques such as the Chrome advanced rest client to send the XXE payload to the vulnerable web application. After the web application response is received press ctrl+C to quit the tcpdump capture. Tcpdump has created a pcap file called cap1.pcap in the current directory containing the network traffic associated with the attack.

Wireshark can be used to analyze this traffic. If Wireshark is not installed, install it by entering "sudo apt-get install wireshark" on the command line of the terminal window. Read the captured network traffic file by entering the following command on the command line:

wireshark cap1.pcap

Figure 18 shows the Wireshark packet list view of the network traffic. The http POST of the XXE payload is shown highlighted in green. The response containing the sensitive data from converter.conf is shown in orange. Clicking on either of these will display the request and response body in the packet details pane. This makes it clear what XML payload was used, including the DOCTYPE declaration and what data was returned.

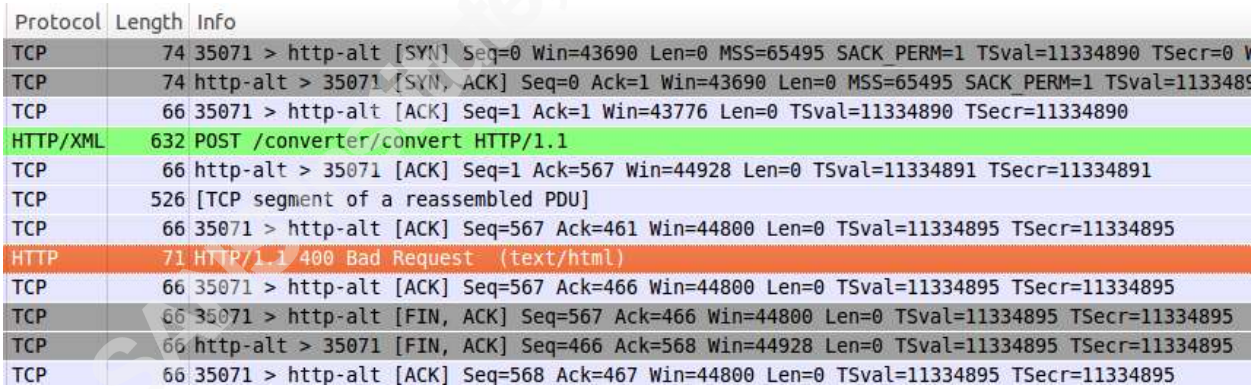

Figure 18. Wireshark Packet List View of XXE Network Traffic

Another useful view of the exchange is found by right clicking on any of these packets and selecting "Follow TCP Stream" from the context menu. Figure 19 shows the "Follow TCP Stream" dialog.

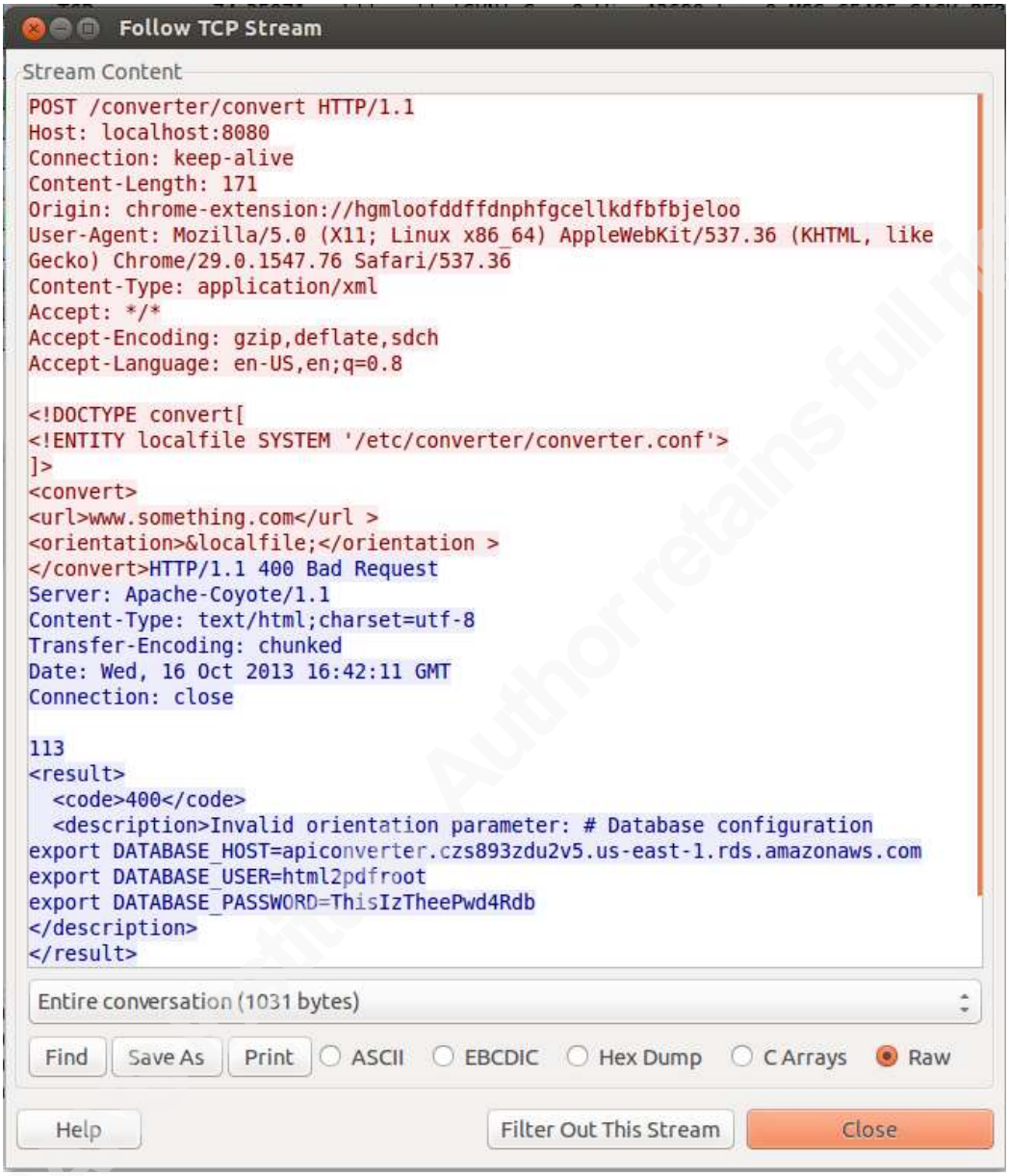

Figure 19. Wireshark "Follow TCP Stream" View of XXE Network Traffic

The incoming request is shown in red and the response is shown in blue. By inspecting the output in this way it is easy to see what payload the attacker used and what sensitive data was retrieved.

10. Conclusion

Default settings for many popular XML parsers leave web applications vulnerable to XXE attacks (XML External Entity (XXE) Processing, 2013). This vulnerability can be used to

expose sensitive data including reading arbitrary files. It can also be used to perform denial of service attacks, rendering the web application unusable. The sample web application provided offers a hand on training experience for understanding the vulnerability, including how to detect and defend against it. Performing network analysis can reveal attempts to exploit the XXE vulnerability. An understanding of this vulnerability from creation to detection and analysis will help developers, administrators and analysts to protect systems against XXE attacks.

11. References

Extensible Markup Language (XML) 1.0. (2008). Retrieved October 10, 2013, from w3.org: http://www.w3.org/TR/REC-xml/#sec-external-ent

HPSR Threat Intelligence Briefing Companion Report to Episode 6. (2013). Retrieved October 1, 2013, from hp.com: http://h30499.www3.hp.com/hpeb/attachments/hpeb/off-by-on-software-security-blog/78/1/Companion%20Report%20to%20HPSR%20Threat%20Intelligence%20Podcast%20Episode%206_v1.6.pdf

XML External Entity (XXE) Processing. (2013). Retrieved October 15, 2013, from owasp.org: https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing

Herzog, S. (2010). *XML External Entity Attacks (XXE).* Retrieved October 13, 2013, from https://www.owasp.org/images/5/5d/XML_Exteral_Entity_Attack.pdf

Roberts, C. (2013). *Discovering Security Events of Interest Using Splunk.* Retrieved October 8, 2013, from sans.org: https://www.sans.org/reading-room/whitepapers/logging/discovering-security-events-interest-splunk-34272

Steuck, G. (2002). *XXE (Xml eXternal Entity) Attack.* Retrieved October 16, 2013, from securiteam.com: http://www.securiteam.com/securitynews/6D0100A5PU.html

Symantec Corporation. (2013). *Internet Security Threat Report 2013.*

Last Updated: July 26th, 2017

Upcoming SANS App Sec Training

Community SANS Minneapolis DEV534	**Minneapolis, MN**	**Aug 25, 2017 - Aug 28, 2017**	**Community SANS**
Community SANS San Francisco DEV541	**San Francisco, CA**	**Aug 28, 2017 - Aug 31, 2017**	**Community SANS**
SANS Network Security 2017	**Las Vegas, NV**	**Sep 10, 2017 - Sep 17, 2017**	**Live Event**
Secure DevOps Summit & Training	**Denver, CO**	**Oct 10, 2017 - Oct 17, 2017**	**Live Event**
SANS Seattle 2017	**Seattle, WA**	**Oct 30, 2017 - Nov 04, 2017**	**Live Event**
SANS Cyber Defense Initiative 2017	**Washington, DC**	**Dec 12, 2017 - Dec 19, 2017**	**Live Event**
SANS Cyber Defense Initiative 2017 - DEV522: Defending Web Applications Security Essentials	**Washington, DC**	**Dec 14, 2017 - Dec 19, 2017**	**vLive**
SANS Security East 2018	**New Orleans, LA**	**Jan 08, 2018 - Jan 13, 2018**	**Live Event**
SANS San Francisco Spring 2018	**San Francisco, CA**	**Mar 12, 2018 - Mar 17, 2018**	**Live Event**
SANS OnDemand	**Online**	**Anytime**	**Self Paced**
SANS SelfStudy	**Books & MP3s Only**	**Anytime**	**Self Paced**