XML External Entity Attacks and Mitigation in XML Parsers

[1]Saurabh Bisht, [2]Satendra Singh, [3]Anju Rabi, [4]Chavda Ashish Nathu

[1,2,3,4]Student, School of Computer Science and Engineering,
Lovely Professional University, Punjab, India.

Abstract : *The increasing dependency of almost every organization in web applications has introduced a whole new set of vulnerabilities and attack vectors. Due to the ever-evolving nature of the information security space, new vulnerabilities, attack methods, and mitigation strategies are being introduced each day. XML External Entity Attack is such an attack. Being recently featured in the updated Open Web-Application Security Project (OWASP) Top 10 2017 list, this attack is prevalent in modern-day web applications and XML parsers. Recent statistics have shown a steep increase in XXE injections. But in spite of the fact, limited literature is available regarding this vulnerability. As XXE usually has a high-security impact on web applications, it is important to implement protective measures against XXE attacks. In this paper, we focus on XXE attack mitigation in various XML parsers. Firstly, we give a brief introduction to various XXE injection attacks and XML parsers. Then we test popular XML parsers for XXE injections. In addition to this, we suggest preventive measures for XML parsers in popular programming languages.*

IndexTerms - XML, XML External Entity Attack (XXE), XML Parsers, Billion Laugh Attack (BIL), Server Side Request Forgery (SSRF), Security Testing

I. INTRODUCTION

Web applications have an important role in everybody's life in today's world. Due to the unexpected circumstances, the pandemic has put forth, the demand for web applications is at an all-time high. Work from home culture has made internet and web technologies an important part of millions of users. But this surge in the usage of Web applications and their high-security implications makes them an attractive target for attackers. As remote working becomes a norm, remote workers will be a prime target for attackers. Acunetix Web Application Vulnerability Report 2021 [1] shows that almost 99% of web applications (from a sample of 3500 randomly selected targets) contain at least one vulnerability. 26% of the websites have high severity vulnerabilities, and 63% of the websites have Medium Severity Vulnerabilities.

The Open Web Application Security Project (OWASP) compiles and releases the top 10 web application security vulnerabilities [2] every three to four years as per the requirement. According to the latest OWASP top 10 list ranking, Injection Vulnerabilities (such as SQL injection, LDAP Injection, XPATH injection, NoSQL Injection, etc) are the most dangerous attacks with high impact.

In 2017, Open Web Application Security Project (OWASP) released an updated list of the top 10 most critical web application vulnerabilities. In this list, a new category of vulnerability known as XML External Entities (XXE) has been added [2]. Due to nature and impact, this vulnerability has been ranked relatively higher at number four. This vulnerability affects web-based infrastructure as well as applications that are XML-based and is known for more than a decade but still exists on poorly configured modern web applications.

Recent studies and statistics have shown a rapid increase in security vulnerabilities in XML-based technologies in the past few years. Different CVEs were assigned to various vulnerabilities including CVE-2019-11253 and CVE-2018-1905. The vulnerabilities that lead to attacks on XML-based web applications are significantly higher, however, limited resources and literature are available to study and mitigate these attacks.

II. GOALS AND CONTRIBUTIONS

The goal of this research paper is to test popular XML parsers for XML external entity attacks. This will help web developers, as well as web-security researchers, secure web applications and systems. This paper is divided into the following four sections as follows:

Section 2 discusses background and related work.
Section 3 discusses about XML based attacks.
Section 4 discusses of the experiment methodology.
Section 5 discusses the results and suggest preventive measures and secure coding practices to mitigate the vulnerability in popular XML parsers and web applications.

2. BACKGROUND AND RELATED WORK

Since the Early stages of XML, several vulnerabilities have been discovered in web applications. These attacks range from Denial-of-Service attacks (DOS) to disclosure of confidential data via XXE attacks. There are a number of web-security researchers continuously looking for vulnerabilities in web applications [3] [4] [5]. Even after the effort, incomplete testing due to lack of time and resources is a major reason why this vulnerability is still a common attack vector.

While the issues are known for a long time, XML-based vulnerabilities still exist in poorly configured applications. Sadeeq et al. [6] described various vulnerabilities that exist in modern XML based applications including XML Bombs (BIL), XXE (XML external entities). Following a systematic and detailed study of different XML parsers in open-source applications, they were able to conclude a large number of applications were vulnerable to XML-based vulnerabilities.

Gupta et al. [7] in their paper survey and classification of XML-based attacks shed light on numerous attacks on XML-based vulnerabilities and attacks. They studied various attacks such as Xpath injection, Ping of Death Attack, Hash Collision, etc., and classified each attack vector based on a number of parameters.

Attacks on web applications including Denial of Service and Injection attacks were described by Tiwari and Singh [8] as well as their mitigation. DOS attacks were studied by various researchers [9] [10]. Along with the manual study, automated vulnerability detection in web applications has been discussed by Milad Hanna et al. [11] and Zhe Sun et al.[12] using automation tools and proposes different testing techniques.

In summary, existing work has discussed a wide variety of vulnerabilities and attacks on XML-based applications, their impact, and their detection. However, limited work has been carried out to suggest the safe implementation of XML parsers used in everyday applications.

3.1 XML Based Attacks

Numerous vulnerabilities arise due to the complex processing and multiple processing states of an XML document. Recent reports published by the National Vulnerability Database show a steep increase in XML-based vulnerabilities since 2016 [13]. XML-based vulnerabilities may lead to Remote Code Execution, Denial of Service, Sensitive data exposure, and various other vulnerabilities. Table 1 shows the effect of different attacks on the Confidentiality, Integrity, and Availability (CIA) triad.

Attack	Confidentiality	Integrity	Availability
Billion Laughs Attack	No	No	Yes
Quadratic Blowup	No	No	Yes
Server Side Request Forgery	Yes	No	No
SQL injection via XXE	Yes	Yes	Yes
XXE	Yes	No	Yes
Remote Code Execution	Yes	Yes	Yes

Table 1: Impact of various vulnerabilities on the CIA Triad

3.2 XML External Entity Injection (XXE)

XML external entities allow data to be dynamically loaded from a resource, either local or remote, at the time of parsing. This functionality can be exploited by attackers to include malicious data from external resources or sensitive data residing on the local system. Successful exploitation of XXE may result in Local File Inclusion, Sensitive data disclosure, Denial of Service, and unauthorized access to system resources.

Furthermore, other attacks such as Billion Laugh Attack (BIL), Quadratic Blowup, Server Side Request Forgery (SSRF), local file inclusion (LFI), port scanning of the internal network, and sometimes even Remote Code Execution (RCE) is possible.[7]

3.3 SSRF due to XXE

Server-Side Request Forgery (SSRF) attack is a server-side vulnerability that gives an attacker the ability to make HTTP requests on behalf of the webserver. In a typical SSRF attack, the attacker issues request to internal network endpoints of the organization which is not accessible from an external network, i.e. the Internet. SSRF attack has a high impact as it compromises both, confidentiality and integrity of the organization's network and may disclose sensitive files, port scan the internal network, etc.

3.4 Billion Laughs Attack (BIL)

BIL (Billion laugh attack) is an XML-based attack that affects the availability of a system and may be used to launch a Denial-of-Service attack [6]. In a BIL attack, an entity is referenced recursively which results in the consumption of a large amount of system memory which hampers the overall performance of the system. A successful BIL attack may result in the unavailability of the service. Figure 1 shows BIL attack payload.

```
<?XML version="1.0"?>
<!DOCTYPE lolz [
<!ELEMENT lolz (#PCDATA)>
<!ENTITY lol "lol">
<!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
<!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">]>

<lolz>&lol9;</lolz>
```

Fig 1. BIL attack payloads

3.5 Quadratic Blowup

Similar to the BIL attack, the Quadratic Blowup attack also exploits entity expansion. Instead of implementing a number of nested entities, this attack uses one large entity containing thousands of characters. When an application filters recursive entity

expansion to mitigate BIL attack, quadratic blowup can still be exploited by attackers to launch DoS attacks. Figure 2 shows a a Quadratic Blowup payload. The entity "blowup" contains the character "a" couple of hundred times. It is then referenced repetitively resulting in a large amount of memory consumption.

```
<?XML version="1.0"?>
<!ENTITY blowup "aaaaaaaaaaaaaaaa...">]>

<data>&blowup;&blowup;&blowup;&blowup;...</data>
```

Fig 2. Quadratic Blowup Attack Payload

XML-based vulnerabilities offer a large attack vector for various attacks to be executed. For example, recent vulnerability (CVE-2019-5893) allowed SQL Injection via the data.xml query parameter. This vulnerability was patched by implementing a white-list approach that disallowed special characters in the user input search field [7]. Similarly, CVE-2019-11253 allowed the attacker to initiate a BIL attack which resulted in a DoS attack. Mitigation strategies have been discussed with an emphasis on validating user input, implementing whitelists and parameterized queries [7]. While implementing whitelists and blacklists is an easy solution to implement, they can be bypassed by highly trained malicious actors. Furthermore, blacklists may adversely affect system performance as they restrict usage of certain keywords that may be required by the application internally for proper functioning.
Hence, the very first step for mitigating XML-based attacks is to properly configure XML parsers and disabling XML external entities and DTD processing. This approach is effective to avoid a large number of XML-based vulnerabilities.

I. EXPERIMENTAL STUDY

To conduct the study, we evaluated popular XML parsers and libraries based on the results acquired from Github. Github is the most popular code hosting platform available. According to Github, Python and JAVA are ranked at number 2 and 3 respectively. This made parsers in these languages a prime choice. Also, we tried to select XML parsers that are included in modern programming languages. In total, we selected a total of 21 parsers to test including but not limited to Python minidom, PHP SimpleXML, JAVA DOM parser, etc.
To study different attack scenarios, each parser was supplied with specially crafted malicious payloads to mimic an attack. It was made sure to keep the default parser properties intact as this is a common coding tendency among developers to keep things at default settings. The result was then manually analyzed. It was observed that vulnerable parsers expand external entities in the XML file and return the contents of the files specified in the XML document. Figure 3 illustrates successful XXE injection and the contents of "/etc/passwd" file are displayed.
Similarly, BIL and Quadratic Blowup attacks were tested one by one. Malicious XML payloads were supplied to the vulnerable parser and the behavior was observed. This was an iterative process as a number of recursions were needed to properly test the BIL attack. To test the SSRF vulnerability, a text file was created and kept on the internal network which was not accessible by the user. The text file was then requested using an XXE payload which resulted in a successful SSRF attack. After all the parsers and results were analysed, proper and safe implementation of parsers was done as suggested by the Open Web Application Security Project (OWASP) to study the effectiveness of the mitigation strategies.

Fig 3. Successful XXE Injection to read contents of /etc/passwd file.

II. RESULTS AND DISCUSSION

4.1 Test Results of Different XML Parsers

Since different programming languages implement different XML parsers, there isn't a universal solution for every programming language present. Therefore, different techniques are to be considered while implementing a language-specific XML parser. Table 2 shows the result of the tested XML parsers.

Parser	Billion Laughs Attack (Bill)	Quadratic Blow up	XXE Injection	Server Side Request Forgery(SSRF) via XXE
Python: xml.sax	Vulnerable	Vulnerable	Vulnerable	Vulnerable
Python: etree	Vulnerable	Vulnerable	Not Vulnerable	Not Vulnerable
Python: minidom	Vulnerable	Vulnerable	Not Vulnerable	Not Vulnerable
Python: pulldom	Vulnerable	Vulnerable	Vulnerable	Vulnerable
Python: lxml	Not Vulnerable	Vulnerable	Vulnerable	Not Vulnerable
Python: defusedxml.lxml	Not Vulnerable	Not Vulnerable	Not Vulnerable	Not Vulnerable
Python: defusedxml.etree	Not Vulnerable	Not Vulnerable	Not Vulnerable	Not Vulnerable
Python: defusedxml.sax	Not Vulnerable	Not Vulnerable	Not Vulnerable	Not Vulnerable
Python: defusedxml.pulldom	Not Vulnerable	Not Vulnerable	Not Vulnerable	Not Vulnerable
Python: defusedxml.minidom	Not Vulnerable	Not Vulnerable	Not Vulnerable	Not Vulnerable
PHP: SimpleXML	Not Vulnerable	Vulnerable	Not Vulnerable	Vulnerable
PHP: DOMDocument	Not Vulnerable	Vulnerable	Not Vulnerable	Not Vulnerable
PHP: XMLReader	Not Vulnerable	Not Vulnerable	Not Vulnerable	Not Vulnerable
.NET: Xmlreader	Not Vulnerable	Not Vulnerable	Not Vulnerable	Not Vulnerable
.NET: LINQ to XML	Not Vulnerable	Not Vulnerable	Not Vulnerable	Not Vulnerable
.NET: XmlDocument	Not Vulnerable	Not Vulnerable	Vulnerable	Vulnerable
.NET: XMLTextReader	Not Vulnerable	Not Vulnerable	Vulnerable	Not Vulnerable
JAVA: Oracle Dom Parser	Vulnerable	Vulnerable	Vulnerable	Vulnerable
JAVA: DocumentBuilderLibrary	Vulnerable	Vulnerable	Vulnerable	Vulnerable
JAVA: TransformerFactory	Vulnerable	Vulnerable	Vulnerable	Vulnerable
JAVA: Validator	Vulnerable	Vulnerable	Vulnerable	Vulnerable

Table 2. Test result of different parsers.

4.2.1 Python

As observed in Python's lxml module, while parsing XML documents, lxml module resolves external entities by default. This makes an application vulnerable to XXE, Quadratic Blowup, and SSRF attacks. But since recursive entities are not expanded by default, this parser is safe against BIL attack. Disabling entity resolution mitigates all attack vectors in the lxml module. To prevent SSRF, while processing an XML document, access to the network and file system should be disabled.

Sax parser processes XML Entities by default which renders them vulnerable to XXE attacks. The proposed countermeasure is to remove Entity processing. Generally, using of sax parser in Python is not recommended for security reasons.

Furthermore, Python provides two modules defusedxml and defusedexpat to mitigate XML-based vulnerabilities. An instance of defusedxml should be implemented if applicable. These packages sanitize user input before the parser processes the document. To use the package, import the defusedxml / defusedexpat package and use the required methods from the package instead of the original module. If another parser is required, disabling external entities, DTD processing, and network access is recommended to mitigate XXE vulnerabilities. Other countermeasures are not available for the listed Python XML parsers.

4.2.2 PHP

Based on the tests, XMLReader was not found to be vulnerable to any XML attack vector whereas SimpleXML and DOMDocument are both vulnerable to DoS attacks. As suggested in the PHP documentation, XXE in DOMDocument can be mitigated by using the "libxml_set_external_entity_loader()" function. This function changes the default external entity loader and stops the expansion of arbitrary external entities. But this countermeasure has adverse effects too as XMLReader and

DOMDocument both support Xinclude and using libxml_set_external_entity_loader() to disable external entities is not suitable for XMLReader and DOMDocument. This prevents the input to be parsed and has an adverse effect on the functioning of the parser.
Furthermore, enabling DTD processing renders all PHP parsers are vulnerable to XXE attacks.

4.2.3 .NET

Various XML libraries are provided by the .NET framework, XmlReader being the most popular, is not vulnerable to XXE since the DTD processing is turned off by default. Similarly, LINQ to XML is not vulnerable to any XXE attack vector. XmlDocument prior to 4.5 was found to be unsafe and vulnerable to SSRF due to XXE. The proposed solution is to turn off external DTD processing if using XmlDocument is necessary. XMLTextReader is vulnerable to XXE as parsing XML entities is turned on by default. Turn off entity processing to secure the application.

4.2.4 JAVA

All tested JAVA parsers are vulnerable to XML-based attacks as External Entity processing is enabled by default. Oracle DOM Parser expands external entities by default which renders it vulnerable to recursive entity resolution attacks and leads to DoS. The proposed solution is to disable entity expansion explicitly. If recursion is required then defining recursion depth is advised.
DocumentBuilder Library, TransformerFactory, and Validator are vulnerable to XXE as they allow DTD processing by default. To prevent XXE, both internal, as well as external DTD processing, should be turned off. If disabling of DTDs is not possible, disallow external entities along with parameter entities.

III. Conclusion and Future Work

XML-based vulnerabilities are still prevalent in popular XML parsers. The inclusion of XXE in OWASP Top 10 clearly signifies the impact of the vulnerability. Multiple parsers implemented in different programming languages are prone to attacks. In this paper, we studied different XML-based attacks, their impact, and their prevention in web applications. In general, developers should disable network access to sensitive data and systems. Also, implementing XML parsers as per the guidelines in the OWASP XXE mitigation cheat-sheet is advised. In addition to this, processing of external entities should be done only when necessary, else discussed methods for language-specific parser should be implemented. Being a pervasive technology, further research is needed to correctly estimate the risk and to provide mitigation strategies for other parsers that were not tested in this paper.

References

[1] Acunetix Web Report 2021 https://www.acunetix.com/white-papers/acunetix-web-application-vulnerability-report-2021/

[2] OWASP Top 10 Web Application Vulnerabilities 2017 https://owasp.org/www-project-top-ten/

[3] E. Bertino, L. Martino, F. Paci, and A. Squicciarini. Security for Web Services and Service Oriented Architectures. Springer, 2010.

[4] M. Jensen, N. Gruschka, and R. Herkenhner. A survey of attacks on web services. Computer Science - Research and Development, 24(4):185–197, 2009

[5] A. N. Gupta and D. P. S. Thilagam. Attacks on web services need to secure xml on web. Computer Science and Engineering, An International Journal (CSEIJ), 3(5), 2013.

[6] Sadeeq Jan, Cu D. Nguyen, Lionel Briand] [Jan, Sadeeq, Cu D. Nguyen, and Lionel Briand. "Known XML vulnerabilities are still a threat to popular parsers and open source systems." 2015 IEEE International Conference on Software Quality, Reliability and Security. IEEE, 2015

[7] Gupta, C., Singh, R. K., & Mohapatra, A. K. (2020). A survey and classification of XML based attacks on web applications. Information Security Journal: A Global Perspective, 29(4), 183-198.

[8] Tiwari, Sunita, and Pratibha Singh. "Survey of potential attacks on web services and web service compositions." 2011 3rd International Conference on Electronics Computer Technology. Vol. 2. IEEE, 2011.

[9] X. Ye. Countering ddos and xdos attacks against web services. In Embedded and Ubiquitous Computing, 2008. EUC '08. IEEE/IFIP International Conference on, volume 1, pages 346–352, Dec 2008.

[10] S. Padmanabhuni, V. Singh, K. Senthil Kumar, and A. Chatterjee. Preventing service oriented denial of service (presodos): A proposed approach. In Web Services, 2006. ICWS '06. International Conference on, pages 577–584, Sept 2006

[11] Hanna, M., Aboutabl, A. E., & Mostafa, M. S. M. (2018). Automated software testing framework for web applications. International Journal of Applied Engineering Research, 13(11), 9758-9767.

[12] Sun, Z., Zhang, Y., & Yan, Y. (2019, December). A Web testing platform based on hybrid automated testing framework. In 2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC) (Vol. 1, pp. 689-692). IEEE.

[13] National Vulnerability Database, https://nvd.nist.gov/