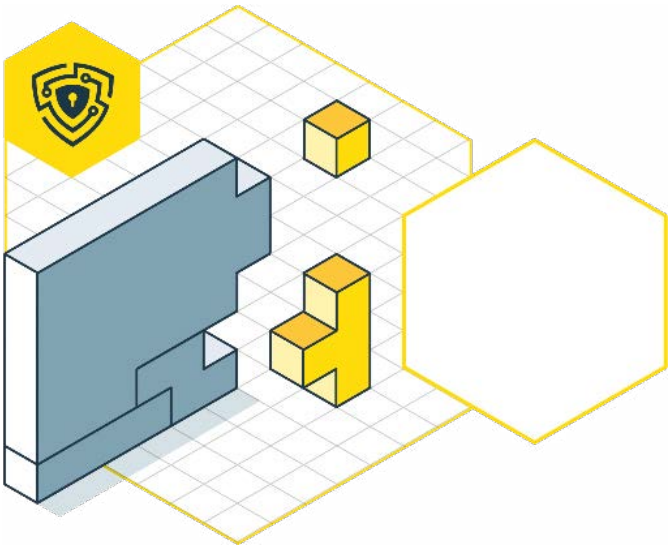
CRASHTEST **SECURITY**

GUIDE FOR PREVENTING XXE ATTACKS

WHAT ARE THE STEPS TO KEEP YOUR WEB APP OR API SAFE FROM SUCH VULNERABILITY

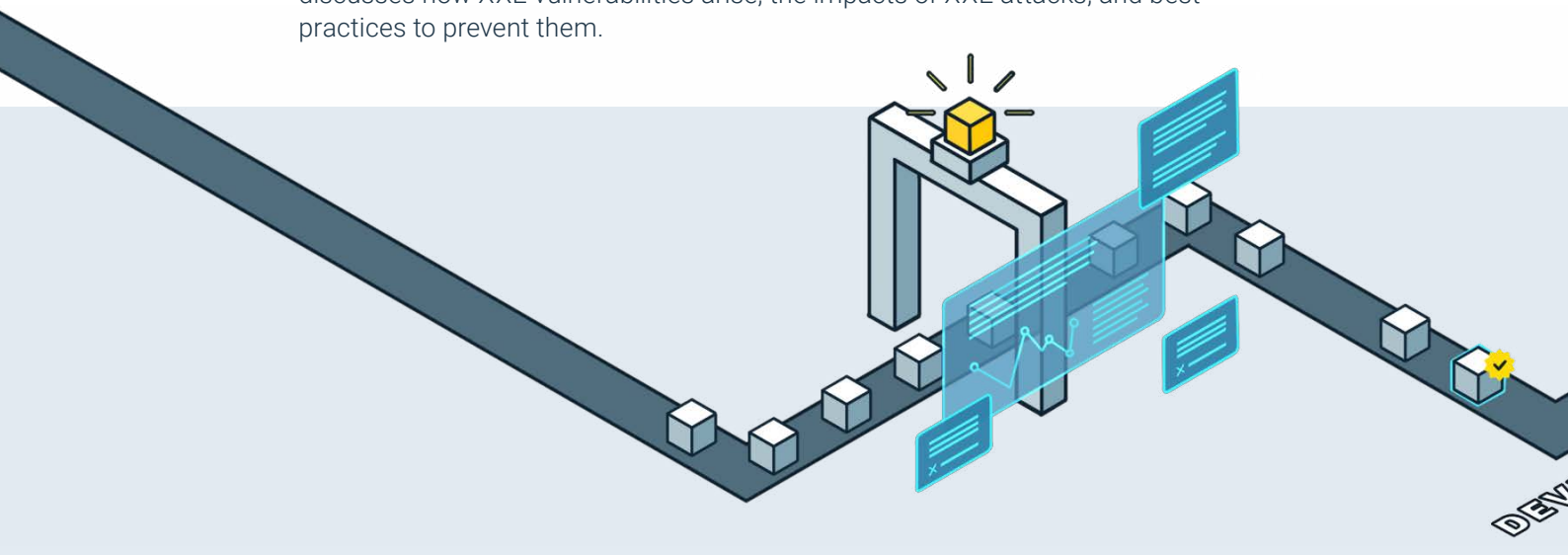
GUIDE FOR THE XXE ATTACK PREVENTION

Table of Contents



INTRODUCTION TO THIS GUIDE

The Extensible Markup Language (XML) format is widely used to share structured data between programs, people and machines, both locally and in area networks. XML is a simple, text-based format that represents information such as documents, configurations, books, transactions, application data, and invoices, among much more. While the format offers advanced benefits for modern application delivery, XML is also vulnerable to severe forms of hacking attacks. XML External Entity (XXE) injection is one such attack that occurs when an adversary interferes with an application's XML parsing mechanism. An attack is usually exploited over the XXE vulnerability that allows the threat actor to forge requests, access sensitive data, and perform port scanning, among other malicious actions. This guide discusses how XXE vulnerabilities arise, the impacts of XXE attacks, and best practices to prevent them.



WHAT ARE XXE VULNERABILITIES?

XML external entity vulnerabilities enable hackers to alter how applications process XML documents. XML external entities are mostly custom XML entities whose values are not loaded from the Document Type Definition (DTD) in which they are declared. In instances where the XML parsers are weakly configured, attackers can inject malicious XML input and use external references to a malicious external entity. Such attacks allow hackers to view files on the application server, and connect with the backend, leading to a broad range of application security issues such as **denial of service, sensitive data disclosure,** and **server-side request forgery**.

TYPES OF XXE ATTACKS

XXE attacks are categorized according to the process/action targeted by the malicious user. Some common forms of XML entity attacks include:

EXPLOITING XXE TO RETRIEVE FILES

In this attack, malicious actors define an XML entity containing the contents of external files, which the application server returns in its response. To retrieve a file from the application server file system, the malicious user usually modifies the XML document in two ways:

- Introducing an external DOCTYPE declaration that defines the XML entity containing the paths to external sources
- Editing a data value in the XML document that the application uses in its response to define external entities.

Let us take an attack scenario to comprehend this better.

Suppose an application processing e-commerce transactions checks the remaining stock levels using the following XML input:

<?xml version="1.0" encoding="UTF-8"?>

<stockamount><ProductID>215</productID></stockamount>

Let us assume that the application processing this statement does not employs any XXE prevention mechanism. This allows the attacker to exploit XML vulnerabilities for retrieving passwords by sending a malicious XML file similar to:

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE darwin [<!ENTITY xxe SYSTEM „file:///etc/passwd">]>

<stockamount><ProductID>&xxe;</ProductID></stockamount>

The above malicious payload contains an external XML entity xxe, whose reference file is ***/etc/passwd*** and uses the XML entities within ***ProductID***. The application's unsafe XML parser subsequently includes the contents of the file, as shown:

Invalid Product ID: root:x:0:0:root:/root:/bin/bash

daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin

bin:x:2:2:bin:/bin:/usr/sbin/nologin

...

EXPLOITING XXE TO ORCHESTRATE SERVER-SIDE-REQUEST-FORGERY (SSRF)

A Server-Side Request Forgery (SSRF) attack is considered one of the significant security risks in which malicious actors force the backend server into sending requests to any URL it can reach through network access. To exploit XXE vulnerabilities for SSRF attacks, attackers define external XML entities using the target URLs. To achieve this, adversaries use these defined entities within an XML data value.

A reference attack can be orchestrated by using the following XXE payload to send a backend request to an internal entity for the web structure of the ***darwin*** web application from an external entity:

<!DOCTYPE foo [<!ENTITY xxe SYSTEM „http:/darwin.com/">]>

In some SSRF attacks, malicious actors can view the external entity within the value in the application's response. On the other hand, in blind SSRF attacks, there is no two-way interaction between the vulnerable server and the hacker.

EXPLOITING BLIND XXE EXFILTRATE DATA OUT-OF-BAND

In blind XXE attacks, the application server does not return the external entities defined in HTTP responses. While this prevents access to files on the server-side, attackers mostly utilize out-of-band methods to detect XML vulnerabilities. Attackers also use these vulnerabilities to exfiltrate errors from the XML parsing library and obtain sensitive information contained in error messages.

An example of exploiting blind XXE would be a payload design to access a local resource that may not return. The malicious user achieves this using a payload of the form:

```
<?xml  version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE darwin [

  <!ELEMENT darwin ANY >

  <!ENTITY xxe SYSTEM  „file:///dev/random" >]>

<darwin>&xxe;</darwin>
```

SEVERITY LEVEL OF XXE VULNERABILITIES

Possible impacts of an XXE attack include:

- In instances where the default XML parser is insecure, attackers can control the browser via remote code execution and run arbitrary code on the application server through SSRF
- Web service shut down due to XML denial of service
- Upload of arbitrary files to the XML document's DTD processing modules
- Exposure to sensitive information
- Unauthorized access to the program's source code

The XXE vulnerability was ranked number 4 on the OWASP Top 10 2017 list with an exploitability rating of 2, a prevalence score of 2, and a detectability rating of 3. In the OWASP Top 10 2021 list, XXE vulnerabilities are now reclassified to fall under **Security Misconfigurations (A05)** category, with an **average incident rate of 4.51%** and an **average weighted impact of 6.56**.

IDENTIFYING XXE VULNERABILITIES WITH CRASHTEST SECURITY

XXE vulnerabilities commonly arise when the application's library supports arbitrary XML entities that are non-essential for the application. Crashtest Security Suite offers several vulnerability scanners that collectively help identify XML vulnerabilities before malicious actors can orchestrate external entity injection. The platform's XXE vulnerability scanner enforces security checks on each type of XML entity and generates detailed security information about the XML processor.

Crashtest Security also includes an API vulnerability scanner that examines every request submitted to the application server for malicious requests. The platform's automated penetration testing tool additionally helps reduce manual overhead on security testing while decreasing risk exposure by mitigating significant security risks.

Try Crashtest Security for a free, 14-day trial to find out how you can mitigate exploits orchestrated over external XML entities.

XML EXTERNAL ENTITY VULNERABILITIES – PREVENTION TECHNIQUES

XXE attack prevention techniques include:

Continuous vulnerability scanning

Continuous, automated vulnerability scanning helps security teams detect XXE vulnerabilities as soon as they arise, preventing potential exploits. Application developers should enforce unit tests, and constant source code scans to protect applications from unsupported custom entities.

Using an updated XML parser

Modern XML parsers based on the libxml2 library typically have built-in XXE prevention by default. When relying on third-party XML parsers, security experts should run adequate tests to administer a secure parser and maintain the default behavior of the XML parsing library.

BEST PRACTICES FOR PREVENTING XXE ATTACKS

While relying on prevention techniques is mostly a one-time activity, developers should also adopt best practices as a continuous development workflow process to prevent XXE attacks and vulnerabilities. These include:

Disabling default DTDs

When Document Type Definition (DTD) processing is enabled, the XML parser can accept arbitrary code in the incoming XML document. Disabling DTDs by default removes external entity support, discouraging attackers from injecting malicious XML payloads.

Identify hidden XXE injection attack vectors through application security testing

It is recommended to leverage the right combination of SAST, DAST, and IAST tools to ensure the application uses strong XML parsers. These tools automate and expedite the detection and identification of security vulnerabilities that can be used to facilitate an XXE attack.

In addition to automated tools, security experts should also classify attack vectors hidden in requests that do not contain XML. Some of these can be categorized as:

- **XInclude attack vectors**: This is where the hacker references an XInclude namespace and a path to subdocuments for building a malicious XML file.
- **XXE via file upload**: In this attack scenario, the attacker submits the XML file in an SVG image to applications that inherently support the SVG format.
- **XXE via modified content type**: Applications that accept content types generated by HTML forms also tolerate other content types. If the application tolerates and parses input content as XML, reformatting requests to use XML potentially conceives an XXE attack surface.

As some of such attack surfaces are undetectable through automated scanning tools, application developers should adopt practices to manually scan for these attack surfaces by referring to the OWASP cheat sheet.

Perform whitelisting for server-side input validation

It is crucial to sanitize the type of input submitted thoroughly for applications that accept data from users and use it to form a section of output code. Hackers typically use input validation flaws to submit arbitrary code or malformed XML files for XXE and server-side request forgery attacks. It is also recommended to ensure server-side input validation is based on a whitelist that instructs the server only to accept limited types of input data.

Use simple data formats

Use simpler data formats, such as JSON, which is purpose-built for rapid data interchange. JSON also enables stronger encoding and is less complex, requiring less memory and processing overhead. Since JSON parsing does not rely on DTD and is not prone to expanding XML entities, the file format is considered secure for

ELIMINATE XXE VULNERABILITIES WITH CRASHTEST SECURITY

XXE attacks are attributed as severe, ranging from **reduced application availability, disclosure of sensitive information, remote code execution,** and **arbitrary file uploads**, among others. **Crashtest Security** helps security experts and application developers to proactively identify attack vectors before malicious attackers get a chance to exploit them. Out of the suite of various vulnerability scanners, Crashtest Security also offers XXE scanner as one of its core offerings that helps reduce the attack surface commonly exploited by external XML entities.

To know more about how Crashtest Security can help your organization administer robust security while preventing XXE vulnerabilities, get a free 14-day trial here.

Start 2-Week Trial for Free

CRASHTEST **SECURITY**

WWW.CRASHTEST-SECURITY.COM