Chapter 30. Defending Against XXE

Generally speaking, XXE is indeed easy to defend against—simply disable external entities in your XML parser (see Figure 30-1). How this is done depends on the XML parser in question, but it is typically just a single line of configuration:

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

XXE is noted by OWASP to be particularly dangerous against Java-based XML parsers because many have XXE enabled by default. Depending on the language and parser you are relying on, it is possible that XXE is disabled by default. You should always check your XML parser's API documentation to make sure. Don't just expect it is disabled by default.

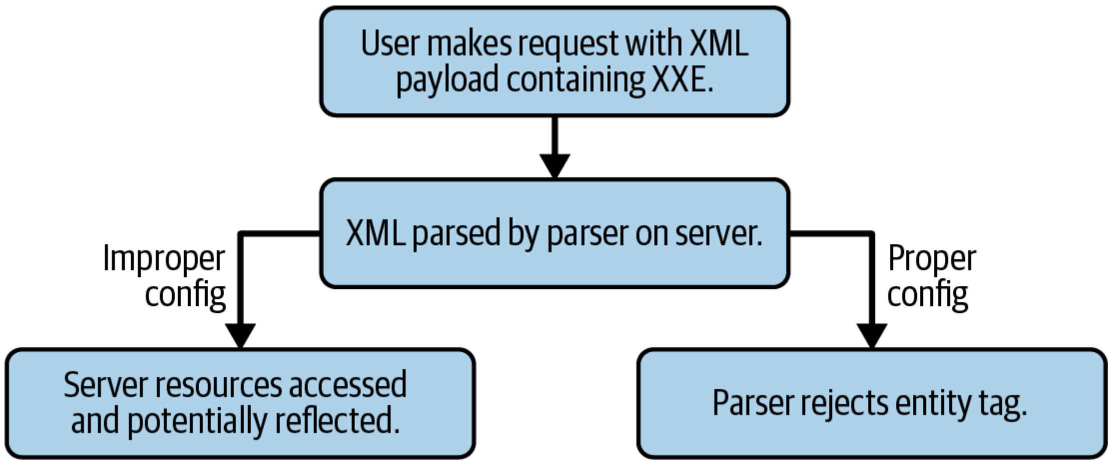
User makes request with XML payload containing XXE.
XML parsed by parser on server.
Improper config
Proper config
Server resources accessed and potentially reflected.
Parser rejects entity tag.

Figure 30-1. XXE attacks can be easily blocked by properly configuring your XML parser

Evaluating Other Data Formats

Depending on your application's use cases, it may be possible to re-architect the application to rely on a different data format rather than XML. This type of change could simplify the codebase while eliminating any

XXE risk. Typically, XML can be interchanged with JSON, making JSON the default when looking at other formats.

JSON, on the other hand, would not be practical if your application is parsing actual XML, SVG, or other XML-derived file types. It would, however, be a practical solution if your application is sending standard hierarchical payloads that just happen to be in XML shape.

Generally speaking, JSON and XML can be compared side-by-side as if they were direct competitors, as Table 30-1 shows.

Table 30-1. XML versus JSON

Category	XML	JSON
Payload size	Large	Compact
Specification complexity	High	Low
Ease of use	Requires complex parsing	Simple parsing for JavaScript compatibility
Metadata support	Yes	No
Rendering (via HTML-like structuring)	Easy	Difficult
Mixed content	Supported	Unsupported
Schema validation	Supported	Unsupported
Object mapping	None	JavaScript
Readability	Low	High
Comment support	Yes	No
Security	Lower	Higher

The comparison of the two formats could go on for an extensive amount of time, but you should grasp a few things right off the bat from Table 30-1:

- JSON is a much more lightweight format than XML.

- JSON offers less rigidity, but brings with it faster and easier-to-work-with payloads.
- JSON maps to JavaScript objects, while XML more closely maps to DOM trees (as the DOM is an XML-derived format).

From this we can conclude that JSON should be an acceptable alternative for any API that is dealing with lightweight structured data to be interpreted by JavaScript, while XML is probably still ideal in any case where the payload will eventually be rendered.

Because XML has schema validation, it may also be useful for applications where deeply rigid data structure is required. JSON, on the other hand, is less rigid, making it perfect for APIs with ongoing development such that the contract between the client and server does not need constant maintenance.

The security risks from XML mostly come from the power of its specification and the fact that it can incorporate external files and multimedia. As such, it is naturally less secure than JSON, a format that simply stores key/value pairs in a string-based format.

If your organization does not like the idea of moving to JSON, YAML, BSON, or EDN are all suitable alternatives but should require a similar analysis prior to commitment.

Advanced XXE Risks

Note that XXE attacks often start as read-only attacks but may progress into more advanced forms of attack. XXE is a “gateway” attack of sorts because it provides the attacker with a recon platform that permits them to access data otherwise inaccessible to the world outside of the web server.

Using this data, other parts of the application may be more easily compromised. The result is that the final impact of an XXE attack can be anywhere from read-only data access to remote code execution and full server takeovers. This is why XXE attacks are so incredibly dangerous.

Summary

I believe XXE deserved attention in this book because of how common improperly configured XML parsers are in production web applications, in addition to how much risk an external entity attack presents to an organization. XXE attacks are often easy to mitigate, yet they are still widespread. As a result, it is imperative to double-check each XML parser configuration prior to publishing any application that makes use of XML or XML-like data types.

XXE attacks are serious and can cause significant damage to an organization, application, or brand. All precautions should be taken when working with a server-side XML parser to prevent an accidental XXE vulnerability from slipping into your codebase.