It's time to take XML out back and shoot it

XML Anxiety

XML security problems are numerous, but you can take steps to limit your exposure – or you can use a different standard. *By Kurt Seifried*

For this month's column, I intended to write about XML security and how to avoid all the attacks and problems that can occur. I started making a list of issues both well known and not so well known. After listing 20 items, I realized I wouldn't have enough space to cover everything [1], so I moved on to plan B: Instead of focusing on the problems, I'd look at the solutions. This worked reasonably well until I realized one small problem: Even if you use software like Python's new defusedxml [2] and defusedexpat [3] a number of problems are still difficult to deal with.

A Brief History of XML

XML came from the W3C (World Wide Web Consortium), who also brought us SGML (from which XML comes), SOAP, HTML, you name it. To say that XML and its related family of standards is complicated is a gross understatement – with XML, XML Schema, RELAX NG, XPath, XSLT, XML Signatures, and XML Encryption to name a few. XML also has been extended into XHTML, RSS, Atom, and KML, to name a few more standards. About the only good news I have is that XML and most of its family of standards are NOT Turing complete [4] (unlike, say, PostScript), but you can embed some pretty funky logic into XML files that can cause problems in the various XML parsers.

One note: Almost no one directly uses XML; it's most often used as an interchange format to move data from an application on system A to another application on system B. As such, many virtualization systems, for example, use XML for manifests/control files. In general, XML is a lot like plumbing – everyone uses it but nobody actually thinks about it until it breaks and you have to call in a plumber to fix it.

Attacks against XML fall into two broad categories: those against the XML parsing/handling layer and those against the back-end software using the data.

XML Parsing

As mentioned before, XML is a very complex format. It allows schemas to be defined for XML documents; for example, an XML document representing an order might be defined such that it must contain one customer ID, one or more order codes, and an option discount code. This allows very high level logic to be specified to ensure documents are well formed and correct.

XML also allows for cryptographic signing of the data within the XML file and encryption of the data. This encryption lets data be transferred securely from end to end in systems using XML regardless of the systems and networks in between. Signing can also be used to ensure that fake XML documents are not injected (e.g., if you run an online store that takes orders from other businesses via XML, you could use this feature to authenticate orders).

XML parsing is just as complicated. You can parse an XML document all at once, but you can also parse it as a stream of data. This approach allows systems to open a connection and start streaming in XML (e.g., orders in real time) and also enables parsing of very large documents (e.g., several gigabytes in size) without completely hammering the system. A short list of attacks against XML parsers includes:

- Internal entity expansion
- External entity expansion

KURT SEIFRIED

Kurt Seifried is an Information Security Consultant specializing in Linux and networks since 1996. He often wonders how it is that technology works on a large scale but often fails on a small scale.

- Non-deterministic elements
- XML choice elements
- Entity references in content models
- Expansion of content models
- General regular expression issues
- Other kinds of external references

XML documents can contain entities that are then expanded. For example, in a sales report, a description of a product can be included that is then referenced in the various orders that include the product. External entities can also be included so that you can include other XML documents, for example. The popular DocBook software does this, allowing you to create one XML file per chapter of a book and then have a single master XML file that includes references to all chapters.

Both of these features can be abused: Create a large text string labeled "a"; create a second string comprising 10 copies of "a" and call that "b"; repeat as needed. When the file is processed, the original string of text will be expanded 10 times in memory at each step. If you do this a few times, a 100-byte XML file could easily consume gigabytes of memory when it is parsed. To deal with these situations, XML parsers have started either disabling entity expansion entirely or placing limits on how fast memory can be consumed (in general, entity expansion should only result in linear growth, not quadratic or exponential growth, in memory use).

External entity expansion can be used to connect to external systems via HTTP and several other protocols; thus, by feeding an XML document to a server, you can cause that server to connect to other web servers – a lot (like, thousands of times). Because you can also include a variety of resources like schemas and document type definitions within XML files, even disabling entity expansion won't prevent all avenues of exploitation.

XML Data Handling

Generally, all the rules about trusting user-supplied input apply to handling XML data. Additionally, you need to worry about the parsing layer; for example, XML data injection [5] (e.g., inserting a " > " character and then more data) can allow an attacker to modify or insert additional records into the XML document.

This approach would allow an attacker to set the price of an order to $0 or change the number of items being included in the order once payment has been made. By including an external entity, an attacker could cause a back-end system to include the data. In short, you need to make sure your parsing layer doesn't do anything stupid or modify the data in unexpected ways, which is basically impossible because XML parsing is so complicated.

TABLE 1: Basic JSON Standard and Data Types

object	array	int
{ }	[ ]	*digit*
{ *members* }	[*elements*]	*digit*1-9 *digits*
members	elements	*-digit*
pair	*value*	*-digit*1-9 *digits*
pair, members	*value, elements*	frac
pair	char	*. digits*
string : *value*	<Any Unicode character except " or \ or control character>	
string	\"	exp
" "	\\	*e digits*
" *chars* "	\/	digits
value	\b	*digit*
string	\f	*digit digits*
number	\n	e
object	\r	e
array	\t	e+
true	\u<four hex digits>	e-
false	number	E
null	*int*	E+
chars	*int frac*	E-
char	*int exp*	
char chars	*int frac exp*	

Safely Using XML

Probably the most important and effective step to secure XML is to use a well-maintained XML parser like libxml2 or expat. A lot of XML parsing libraries are available, and most of them are terrible. The next step is to disable any XML features you don't need, such as entity expansion. Once you have done this, you can create schemas for the XML data you need to process. Properly specified schemas can prevent an attacker from successfully conducting XML injection attacks or modifying the data in ways that can cause problems.

If you need to sign or encrypt your XML documents, I recommend thinking really hard about alternative solutions. Several major XML libraries do not support signatures, and it's all moot because the XML encryption standard has significant technical problems that basically render it largely broken [6].

XML Alternative – JSON

If you can't really secure XML, what should you do? If you need to accept user-supplied data, I strongly recommend JSON [7]. The basic definition for JSON (Table 1) literally fits on the back of a business card. Although JSON is certainly more limited in the data types it can represent, in general, it offers enough to satisfy most people. JSON validation can also be accomplished using any number of third-party libraries, most of which are relatively simple (and in security, simple is better than complicated). To find one for your language, simply Google your favorite programming language with terms like *json* and *validation* or *schema* (you won't be lacking choice). ■■■

INFO

[1] Your XML parser will destroy everything you have ever loved: *http://portal.sliderocket.com/CJAKM/xml-attacks*

[2] Python defused XML: *https://bitbucket.org/tiran/defusedxml*

[3] Python defused expat: *https://bitbucket.org/tiran/defusedexpat*

[4] Turing completeness: *http://en.wikipedia.org/wiki/Turing_completeness*

[5] Testing for XML injection (OWASP-DV-008): *https://www.owasp.org/index.php/Testing_for_XML_Injection_(OWASP-DV-008)*

[6] XML Encryption is Insecure: *http://aktuell.ruhr-uni-bochum.de/pm2011/pm00330.html.en*

[7] JSON: *http://www.json.org/*