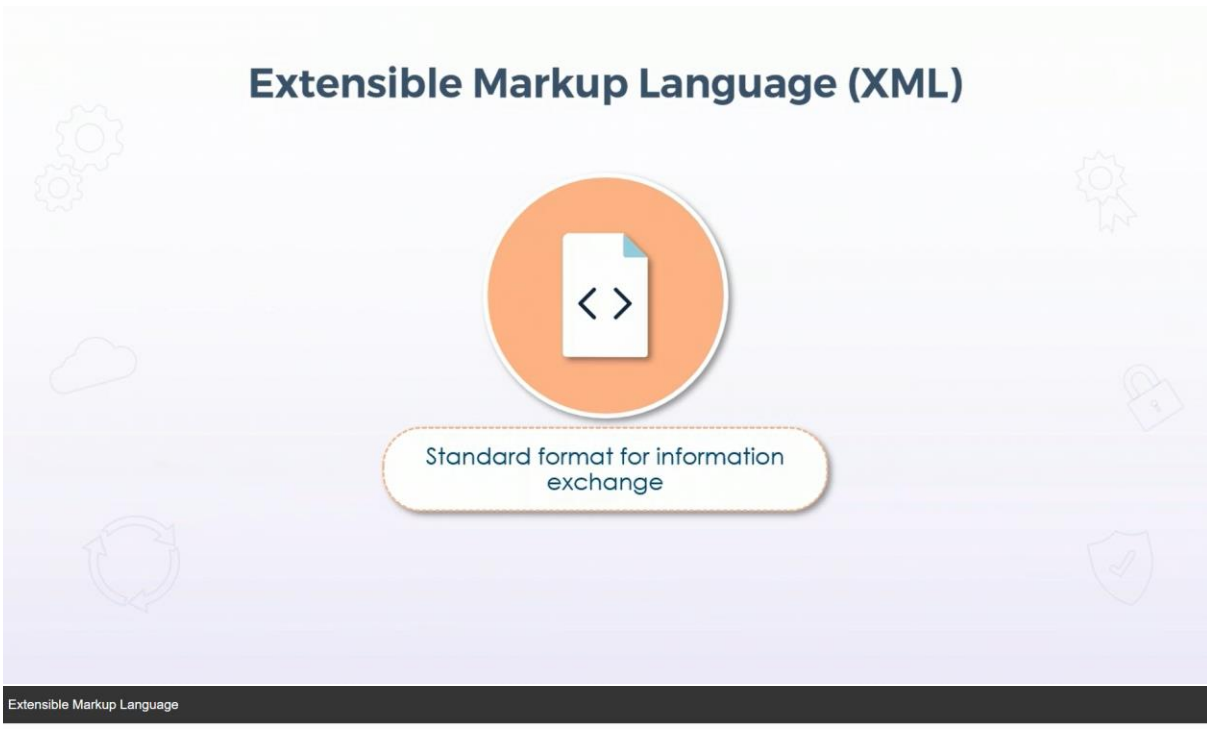
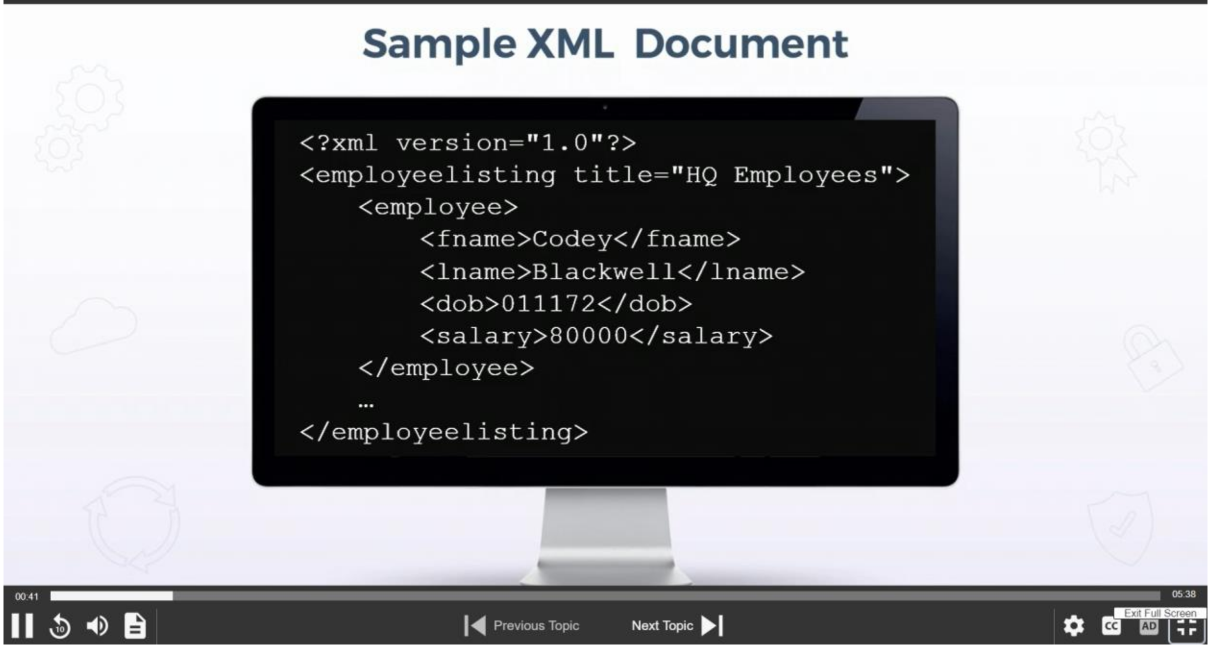


Extensible Markup Language (XML)

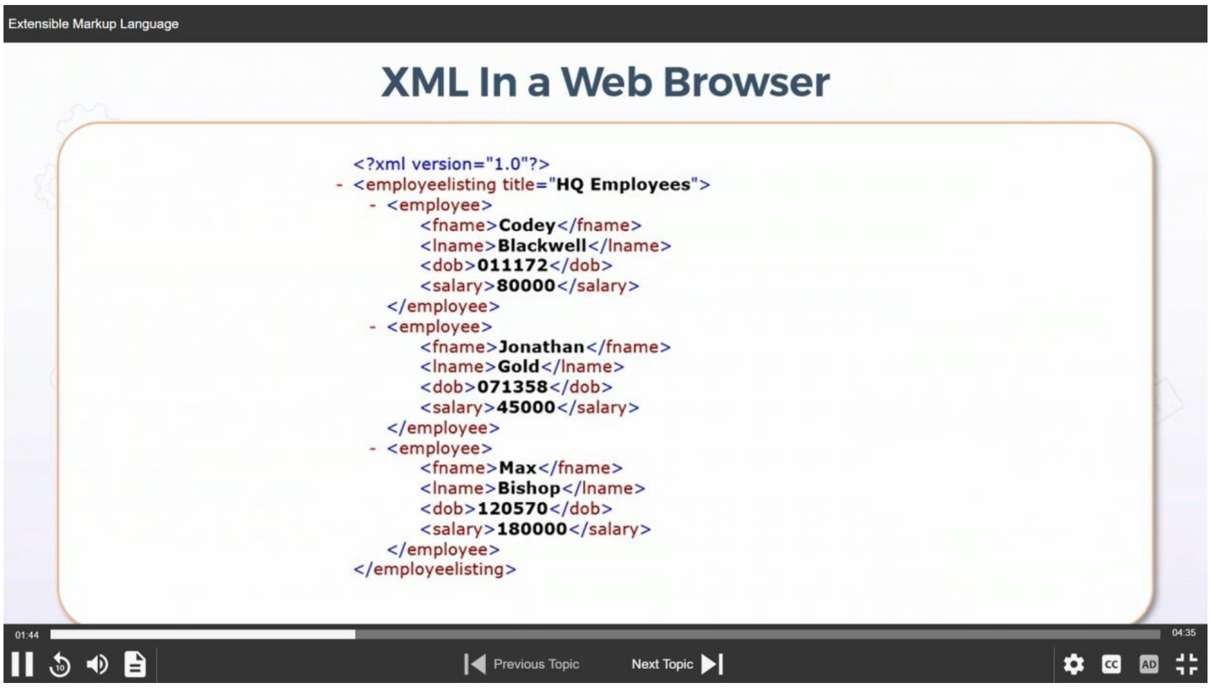
Standard format for information exchange

Extensible Markup Language



Sample XML Document

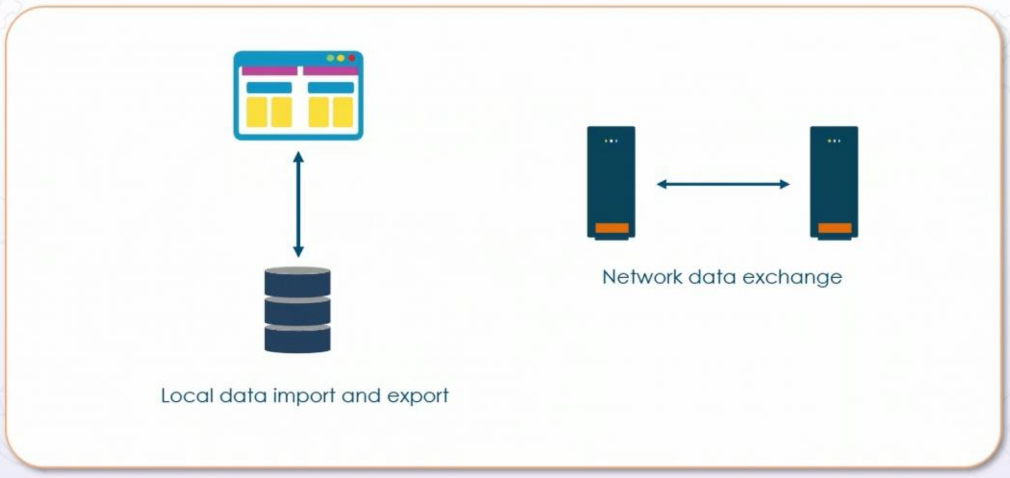
```
<?xml version="1.0"?>
<employeelisting title="HQ Employees">
    <employee>
        <fname>Codey</fname>
        <lname>Blackwell</lname>
        <dob>011172</dob>
        <salary>80000</salary>
    </employee>
    …
</employeelisting>
```

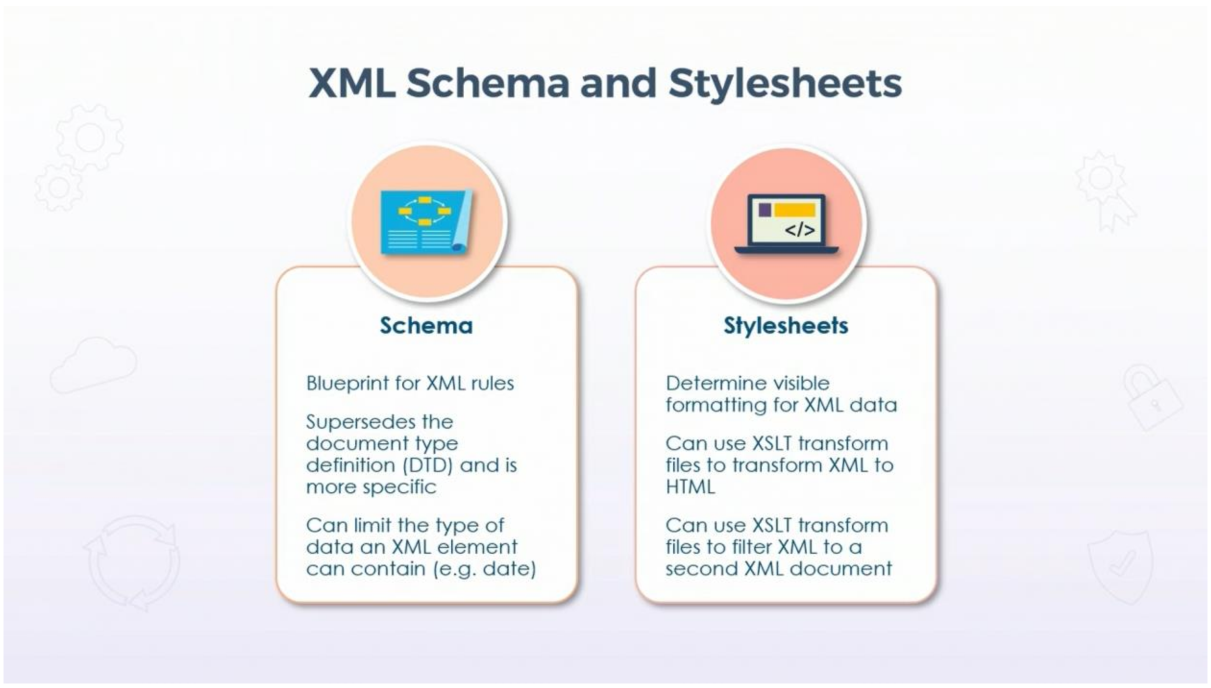


XML In a Web Browser

```
  <?xml version="1.0"?>
- <employeelisting title="HQ Employees">
  - <employee>
        <fname>Codey</fname>
        <lname>Blackwell</lname>
        <dob>011172</dob>
        <salary>80000</salary>
    </employee>
  - <employee>
        <fname>Jonathan</fname>
        <lname>Gold</lname>
        <dob>071358</dob>
        <salary>45000</salary>
    </employee>
  - <employee>
        <fname>Max</fname>
        <lname>Bishop</lname>
        <dob>120570</dob>
        <salary>180000</salary>
    </employee>
  </employeelisting>
```

XML Local and Remote Data Exchange





XML Schema and Stylesheets

Schema

Blueprint for XML rules

Supersedes the document type definition (DTD) and is more specific

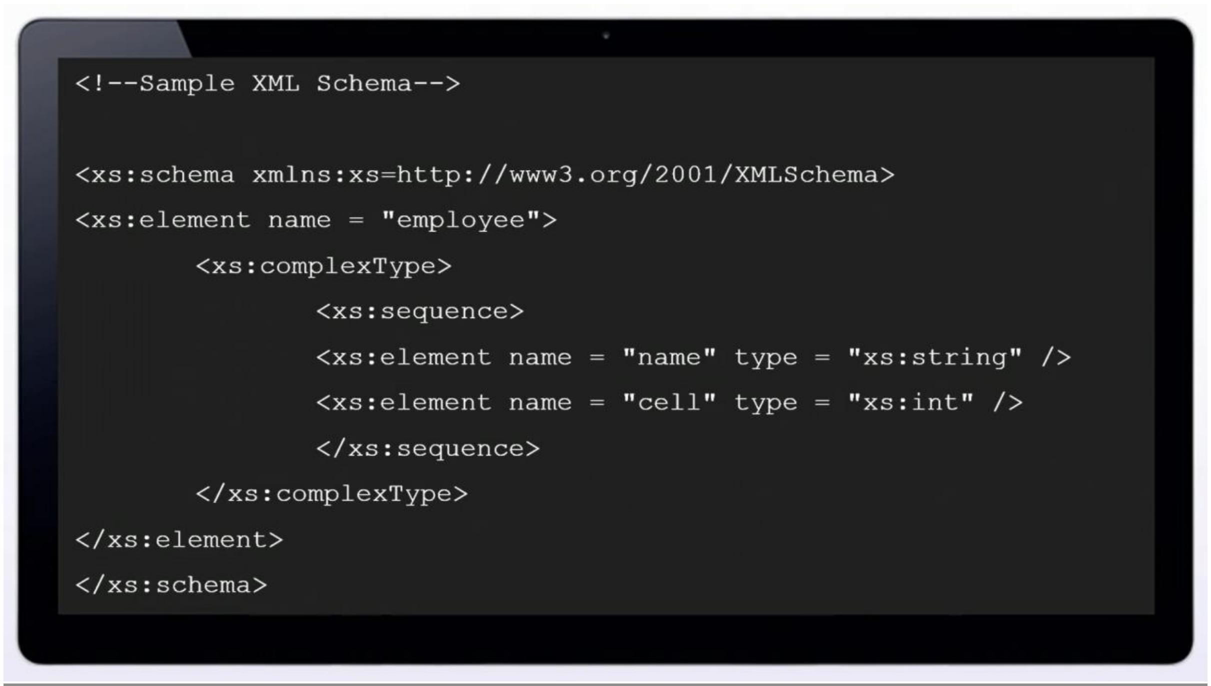
Can limit the type of data an XML element can contain (e.g. date)

Stylesheets

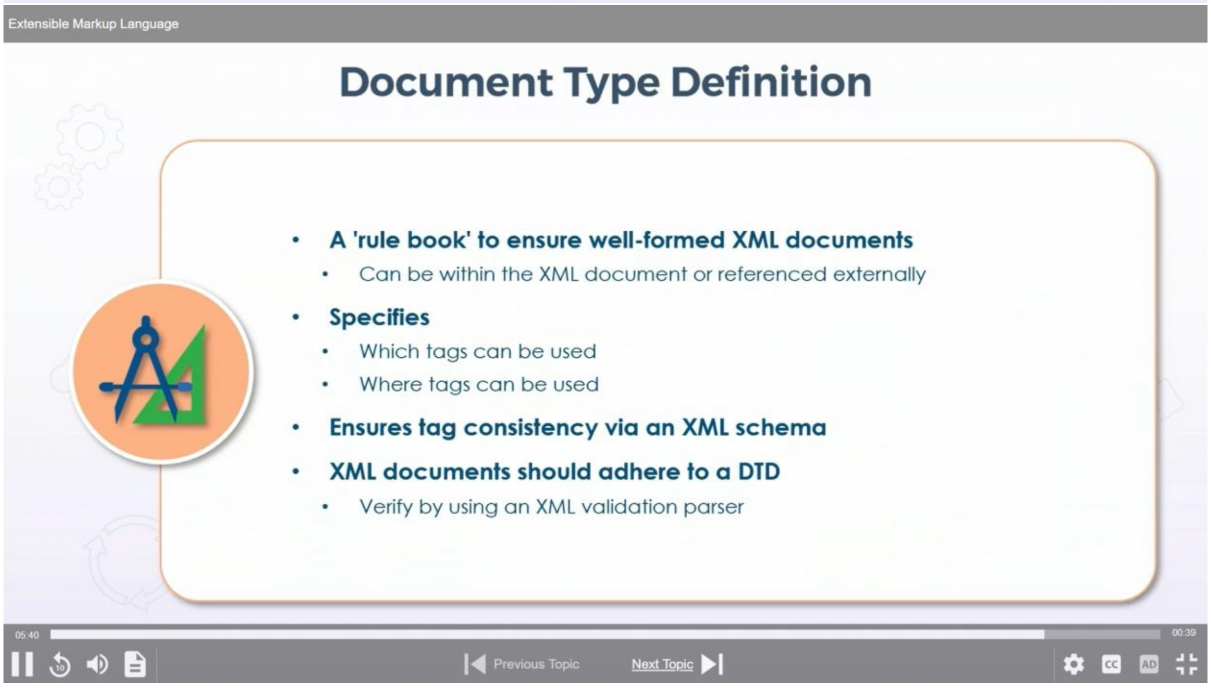
Determine visible formatting for XML data

Can use XSLT transform files to transform XML to HTML

Can use XSLT transform files to filter XML to a second XML document



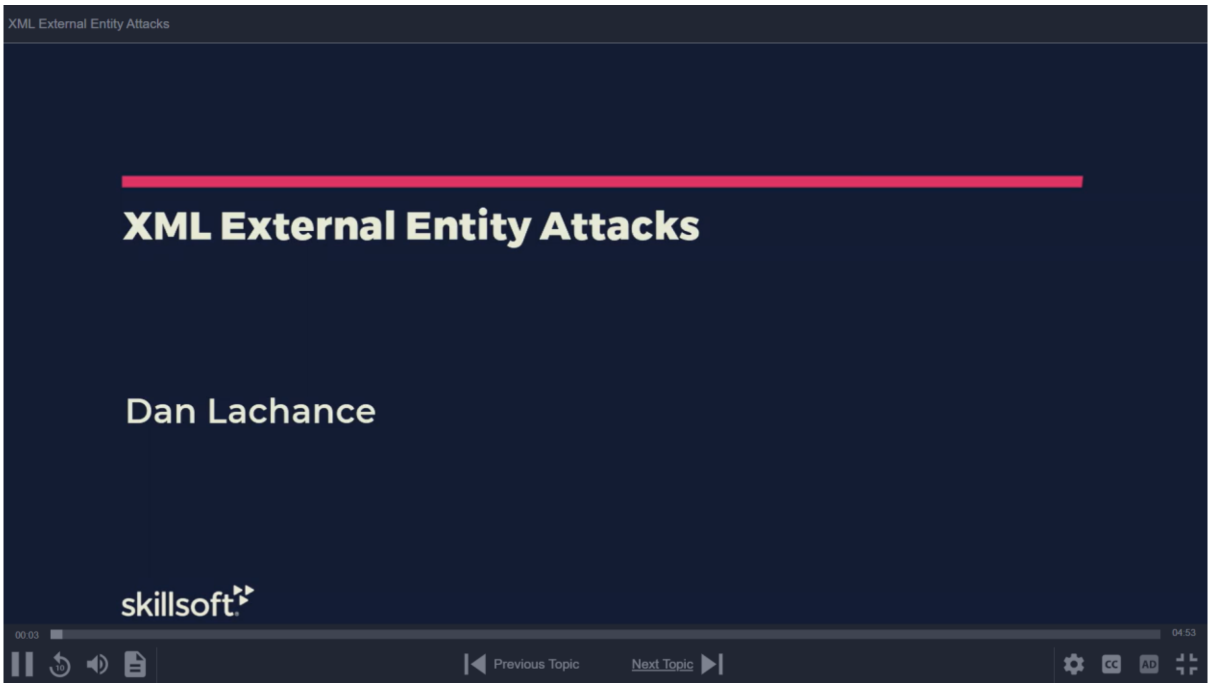
```
<!--Sample XML Schema-->

<xs:schema xmlns:xs=http://www3.org/2001/XMLSchema>
<xs:element name = "employee">
        <xs:complexType>
                <xs:sequence>
                <xs:element name = "name" type = "xs:string" />
                <xs:element name = "cell" type = "xs:int" />
                </xs:sequence>
        </xs:complexType>
</xs:element>
</xs:schema>
```



Extensible Markup Language

Document Type Definition

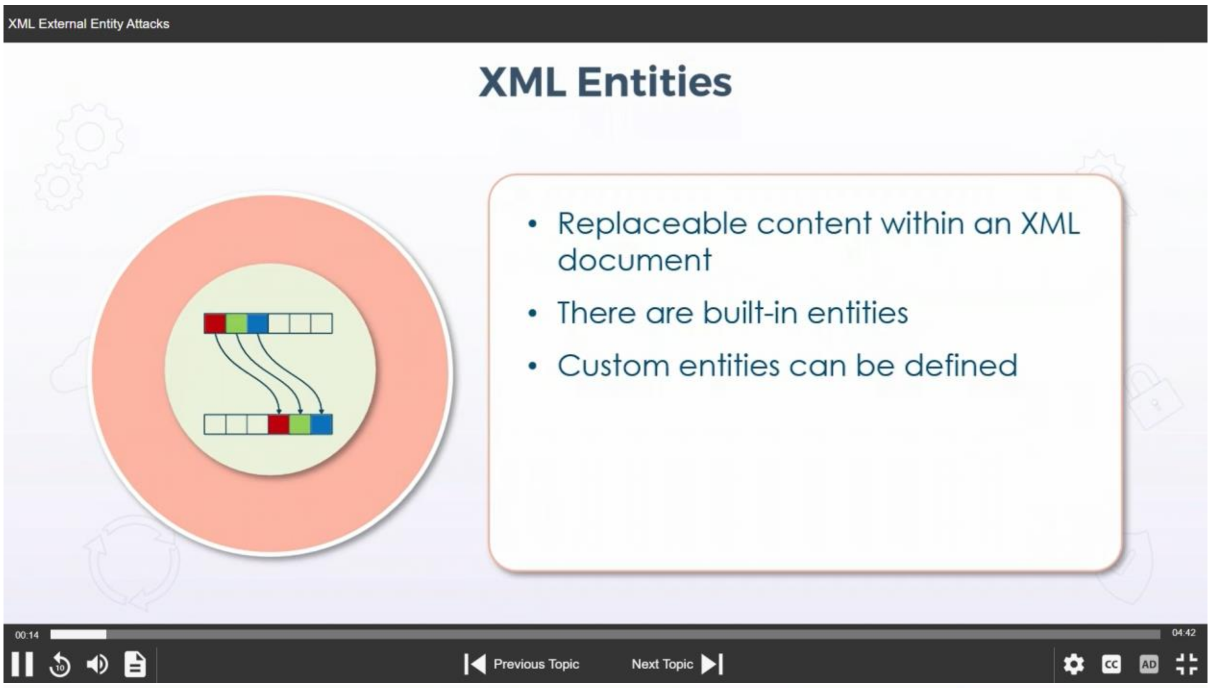
- **A 'rule book' to ensure well-formed XML documents**
  - Can be within the XML document or referenced externally
- **Specifies**
  - Which tags can be used
  - Where tags can be used
- **Ensures tag consistency via an XML schema**
- **XML documents should adhere to a DTD**
  - Verify by using an XML validation parser


XML External Entity Attacks

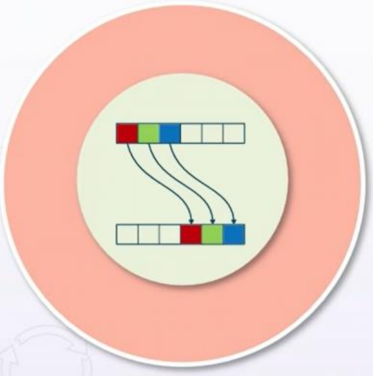
XML External Entity Attacks

Dan Lachance

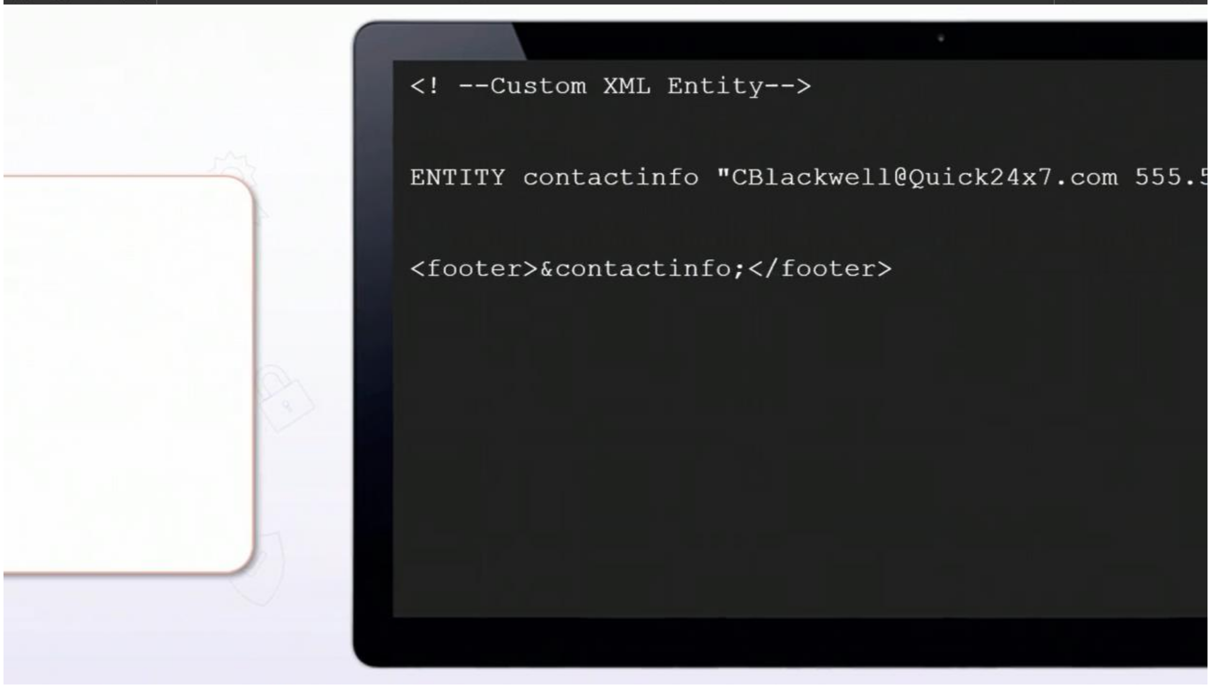
skillsoft


XML External Entity Attacks

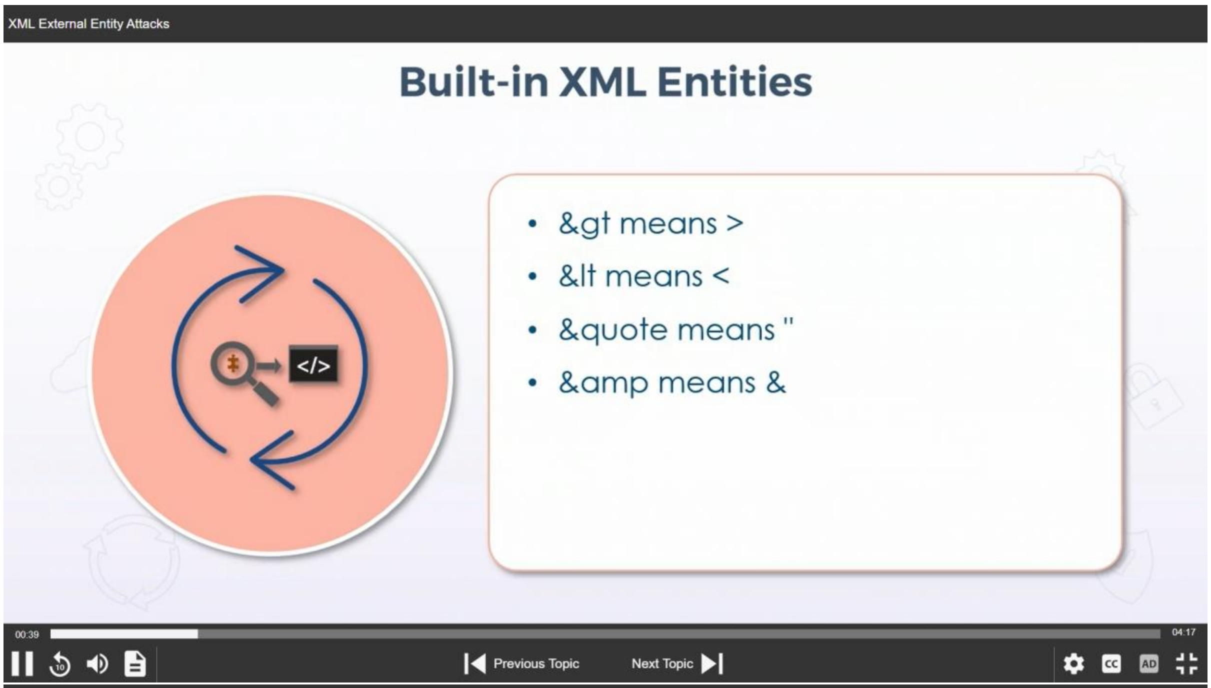
XML Entities



- Replaceable content within an XML document
- There are built-in entities
- Custom entities can be defined



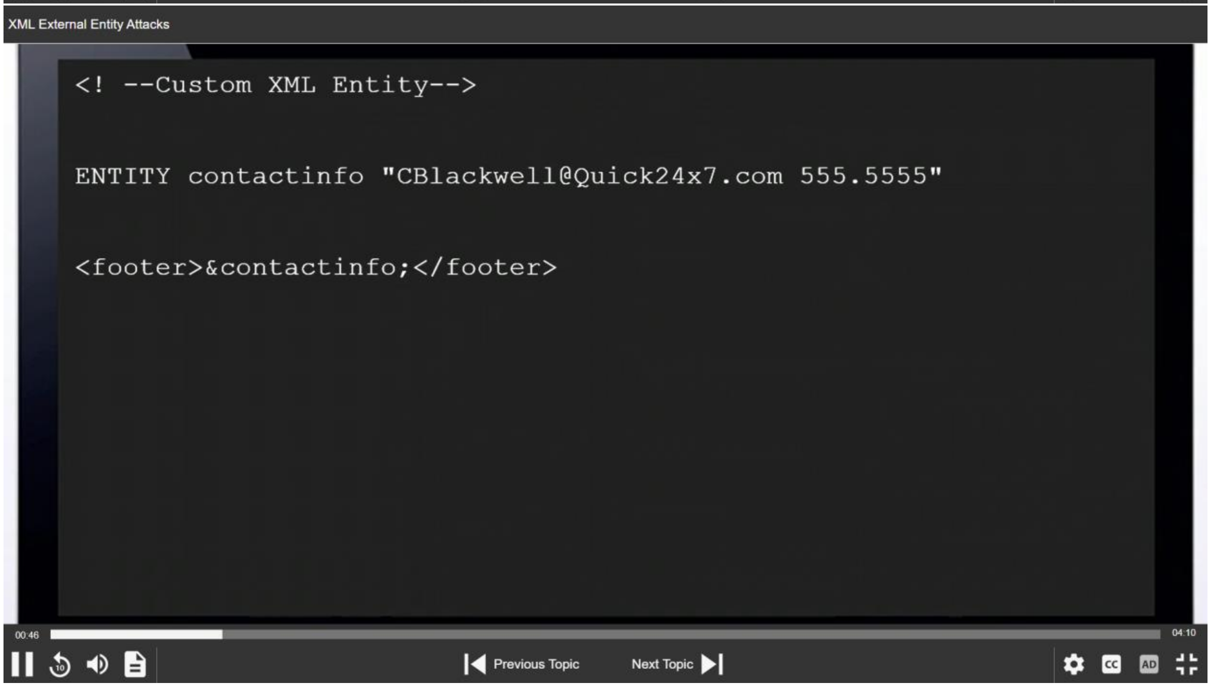
```
<! --Custom XML Entity-->

ENTITY contactinfo "CBlackwell@Quick24x7.com 555.5

<footer>&contactinfo;</footer>
```



XML External Entity Attacks

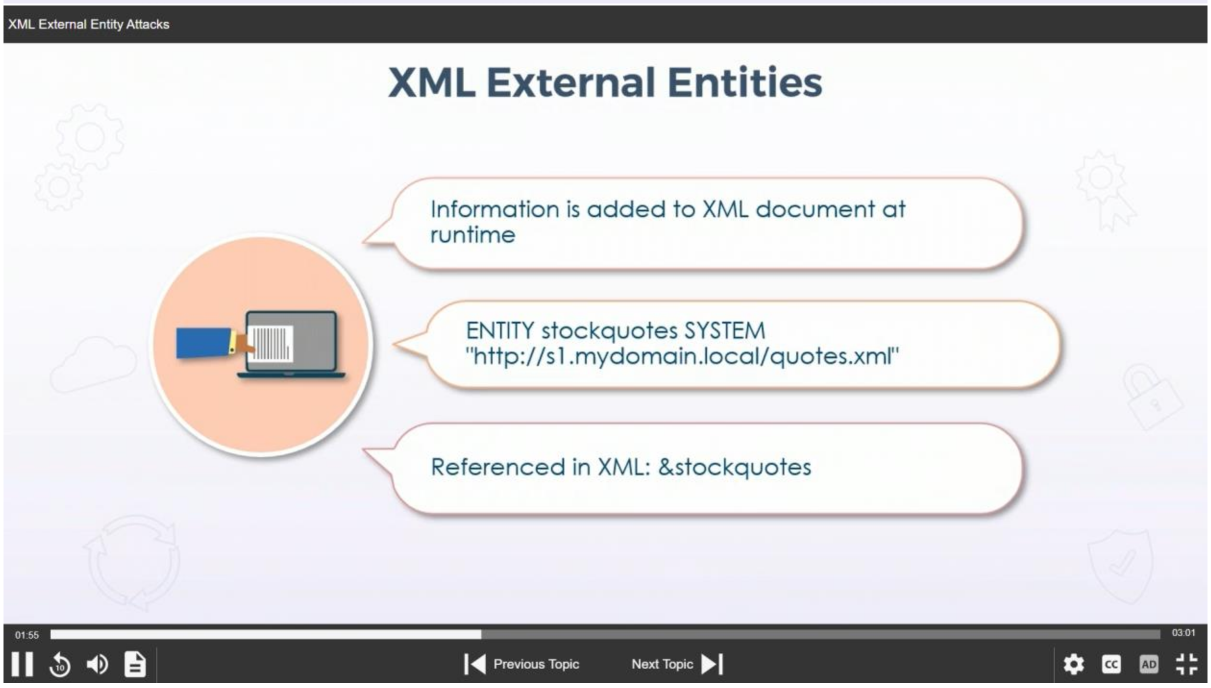
Built-in XML Entities

- &gt means >
- &lt means <
- &quote means "
- &amp means &



XML External Entity Attacks

```
<! --Custom XML Entity-->

ENTITY contactinfo "CBlackwell@Quick24x7.com 555.5555"

<footer>&contactinfo;</footer>
```

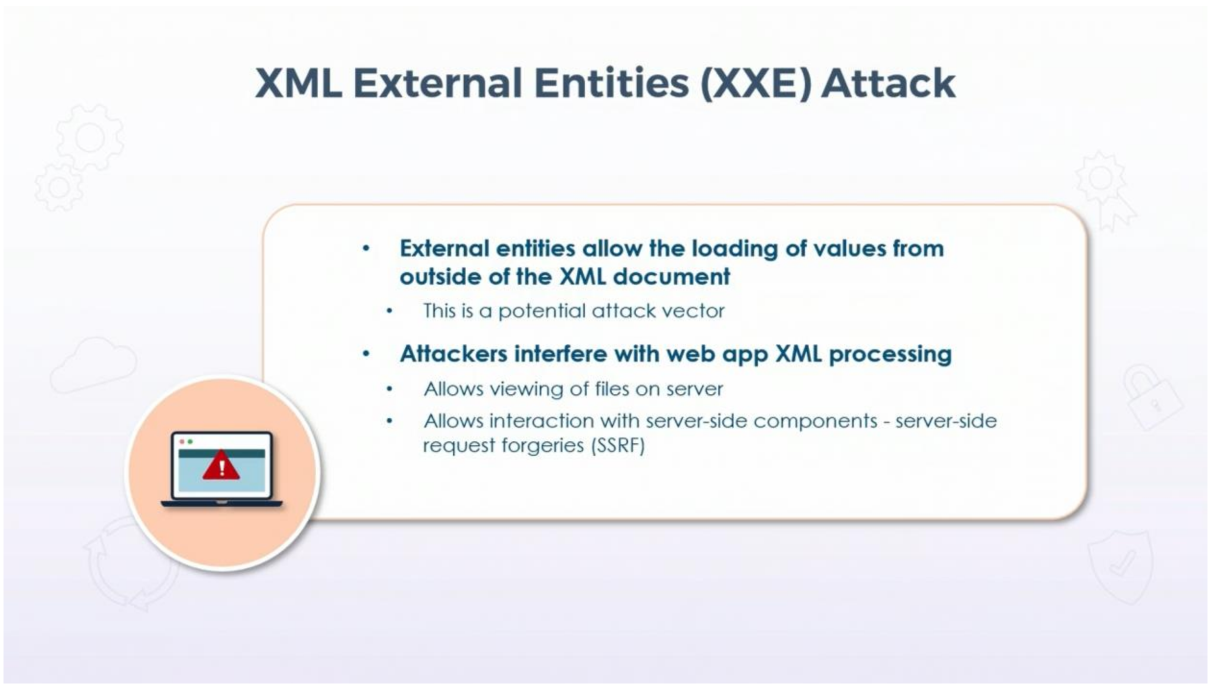


XML External Entities

Information is added to XML document at runtime

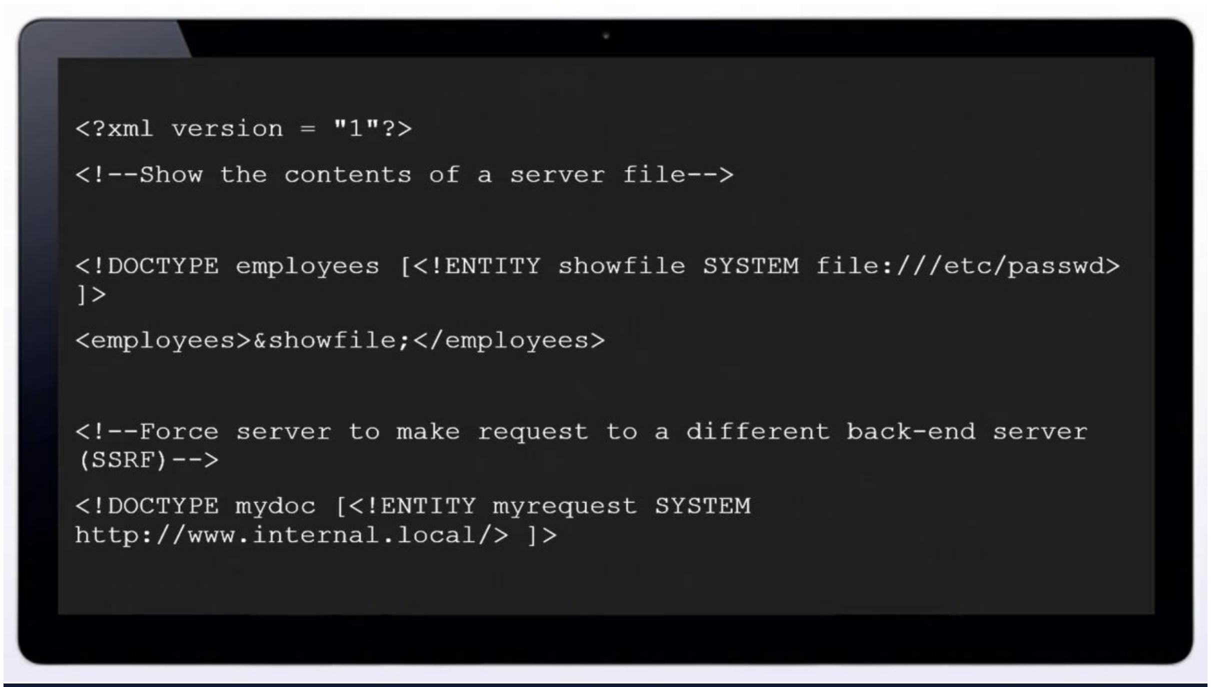
ENTITY stockquotes SYSTEM "http://s1.mydomain.local/quotes.xml"

Referenced in XML: &stockquotes



XML External Entities (XXE) Attack

- **External entities allow the loading of values from outside of the XML document**
  - This is a potential attack vector
- **Attackers interfere with web app XML processing**
  - Allows viewing of files on server
  - Allows interaction with server-side components - server-side request forgeries (SSRF)

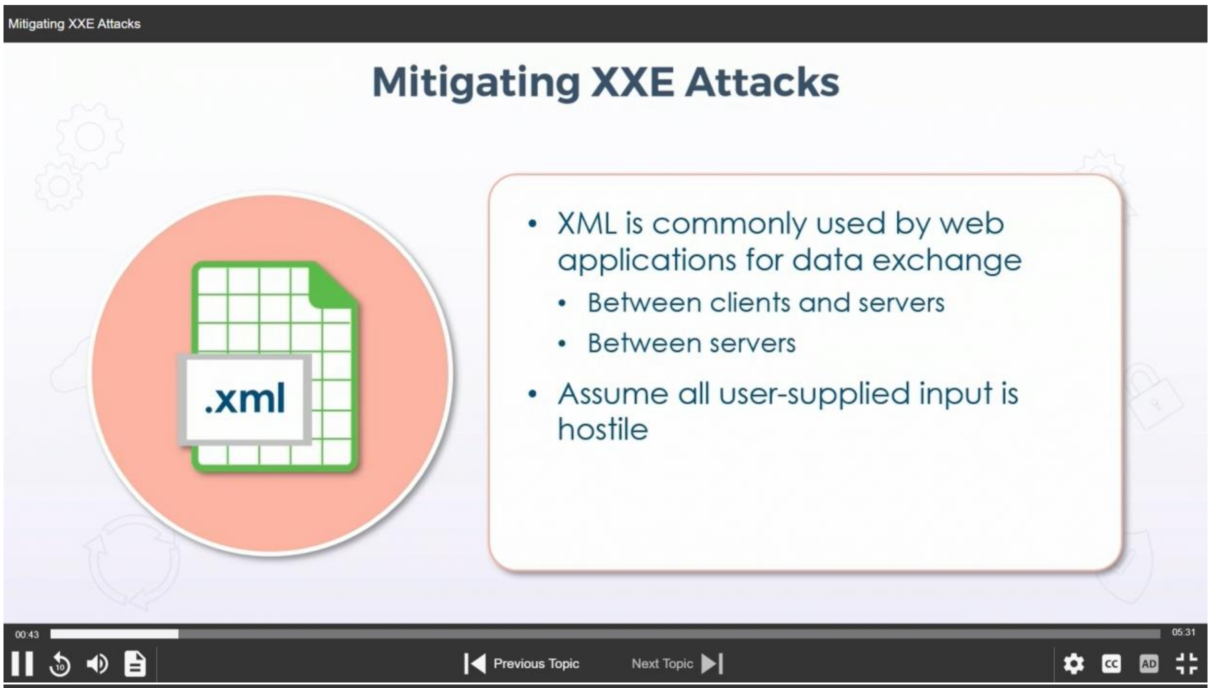


```
<?xml version = "1"?>
<!--Show the contents of a server file-->

<!DOCTYPE employees [<!ENTITY showfile SYSTEM file:///etc/passwd>
]>
<employees>&showfile;</employees>

<!--Force server to make request to a different back-end server
(SSRF)-->
<!DOCTYPE mydoc [<!ENTITY myrequest SYSTEM
http://www.internal.local/> ]>
```

Mitigating XXE Attacks

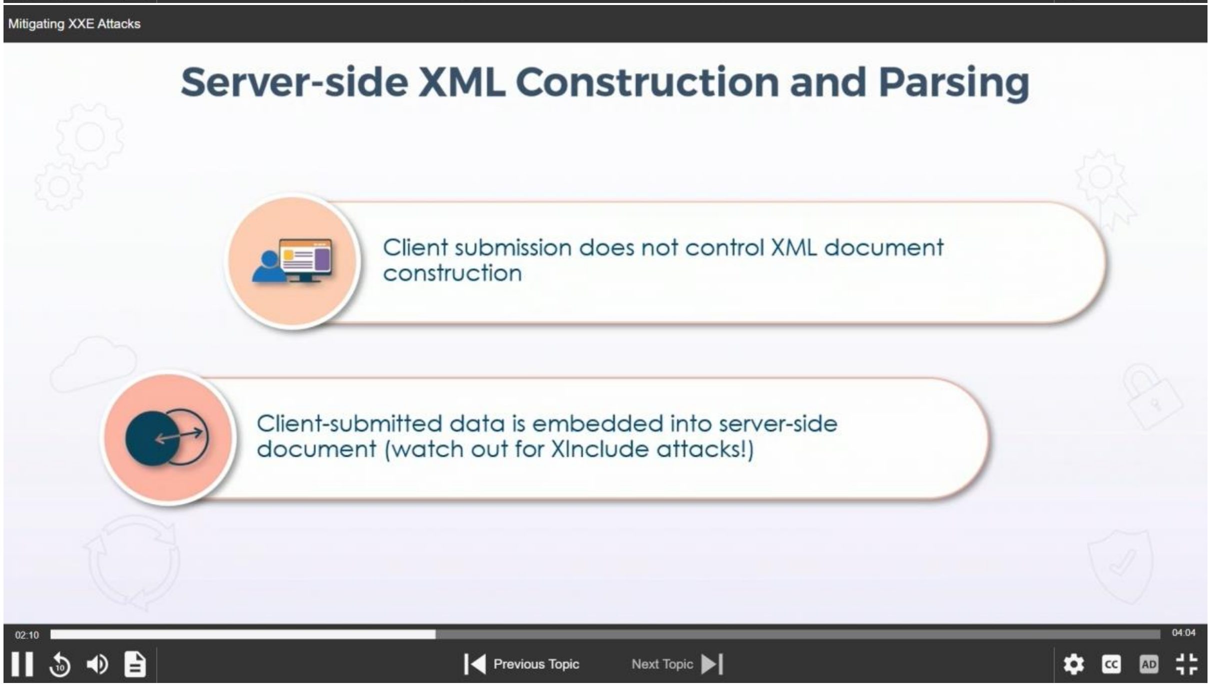
Dan Lachance

skillsoft


Mitigating XXE Attacks

Mitigating XXE Attacks

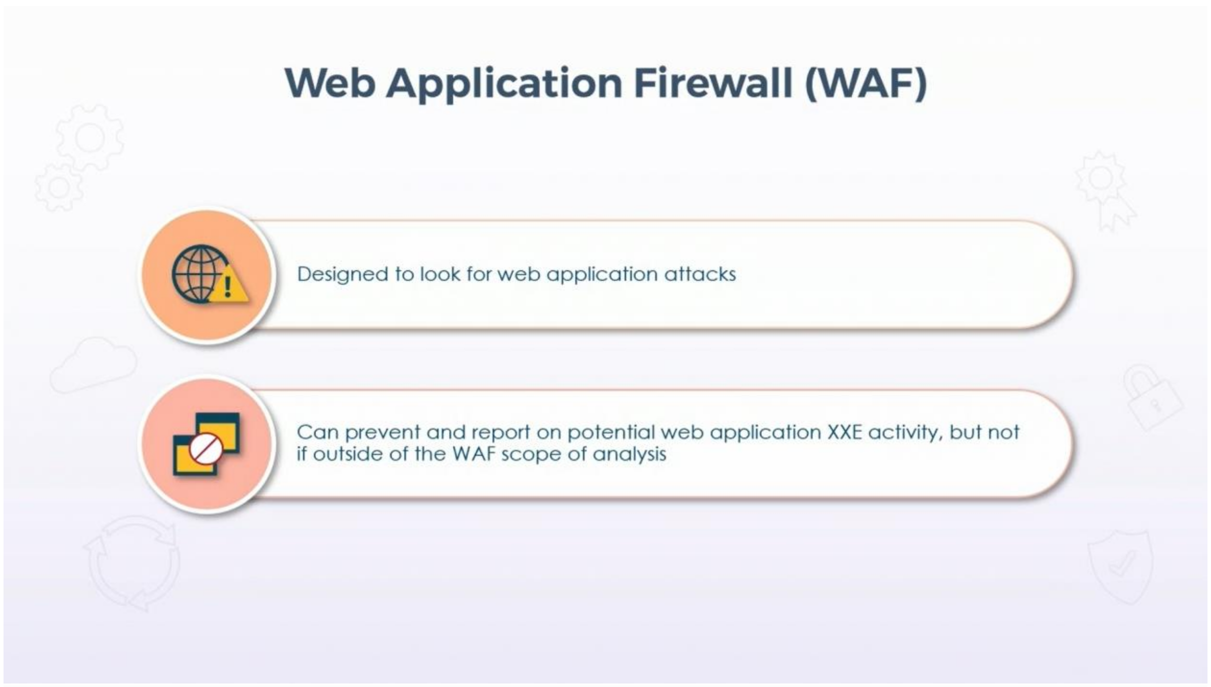
- XML is commonly used by web applications for data exchange
  - Between clients and servers
  - Between servers
- Assume all user-supplied input is hostile


Mitigating XXE Attacks

Server-side XML Construction and Parsing

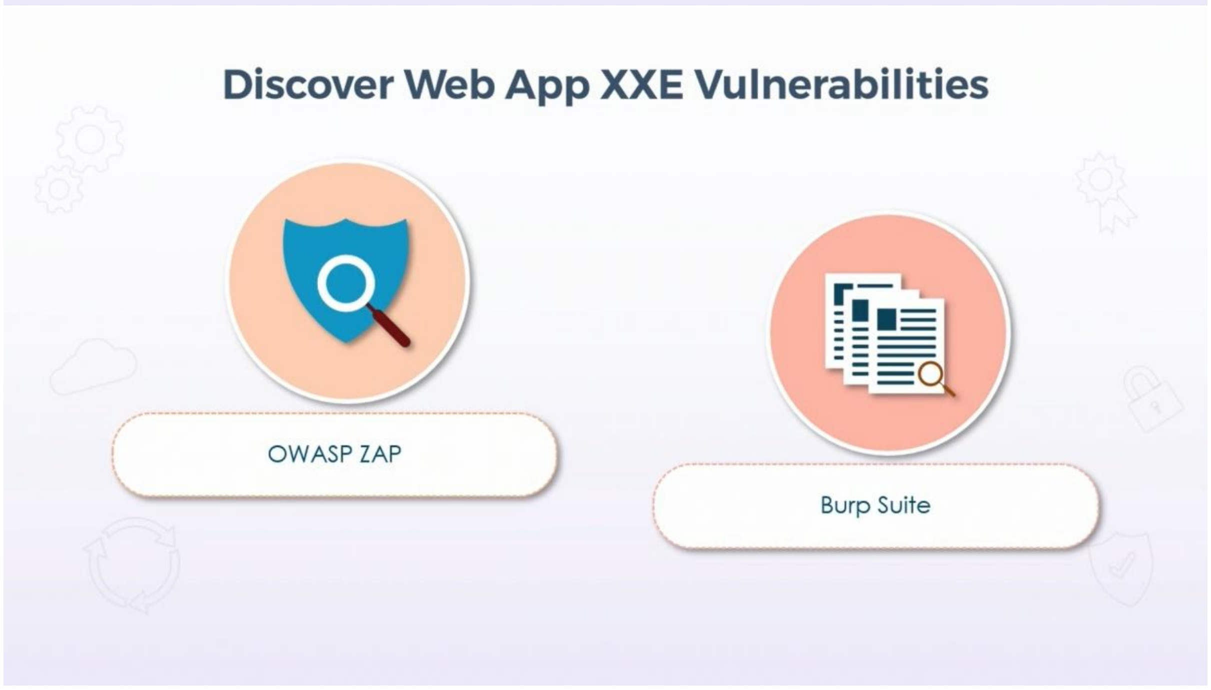
Client submission does not control XML document construction

Client-submitted data is embedded into server-side document (watch out for XInclude attacks!)



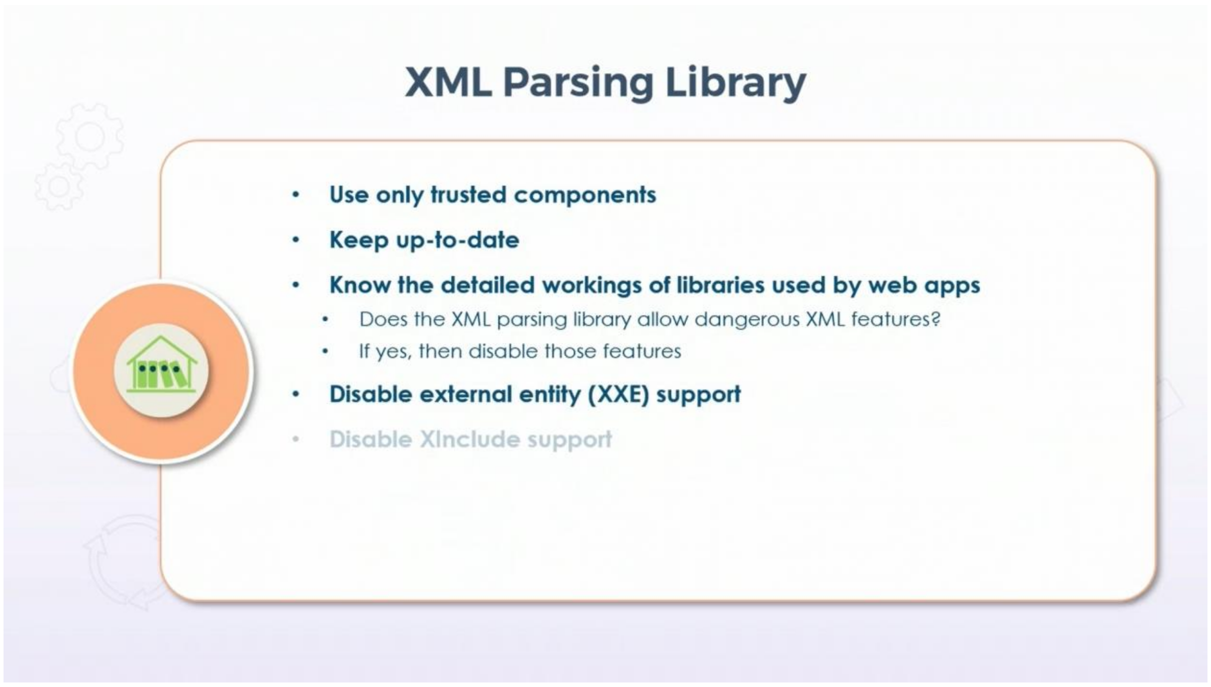
Web Application Firewall (WAF)

- Designed to look for web application attacks
- Can prevent and report on potential web application XXE activity, but not if outside of the WAF scope of analysis



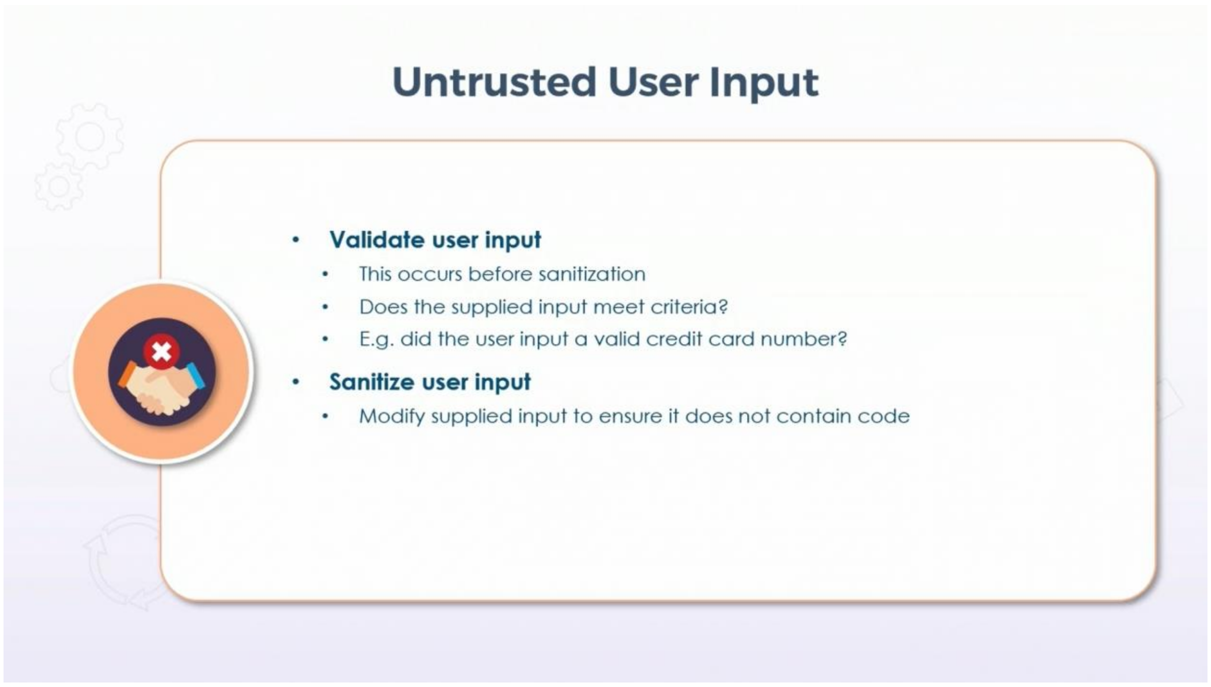
Discover Web App XXE Vulnerabilities

- OWASP ZAP
- Burp Suite



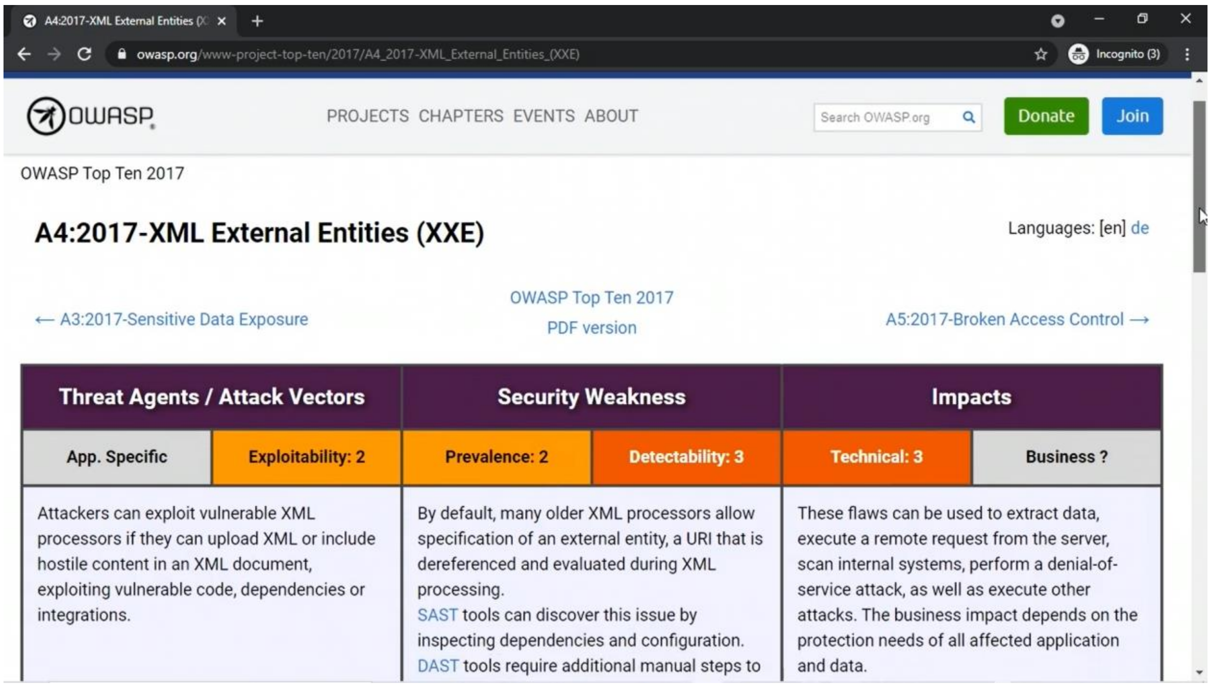
XML Parsing Library

- **Use only trusted components**
- **Keep up-to-date**
- **Know the detailed workings of libraries used by web apps**
  - Does the XML parsing library allow dangerous XML features?
  - If yes, then disable those features
- **Disable external entity (XXE) support**
- **Disable XInclude support**



Untrusted User Input

- **Validate user input**
  - This occurs before sanitization
  - Does the supplied input meet criteria?
  - E.g. did the user input a valid credit card number?
- **Sanitize user input**
  - Modify supplied input to ensure it does not contain code


A4:2017-XML External Entities (X
owasp.org/www-project-top-ten/2017/A4_2017-XML_External_Entities_(XXE)
Incognito (3)
OWASP
PROJECTS CHAPTERS EVENTS ABOUT
Search OWASP.org
Donate
Join

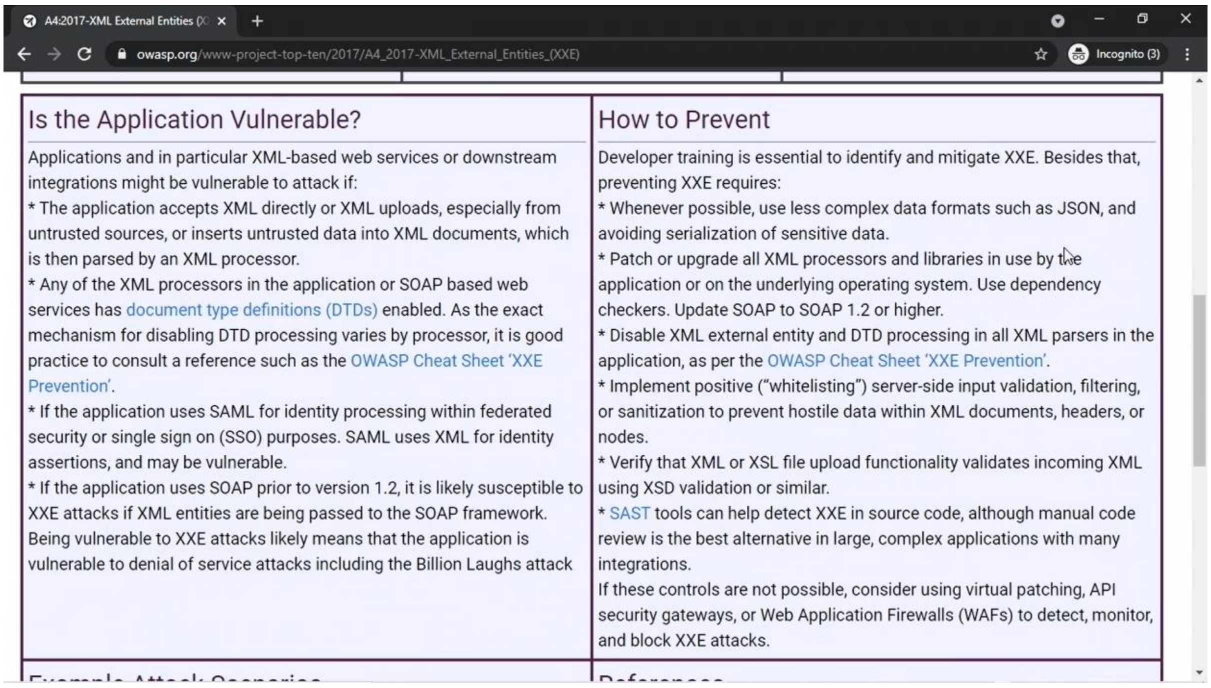
OWASP Top Ten 2017

A4:2017-XML External Entities (XXE)

Languages: [en] de

← A3:2017-Sensitive Data Exposure | OWASP Top Ten 2017 PDF version | A5:2017-Broken Access Control →

Threat Agents / Attack Vectors		Security Weakness		Impacts	
App. Specific	Exploitability: 2	Prevalence: 2	Detectability: 3	Technical: 3	Business ?
Attackers can exploit vulnerable XML processors if they can upload XML or include hostile content in an XML document, exploiting vulnerable code, dependencies or integrations.		By default, many older XML processors allow specification of an external entity, a URI that is dereferenced and evaluated during XML processing. SAST tools can discover this issue by inspecting dependencies and configuration. DAST tools require additional manual steps to		These flaws can be used to extract data, execute a remote request from the server, scan internal systems, perform a denial-of-service attack, as well as execute other attacks. The business impact depends on the protection needs of all affected application and data.	

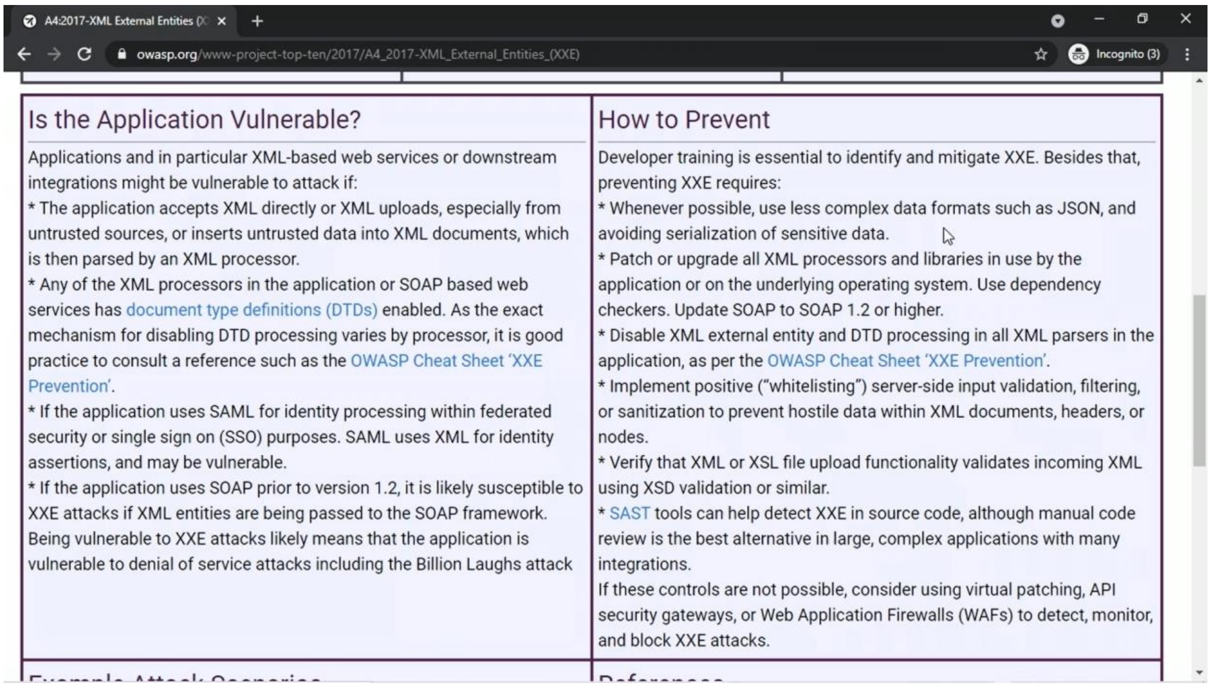

A4:2017-XML External Entities (X
owasp.org/www-project-top-ten/2017/A4_2017-XML_External_Entities_(XXE)
Incognito (3)

Is the Application Vulnerable?

Applications and in particular XML-based web services or downstream integrations might be vulnerable to attack if:
* The application accepts XML directly or XML uploads, especially from untrusted sources, or inserts untrusted data into XML documents, which is then parsed by an XML processor.
* Any of the XML processors in the application or SOAP based web services has document type definitions (DTDs) enabled. As the exact mechanism for disabling DTD processing varies by processor, it is good practice to consult a reference such as the OWASP Cheat Sheet 'XXE Prevention'.
* If the application uses SAML for identity processing within federated security or single sign on (SSO) purposes. SAML uses XML for identity assertions, and may be vulnerable.
* If the application uses SOAP prior to version 1.2, it is likely susceptible to XXE attacks if XML entities are being passed to the SOAP framework.
Being vulnerable to XXE attacks likely means that the application is vulnerable to denial of service attacks including the Billion Laughs attack

How to Prevent

Developer training is essential to identify and mitigate XXE. Besides that, preventing XXE requires:
* Whenever possible, use less complex data formats such as JSON, and avoiding serialization of sensitive data.
* Patch or upgrade all XML processors and libraries in use by the application or on the underlying operating system. Use dependency checkers. Update SOAP to SOAP 1.2 or higher.
* Disable XML external entity and DTD processing in all XML parsers in the application, as per the OWASP Cheat Sheet 'XXE Prevention'.
* Implement positive ("whitelisting") server-side input validation, filtering, or sanitization to prevent hostile data within XML documents, headers, or nodes.
* Verify that XML or XSL file upload functionality validates incoming XML using XSD validation or similar.
* SAST tools can help detect XXE in source code, although manual code review is the best alternative in large, complex applications with many integrations.
If these controls are not possible, consider using virtual patching, API security gateways, or Web Application Firewalls (WAFs) to detect, monitor, and block XXE attacks.


A4:2017-XML External Entities (X
owasp.org/www-project-top-ten/2017/A4_2017-XML_External_Entities_(XXE)
Incognito (3)

Is the Application Vulnerable?	How to Prevent
Applications and in particular XML-based web services or downstream integrations might be vulnerable to attack if: * The application accepts XML directly or XML uploads, especially from untrusted sources, or inserts untrusted data into XML documents, which is then parsed by an XML processor. * Any of the XML processors in the application or SOAP based web services has document type definitions (DTDs) enabled. As the exact mechanism for disabling DTD processing varies by processor, it is good practice to consult a reference such as the OWASP Cheat Sheet 'XXE Prevention'. * If the application uses SAML for identity processing within federated security or single sign on (SSO) purposes. SAML uses XML for identity assertions, and may be vulnerable. * If the application uses SOAP prior to version 1.2, it is likely susceptible to XXE attacks if XML entities are being passed to the SOAP framework. Being vulnerable to XXE attacks likely means that the application is vulnerable to denial of service attacks including the Billion Laughs attack	Developer training is essential to identify and mitigate XXE. Besides that, preventing XXE requires: * Whenever possible, use less complex data formats such as JSON, and avoiding serialization of sensitive data. * Patch or upgrade all XML processors and libraries in use by the application or on the underlying operating system. Use dependency checkers. Update SOAP to SOAP 1.2 or higher. * Disable XML external entity and DTD processing in all XML parsers in the application, as per the OWASP Cheat Sheet 'XXE Prevention'. * Implement positive ("whitelisting") server-side input validation, filtering, or sanitization to prevent hostile data within XML documents, headers, or nodes. * Verify that XML or XSL file upload functionality validates incoming XML using XSD validation or similar. * SAST tools can help detect XXE in source code, although manual code review is the best alternative in large, complex applications with many integrations. If these controls are not possible, consider using virtual patching, API security gateways, or Web Application Firewalls (WAFs) to detect, monitor, and block XXE attacks.