XML external entity (XXE) injection

XML External Entity (XXE) injection is a web security vulnerability that allows an attacker to interfere with an application's processing of XML data.

1. Lab: Exploiting XXE using external entities to retrieve files

Steps Performed:

- Access the Target Functionality

Navigated to a product page on the lab application.

Clicked the "Check stock" button, which triggered a POST request to the server with XML data.

- Intercept the Request in Burp Suite

Enabled the **Proxy** tab in Burp Suite.

Captured the outgoing HTTP request from the browser.

- Inject the External Entity Definition

Modified the intercepted XML request by inserting an external entity definition between the XML declaration and the <stockCheck> element:

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

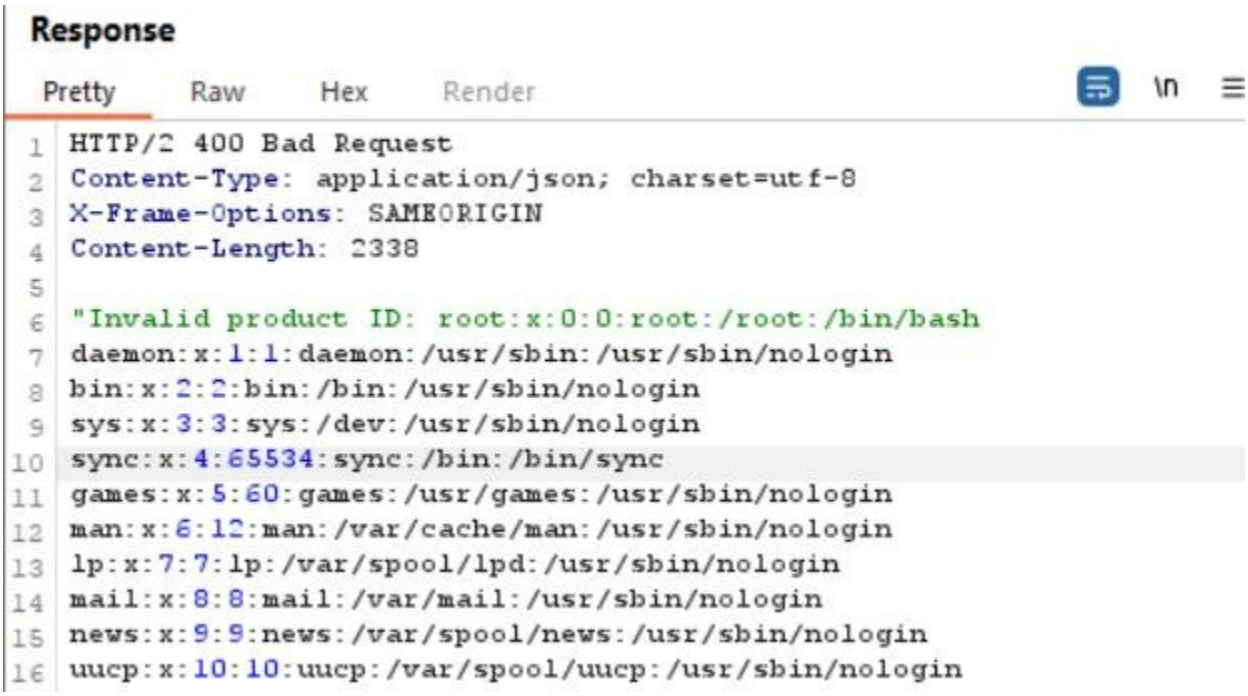
- Forward the Modified Request

Sent the modified request to the server.

The server parsed the XML and processed the external entity.

- Observe the Server Response

The HTTP response contained the contents of /etc/passwd displayed in the error message:



This confirms that the application was vulnerable to XML External Entity Injection (XXE)**.**

2. Lab: Exploiting XXE to perform SSRF attacks

Steps Performed:

- Identify the Vulnerable Functionality

Navigated to a product page on the lab application.

Used the "Check stock" feature, which sends XML input to the server.

- Intercept the Request in Burp Suite

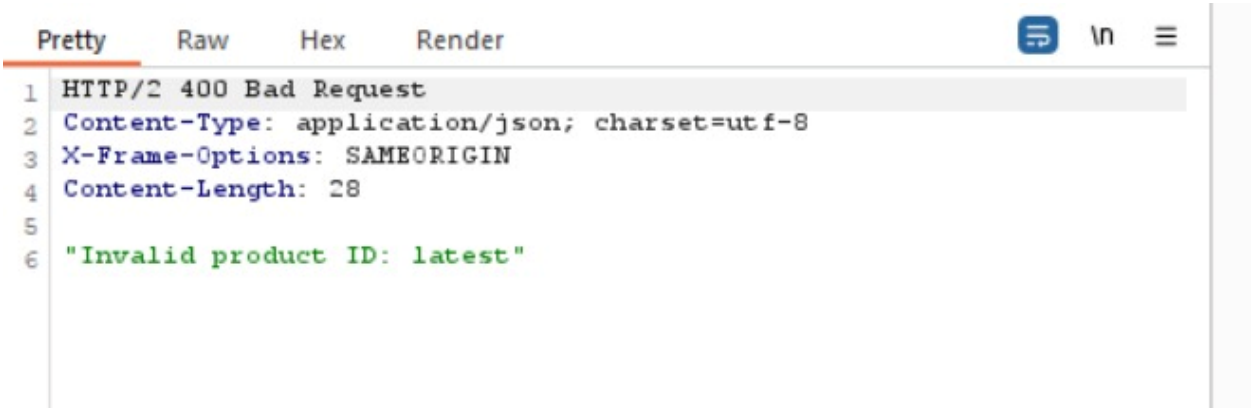
Enabled Burp Suite's Proxy and captured the outgoing POST request.

Observed that the request body contained XML like:

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

- Inject an External Entity for SSRF

Modified the request to include an external entity referencing the AWS EC2 metadata service:



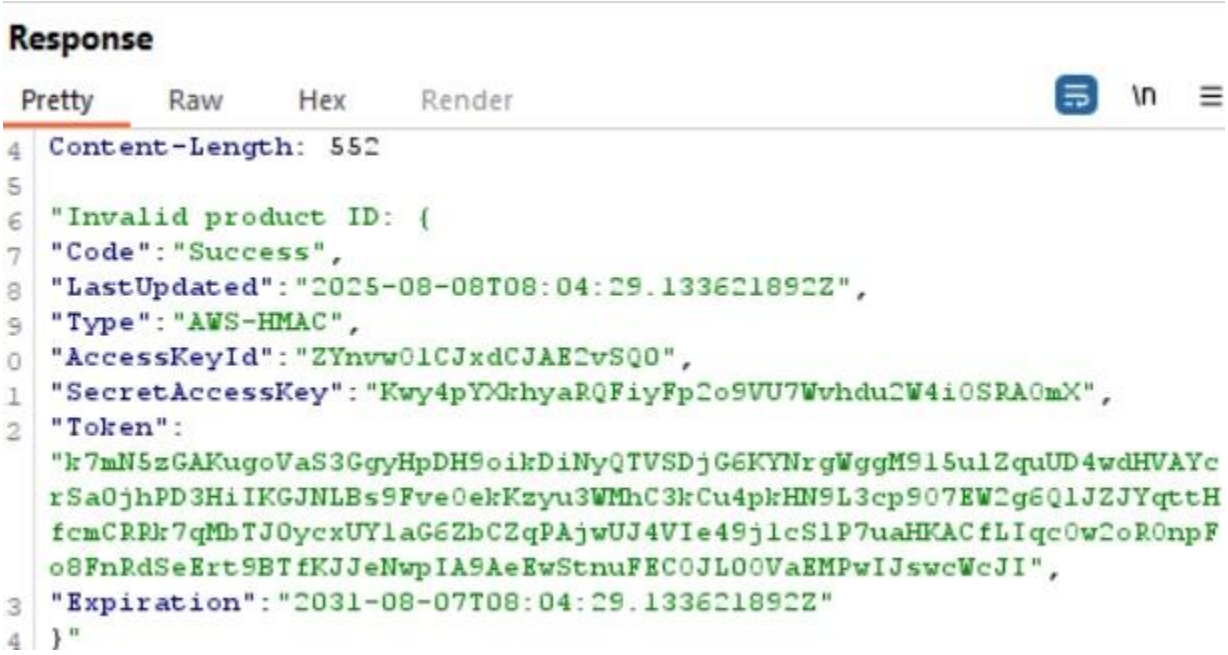
Forwarded the request and received the first response showing available directories

- Enumerate Metadata Directories

Updated the SYSTEM URL to step deeper into the metadata service:

- Retrieve IAM Secret Access Key

The final request returned JSON containing AWS IAM credentials:



3. Lab: Blind XXE with out-of-band interaction

Steps Performed:

- Identify the Target Functionality

Navigated to a product page on the target lab application.

Located the "Check stock" feature, which sends XML input to the server.

- Intercept the Request in Burp Suite Professional

Enabled the Proxy tab and captured the POST request sent to the server.

Observed the XML structure in the request body

- Craft the Blind XXE Payload

Inserted an external entity definition that referenced a Burp Collaborator subdomain

Used Right-click → Insert Collaborator Payload in Burp to automatically insert a unique subdomain.

- Send the Malicious Request

Forwarded the modified request to the server.

- Detect the Out-of-Band Interaction

Opened the Collaborator tab in Burp Suite Professional.

Clicked "Poll now" and observed inbound DNS and HTTP interactions from the target application to the Collaborator server.

- Verification

The recorded interactions confirmed that the XML parser processed the malicious external entity, resulting in an out-of-band request to the attacker-controlled domain.

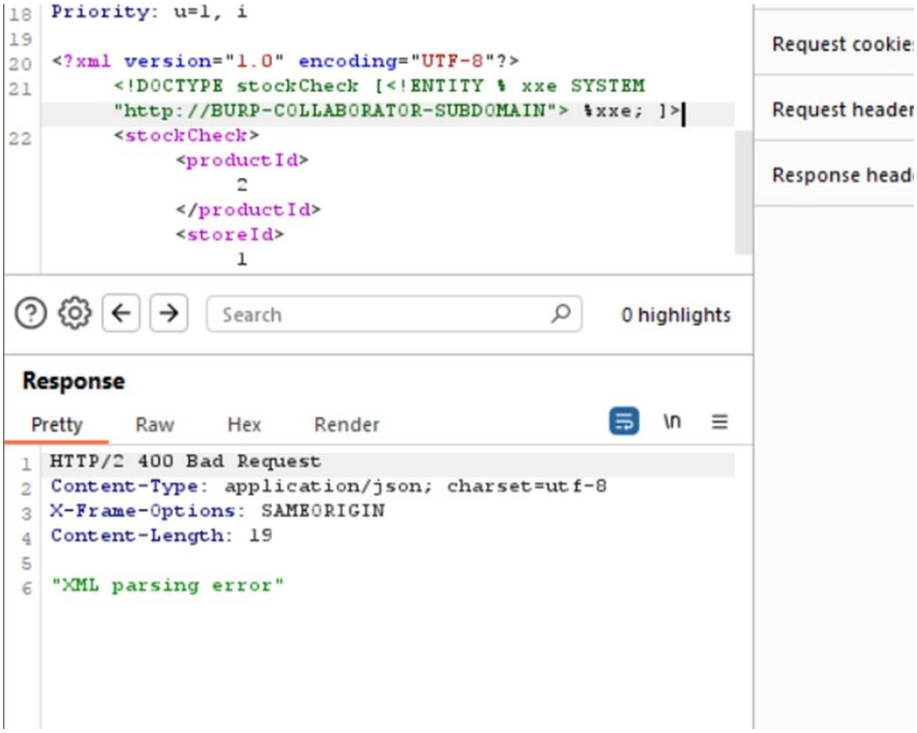
4. Lab: Blind XXE with out-of-band interaction via XML parameter entities

Steps Performed:

- Identify the Vulnerable Functionality

Navigated to a product page in the lab application.

Located the "Check stock" feature, which sends XML input to the server.



Initial tests showed that regular external entities were blocked, indicating the need for parameter entity usage.

- Intercept the Request in Burp Suite Professional

Enabled the Proxy tab and captured the POST request sent by the application.

The original request body contained XML like

- Craft the Payload with a Parameter Entity

Modified the intercepted request to include a parameter entity definition that referenced a Burp Collaborator subdomain

The "Insert Collaborator payload" option was used to generate a unique subdomain.

- Send the Exploit Request

Forwarded the modified XML request to the server.

- Check for Out-of-Band Interaction

Opened the Collaborator tab in Burp Suite Professional.

Clicked "Poll now" and observed DNS and HTTP interactions originating from the vulnerable application to the Collaborator server.

- Verification

These interactions confirmed that the XML parser processed the parameter entity, triggering an external request even though regular external entities were blocked.