SAFE SECURITY

URL Exploiting

XXE to SSRF

Research Paper

www.safe.security

V.19.02.01.01

Table of Contents

We will learn how we can use XXE vulnerability to perform an SSRF attack. For this, we will cover the required areas and look into demonstration:

1. Introduction to XML
2. What are external entities, and how it's used?
3. Taking a look at XXE with a simple example
4. Understanding SSRF vulnerability
5. A simple example of how SSRF works
6. Using XXE to exploit SSRF vulnerability in a Web application

1
Introduction to XML

XML stands for eXtensible Markup Language much like HTML. It's used to store and transport data and was designed to be self-explanatory.
"Self-explanatory" can be understood from an example code which stores some information:

```
<?xml version="1.0" encoding="UTF-8"?>
<Person>
<Name>Xyz</Name>
<Age>22</Age>
</Person>
```

Here we are storing a person's information with his Name and Age. But how will we use this information? The code does not do anything in its current form, but URLwe has to make another code or software to use this information wrapped in tags for using it.

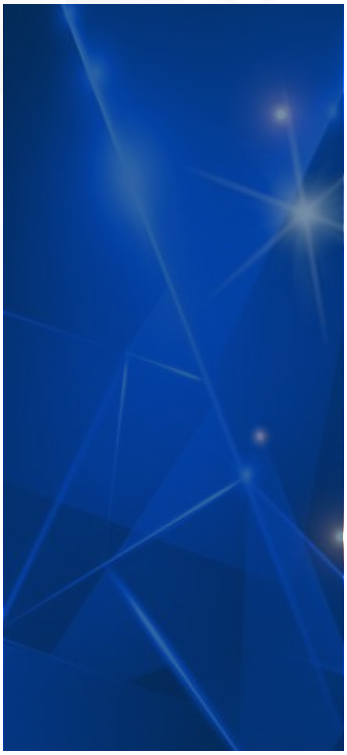
The XML tags are not pre-defined, which means that the author of the XML document invents the tag names just like in the case above. We have the Person, Name, and Age tags that are not pre-defined in XML standards but are made by the code developer.

So how can we use this XML data? XML does not tell us how it will be presented, and the stored data can be shown in any way by our application. Hence XML helps in completely separating the data from the presentation.

In general the XML documents have a definite structure:

```
<root>
        <child>
                <subchild>.....</subchild>
        </child>
</root>
```

The document can have only one root element. Also, each tag needs to be closed with a supporting closing tag. The tags are case sensitive, that is, <name> and <Name> tags are different.



Again from the first code example,
<?xml version="1.0" encoding="UTF-8"?>: The first line is the prolog, defining the version and character encoding to avoid errors due to international characters like Norwegian or French.

<Person>: Root Tag

<Name> and <Age>: Child Tags

Next, we need to take a look at Entities in XML. Entities are simple storage units in XML, just like variables used to assign values throughout the document.

Also, these entities are found in DTD, Document Type Definition. Syntax to create an entity is:

```
<!ENTITY entity-name "entity-value">
```

An example of creating an entity in XML and using it:

```
<?xml version="1.0" encoding="UTF-8"?>
<! DOCTYPE Person [
        <! ENTITY name “Xyz” >
]>
<Person>
<Name>&name;</Name>
<Age>22</Age>
</Person>
```

We can see that we have updated the previous example. We now used DTD and created an Entity inside it as “name,” which stores the person’s name and used it later in the document.

2

What are external entities, and how it's used

Now that we know how to make an entity and use it in the document, let's create an external entity used in the XXE exploit.

Using the SYSTEM keyword, we can create an external entity. This keyword tells the parser that the entity is of the external type, telling it to fetch the external resource and store it in the external entity.

But entities can not only store values but local or remote files and hence can be taken advantage of by a hacker. Lets take an example of how to create an external entity:

```
<?xml version="1.0" encoding="UTF-8"?>
<! DOCTYPE Pwn [
  <! ENTITY external SYSTEM "secret.txt" >
]>
<Pwn>&external;</Pwn>
```

The secret.txt file can be a secret local file present on your server that should not be accessible. But using an external entity, we can get the secret file and view the content on a vulnerable web application. Hence XML eXternal Entity Vulnerability.

3

Taking a look at XXE Vulnerability using a simple example

XXE is a vulnerability where we take advantage of the XML Parser and get our bad data executed. From bad data, we mean the malicious external entity we create and use in an XML document.

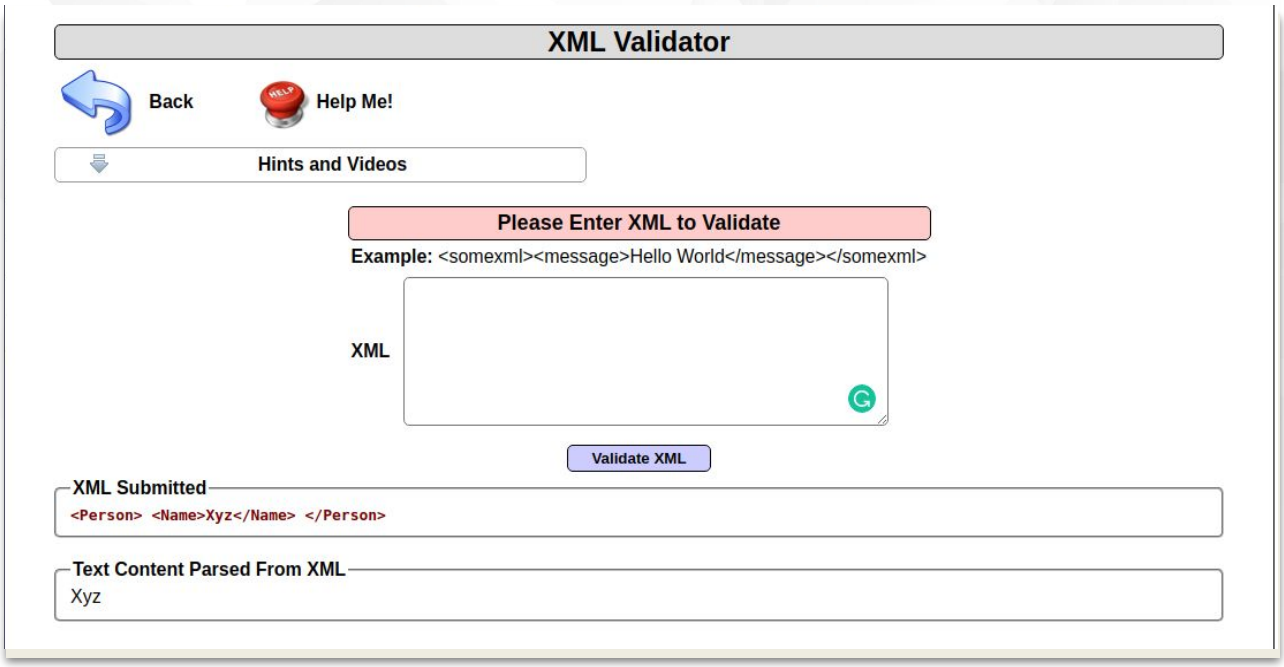
Sometimes XML Parsers refer to an external entity that allows an attacker to enter a payload in place of the external entity and execute it using the XML Parser. XXE can disclose local files in the file system of the website. A famous example of this vulnerability exploit is the Billion Laughs attack.

An example of XXE to fetch passwd file from the system:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Person [
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >
]>
<Person>
<Name>&xxe;</Name>
<Age>22</Age>
</Person>
```

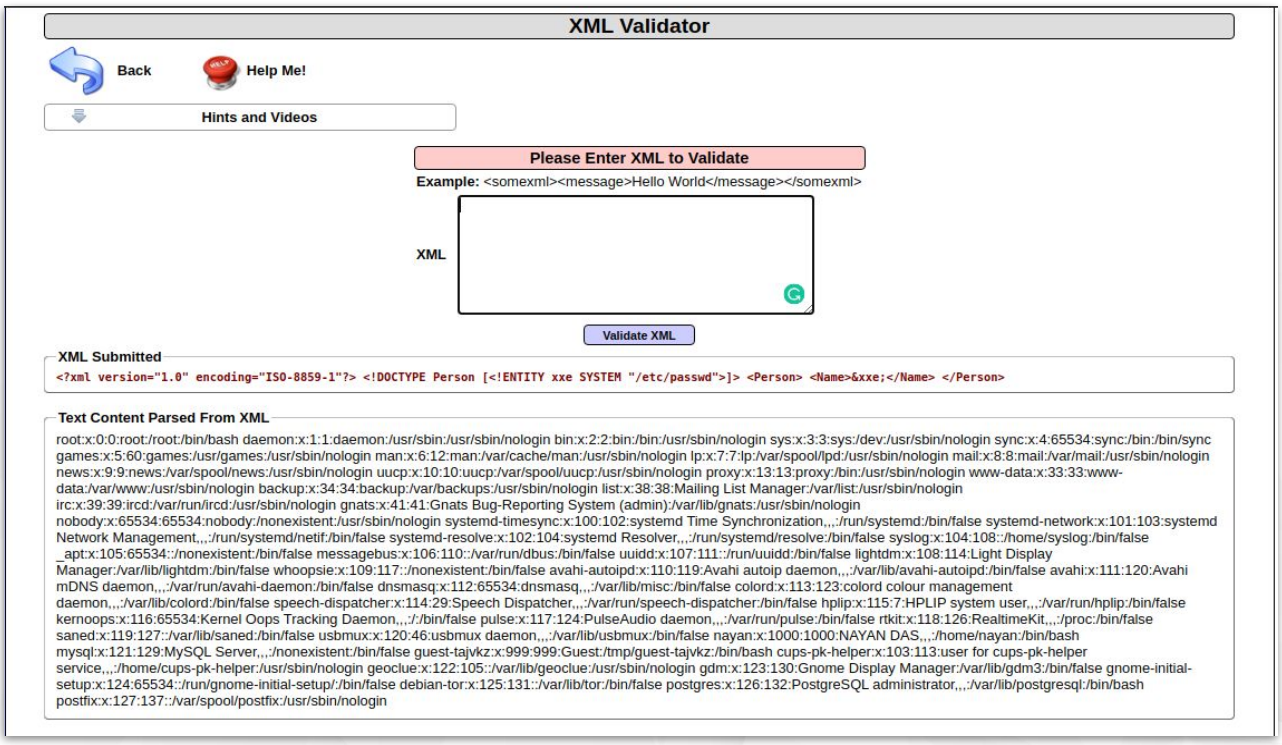
We can look at the implementation of XXE on a test application. This application allows us to enter an XML code and execute it using the XML parser and get the output. On entering the sample code, we can see the output:

```
<Person>
<Name>Xyz</Name>
</Person>
```



We can create an external entity that will fetch the passwd file from the server and present us the data. For this we will enter:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Person [
  <!ENTITY xxe SYSTEM "/etc/passwd" >
]>
<Person>
<Name>&xxe;</Name>
</Person>
```

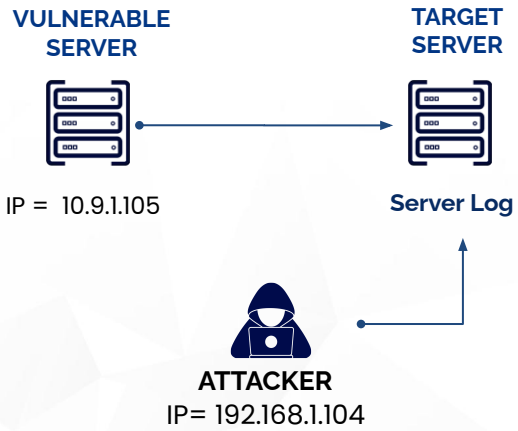


4

Understanding SSRF vulnerability

Server Side Request Forgery is a web application vulnerability allowing an attacker to send an HTTP request to an arbitrary domain. The advantage of this vulnerability is that the victim server's logs will not contain the attacker's IP address but the vulnerable web application server's IP address sending the request from the attacker's side.
This can happen when the vulnerable web application server requests external domains for users through its parameters taking external URLs.

We can understand this from the diagram:



abc.com/contactus :ip=192.168.1.104
abc.com/index.php :ip=192.168.1.104
abc.com:21/index.php :ip=10.9.1.105

In the logs of the target server, the first logs were generated by the attacker. But as the attacker found a vulnerable web application with SSRF, he started sending a request from the vulnerable machine to the target machine. The attacker's IP is now hidden.

The target parameters for this attack are uri, URL URL, file, dist, redirect, etc. If a web application is taking user input in a parameter in the form: www.abc.com/?file=page1, then an attacker can simply replace the value with a target remote domain like www.abc.com/?file=http://victim.com/

Now in the log files of victim.com, the IP address sending the request will be abc.com. And now the attacker can check for open ports, their version, initiate a DOS attack, or find files with sensitive information.

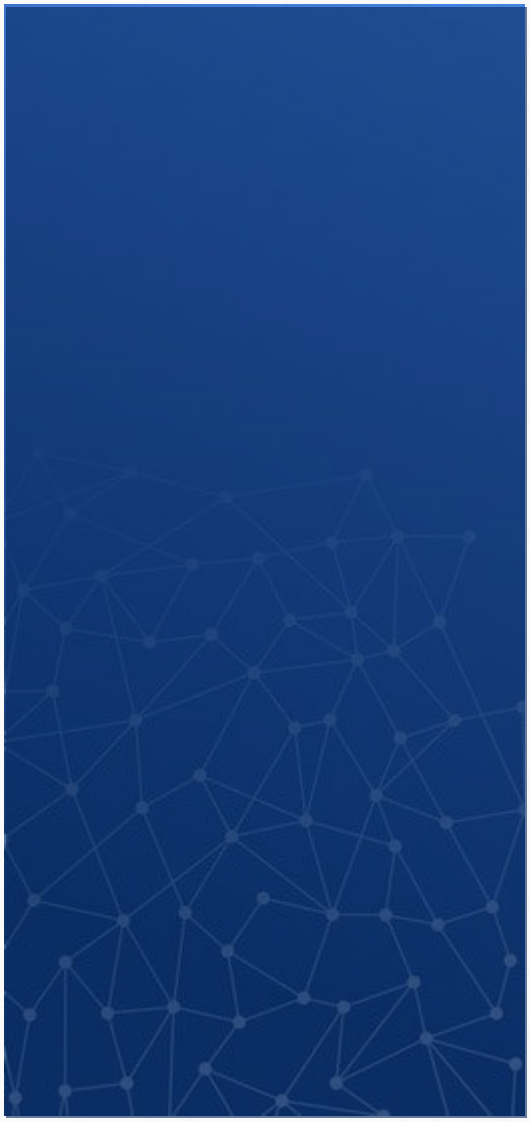
5

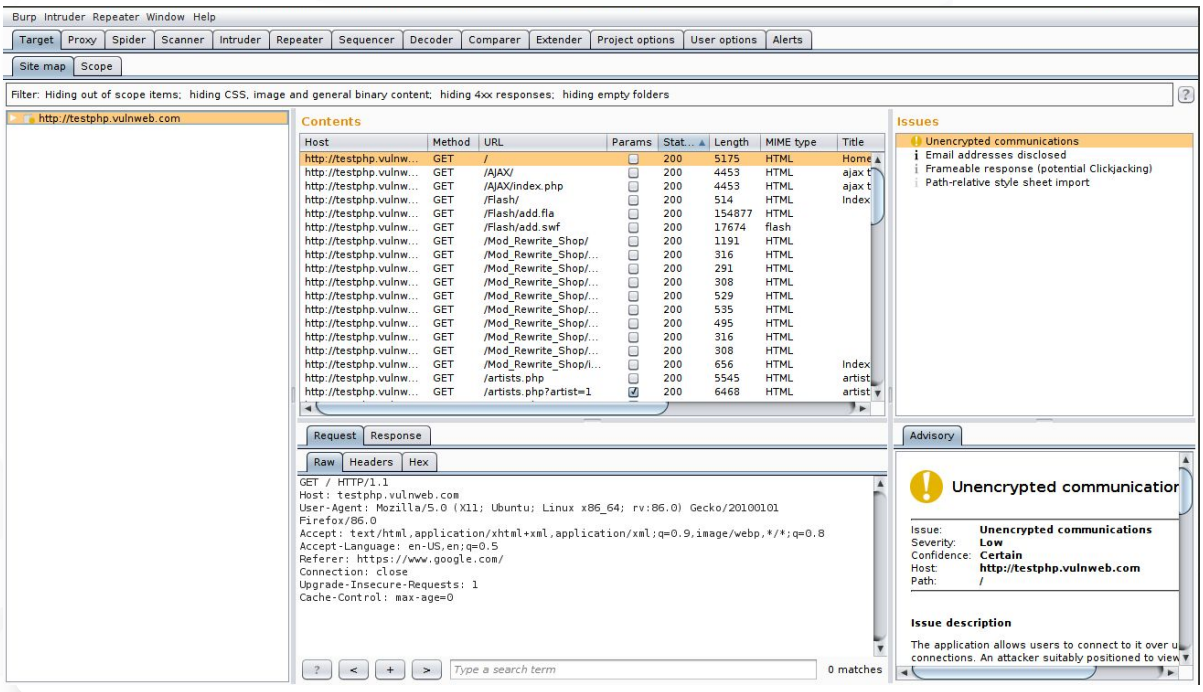
A simple example of how SSRF works

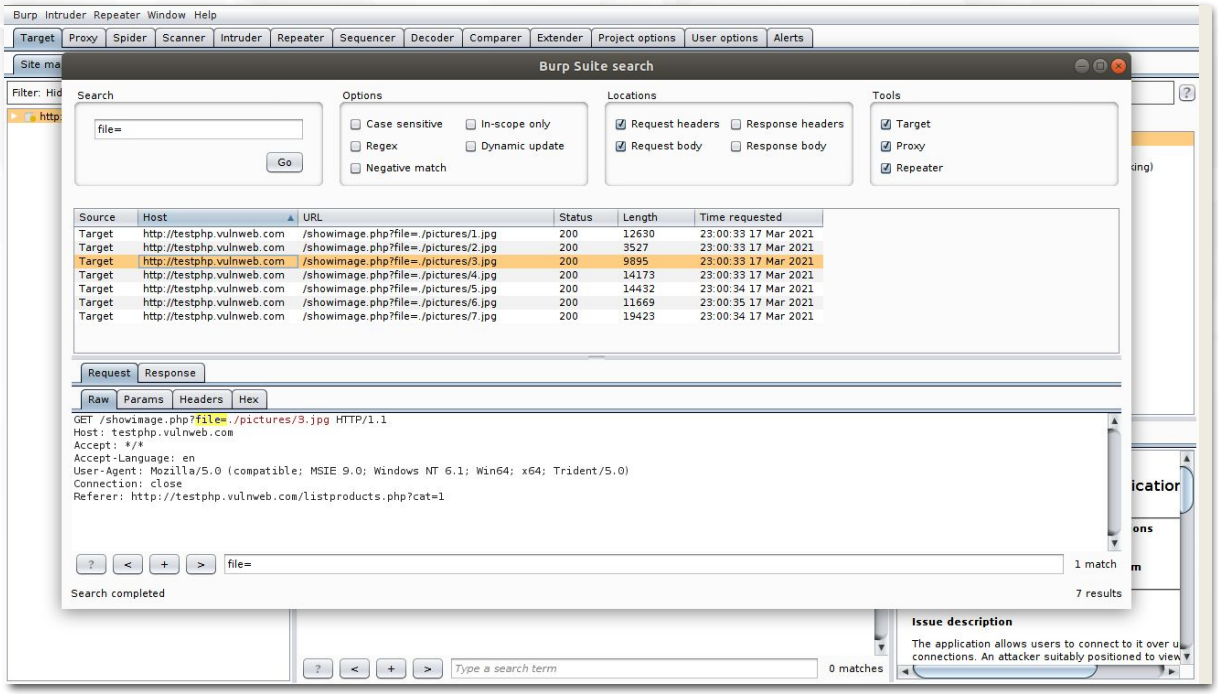
We can use a test web application to check for SSRF vulnerability. But the challenge is we do not have a secondary domain to check the logs if containing the vulnerable server's IP address.

First, let's check our IP address using an online service. Go to https://www.expressvpn.com/what-is-my-ip Here we will get our IP. But what if the vulnerable web application sends a request to this domain. The domain will give the IP address of the vulnerable web application server.

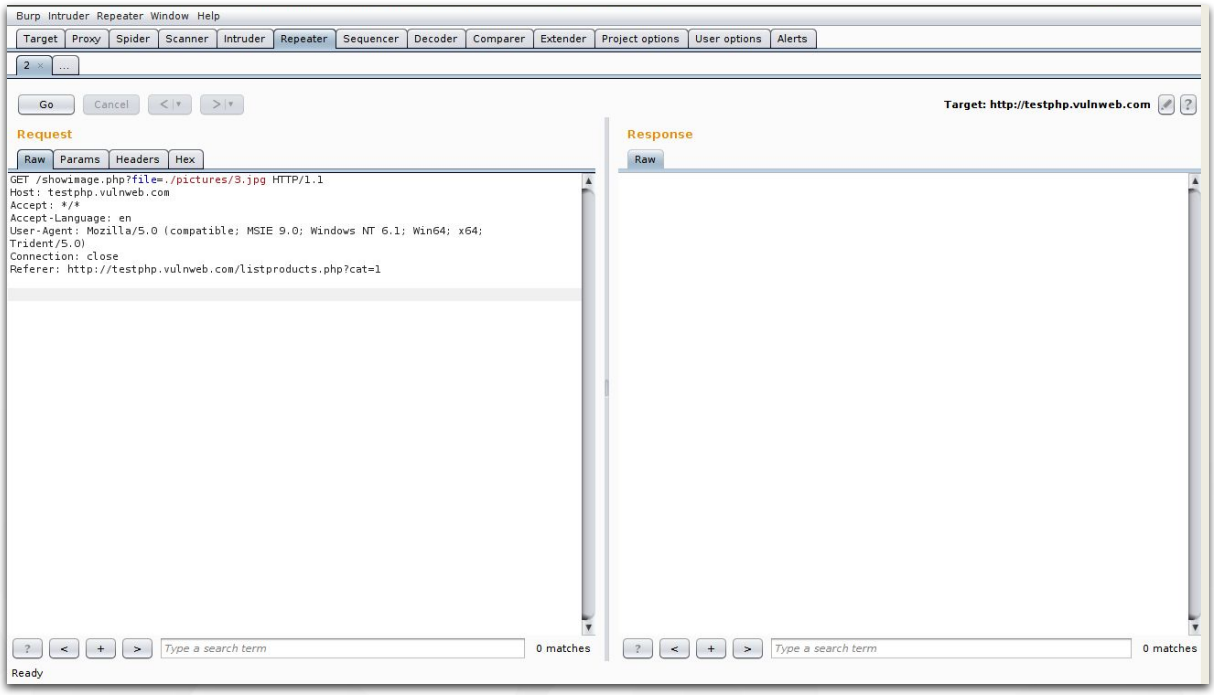
We will use Burp Suite to spider the vulnerable web application and find a parameter that will send our request to an external domain.



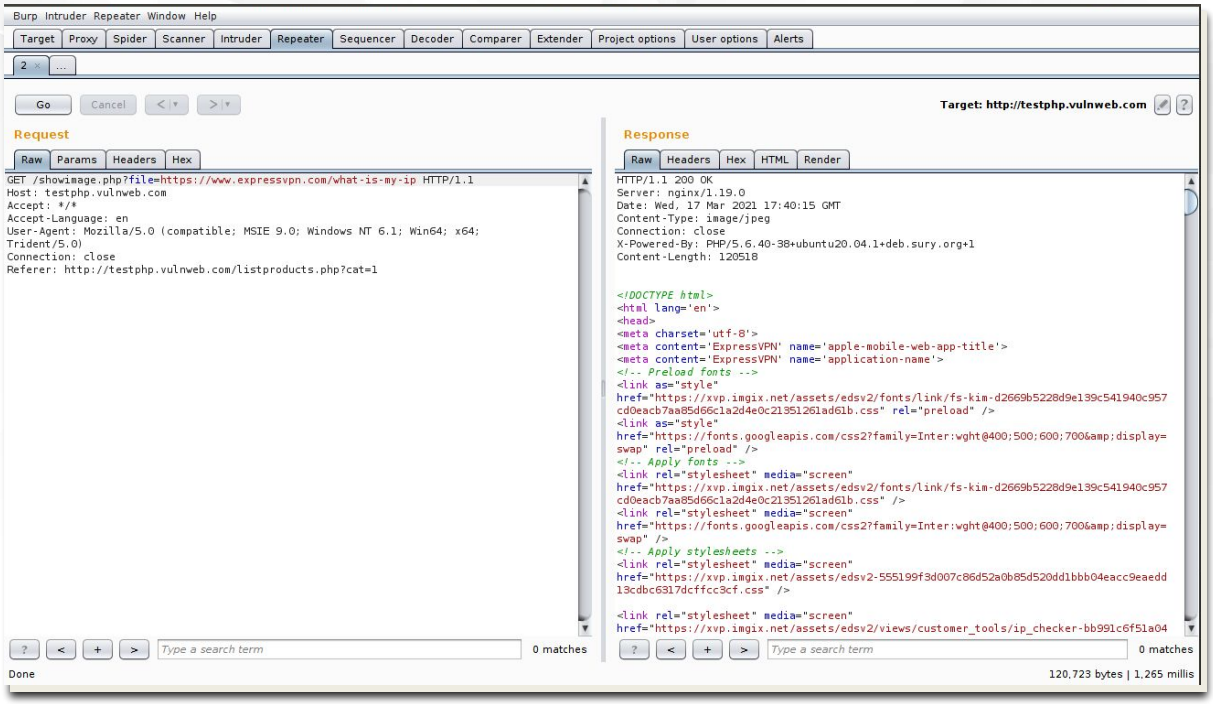




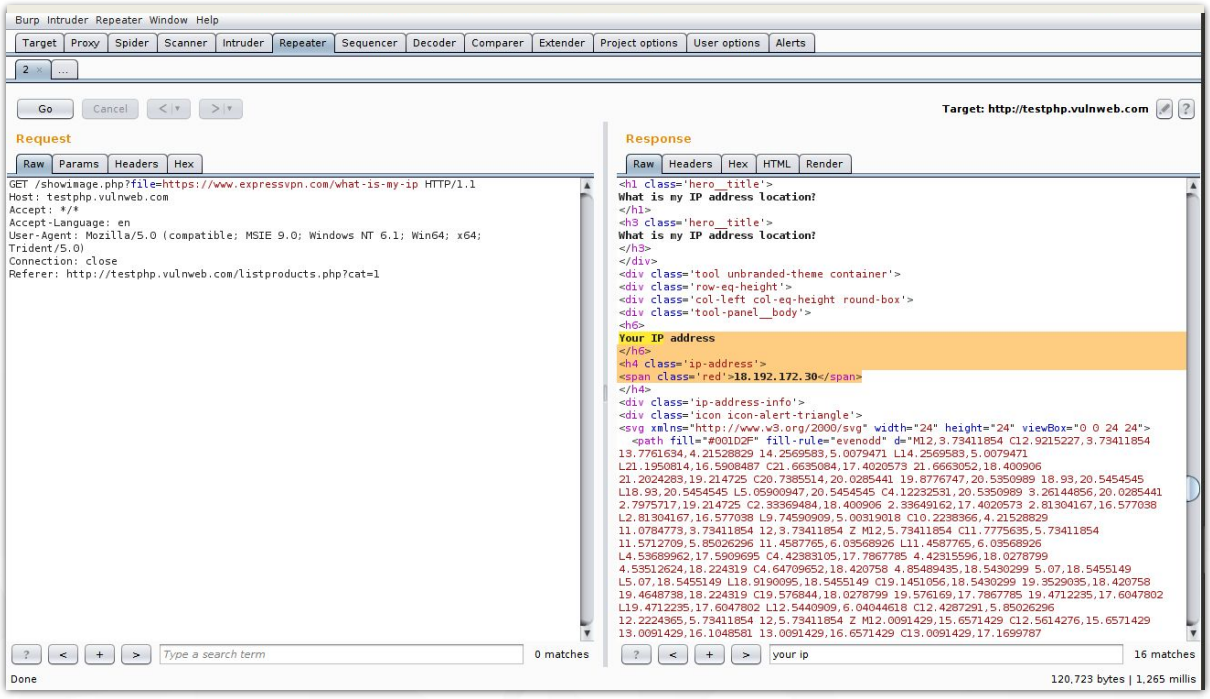
Now we can select any of the requests with the “file” parameter and send it to the repeater tab



Now we can try replacing the parameter value with the URL we used before to find our IP address.



From the response, it's clear that the request was successful. Check the response we found that the IP address was different from our IP address.



This means that the web application is vulnerable to SSRF, and if we send any request to the remote domain from here, the attacker's IP will be hidden.

Now that we know how XXE and SSRF work, we can use them together and exploit SSRF from XXE.

6

Using XXE to exploit SSRF vulnerability in a Web application

We will use a test application that checks for the number of products left in stock. The application is vulnerable to both XXE and SSRF, and to exploit SSRF, we will take advantage of XXE vulnerability.

We can capture the request in Burp Suite. The request sent is handled using XML, and the product id helps us find the stock value.

Description:

The Six Pack Beer Belt - because who wants just one beer?

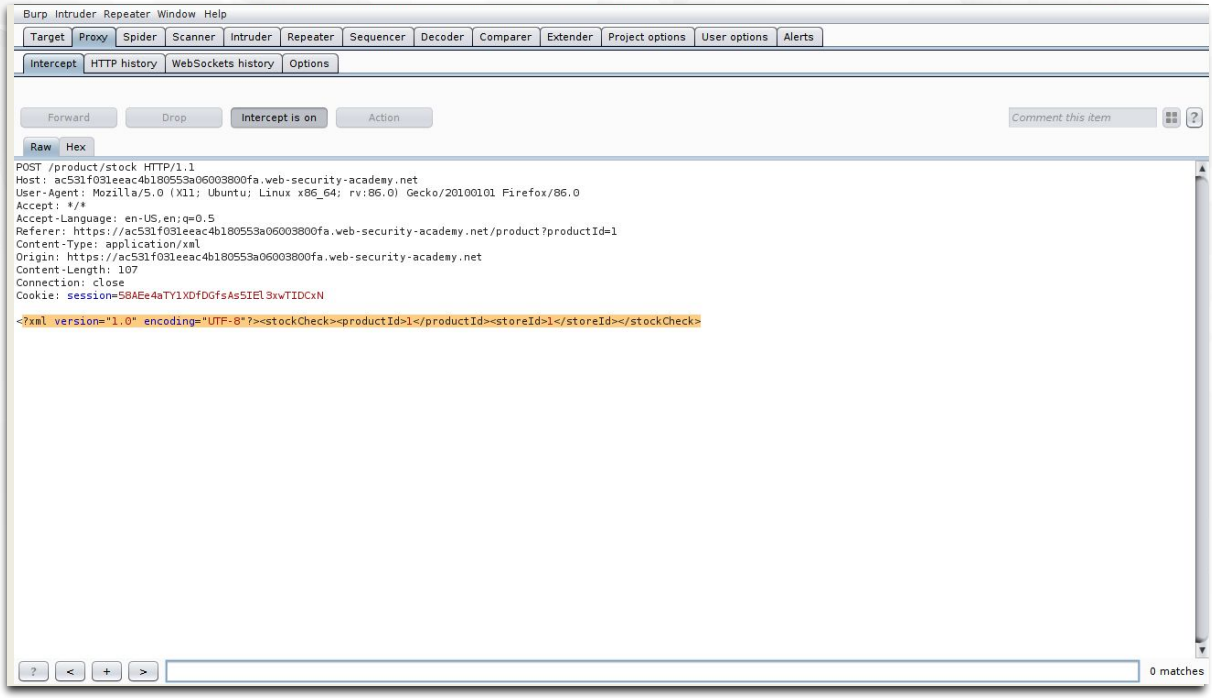
Say goodbye to long queues at the bar thanks to this handy belt. This beer belt is fully adjustable up to 50' waist, meaning you can change the size according to how much beer you're drinking. With its camouflage design, it's easy to sneak beer into gigs, parties and festivals. This is the perfect gift for a beer lover or just someone who hates paying for drinks at the bar!

Simply strap it on and load it up with your favourite beer cans or bottles and you're off! Thanks to this sturdy design, you'll always be able to boast about having a six pack. Buy this adjustable belt today and never go thirsty again!

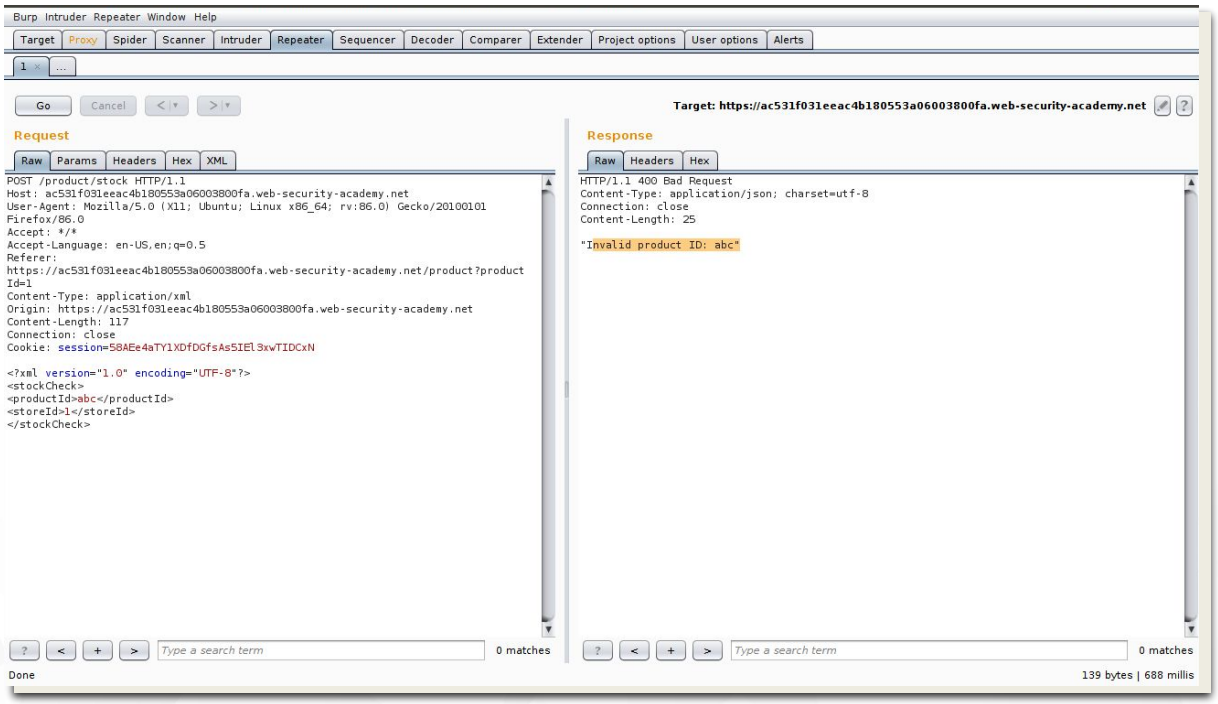
London Check stock

933 units

< Return to list



So we will send this request to the repeater and check if altering the product id reflects in the response and then start creating an external entity.



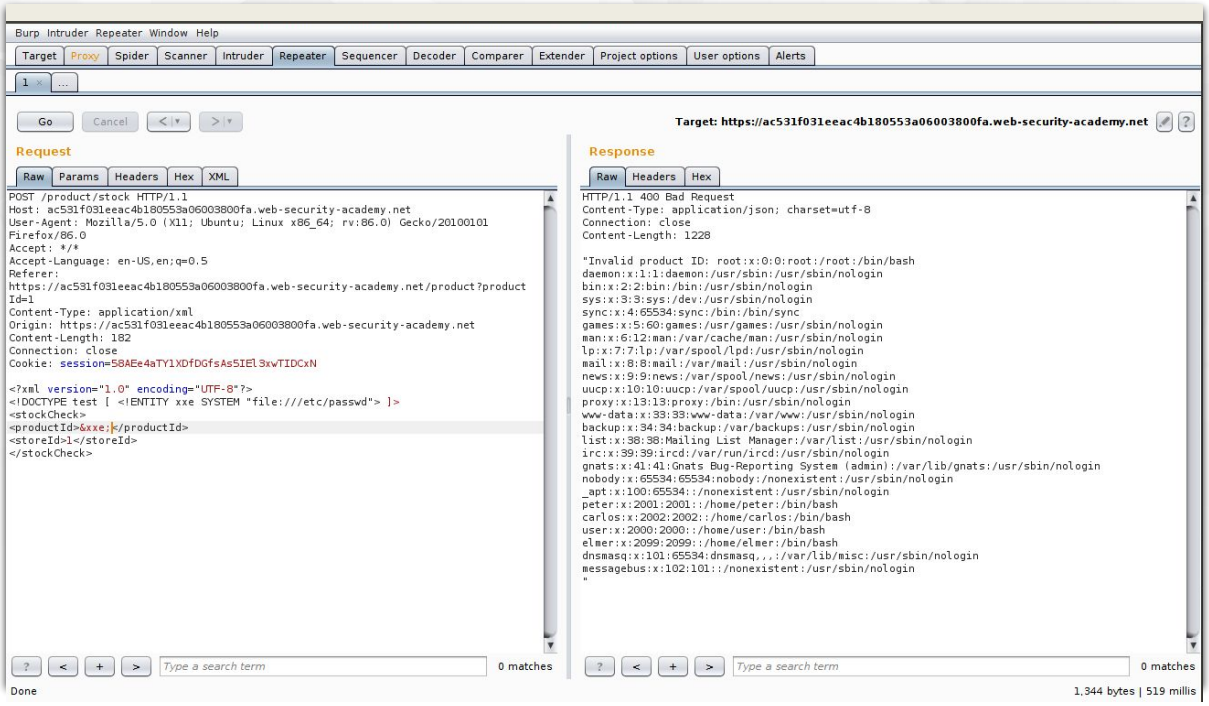
As the change in value is now visible, we can create an external entity and fetch a local file like a password file.

For this, we added a line:

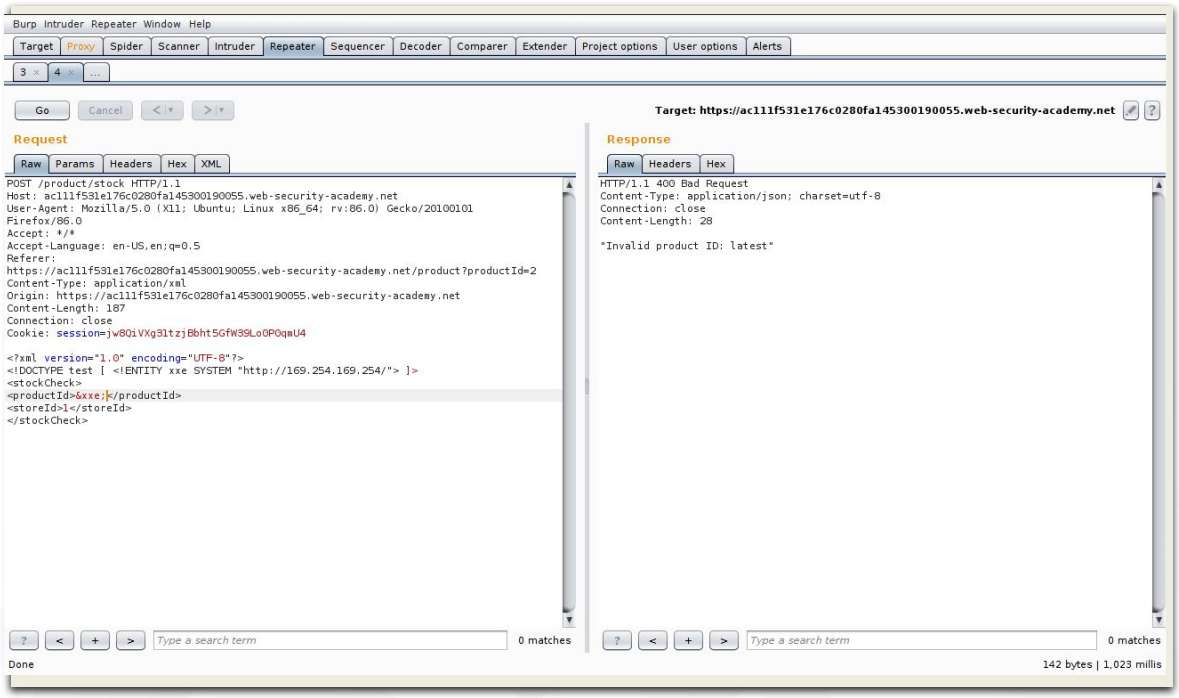
```
<!DOCTYPE test [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
```

Now we used the “xxe” external entity at the place of product id and checked for the response.

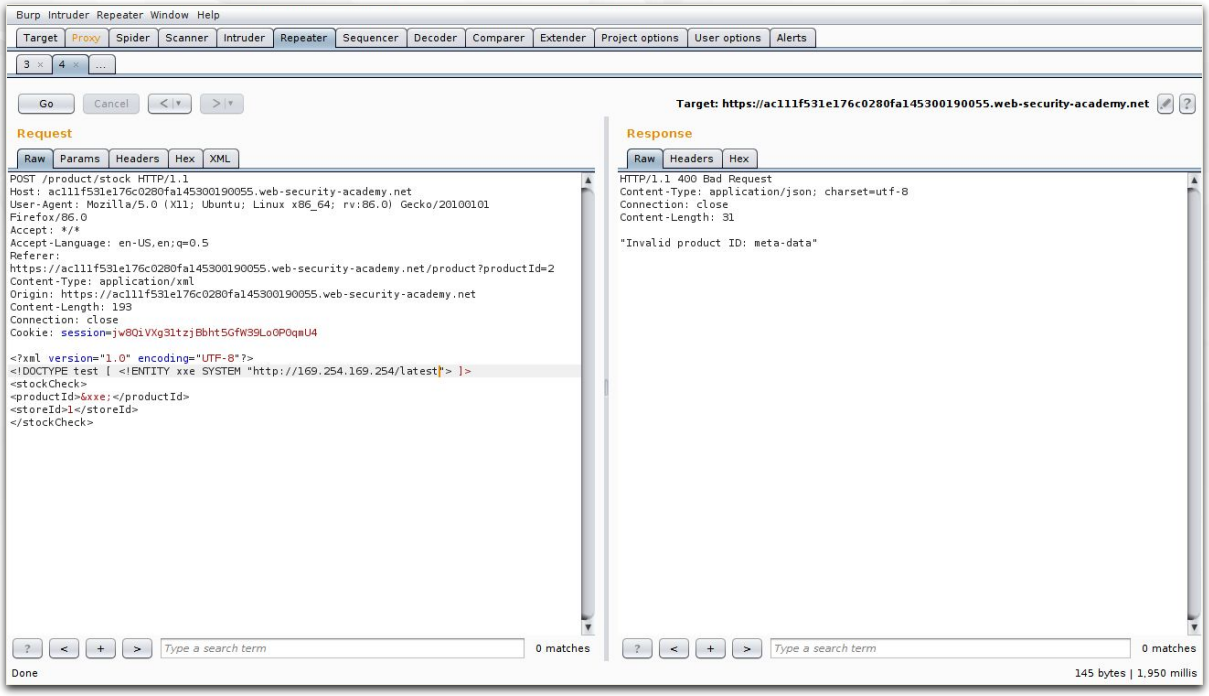
.



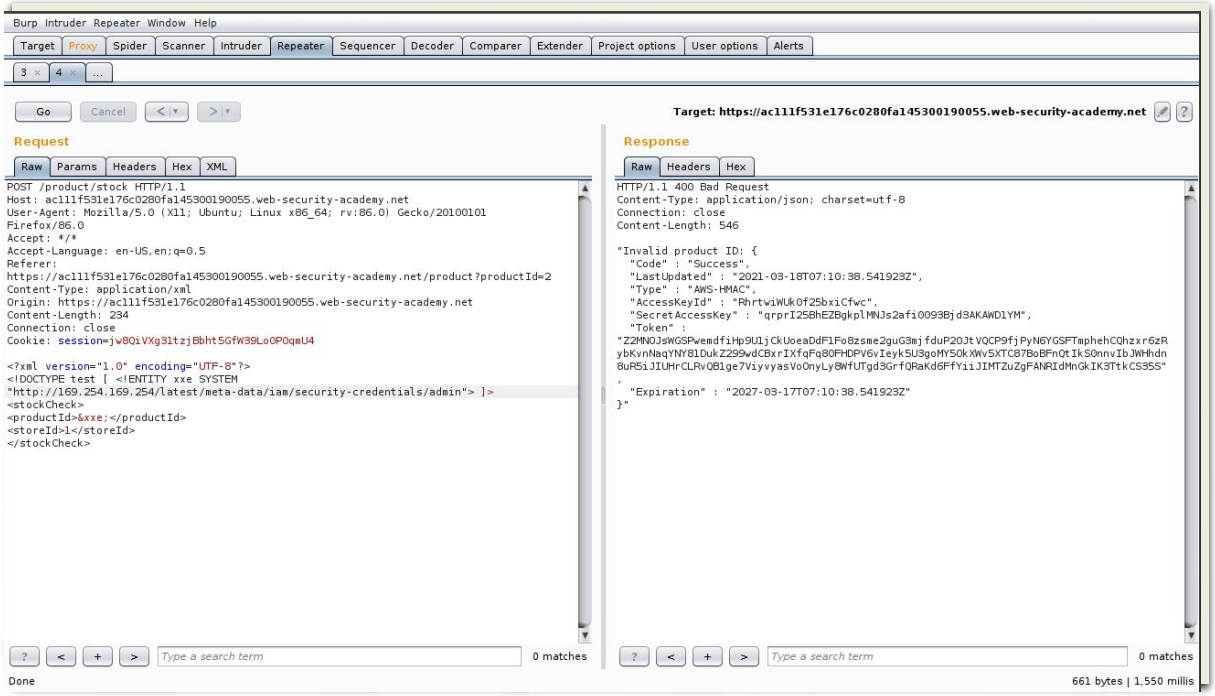
Now that the value is visible, we can perform SSRF from the same product id parameter by creating an external entity that requests a target domain. The domain is at http://169.254.169.254/, and we will again use the same steps but replace the local file with the target URL and check for a response.



We can see that we got an error and received a file name “latest” in the response. Let’s use this file name in our given URL and send the request again.



We again received a path. Let's keep adding the file names till we find some information of our interest.



We finally reached the admin page with all the sensitive information. In this way, we could send requests from vulnerable web application servers to the target server and perform SSRF using the XXE vulnerability.

www.safe.security | info@safe.security | +91 11 2632-2632