# Objective: Recover the Web Ring—Answers Only

## Naughty IP

### Answer

Open the packet capture file in Wireshark and select Statistics, Endpoints  
Graphical user interface, text, application

Description automatically generated with medium confidence

Select the IP tab and click on the Bytes column until is sorted with the biggest numbers at the top.  
Graphical user interface, application, table, Excel

Description automatically generated

The number two address is 18.222.86.32, which is the attacker according to the hint.

The trick with tshark is to read the pcap file and then follow the same sort of steps you would in Wireshark. In this case, we looked at statistics for endpoints. A search of the [tshark man pages](https://www.wireshark.org/docs/man-pages/tshark.html) for endpoints gives us this.  
Graphical user interface, text, application, email

Description automatically generated  
   
<snip>  
Text

Description automatically generated

Same answer as before!

## Credential Mining

### Answer

We know that the attacker’s IP address is 18.222.86.32, and that we are looking for some sort of HTTP request. I use those two filters often so it is easy to construct a display filter for them.  
ip.addr==18.222.86.32 && http.request  
Graphical user interface, application, table, Excel

Description automatically generated

The string of POSTs to /login.html probably marks the start of the brute force attack. We can investigate further by looking specifically at /login.html. Find login.html in the request body, right-click, select Prepare as a Filter -> Selected, and we get a filter, http.request.uri == "/login.html".

Graphical user interface, text, application, Word, email

Description automatically generated

That give us this filter:  
http.request.uri == "/login.html" && ip.addr == 18.222.86.32

Lots of interesting data there.  
Graphical user interface, text, application, email

Description automatically generated

The first POST has the username alice and password phillip, and that is the answer to the objective.

For fun, we can extract the usernames and passwords attempted in the attack using tshark.  
tshark -r victim.pcap -Y 'ip.addr==18.222.86.32 && http.request.method==POST && http.request.uri == "/login.html"' -T fields -e 'urlencoded-form.value' | head

The option -Y is the display filter we used to select the packets in Wireshark, with the addition of http.request.method==POST to exclude GET requests. The option -T fields tells tshark we want to output fields, and the -e option urlencoded-form.value specifies the field. I found that field name from Wireshark as before, using Prepare as a Filter -> Selected. Note that our choices are only urlencoded-form, urlencoded-form.key, and urlencoded-form.value. The fields do not distinguish between password and username, so we will get both from tshark.  
Text

Description automatically generated

Text

Description automatically generated

Now we can easily count the usernames and passwords that the attacker used (no one asked, but…)  
Text

Description automatically generated

The attacker tried 9 usernames with 101 unique passwords (field 1 is the username, 2 is the password.)

## 404 FTW

### Answer

Find an HTTP Status Code in a server reply and use the Prepare a Filter -> Selected method to create a display filter. Be sure to check for a response code of 404.  
Table

Description automatically generated

It appears the attack started with packet 23355 at time 175.372379. Any 200 OK after that, directed to the attacker’s IP 18.222.86.32, is a successful result for the attacker. The next search is for 200 OK responses to the attacker. There are a lot of OK responses before the attack, so I ruled them out with frame.number > 23355 (you could just scroll past the earlier frames, though.)  
Graphical user interface, application, table, Excel

Description automatically generated

The first OK after 23355 is 26774, shown here.  
Graphical user interface, text, application, email

Description automatically generated

Note that Wireshark kindly inserts the Request URI, <http://www.toteslegit.us/proc>. The answer for the objective is /proc, as they only want the URL path.

If you like tshark, this will build a list of all the response codes and what was requested. Then you can scroll through it to see where the attack started.  
tshark -r victim.pcap -Y 'ip.addr == 18.222.86.32 && http.response.code' -T fields -e frame.number -e http.response.code -e http.response\_for.uri  
Graphical user interface, text

Description automatically generated

You can scroll through the output and see that the attack started at frame 23355  
Text

Description automatically generated

Then you can use the same display filter as we used in Wireshark. The first 200 OK that appears is the answer we seek.  
tshark -r victim.pcap -Y 'frame.number > 23355 && ip.addr==18.222.86.32 && http.response.code==200' -T fields -e frame.number -e http.response.code -e http.response\_for.uri | head  
Text

Description automatically generated

## IMDS, XXE, and Other Abbreviations

### Answer 1

The address in the article is 169.254.169.254. It is an [APIPA address](https://study-ccna.com/apipa-automatic-private-ip-addressing/), so it is only available from within the AWS infrastructure and not from the public Internet.

Graphical user interface

Description automatically generated

This is interesting because it captures traffic between the server and its backend infrastructure as well as the attack packets. Note that attack traffic is all Content-Type: application/xml, POST

### Answer 2

In Wireshark, you can use the display filter  
ip.addr==18.222.86.32 && http.content\_type == "application/xml"  
To find the packets we want. Then you can Follow -> HTTP stream on each packet to see what the attacker did. Or you can use tshark to see them all at once. My ‘Prepare as a Filter -> Selected’ technique did not work well here. Using -e xml showed nothing but the characters, xml. I had to back up one step and use File Data to find my filter, http.file\_data  
Graphical user interface, text, application, email

Description automatically generated

tshark -r victim.pcap -Y 'ip.addr==18.222.86.32 && http.content\_type == "application/xml"' -T fields -e http.file\_data  
Text

Description automatically generated

We can get rid of noise with | cut -d '"' -f 6 (column delimiter “, field 6)Text

Description automatically generated

After a bit of reconnaissance (icanhazip.com returned the IP address of the server), the attacker walked his way down the AWS path until he found the EC2 credentials. This is from Follow -> HTTP Stream on the last packet.  
Text, letter

Description automatically generated

Oops! Keys exfiltrated! Paste this into the objective to get credit:  
<http://169.254.169.254/latest/meta-data/identity-credentials/ec2/security-credentials/ec2-instance>

This is the payload for a successful XXE attack—remember it for later.  
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE foo [

<!ELEMENT foo ANY >

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

<foo>&xxe;</foo>

## Boria Mine Door

### Answer Pin 1

In this case, a comment does give the answer. Copy @&@&&W&&W&&&& from the comment and paste it into the window.

Text

Description automatically generated

Text

Description automatically generated

### Answer Pin 2

There is a nice hint in the Pin 2 comments. It isn’t a clear giveaway like Pin 1, but it does help. It is nice to know that HTML will be unobstructed.

Graphical user interface, text, application

Description automatically generated

I started creating SVG for this challenge with an online editor but found some simple constructs that allowed me to create my SVG by hand, and more quickly. Compare this SVG  
<svg viewBox="0 0 300 300"><line x1="0" y1="100" x2="300" y2="230" stroke="white" stroke-width="40"/></svg>

with one from an online editor:  
<svg id="eCvtJ2qdyUW1" xmlns="<http://www.w3.org/2000/svg>" xmlns:xlink="<http://www.w3.org/1999/xlink>" viewBox="0 0 300 300" shape-rendering="geometricPrecision" text-rendering="geometricPrecision"><line x1="-150" y1="-70.629552" x2="150" y2="50" transform="translate(150 181.451613)" fill="#fff" stroke="#fff" stroke-width="10"/></svg>

Both work. I prefer the simple version.  
<svg viewBox="0 0 300 300">

<line x1="0" y1="100" x2="300" y2="230" stroke="white" stroke-width="40"/>

</svg>

The simple SVG code sets a view box with (0, 0) in the upper left corner and (300, 300) in the bottom right. Note that this does not exactly match the size of the black image from the web page. The bottom y coordinate of the black image seems to be about 250.

### Answer Pin 3

The code for Pin 3 indicates that that JavaScript may not be filtered.  
Graphical user interface, text, application

Description automatically generated

This line works for Pin 3. Apparently SVG works in both HTML and JavaScript.

<svg viewBox="0 0 300 300" >

<line x1="0" y1="140" x2="300" y2="30" stroke="blue" stroke-width="40"/>

</svg>

### Answer Pin 4

For Pin 4, the Content-Security-Policy is gone, and the developer has written their own code for filtering. It removes quotes and angle brackets using the JavaScript [string replace method](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/replace). Graphical user interface, text, application

Description automatically generated

This comment from the documentation is especially useful.  


This developer did not use the g flag, so the method only replaces the first character it finds, and subsequent characters pass through unaltered. So, we can add <>” to the beginning of our SVG and evade the filter.

<>"<svg viewBox="0 0 300 300">

<line x1="0" y1="60" x2="300" y2="60" stroke="white" stroke-width="40"/>

<line x1="0" y1="200" x2="300" y2="200" stroke="blue" stroke-width="40"/>

</svg>

### Pin 5 Answer

Here is the DOM code for Pin 5, taken from the Elements column of Chrome dev tools.  
Graphical user interface, text, application

Description automatically generated

The sanitizeInput() function does the filtering and is difficult to evade because it uses the g, or global, flag to replace all occurrences of the characters. However, the HTML element that calls the sanitizeInput() function is onblur. The onblur event occurs when the object loses the focus; in our case when the mouse pointer leaves the text field to click on the Go button. See [this link](https://www.geeksforgeeks.org/html-dom-onblur-event/) for a nice demonstration.

Since the filtering occurs when the mouse moves out of the text field, we can avoid that by pressing Enter instead of using the mouse to click Go. This method would have worked for Pin 4 as well.

Do not click Go, instead press Enter after pasting this.  
<svg viewBox="0 0 300 300">

<line x1="0" y1="200" x2="300" y2="60" stroke="red" stroke-width="40"/>

<line x1="50" y1="250" x2="300" y2="140" stroke="blue" stroke-width="40"/>

</svg>

### Pin 6 Answer

I could not find any client-side filtering. If there is server-side filtering, it does not object to SVG, which seems to bypass a lot of the protections the challenge designer implemented.

Graphical user interface, text, application

Description automatically generated

This code will draw the three lines we need.  
<svg viewBox="0 0 300 300">  
<line x1="0" y1="50" x2="300" y2="50" stroke="#00ff00" stroke-width="40"/>  
<line x1="0" y1="120" x2="300" y2="160" stroke="red" stroke-width="40"/>  
<line x1="0" y1="180" x2="250" y2="270" stroke="blue" stroke-width="40"/></svg>

## Glamtariel’s Fountain

### Answer 1: Catch a Drop

The browser sends a POST request to /dropped of the fountain site.  
Graphical user interface, application

Description automatically generated

The POST request has a header, Content-Type: application/json, and the payload is indeed JSON,  
{"imgDrop":"img2","who":"princess","reqType":"json"}.  
It may seem strange that we are going to use an attack, XXE Injection, that requires XML when the site uses JSON, but hang in there. Maybe they forgot to turn XML off. The payload tells us which icon was dropped (img2), who received it (princess), and that the request type is JSON.

### Answer 2: Play the Game

After dragging and dropping for a while, we see the frozen eye.  
Text

Description automatically generated

We can recover the name of the image from either Burp or the browser. It is  
<https://glamtarielsfountain.com/static/images/stage2ring-eyecu_2022.png>

It appears we have started stage 2. Continue to drop until the icons change to four rings.

Text

Description automatically generated

### Answer 3: Convert to XML

Our simple JSON converts to this in XML:

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

<root>

<imgDrop>img1</imgDrop>

<who>princess</who>

<reqType>xml</reqType>

</root>

I manually changed reqType to xml. The root and /root can be named anything, but root is common.

### Answer 4: Insert XML

First, turn on Intercept in Burp proxy. Then generate a POST to /drop by dragging an icon to the princess. The POST should appear.

Oops, caught a heartbeat. Let it go by pressing Forward.  
Graphical user interface, text, application

Description automatically generated

Here is the original packet, caught in the proxy.  
Graphical user interface, text, application, email

Description automatically generated

Now, change Content-Type to application/xml, and change the JSON payload to our XML version.  
Graphical user interface, text, application, email

Description automatically generated

Now click Forward and see if the server accepts it. Once you’ve clicked Forward, turn Intercept off so the normal traffic can resume. If you forget to turn Intercept Off, it will appear that your browser is broken (a disadvantage of proxy.) If the XML is accepted, you will see this when you return to Target.  
Graphical user interface, text, application

Description automatically generated

A picture containing text

Description automatically generated

If you see anything else, the XML was probably refused. If the site talks about TAMPER, you will probably have to use the Reset button on the site and do all the draggy/droppy stuff again. Sorry.

### Answer 5: Guess the PATH

The hint starting with “Be careful” has the characters ROOT, PATH, and APP, in upper case. This could be interpreted as “the root of the path is app.” So, we have <file:///app>.

The hint, “I like to keep my things with my images,” tells us the middle of the path. If you look on the Target tab in Burp, you will see the images are kept in static/images. <file:///app/static/images>.

Finally, RINGLIST and SIMPLE FORMAT give a clue about the name of the file. We are expecting a text file, so ringlist.txt. The path is <file:///app/static/images/ringlist.txt>. Easy, yes?

: application/xml and the XXE in the payload.

### Answer 6: XXE with Repeater

The changed packet should look like this.  
Graphical user interface, text, application, email

Description automatically generated

The response should be:  
Graphical user interface, text, application, email

Description automatically generated

Note that Glamtariel sent us a link to an image, static/images/pholder-morethantopsupersecret63842.png. You can recover that from the browser.  
Graphical user interface

Description automatically generated with low confidence

### Answer 7: More Hoops

The bluering.txt file does not help much. Graphical user interface

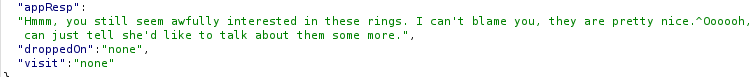
Description automatically generated with medium confidence

A picture containing text

Description automatically generated

The redring.txt file is not helpful either.  
Graphical user interface

Description automatically generated with low confidence



Maybe silvering.txt?  
A picture containing text

Description automatically generated

A picture containing text

Description automatically generated

Let’s look at the image static/images/x\_phial\_pholder\_2022/redring-supersupersecret928164.png.  
A picture containing graphical user interface

Description automatically generated

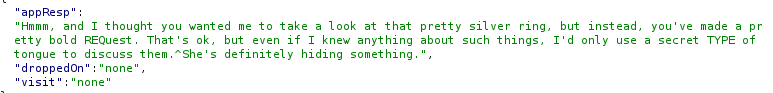
Note the text (defacement) inside the ring.   


It says, goldring\_to\_be\_deleted.txt.

### Answer 8: Yet Another Hoop

Look for the new file.  
A picture containing text

Description automatically generated



Another vague clue.

### Answer 9: The Last Hoop

My intent with the last clue was to tell the player to move their &xxe; out of imgDrop and into the reqType field. We cannot leave the imgDrop field empty, so let us put the filename for a generic silver ring, img1, into it.

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

At last. static/images/x\_phial\_pholder\_2022/goldring-morethansupertopsecret76394734.png Graphical user interface, application, icon

Description automatically generated

Put this into the objective:  
goldring-morethansupertopsecret76394734.png

Whew!

A picture containing text, device, meter, gauge

Description automatically generated