H1-212 CTF Write-Up

By: jackds (Daniel Bakker)

Introduction

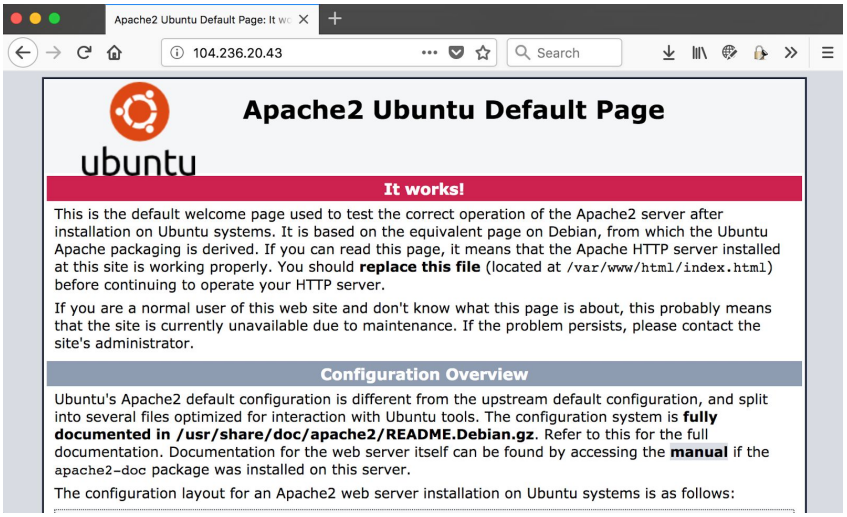
Hello everyone and welcome to my very first write-up of my very first CTF challenge ever. In November 2017 HackerOne organised a high-stakes “capture-the-flag” challenge going by the name H1-212. The prize? An all expenses paid trip to New York City to hack against HackerOne 1337 and a chance to earn up to $100,000 in bounties! That immediately caught my attention.

In the following chapters I will describe my adventure that eventually led me to capturing the flag! It all started with the following small piece of information:

> *An engineer of acme.org launched a new server for a new admin panel at http://104.236.20.43/. He is completely confident that the server can’t be hacked. He added a tripwire that notifies him when the flag file is read. He also noticed that the default Apache page is still there, but according to him that’s intentional and doesn’t hurt anyone. Your goal? Read the flag!*

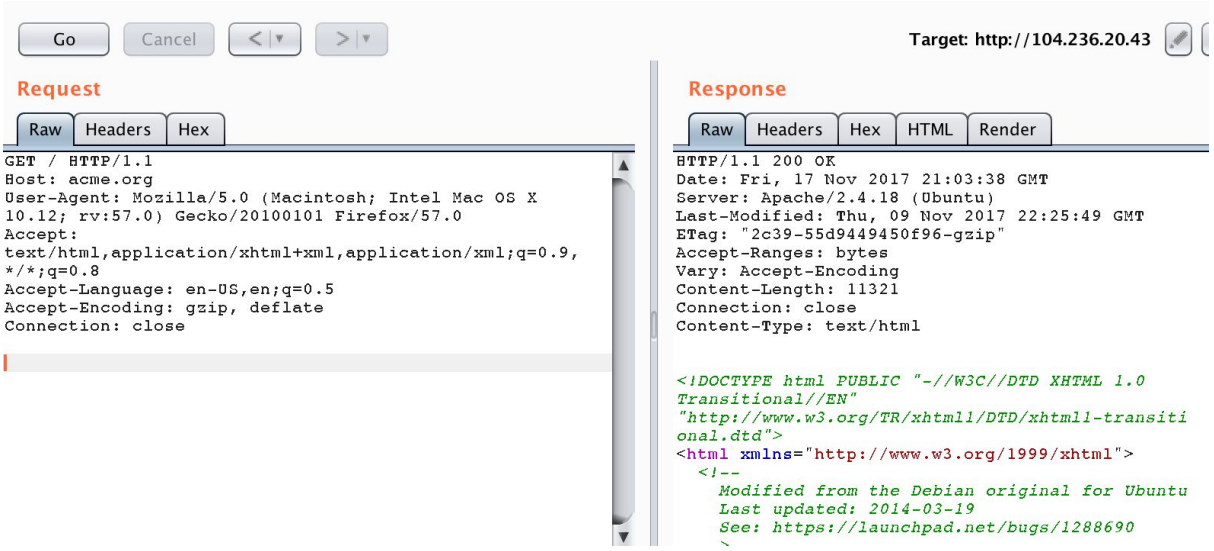
Getting started

I started off by reading the description a few times very carefully. I wrote down the following 3 things which I think were important: acme.org, admin panel and 104.236.20.43. OK, time to see what’s being served on that server:

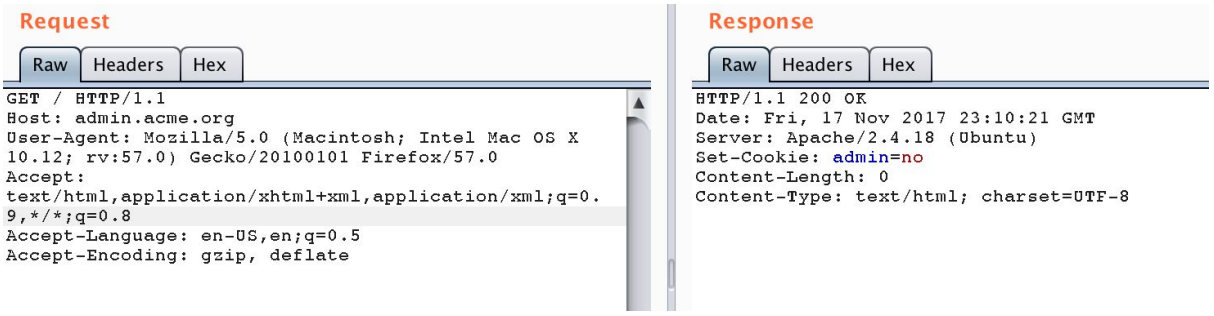


After finding out that there was nothing useful to see I thought it would be a good idea to use “dirsearch” [1] to do some initial discovery. In parallel I also started an nmap [2] session to do a quick port-scan. As expected this didn’t reveal anything useful again…

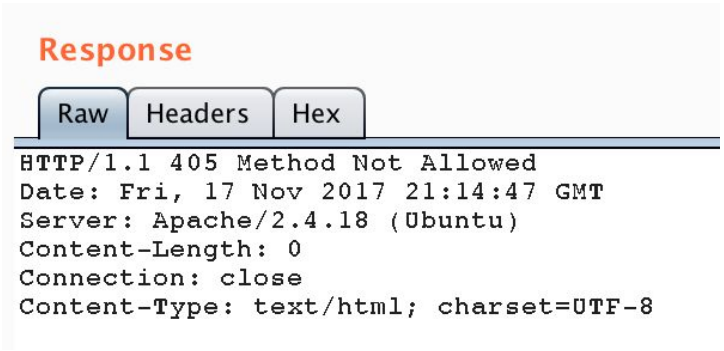
Looking at my notes I asked myself why we were given an IP-address while also a specific domain/host is mentioned. Let’s see if there is any acme.org host configured on that server as well. I started Burp [3] and replayed my request to 104.236.20.43 and added the “Host: acme.org" header:



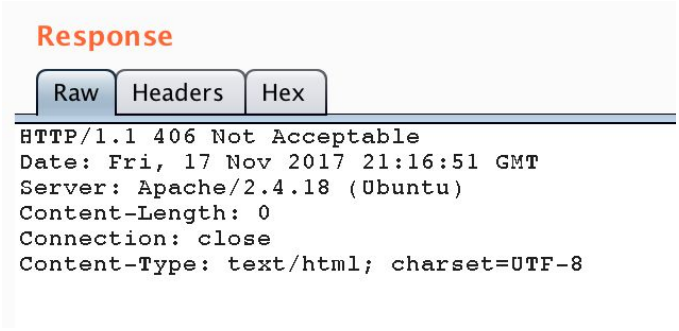
Well apparently that works, but it returns exactly the same page as the IP-address itself. So a dead-end. My next guess turned out more promising: adding admin.acme.org as header:



Finally! My first breakthrough, so let’s see what we can learn from this. As you can see the server responded with an empty response, but at least it sets a “admin=no” cookie as well. Interesting! My next obvious attempt was setting the cookie to “yes”:



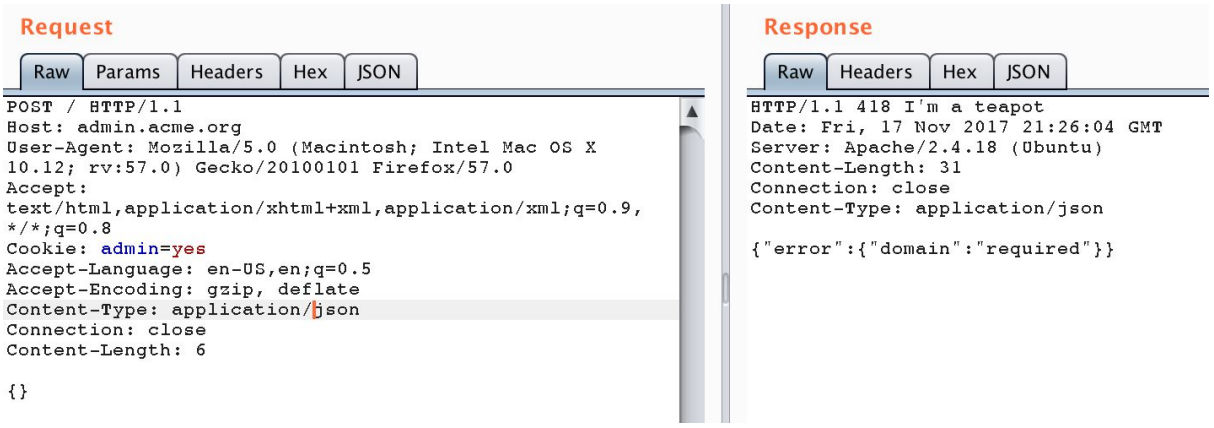
“Method not allowed” it responded. This is typically sent by web-servers to indicate that the requested method (GET in this case) cannot be used. So, let’s switch to POST instead:



“Not Acceptable” this time. So it looks like we’re heading somewhere now. I quickly googled to see what this exact status code meant and found the following “definition”: *The HTTP 406 Not Acceptable client error response code indicates that a response matching the list of acceptable values defined in Accept-Charset and Accept-Language cannot be served.*

Without thinking any further I quickly started to change the mentioned request headers. This turned out to be a waste to time, and a clear signal that I wasn’t thinking clearly anymore. Time for some sleep.

The next day I immediately knew what to do: a POST request requires data/payload and a content-type. So I tried sending an empty JSON payload as first attempt:



Bingo. Another step forward. The response I got back now gave me 2 directions: one was the “418 I'm a teapot” status and the other one was the response data itself. For a second I thought that it would be a good idea to focus on the “teapot” thing, but (luckily as it turned out) I decided to go with the payload first. As the server nicely told me what data was missing I simply added that to my request. My payload now looked like this:

```
{“domain”:”admin.acme.org"}
```

The result I got back was:

```
{"error":{"domain":"incorrect value, .com domain expected"}}
```

OK, so let’s try passing a different domain then:

```
{“domain”:”www.google.com”}
```

But unfortunately (and again a little expected):

```
{"error":{"domain":"incorrect value, sub domain should
contain 212”}}
```

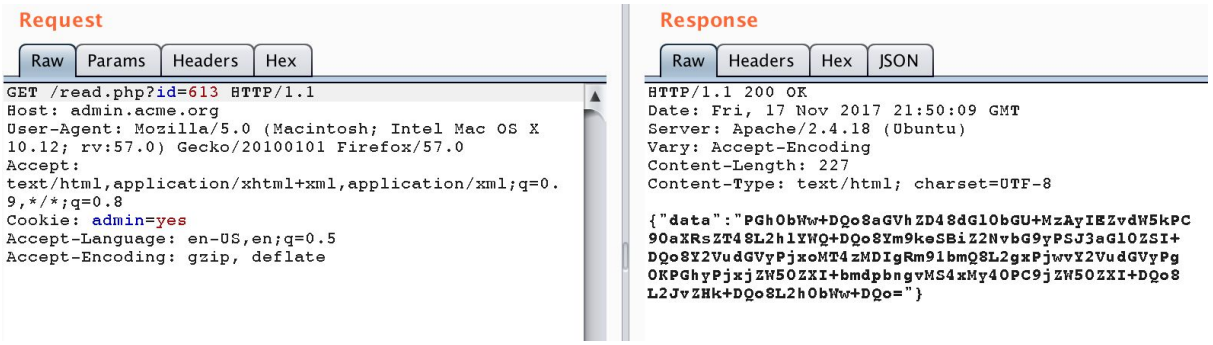
Next try:

```
{“domain”:”212.test.com”}
```

Whohoo! Different result this time and another new endpoint:

```
{“next":"\/read.php?id=0"}
```

But then I suddenly realized it was already way past midnight and since I needed to get up early the next morning I decided to call it a day… The next day, after returning from work, I quickly went back to the last endpoint I got in my last attempt and found another promising result:



As shown above the read.php endpoint returned some kind of base64-encoded payload. Decoding it gave me the following result:

```
<html>
<head><title>302 Found</title></head>
<body bgcolor="white">
<center><h1>302 Found</h1></center>
<hr><center>nginx/1.13.4</center>
</body>
</html>
```

Hmm not very useful I thought, so I started tampering with the “id” parameter from the read.php endpoint. Unfortunately all I got was one of the following errors “incorrect row” or “incorrect type, number expected”. It seemed I was again way to focussed on trying to get around this, because it took me busy for the rest of the night.

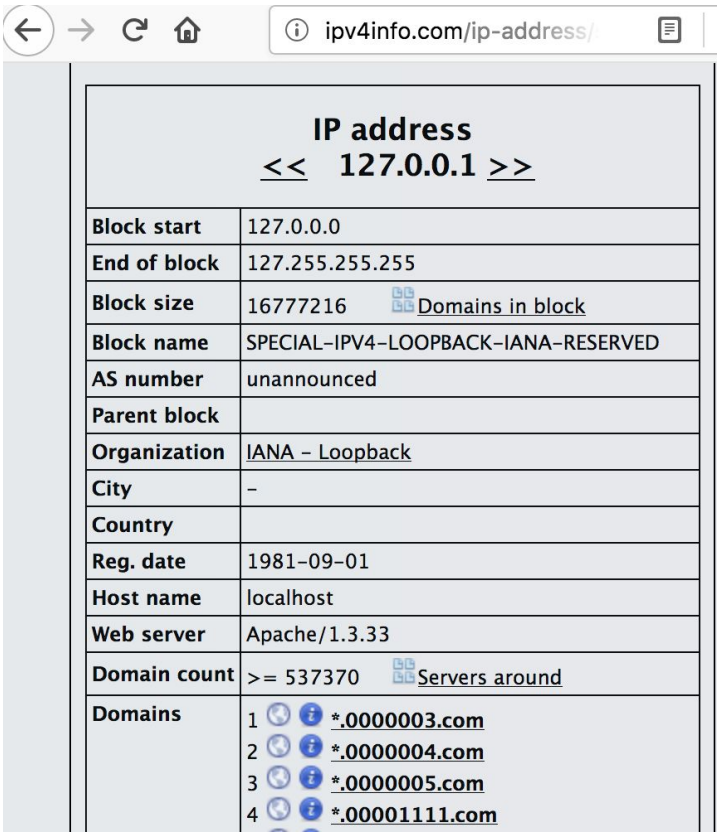
With the deadline of challenge getting closer and closer I decided to take a step back the next day and return to the POST request. I quickly realised that the “id” number in the read.php response was increasing every time I sent a valid POST request. So I figured that the payload I was getting back must have something to do with the “domain” I was posting. To confirm this I used curl to do a similar request to 212.test.com:

```
curl http://212.test.com

<html>
<head><title>302 Found</title></head>
<body bgcolor="white">
<center><h1>302 Found</h1></center>
<hr><center>nginx/1.13.4</center>
</body>
</html>
```

Aha that looks familiar! Seems I was right about that. So back to the POST request again. I figured the next step in this challenge was to perform some kind of SSRF (server-side-request-forgery) request, but with some restrictions on the domain name (212.*.com). Let’s see if we can get around this…

The first thing that came to my mind was to find a domain that acts as CNAME for localhost (127.0.0.1). So I browsed to ipv4info.com and entered 127.0.0.1. This gave me a list of useful domains:



I selected the first one from the list and replayed the POST request with the following data: {“domain":"212.0000003.com"} . As expect this gave me another (incremented) ID number for the next endpoint. When I tried this I immediately noticed that the base64-encoded string was much bigger than the one I got back before. After decoding it I saw that this indeed was the contents of http://104.236.20.43/ (the Apache Default Page). So it looked like I was on the right track again.

In my discovery step (at the beginning of the CTF) I already learned that there is really nothing to be found on that host. So I figured that there had to be another open port which is only internally accessible. Unfortunately all my attempts to accomplish this were unsuccessful for the rest of the night… every time I ran into the problem of having to end the string with “.com”. I was really getting frustrated so I decided to take a break again.

The next day, 2 days before the deadline, I continued. I decided it was a good idea to first find a payload that includes a port number and request a file from a host that I know (so that I could verify the contents). After a few attempts I found the following working payload:

```
{"domain":"212.0000003.com:80/.com"}
```

After decoding the base64 response I got the following output:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL /.com was not found on this server.</p>
<hr>
<address>Apache/2.4.18 (Ubuntu) Server at 212.0000003.com Port 80</address>
</body></html>
```

So at this point I was convinced that this would lead me further. The next step was to figure out which port number I should use, so I quickly wrote a simple script that enumerated all the ports. By looking at the returned responses I was able to tell that the port I needed was 1337 (Why didn't I think of this earlier, hehe). Time to read the flag now (I thought…).

The next hurdle to take was to get rid of the ".com" extension that caused my requests to return a 404 status. I again tried a dozen of things and just as I was about to quit a fellow hacker on BugbountyWorld told me: "take a break and return to the challenge tomorrow". Best hint ever! :) This time I was still clear enough to immediately understand what he meant: apparently I need a carriage-return or a line-feed character to get it to work. In my previous attempts I already tried a few things with a %0a and a %0d character, but was unsuccessful. I realised it might also be possible by using a '\n' or '\r' in the payload.

Almost there

Happy as I was, I was really convinced now that I would be able to solve this in time. So I quickly jumped back to Burp again and tried the following payload:

```
{"domain":"212.0000003.com:1337/\n.com"}
```

I received a new ID number and requested the read.php again, but the response was different that I expected:

```
{"data":""}
```

What the ….!? Why doesn't that work was my immediate reaction. Then I looked closer at the returned ID number (I was at number >600 already!) and noticed it was increased by 2 instead of 1. Let's see what happens if I request ID-1 I though:

"Hmm, where would it be?" was the response after decoding the returned base64 string. So I figured I needed to request a specific file on that host. Obviously "flag" was my first guess, so I changed my payload to:

```
{"domain":"212.0000003.com:1337/flag\n.com"}
```

Again the ID number (luckily) increased by 2, so I requested the returned ID - 1 and got the following response (after base64 decoding):

```
FLAG:
CF,2dsV\/]fRAYQ.TDEp`w”M(%mU;p9+9FD{Z48X*Jtt{%vS($g7\S):f%=P[
Y@nka=<tqhnF<aq=K5:BC@Sb*{[%z"+@yPb/nfFna<e$hv{p8r2[vMMF52y:z
/Dh;{6
```

YEAH!!! I finally got the flag! For a second I doubted whether I should also still decrypt this piece of code, but I quickly found out that this was really the flag! **YEAH!!**

Final words

I would like to end my write-up with thanking HackerOne, especially Jobert and Ben, for building/creating this very nice CTF challenge. It took me quite some blood, sweat and tears but I really enjoyed participating in it. I also would like to thank a few fellow hackers (edoverflow and tomnomnom) for keeping me motivated during this challenge :)

Thanks all!

jackds

[1] https://github.com/maurosoria/dirsearch
[2] https://nmap.org/
[3] https://portswigger.net/burp