SANTA'S BUSINESS CARD

My writeup

<<Riiiiiing, Riiiiiing, Riiiiiing>>

I woke up with a start, the clock said 4 am and the smell of something bad was in the air, or maybe it was the smell of alcohol on my breath. "Who's the crazy one who calls in the middle of the Christmas eve's night?" I thought. I put the pillow on my head pretending to not be there. Unfortunately, that plan did not work; the phone was still ringing and my head was clanking like a bell. I finally decided to pick up the phone and I heard two young and shocked voices:

"Someone kidnapped Santa!" they screamed. I had already heard these two voices from the news, they were the two Dosis' kids that saved last Christmas. Probably they found my number in one of those ads in the newspaper; I knew that writing "Supah-Dupa-Cyber-Detective" in the title would have impressed someone!

I tried to reassure and convince them to give me more details about the fact but I couldn't understand their confused voices, maybe because my hangover that was still kicking in, that's why I decided to reach them.

I quickly took my badge and pistol off the nightstand, put my pants on, took my notebook, kissed my alpaca Robert Downey Jr. and ran out the door. What? Yeh, I own an alpaca; it's quite normal where I live.

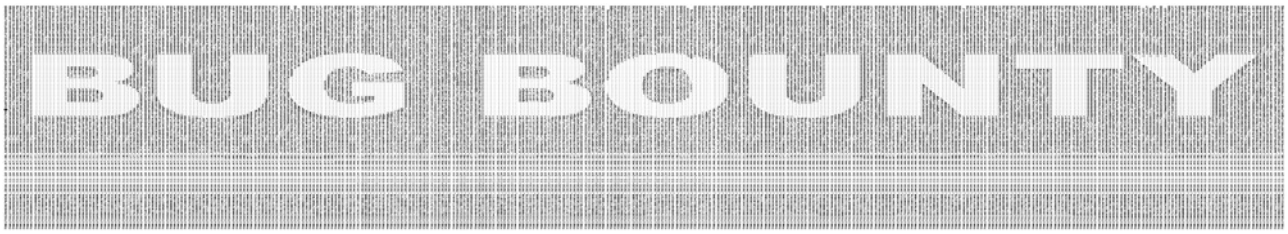
"This is going to be a long night." I thought.

When I opened the Dosis' door I realized that everything was real; something had happened to Santa. The room was a mess, the Christmas tree was broken into pieces and damaged presents had been thrown all around: someone had a bad time there. Santa's sack was lying on the floor and, like if everything wasn't serious enough, milk and cookies were still there as well. These animals didn't even let Santa get his reward!

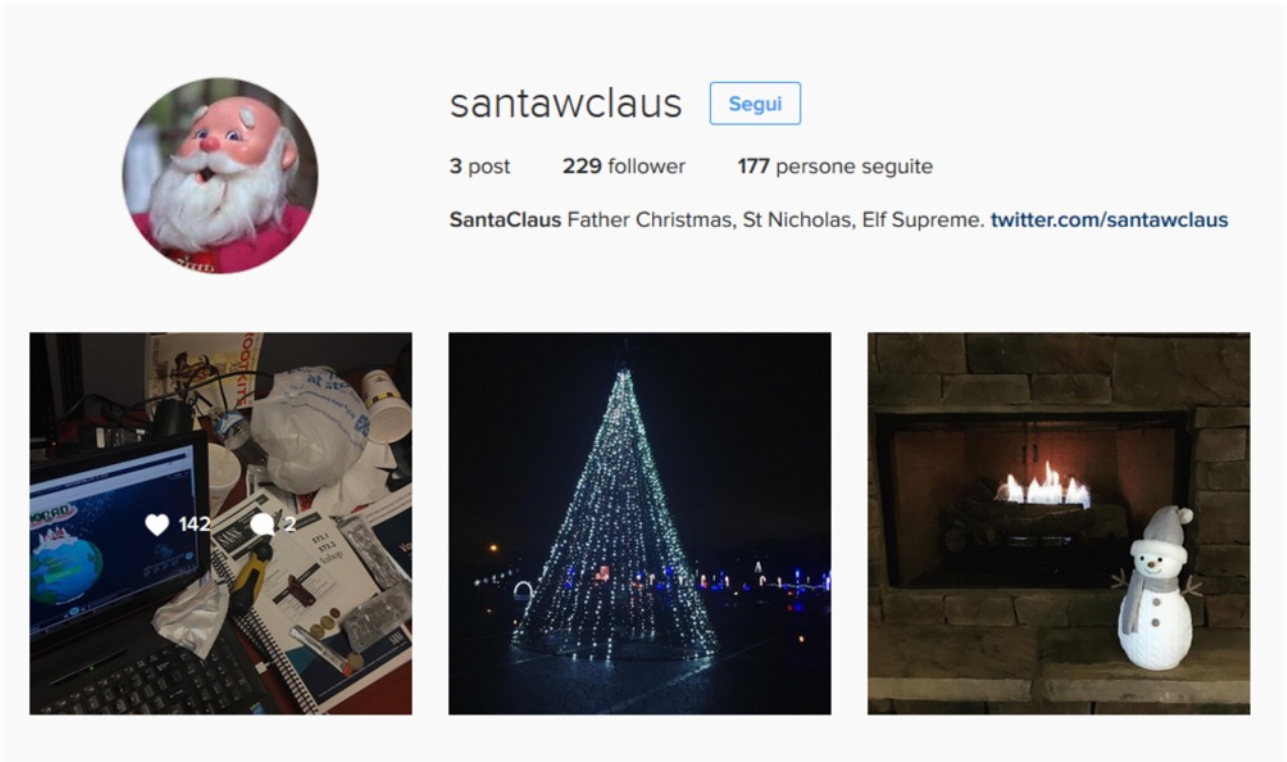
Before my arrival the kids had already searched all over the place looking for clues and Joshua, pointing a small, rectangular card, said: “Take a look at this, it may help you find out what happened!”.
The card had a subtle off-white coloring, a tasteful thickness and... Oh my God. It even had a logo on it. I hoped that psycho of Patrick Bateman had nothing to do with this affair.

I then noticed that, at the bottom of the card, there were two Santa’s account: a Twitter and an Instagram one. “Times are changing kids, now even Santa deals with Social Networks!”
By taking a look at Santa’s Twitter account I saw a huge amount of nonsense tweets, that kind of tweets that neither my alpaca could have written.
A grin appeared on my face and I immediately suspected that some sort of secret message was there.

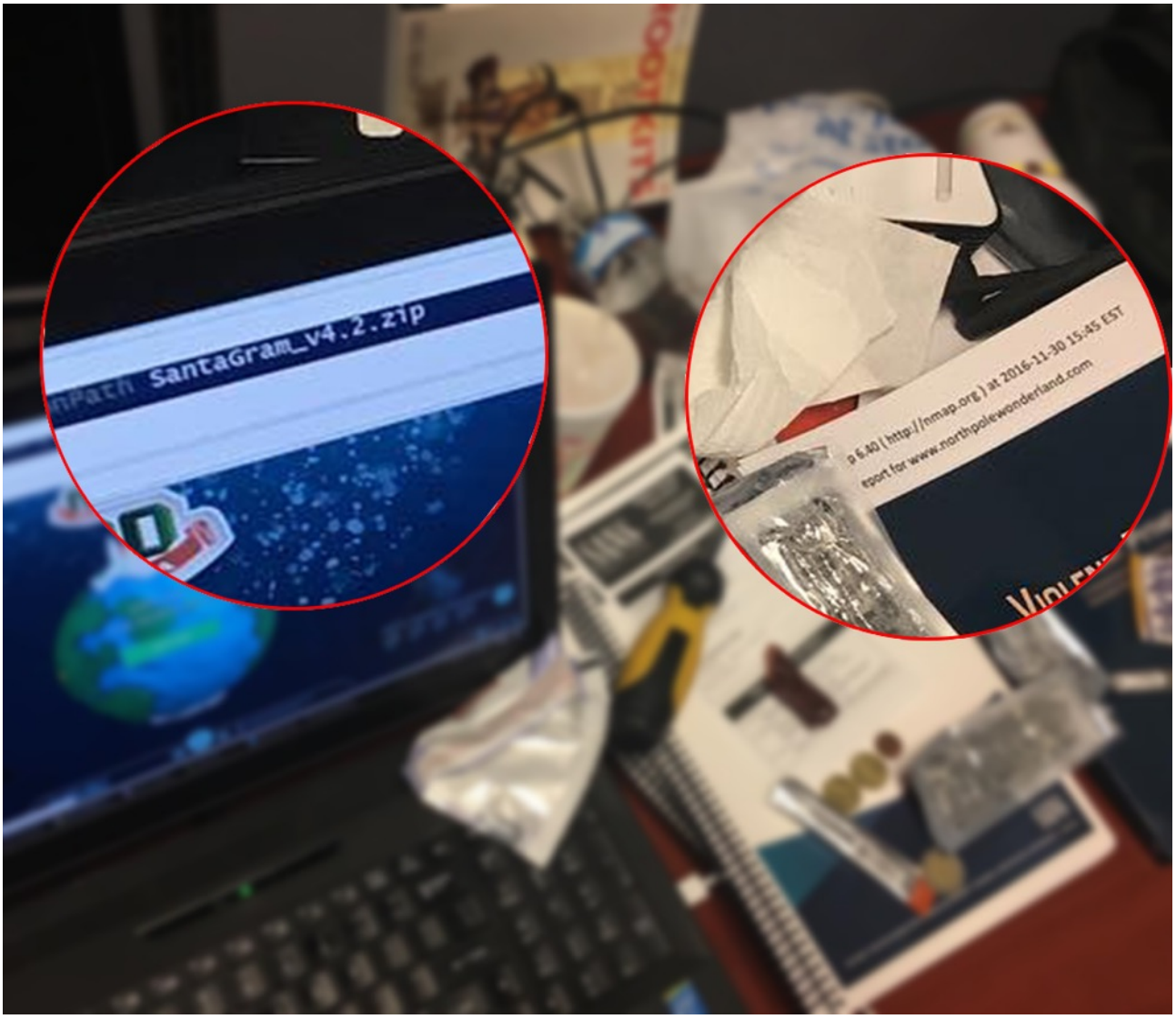
I thought that dumping all the tweets could have helped us figuring out what Santa was trying to say. With my Google-fu skills, obtained through years and years of experience in looking for solutions to problems that other people had already experienced, I found on Github an useful Python script written by Yanofsky (https://gist.github.com/yanofsky/5436496).
Putting the tweets all together, the hidden message jumped right out at me.



"Bug bounty!" I shout.
I did not feel so excited since that time in which I did a drum solo with The Animal from The Muppets Show.
"Kids, we should also check Santa's Instagram profile!" I said. Despite my fear of jumping into some duck faces and half-naked shots, we took a look at his profile.



Luckily for us, no sexy-Santa shots but an intriguing picture of a messy desk. Taking a closer look, I noticed a piece of paper with the output of an NMAP scan and the name of a ZIP archive on the notebook screen.



Putting these two information together, we became able to download the ZIP file at the following URL: *www.northpolewonderland.com/SantaGram_v4.2.zip*. We were thrilled while opening the ZIP, but a password prompt immediately brought down our excitement. "Try with the hidden message from the tweets!" Jess exclaimed.
I followed Jess' suggestion and, inserting the hidden message in the password prompt, we were able to extract the archive, discovering the Android Application Package named *SantaGram_4.2.apk.*

After collecting all these evidences, we were at a dead point again, "What can we do now?" I thought. We needed another track to follow... and an aspirin for my headache.

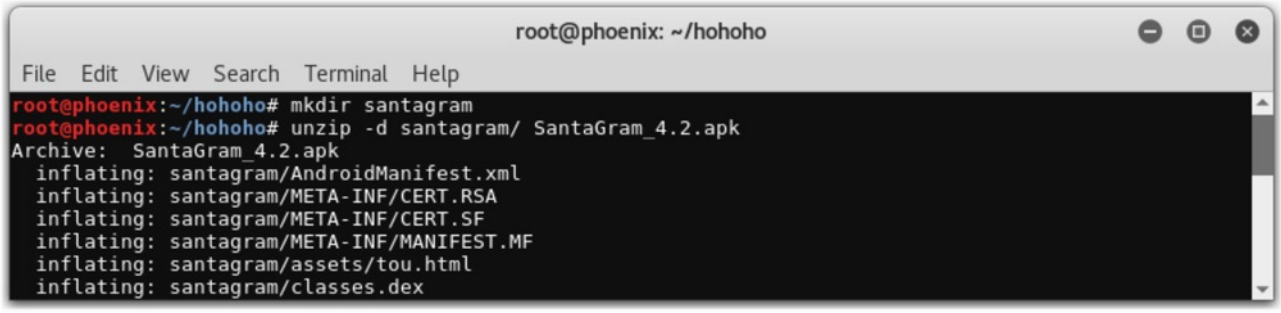
When I looked around in the room again, I spotted a strange glow of light coming from Santa's sack and, thanks to my experience in playing Portal, I knew that this could have meant only one thing: that sack could have lead us somewhere closer to Santa.

"We should follow Santa's tracks kids! Quick, jump into the sack!"
We jumped and, when our feet finally touched the ground, we realized that we had been teleported to an unknown place, where everything was covered in thick blankets of snow and elves were standing everywhere.
"W-w-where are we?" Josh shivered.
The scent of gingerbread and candy was strong and persistent, shiny and pretty lights had been put all over the place and the Christmas magic was so powerful that we could almost touch it.
"Kids, I've a feeling we're not in Kansas anymore" I said.
"Given all the snow and the elves roaming about, I'd say there's a good chance we're at the North Pole itself," Jessica replied.
My outfit was definitely not appropriate for that temperature; we had to find Santa as soon as possible before becoming part of the snowy surrounding.
Josh had a brilliant idea: "Let's take a closer look at that SantaGram mobile application. It might help us find out who kidnapped Santa."
"That's a good idea Josh, but I'm a bit rusty with APK analysis. I think we should talk with some elves first; maybe they can help us with the investigation." I replied.
After walking around for a while, we finally arrived at the Train Station, inside the Workshop, where we met a cordial elf that gave us some good information.

“Hi, my name is Shinny Upatree. I'm one of Santa's bug bounty elves.” He said.
“Hello my little fellow! Is there anyone here who would be able to aid us in reversing Android applications?” I asked.
The elf responded: “You lucky, weird bad-dressed man. I'm the newest elf on Santa's bug bounty team. I've been spending time reversing Android apps.”
“Did you know Android APK files are just zip files?” He continued, “If you unzip them, you can look at the application files. Android apps written in Java can be reverse engineered back into the Java form using JadX.”
“Allright, let’s take a look at the SantaGram source code” I concluded.
I launched JadX on my notebook and, while browsing through the code, I spotted an embedded username and password:

```
    public static void a(final Context context, String str) {
        final JSONObject jSONObject = new JSONObject();
        try {
            jSONObject.put("username", "guest");
            jSONObject.put("password", "busyreindeer78");
            jSONObject.put("type", "usage");
            jSONObject.put("activity", str);
```

Taken by the enthusiasm, I shout: “Quick kids! Take note of these credentials, they may be useful later on in the investigation!”
I then proceeded to unzip the APK:



And while searching for interesting files I found a suspicious audio file:

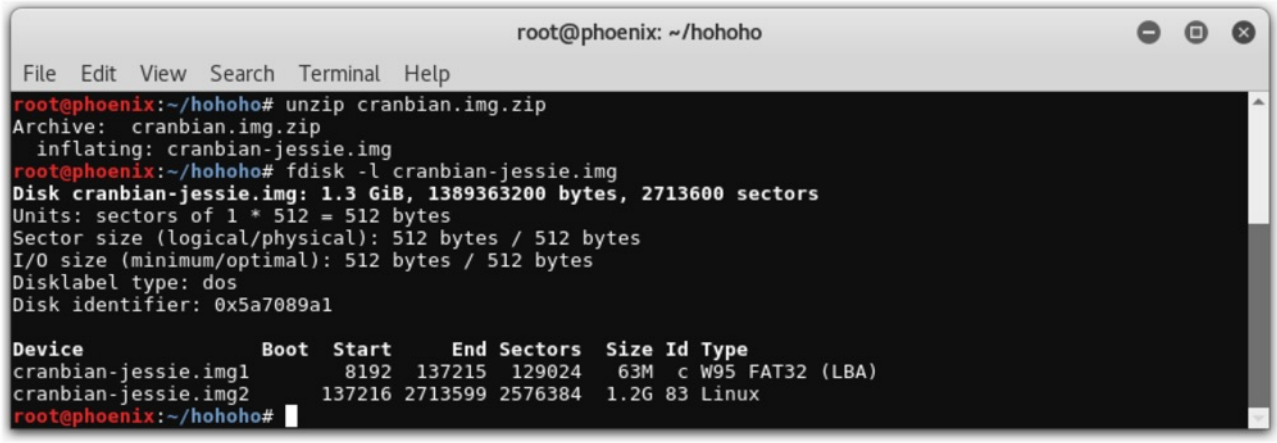
“*Discombobulatedaudio1.mp3*, That’s seems interesting..” I said.

We played it, with the hope of discovering something useful for our investigation; unfortunately, the audio was all distorted and we could not get any information from it.
Jessica was perplexed. "That audio inside of the SantaGram application sounds really strange. I wonder what it means."
It reminded me of the *Devil Ether* scene from *Fear and Loathing in Las Vegas* but, you know, telling that to the kids would have been inappropriate.
Just to thicken the plot even more, during our search, we discovered a couple of suspicious locked doors with little computer terminals next to them.

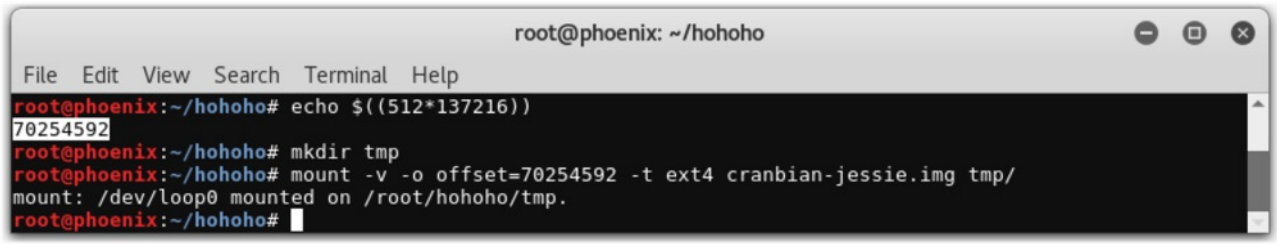
At that point, there were a lot of questions that were waiting to be answered. What was Santa hiding behind those doors? What the audio means? Would we have been able to find Santa and save Christmas? One thing was certain: we needed something to interface with those terminals in order to open those doors.
Just then, Jessica noticed something curious and positively useful. "Heeeey! It looks like someone has left parts of a computer system called a 'Cranberry Pi' strewn all about the North Pole. Perhaps we can fetch all of those pieces and put together a computer we can then use to open those terminals and work on the SantaGram application!"
Driven by this new objective, we searched far and wide, in every corner of the village, finding the power cord, the board, the SD card, the heat sink, the HDMI cable and we reassembled the Cranberry Pi in less than no time.

After reassembling the computer, to be able to make it work, we needed an operating system. It was at this time that another of Santa's elves came to our aid: Holly Evergreen had what we were looking for, but unfortunately he did not know the system password:
"Wow, you found all the pieces of the Cranberry Pi! Great job!" exclaimed Holly and, shrugging continued: "You'll need a Cranbian image to use the Cranberry Pi, but only Santa knows the login password."
"Can you download the image and tell me the password?" asked, handing us a piece of paper with a link written on it.

I then pulled out the PC, downloaded the operating system's image at the link writte in the piece of paper and started to analyze it. I unzipped the Cranbian image and displayed the list of the partitions with ***fdisk***:



Using the sector size (*512 bytes*) and the start sector for the Linux file system (*137216 bytes*), I calculated the number of bytes to the beginning of the file system (*70254592 bytes*).
Then, I created a mount point and mounted the image, specifying the number of offset bytes to the Linux partition and the file system type:



Once I did that, I simply explored the mounted partition, dumping the *shadow* and *passwd* files and combining them using the ***unshadow*** command:

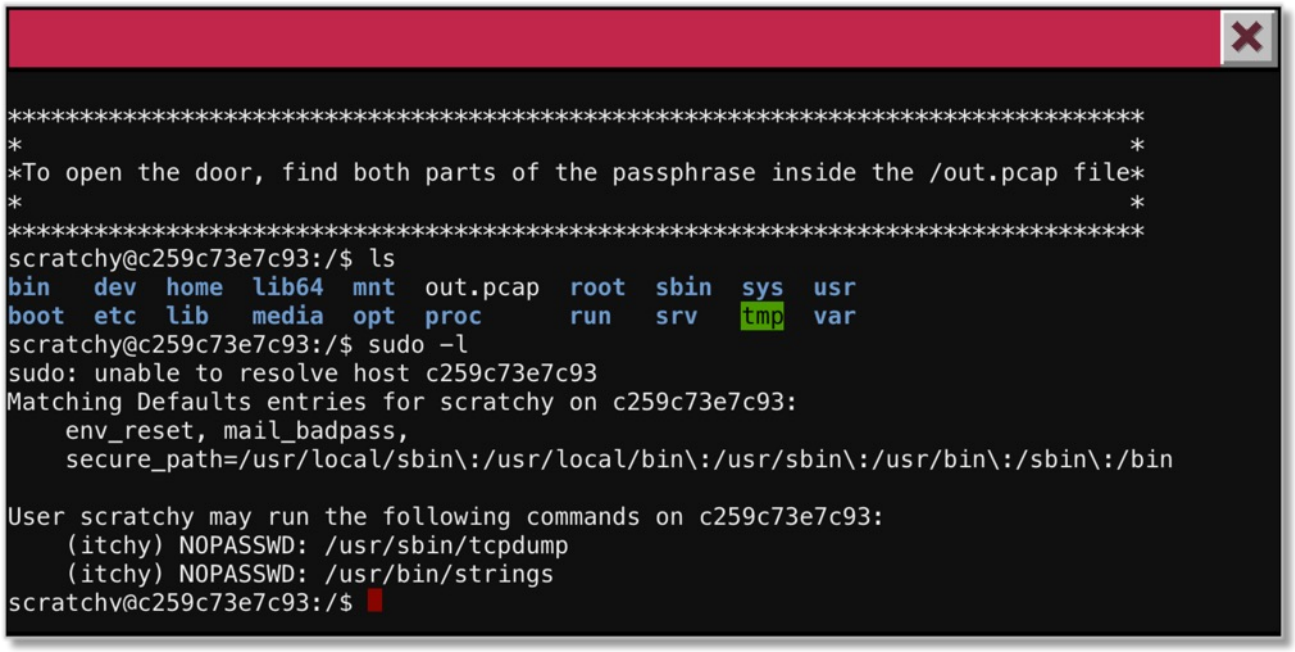
```
root@phoenix: ~/hohoho
File  Edit  View  Search  Terminal  Help
root@phoenix:~/hohoho# cp tmp/etc/passwd .
root@phoenix:~/hohoho# cp tmp/etc/shadow .
root@phoenix:~/hohoho# unshadow passwd shadow > unshadowed.txt
root@phoenix:~/hohoho#
```

I saved this combination into the *unshadowed.txt* file and I started ***John The Ripper*** to crack the password hash, using the *rockyou.txt* file as wordlist.

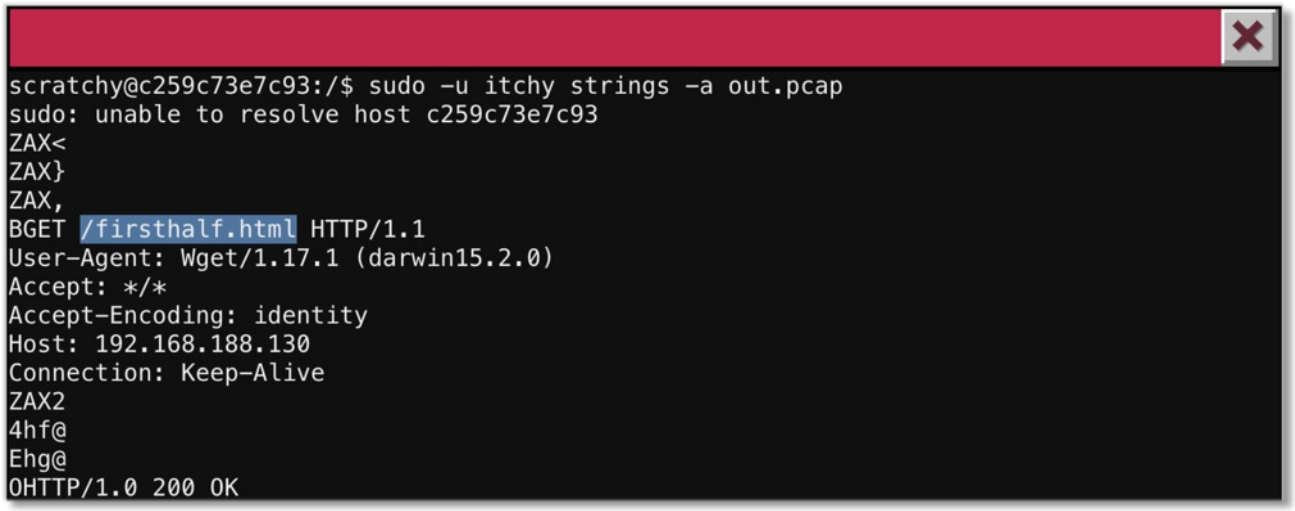
```
root@phoenix: ~/hohoho
File  Edit  View  Search  Terminal  Help
root@phoenix:~/hohoho# john --wordlist=rockyou.txt unshadowed.txt
Warning: detected hash type "sha512crypt", but the string is also recognized as "crypt"
Use the "--format=crypt" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 128/128 AVX 2x])
Press 'q' or Ctrl-C to abort, almost any other key for status
yummycookies     (cranpi)
1g 0:00:00:00 DONE (2017-01-01 12:31) 1.136g/s 621.5p/s 621.5c/s 621.5C/s q..yummycookies
Use the "--show" option to display all of the cracked passwords reliably
Session completed
root@phoenix:~/hohoho#
```

Believing to have identified the password, I once again turned to the elf and asked him: "May the password perhaps be ***yummycookies***?".
And Holly replied: "You're right, that password unlocks the *'cranpi'* account on your Cranberry Pi!"
"With all the pieces of the Cranberry Pi and the Cranbian password, you'll be able to access the terminals throughout the North Pole Wonderland." He added.
With the Cranberry Pi finally functioning, we headed to the Elf House #2, towards the first locked door.

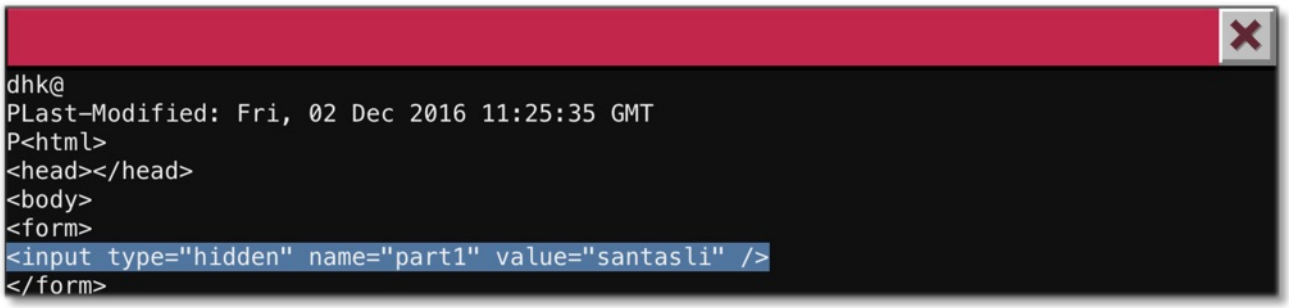
I connected the Cranberry Pi to the terminal and a prompt appeared. It was asking to find the passphrase inside the *out.pcap* file. I needed something to analyze the .pcap file. I quickly listed all the commands that could have been run with higher privileges, and I was amazed to find that *tcpdump* and *strings* were not requiring password for the user *itchy*.



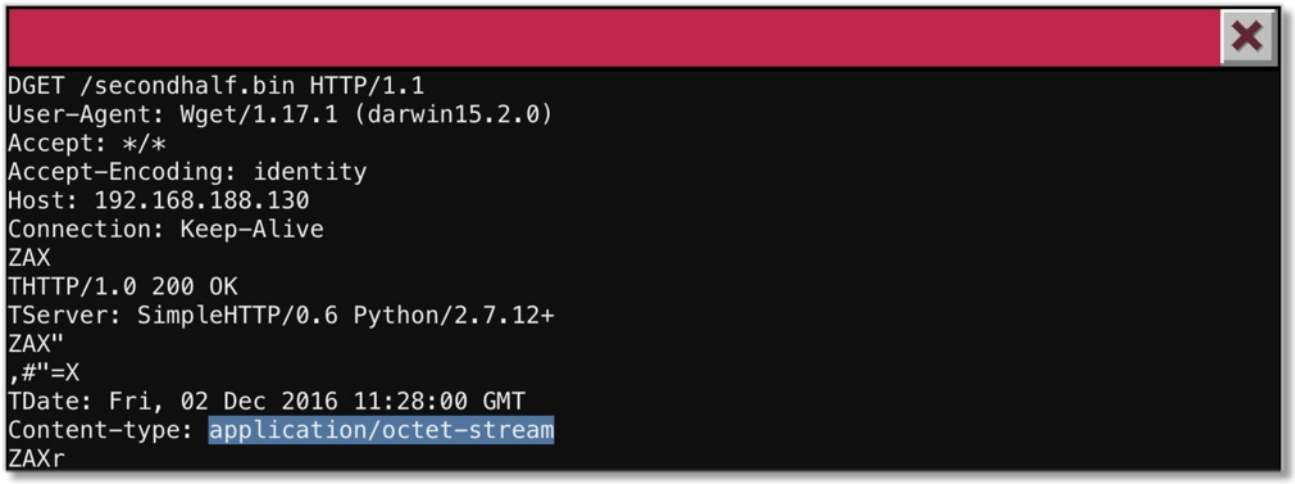
Launching ***tcpdump*** as the user *itchy,* I read the pcap file, discovering that it contained captured HTTP traffic. I decided to use ***strings*** to display only the printable characters contained in the packet capture. Analyzing these strings I saw that the first GET request contained the first half of the passphrase.



The first part of the passphrase was ***santasli***:



The second one, as indicated in the Content-type, contained a binary file:



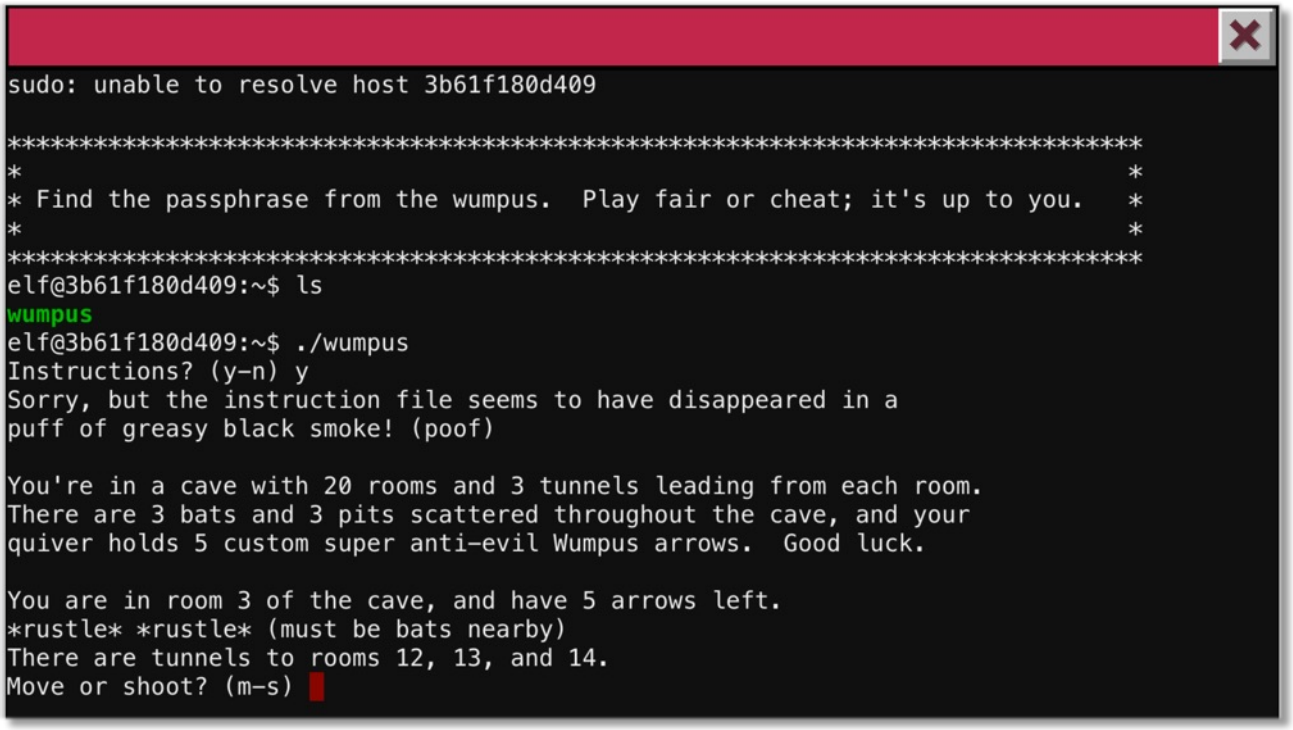
Knowing that, I used the ***strings*** command, changing the encoding type to ***16-bit little endian*** to print the second part of the passphrase:

I inserted the password and, as soon as I clicked on the Enter key, I heard a loud CLANG! The door lock was unbarred. We quickly broke into the room hoping to find Santa who unfortunately was not there either, but instead of him we found Alabaster Snowball, a poor elf who had apparently been accidentally locked inside the room.

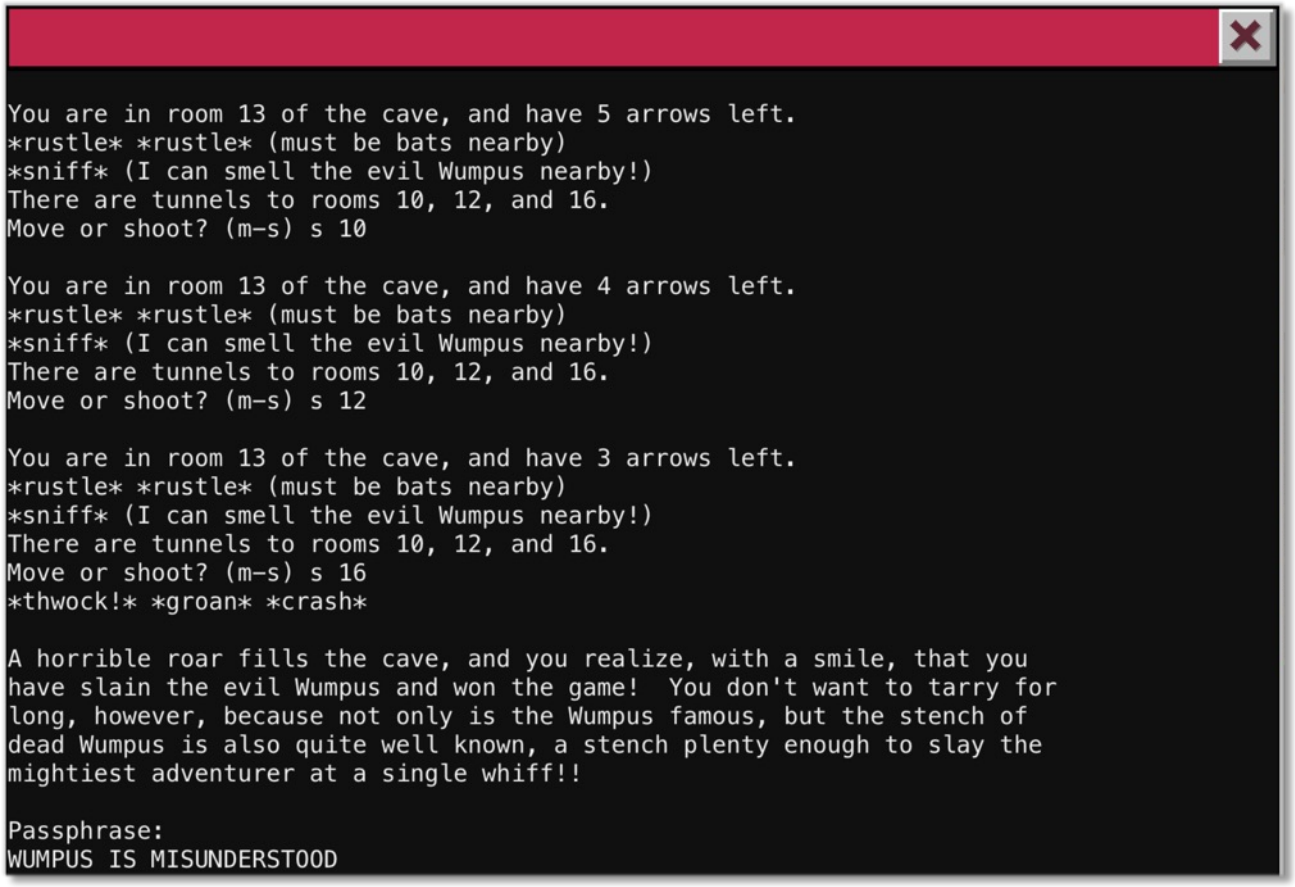
We continued with the second locked door, located inside the Workshop, near the barn.

This prompt was indicating that, to obtain the passphrase, we should have played the *Wumpus* game.

“Let’s do this! Let’s play the game!” said Jess.

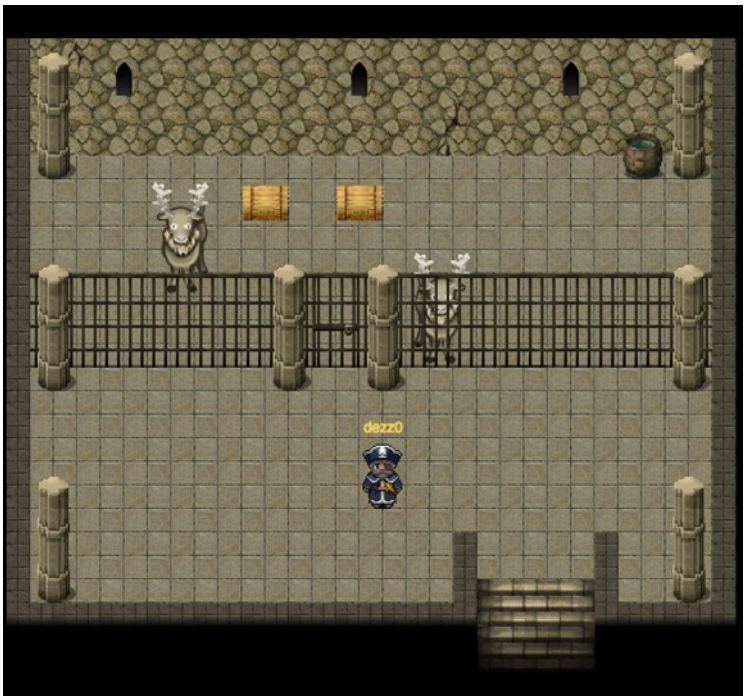


We played that game for a long time, not being able to win it, until Josh came out with this idea: “Maybe, when we smell the wumpus, we should shoot an arrow towards every tunnel that we can see?!”.

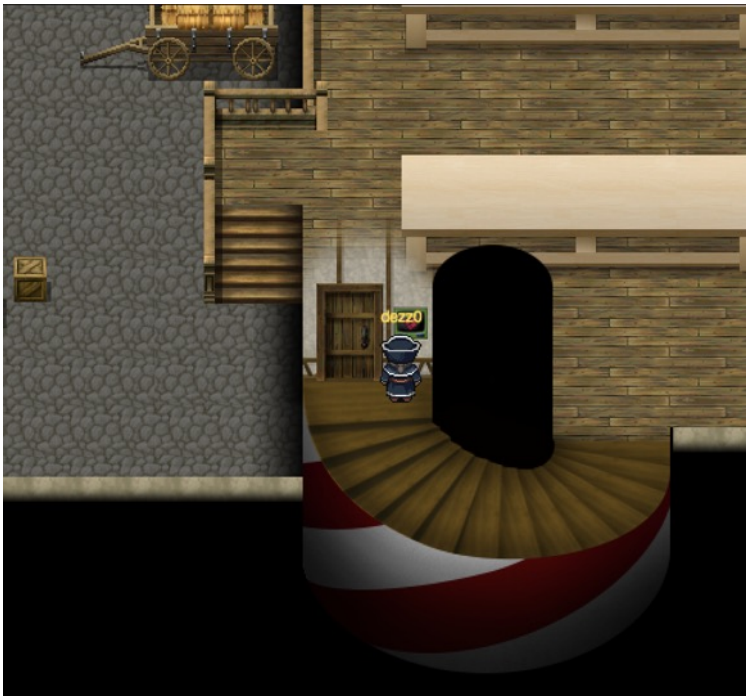


Josh’s idea was a success and we were able to win the game and find the password.

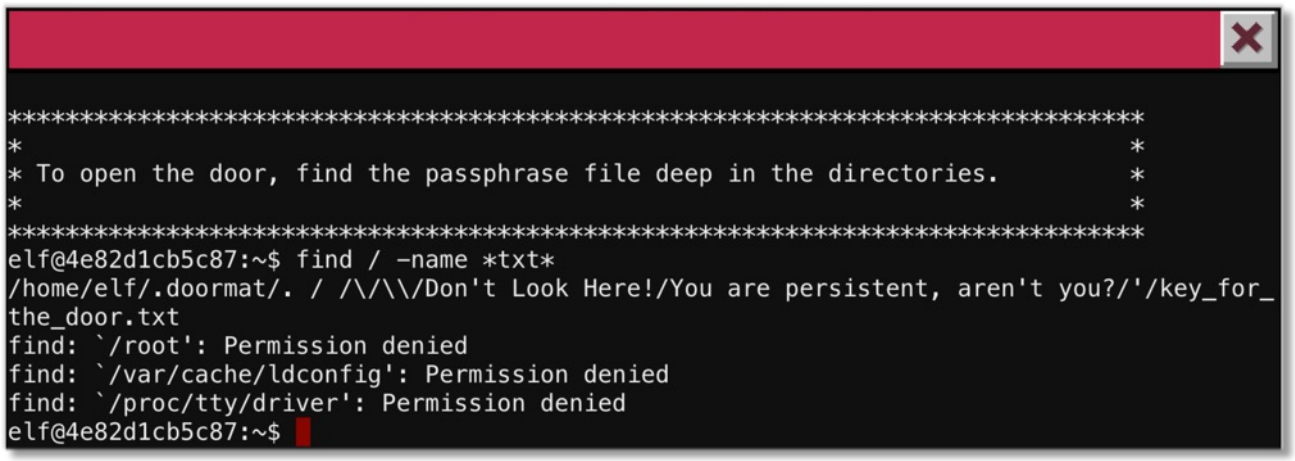
I have to admit that having these two kids by my side helped me a lot while solving this game. We inserted the passphrase and opened wide the door.
We entered in a sort of damp and dark dungeon. Nothing, not even here. We only found a few reindeer behind the bars and, frankly, I do not know what those reindeers had done in order to finish in there.



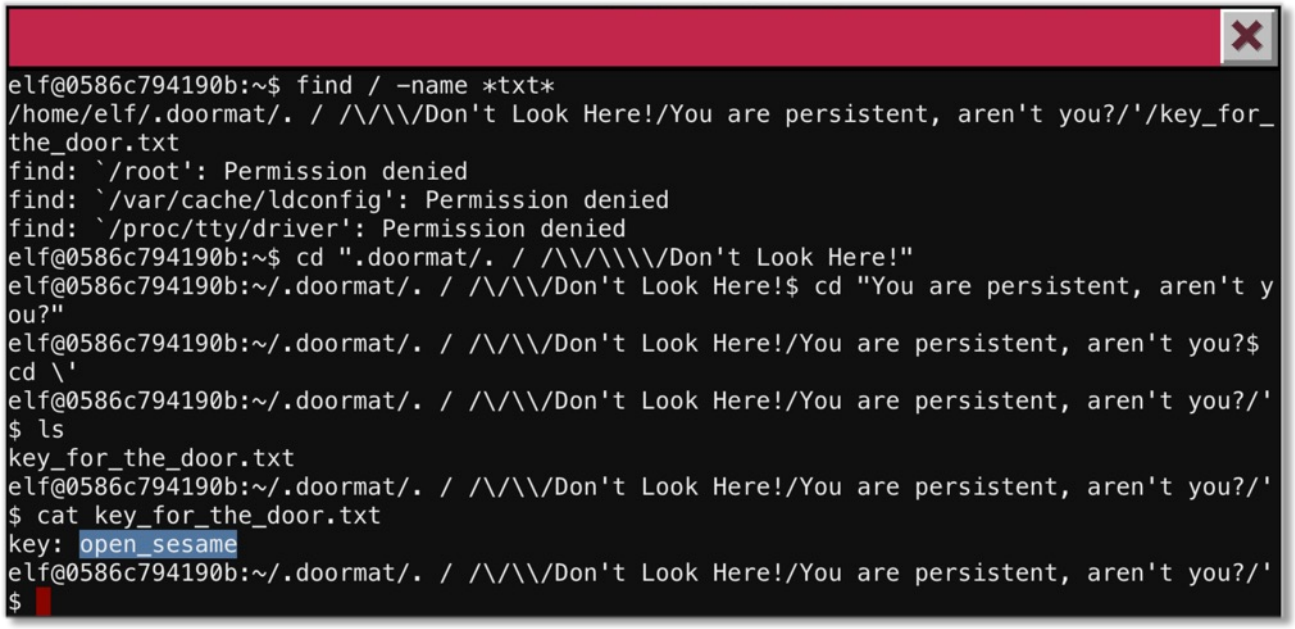
Unfortunately, there was no time to find it out; we were running out of time and we still had to find Santa. Going ahead in our search, we headed to the top floor of the Workshop in which there was another locked door.



To find the passphrase I searched for txt files on the system:



With a bit of escaping I was able to select the right directory and print out the passphrase contained in the *key_for_the_door.txt* file:

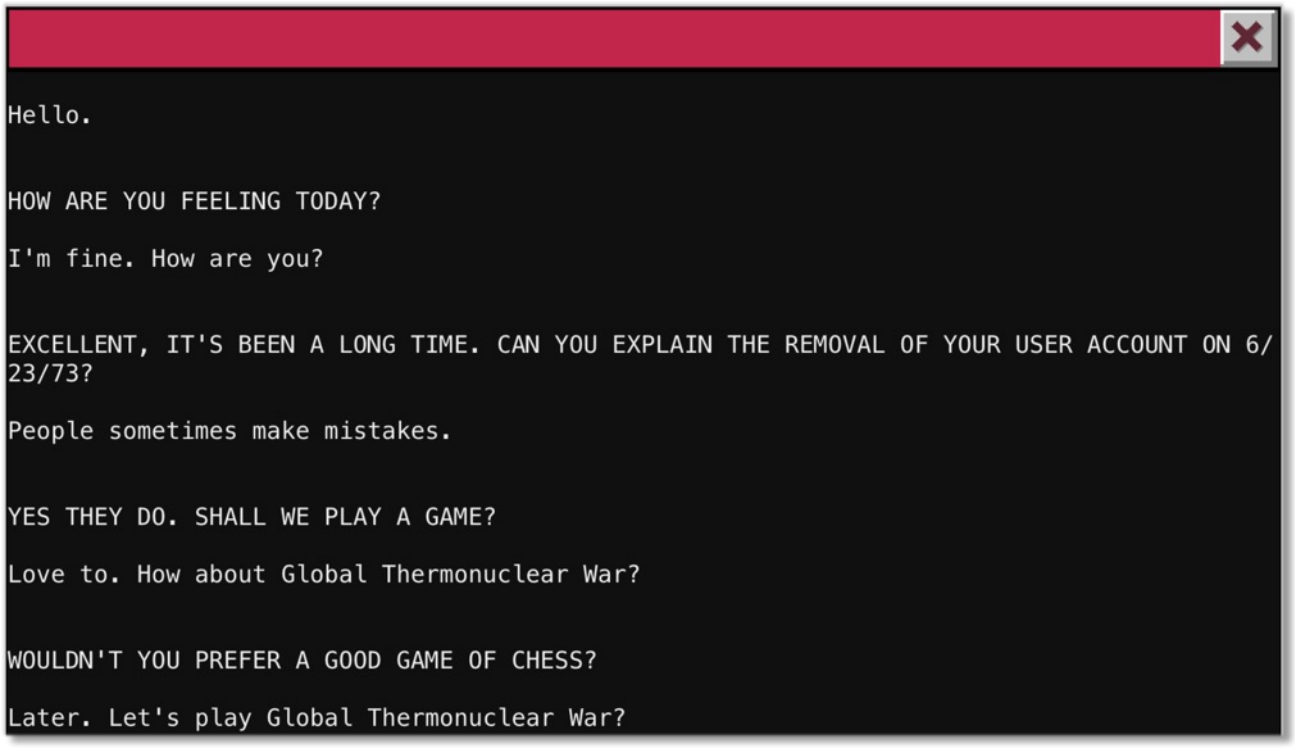


I used the newly discovered password, pressed the Enter key and the door swung open. We found ourselves in Santa's office, but again, there was no sign of him.
Everything was in perfect order: Santa's big desk, the one in which I have always imagined him sitting while reading the endless requests of children from all over the world, was there, in the middle of the room. Behind it a red chair which, to be honest, did not seem to miss that big backside too much!

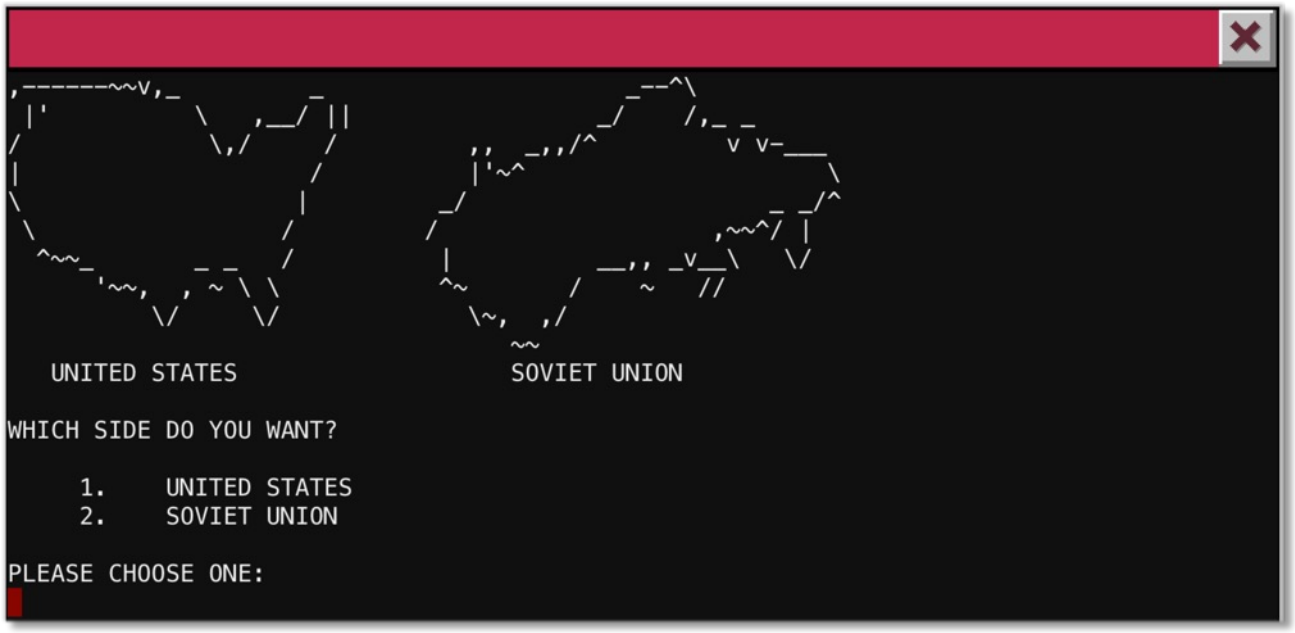
We looked everywhere: under the desk, behind the sofa, even in that big map of the world, almost hoping to see a GPS signal flashing in some corner of the planet.
Between the two bookcases I spotted another terminal and I also noticed that on the floor, at the side of one of them, there were some deep scars which made me guess that something had been dragged repeatedly. Was it really possible that a secret passage was hidden behind there?
Connecting the Cranberry PI to the terminal a prompt came out.

“I already heard that phrase somewhere” I thought.
Then I remembered: “It’s WOPR from the Wargames movie!”

I replied to the Super Computer using the exact same quotes from the movie:



The Super Computer let me chose the Country to play with and I typed *2* to select Soviet Union.

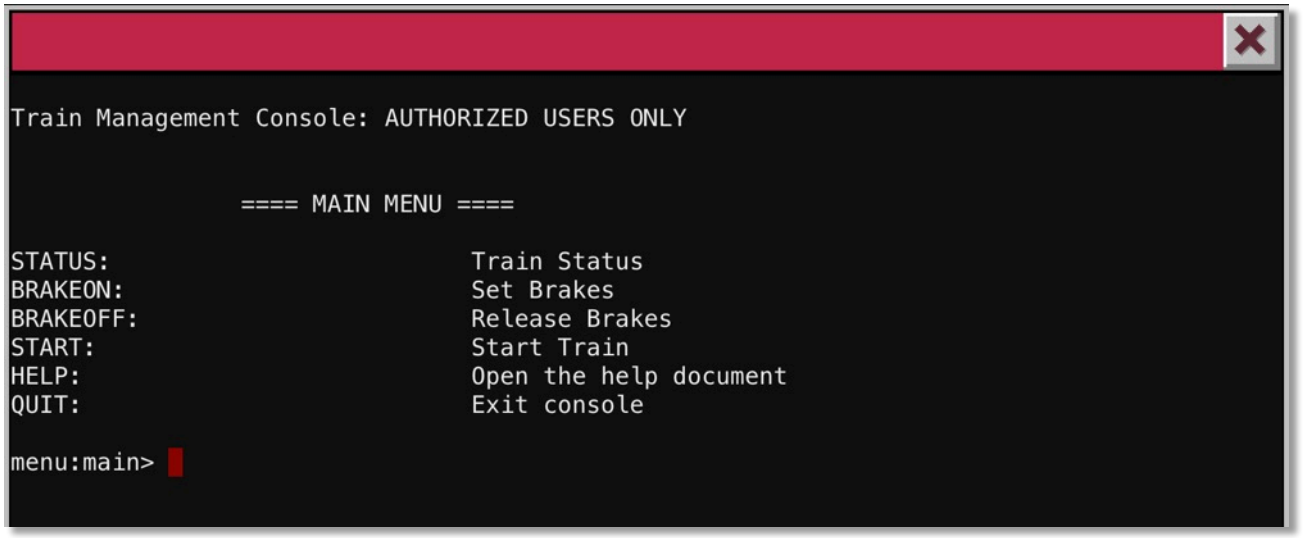


Then It asked for a target; I chose ***Las Vegas*** but my finger hesitated in pressing the button. I was sweating. "Kids, I really hope that this is only a simulation because, if not, this Christmas would be worse than the one where I thought that feeding a Gremlin after midnight was a great idea."

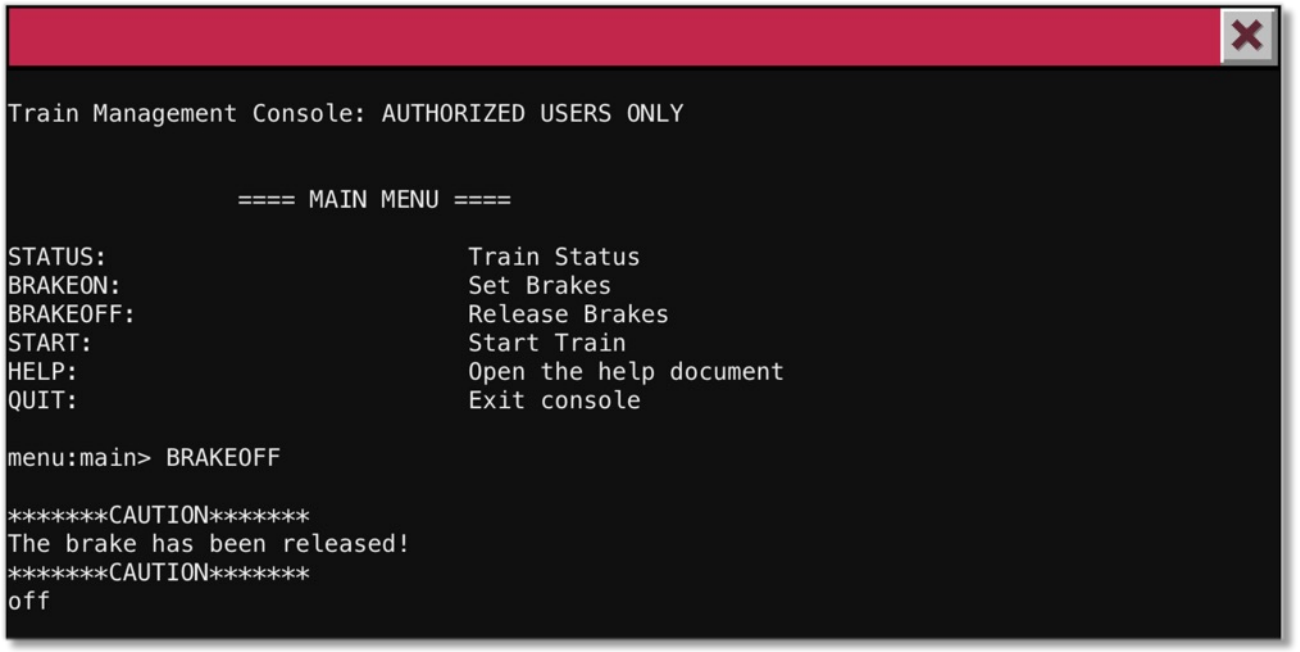
Fortunately for us, unlike its twin, the Super Computer on Santa's office gave us the password instead of burning down the entire Los Angeles. We inserted the passphrase and pushed the Enter key; the library skidded sideways revealing a hidden entrance.
We ventured inside the passage, finding ourselves in a narrow corridor. In front of us another locked door. This time without a terminal to hack. Not knowing how to open that door we had been forced to temporarily abandon the enterprise.

I then remembered that I saw a terminal while we were talking with *Shinny Upatree* at the Train Station in the Workshop.

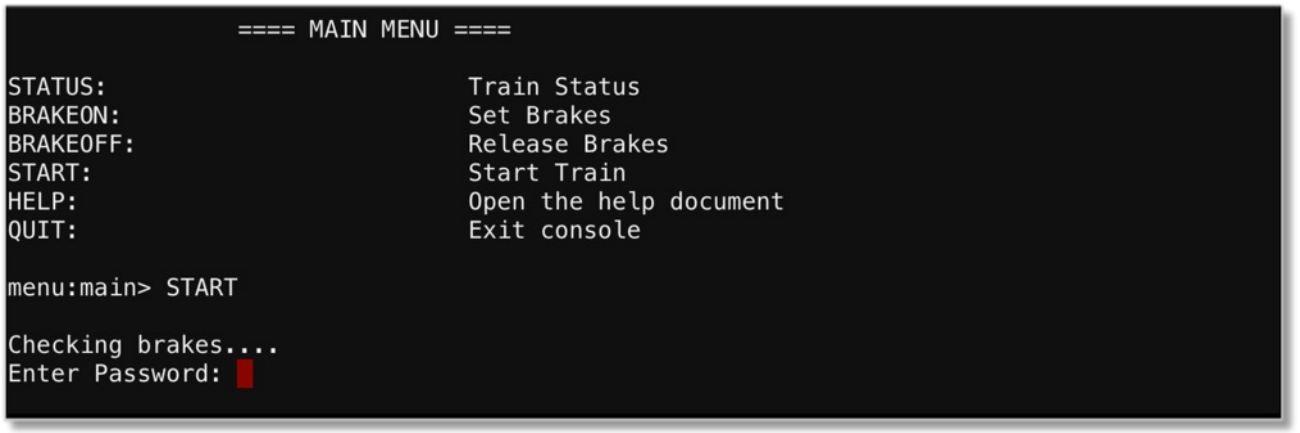
The screen was displaying a Train Management Console:



To move the train, I first sent the command to release brakes:



And I started the train. Unfortunately it requested a password.



I stopped for a second and then I remembered the god ol' *vi* trick to escape restricted shells. I opened the help document by typing *HELP*:

```
               ==== MAIN MENU ====

STATUS:                          Train Status
BRAKEON:                         Set Brakes
BRAKEOFF:                        Release Brakes
START:                           Start Train
HELP:                            Open the help document
QUIT:                            Exit console

menu:main> HELP
```

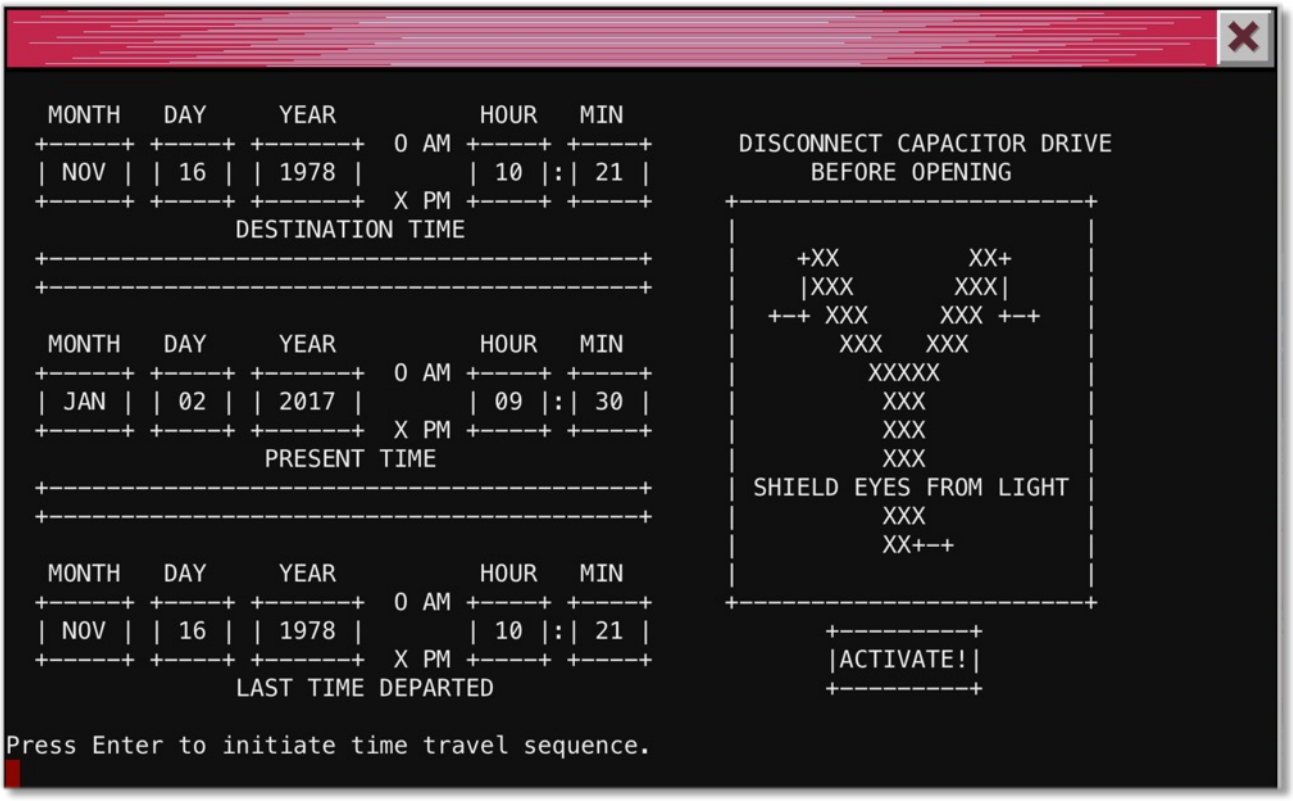
I then tried to spawn a shell from the help document by typing !/bin/sh

```
Help Document for the Train

**STATUS** option will show you the current state of the train (brakes, boiler, boiler t
emp, coal level)

**BRAKEON** option enables the brakes.  Brakes should be enabled at every stop and while
 the train is not in use.

**BRAKEOFF** option disables the brakes.  Brakes must be disabled before the **START** c
ommand will execute.

**START** option will start the train if the brake is released and the user has the corr
ect password.

**HELP** brings you to this file.  If it's not here, this console cannot do it, unLESS y
ou know something I don't.


Just in case you wanted to know, here's a really good Cranberry pie recipe:

Ingredients
1 recipe pastry for a 9 inch double crust pie
1 1/2 cups white sugar
1/3 cup all-purpose flour
1/4 teaspoon salt
1/2 cup water
1 (12 ounce) package fresh cranberries
1/4 cup lemon juice
1 dash ground cinnamon
!/bin/sh
```

Pressing the Enter key I successfully spawned a shell and, listing all the files contained in the directory, I noticed the ActivateTrain script. I launched that script:

```
               ==== MAIN MENU ====

STATUS:                          Train Status
BRAKEON:                         Set Brakes
BRAKEOFF:                        Release Brakes
START:                           Start Train
HELP:                            Open the help document
QUIT:                            Exit console

menu:main> HELP

sh-4.3$ ls
ActivateTrain  TrainHelper.txt  Train_Console
sh-4.3$ ./ActivateTrain
```

On the control panel I read: "Destination Time November 16, 1978 – 10:21, Press Enter to initiate the time travel sequence".

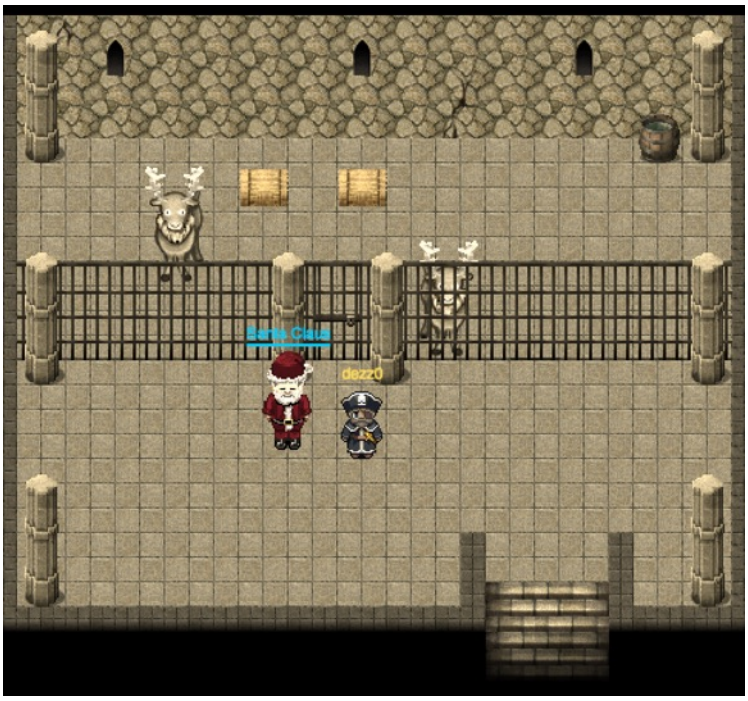


Everything made me think of a time machine. "But how could a train travel back through time?" I thought. I do not know why I was still astonished by these oddities, shortly before that moment I was jumping inside a sack and then found myself talking to Santa's elves at the North Pole! "Damn – I thought – let's do it!"
I pressed the Enter key and heard the train engines turning on. I noticed that the flux capacitor was slowly activating and I shouted to the children:
"Quick, jump on!"

When the flux capacitor was completely activated, the train disappeared in a blinding flash of blue light, leaving a pair of fire trails on the rails.

We got off the train and surprisingly found ourselves in the past! "It's 1978 children! We would still be able to save John Lennon!" I exclaimed.

We scoured the village from cover to cover and finally, checking all the door that we unlocked in the present and, entering the door next to the barn, the one that led to the dungeon, we found Santa. Poor thing, he did not look well at all!



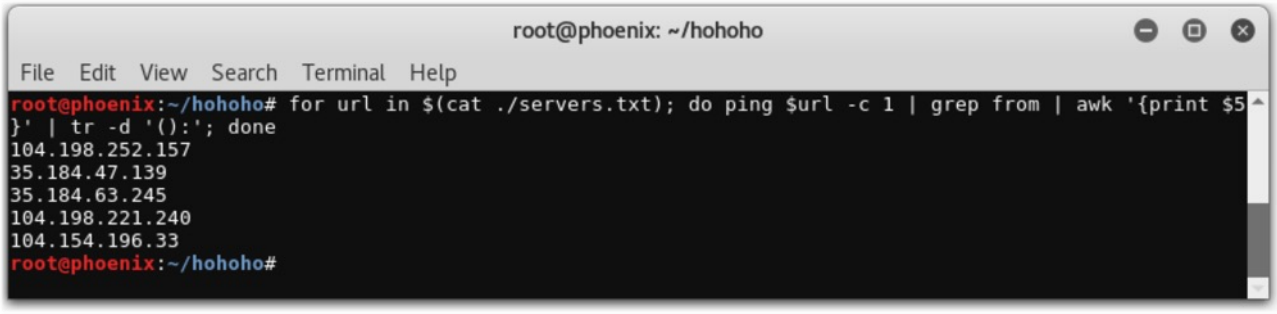
“Well, hello there. You’ve rescued me! Thank you so much!” said Santa as soon as he saw us.

Unfortunately, Santa had lost his memory during the scuffle and he was not able to provide the name of his kidnapper.
We then let Santa go, greeting him and wishing him good luck with his gift delivery.
Jessica proclaimed, "We finally found Santa Claus! We've saved Christmas." The children were exuberant!
But Joshua made us immediately note that we should have not let ourselves be carried away by the enthusiasm; our mission could have been considered completed only after finding the kidnapper and handing him over to justice: that was the only way in which we would have been really sure about Santa Claus' safety.
“Yeh, we are not done yet.” I grunted.
We agreed on the fact that we needed to attack the SantaGram application and its associated servers in order to find other useful hints that could have lead us to the kidnapper.
But there was a problem: How could we know which server was in scope?
Josh lit up. "Hey, in wandering around the North Pole, you'll never believe who I ran into. Mr. Tom Hessman himself! As it turns out, he is up here, and is happy to confirm which IP addresses we are allowed to attack."

So, the first thing that we needed were the server addresses.
Inspecting SantaGram’s source code with ***JadX,*** I discovered some addresses written in the ***strings.xml*** files.

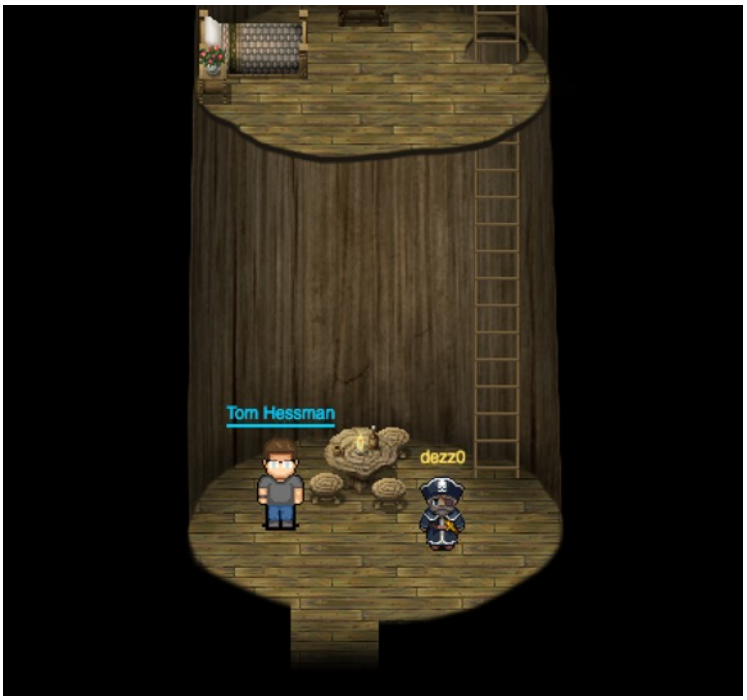
```
<string name="app_name">SantaGram</string>
<string name="appbar_scrolling_view_behavior">android.support.design.widget.AppBarLayout$ScrollingViewBehavior</string>
<string name="banner_ad_url">http://ads.northpolewonderland.com/affiliate/C9E380C8-2244-41E3-93A3-D6C6700156A5</string>
<string name="bottom_sheet_behavior">android.support.design.widget.BottomSheetBehavior</string>
<string name="character_counter_pattern">%1$d / %2$d</string>
<string name="debug_data_collection_url">http://dev.northpolewonderland.com/index.php</string>
<string name="debug_data_enabled">true</string>
<string name="dungeon_url">http://dungeon.northpolewonderland.com/</string>
<string name="exhandler_url">http://ex.northpolewonderland.com/exception.php</string>
```

I obtained the IP addresses from these URL with a quick script:



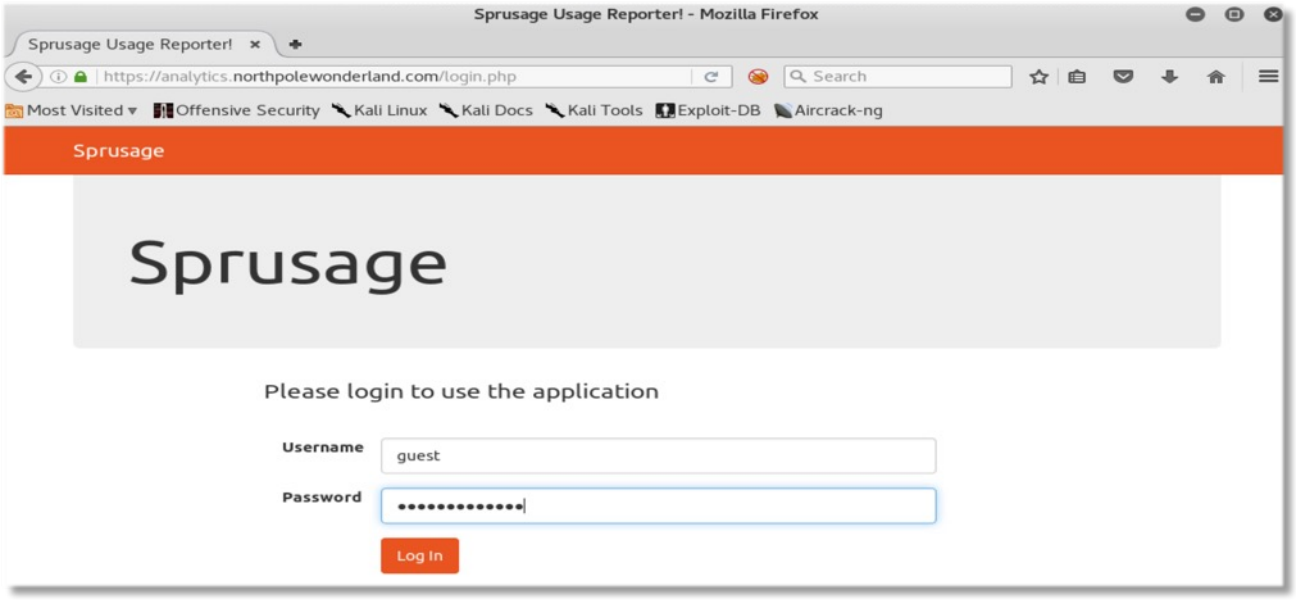
Once the script finished, we wrote down the addresses and we reached Mr. Tom Hessman at The Big Tree House, gaving him the little piece of paper where we wrote all the discovered IP addresses.

Mr. Hessman took the little piece of paper and, after analyzing carefully these addresses, exclaimed:
"Yes! All the IP addresses you listed are in scope! Just make sure you don't launch denial of service attacks, or you will otherwise interfere with the host's production processing. *Dirbuster* will not help you"

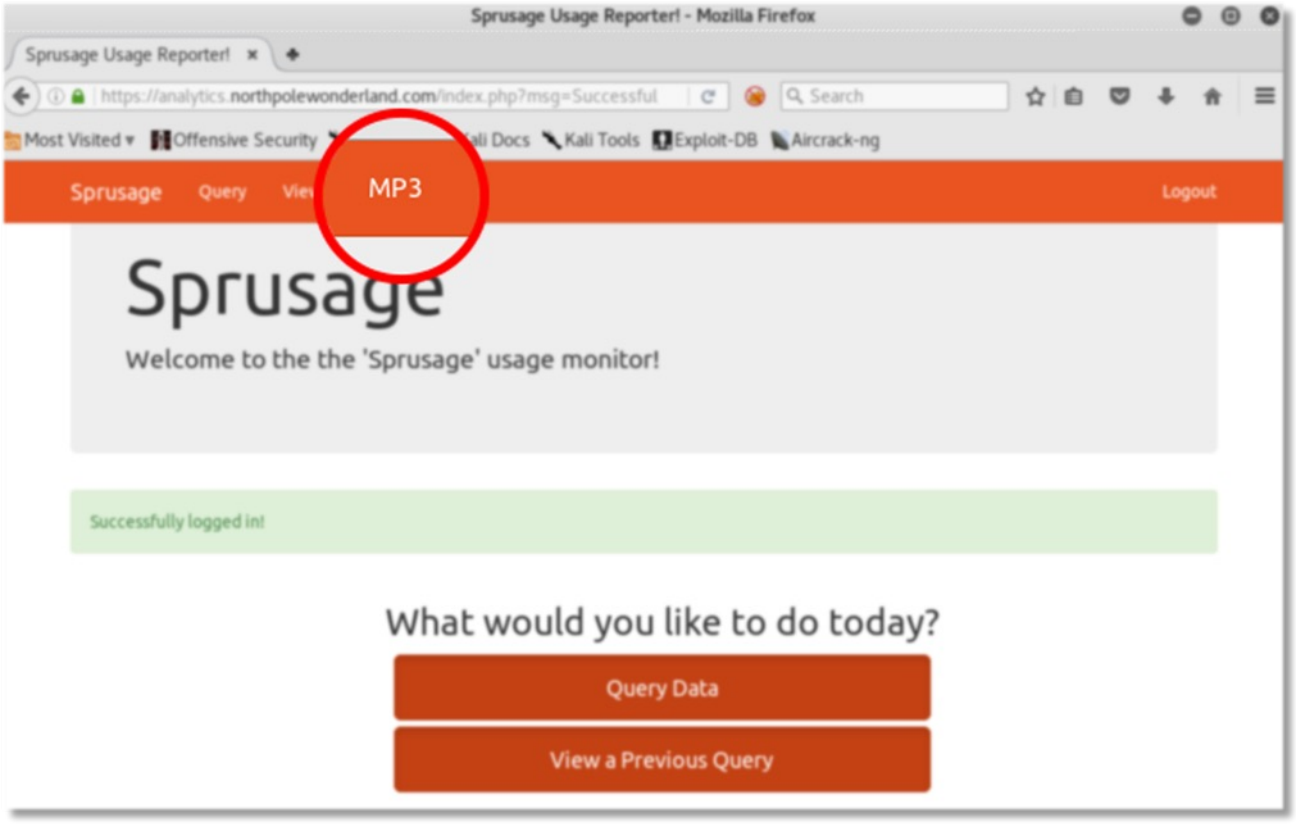


We were ready and determined to attack these servers!
We started with the ***Analytics Server*** at: *https://analytics.northpolewonderland.com*
Browsing to that URL, I landed on a login page. Josh poked me to draw my attention and exclaimed: "It could probably be the right time to use the credentials we gathered while analyzing the APK!"

Following his advice, I inserted the credentials into the login form:



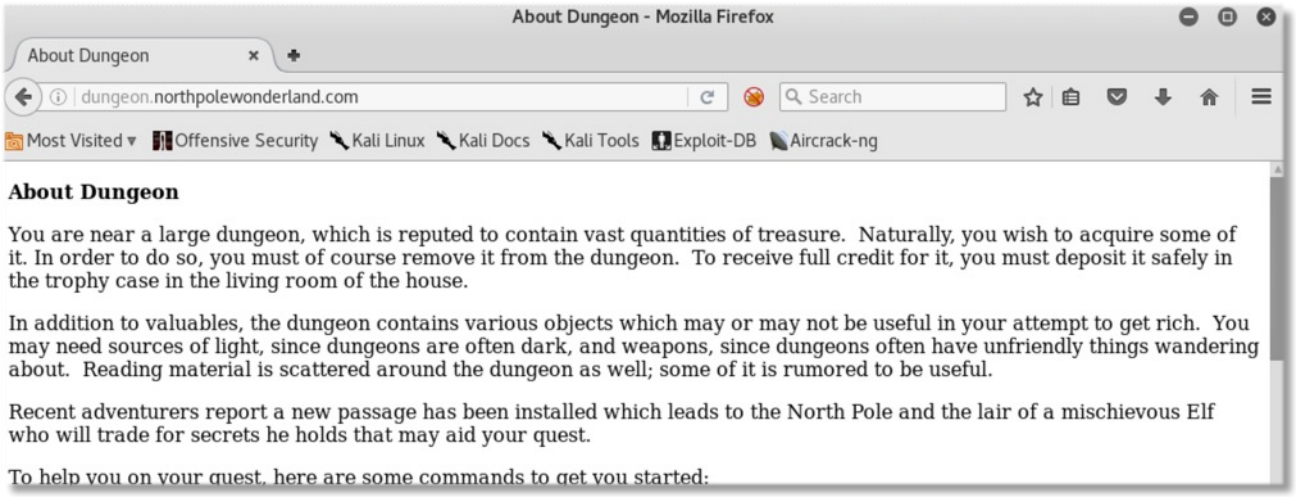
I successfully logged into the web application and I noticed the MP3 button on the menu bar:



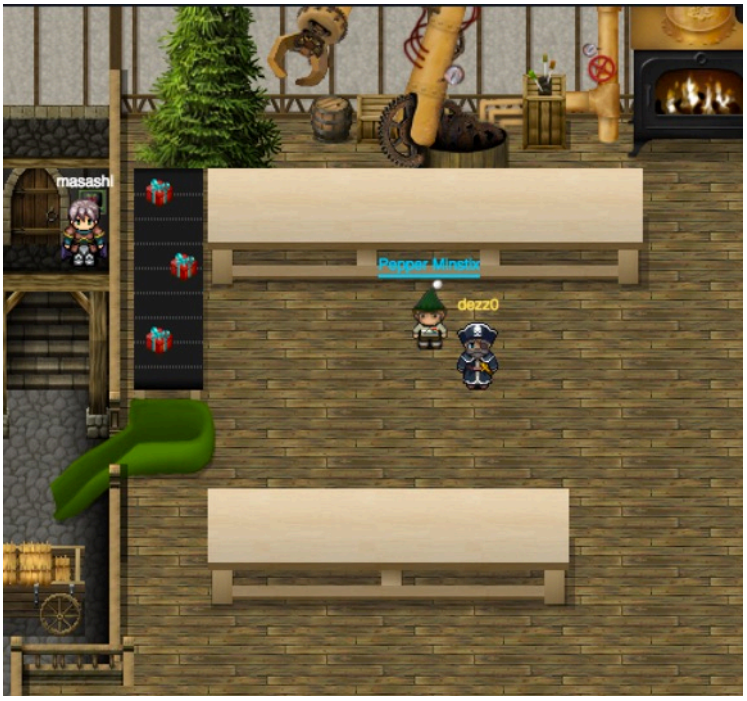
Clicking on it, I was able to download another file audio called: *discombobulatedaudio2.mp3*

"Well, that was easy! Let's attack the next one!" I said.

We decided to target the ***Dungeon Server*** and, browsing to the URL: *http://dungeon.northpolewonderland.com*, I landed into an help page:



The last part of the help page was talking about a mischievous Elf with whom we could have trade for secrets and who could have helped us in our quest.
We tried really hard to find the page into which we could play the game without any luck and, at that point, we were a bit discouraged.
Unexpectedly, while we were discussing about the game, an elf heard us and said:
"Hey! I'm sorry to interrupt you, I'm Pepper Minstix. Are you talking about Dungeon?"

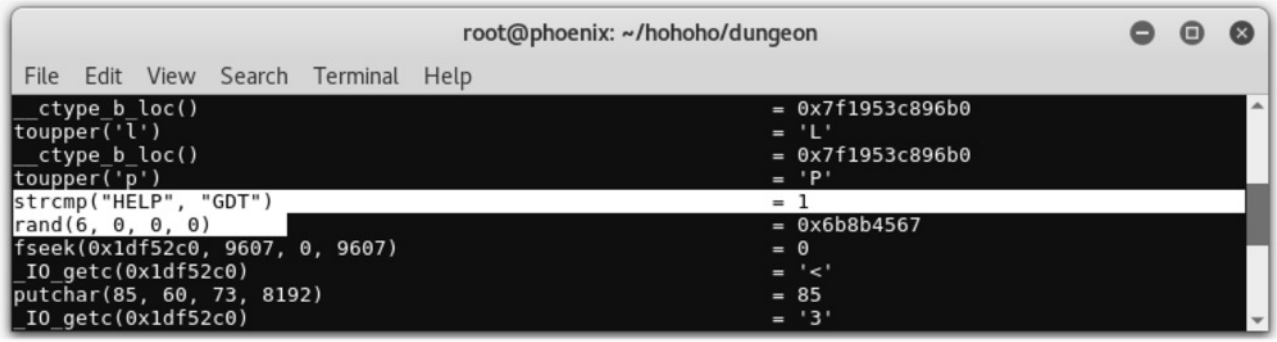


"Yes, we are having problems in figuring out where to find the game" I answered.

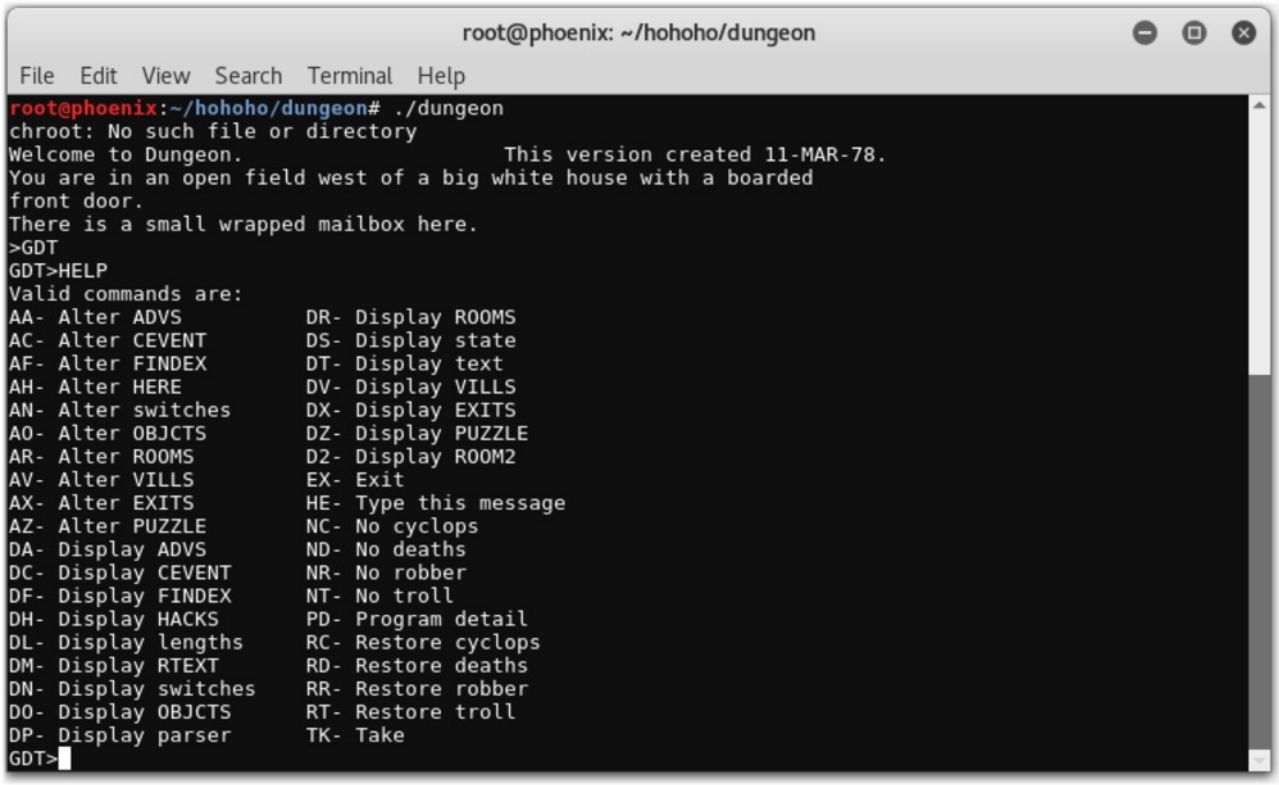
“When I need a break from bug bounty work, I play dungeon. I’ve been playing it since 1978. I still have yet to beat the Cyclops...” and then he added: “Alabaster’s brother is the only elf I’ve ever seen beat it, and he really immersed himself in the game. You can download an old version at *www.northpolewonderland.com/dungeon.zip*”.

We followed that URL, downloaded the file and unzipped it.
I launched the game, noticing that it was suspiciously similar to Zork.

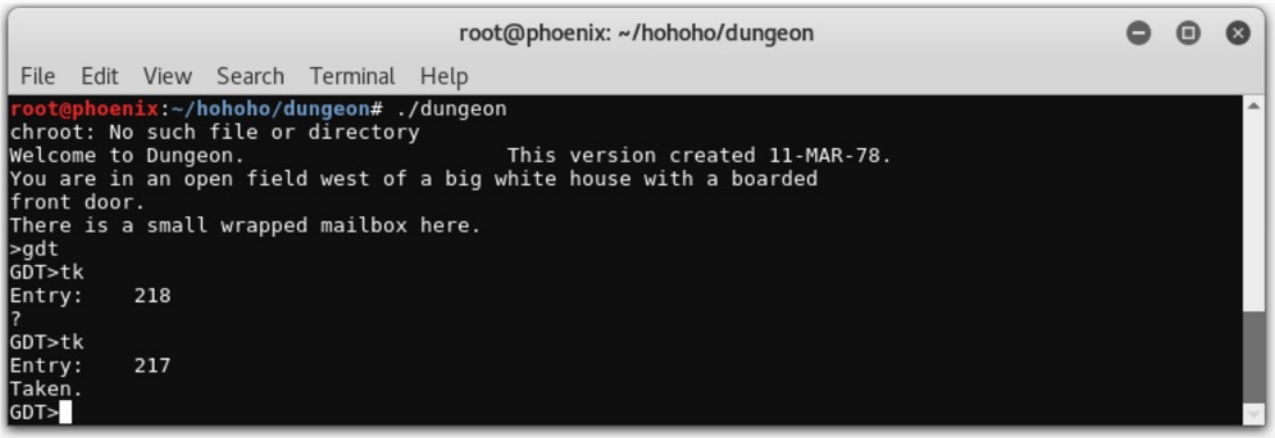
“I wonder if it’s possible to cheat somehow” I thought while looking at the prompt.
I then launched again the game, this time running it with ***ltrace***, to intercept and analyze all the dynamic library and system calls which were called by the game. I noticed that at each command inserted, the game compared it with the string “GDT”.



Searching on the Internet for this acronym, I discovered that Zork included a built-in debugger (http://gunkies.org/wiki/Zork#The_GDT_command). I opened the game again and, by typing ***GDT*** I entered in debug mode.

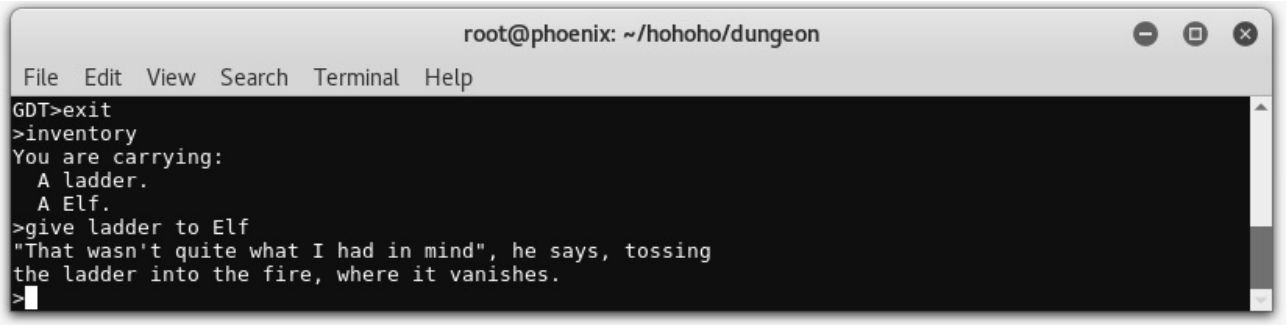


Interestingly, it seemed that I could take items from the game with the command *TK*.
I then tried, using an unorthodox approach, to take the last available item in the hope to get something useful.



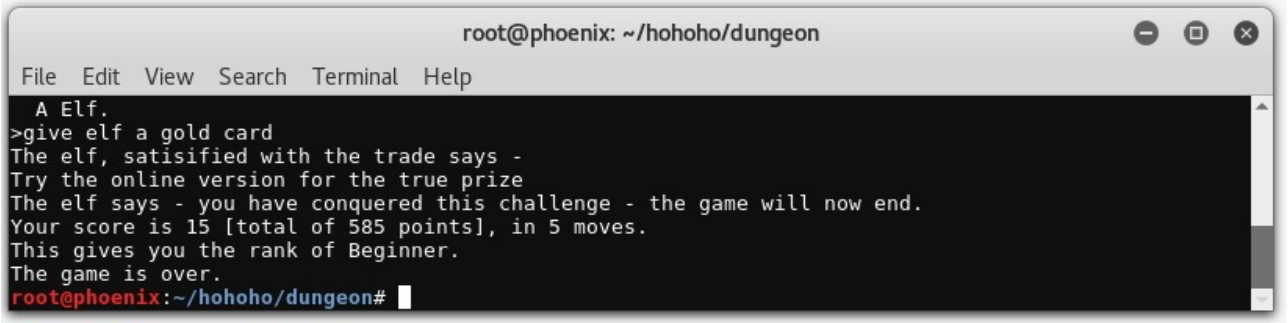
And, with my surprise, I discovered that I put the Elf on my inventory!

After that, I took the next item and I tried to give it to the Elf but he obviously wasn't happy at all about what he got.

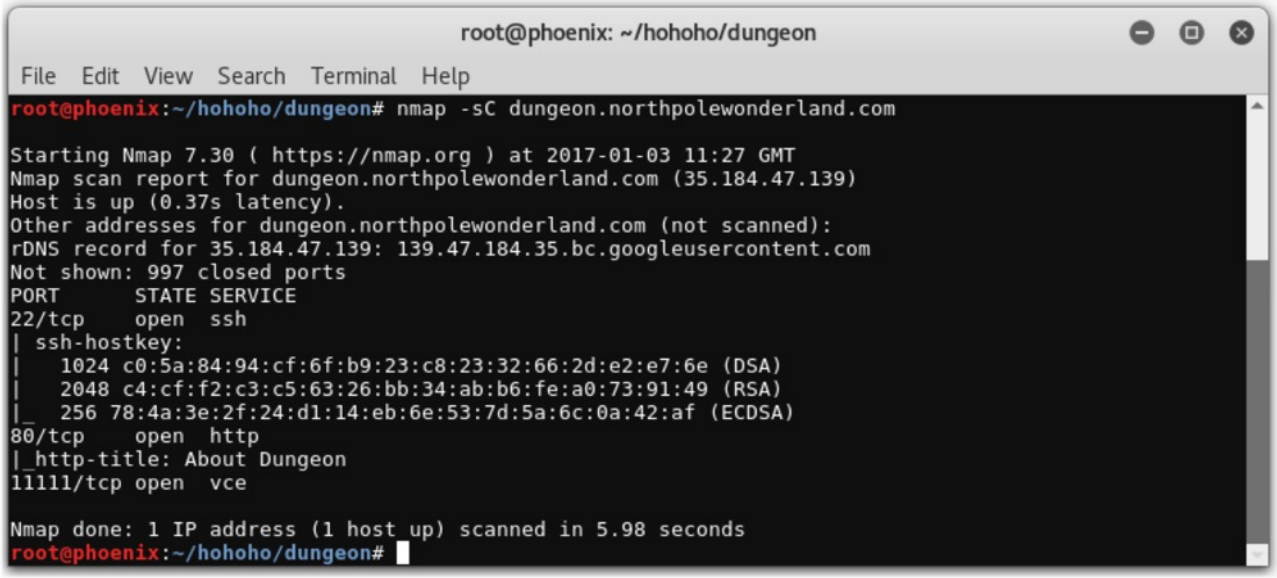


I thought that the Elf would have accepted something precious. The most precious thing that I found was a golden card.

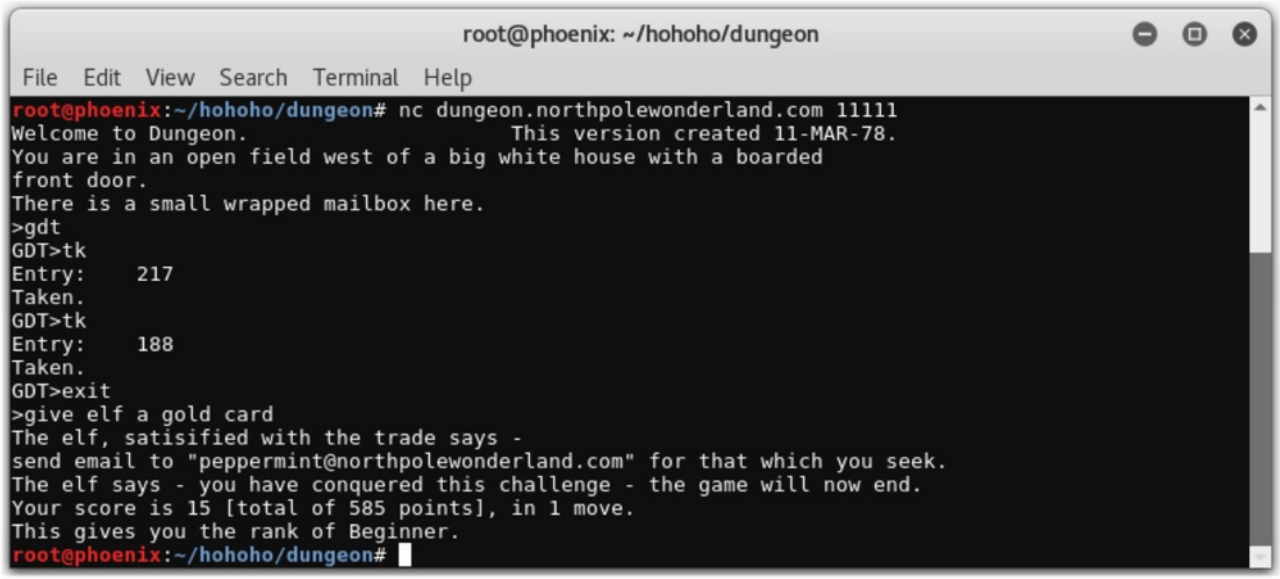
Before pressing the Enter key, with my best Italian voice, I said: "I'm gonna make him an offer he can't refuse".



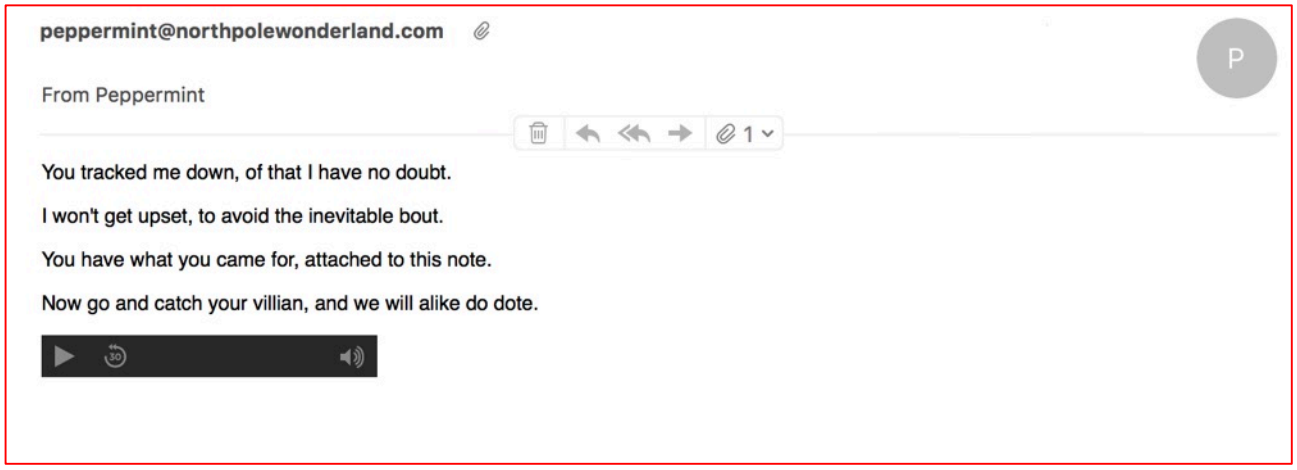
The elf was satisfied and we won the game but, unluckily, the prize was on the server. So, we were definitely missing something on the server; maybe another open port with which we could communicate? I decided to launch an *nmap* scan:



My assumptions were right; *TCP port 11111* was open. I connected to it using ***netcat*** and the game suddenly started.
I followed the same steps that I took in the offline game:



We finished the game and we got instructions to send an email to *peppermint@northpolewonderland.com* as a reward for winning it.
I sent an email to that address and, after a while, I received a response that had the a file audio named **discombobulatedaudio3.mp3***:*



We moved to the ***Debug Server*** but, to attack that one, we needed to first capture a request from the application. I remembered that, while looking at the code, I saw something related to the debug mode on the class *EditProfile* inside the *onCreate* method:

```
protected void onCreate(Bundle bundle) {
    boolean z;
    super.onCreate(bundle);
    setContentView((int) R.layout.edit_profile);
    super.setRequestedOrientation(1);
    b.a(getApplicationContext(), getClass().getSimpleName());
    if (getString(R.string.debug_data_enabled).equals("true")) {
        Log.i(getString(R.string.TAG), "Remote debug logging is Enabled");
        z = true;
    } else {
        Log.i(getString(R.string.TAG), "Remote debug logging is Disabled");
        z = false;
    }
```

And the *debug_data_enabled* flag was set to *false* in the *string.xml* file:

```
<string name="character_counter_pattern">%1$d / %2$d</string>
<string name="debug_data_collection_url">http://dev.northpolewonderland.com/index.php</string>
<string name="debug_data_enabled">false</string>
<string name="dungeon_url">http://dungeon.northpolewonderland.com/</string>
```

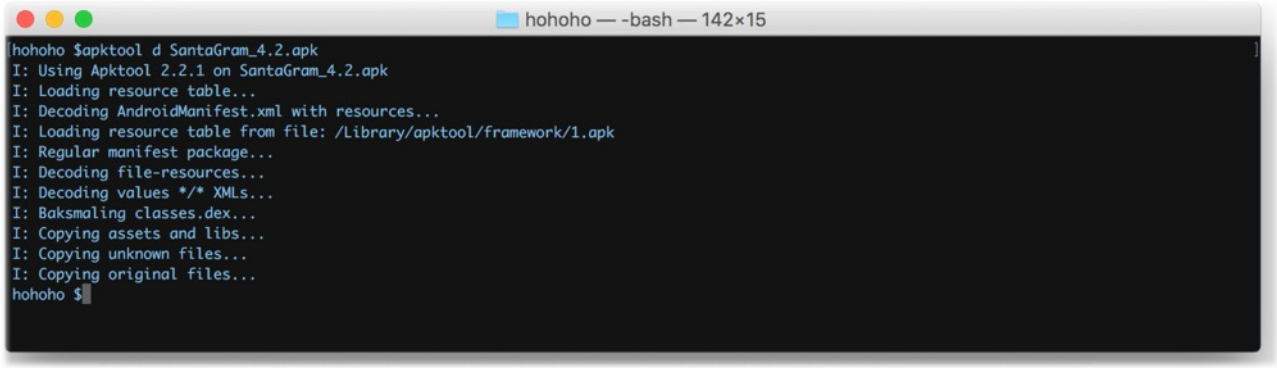
I needed to modify that flag, setting it to true and then recompile the application. Outside the Santa's Workshop we met *Bushy Evergreen*, a gentle elf that had some useful suggestions in order to perform that task.

"You guys could try with *Apktool*, it can preserve the functionality of the app, then change the Android bytecode smali files, it can even change the values in Android XML files, then use *Apktool* again to recompile the app." Bushy suggested.

We thank him and before we left, he added:

"Hey! Just remember that *Apktool* compiled apps can't be installed and run until they are signed. The *Java keytool* and *jarsigner* utilities are all you need for that."

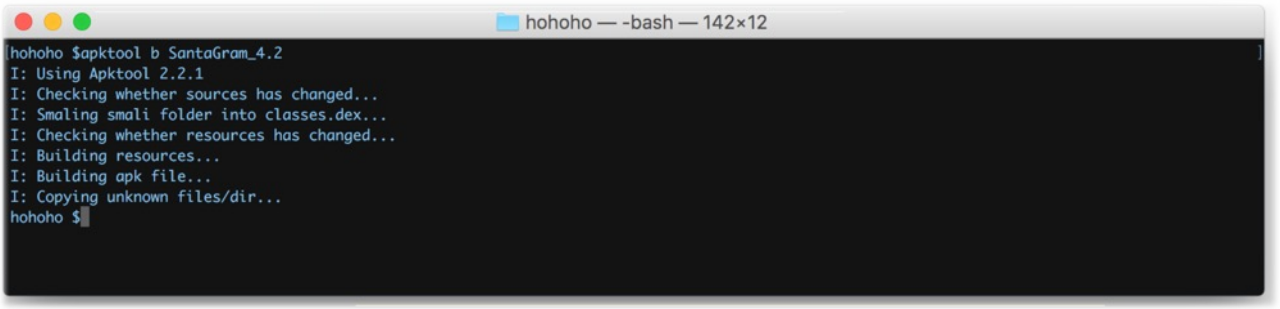
I started by using the ***apktool*** to decompile the SantaGram app:



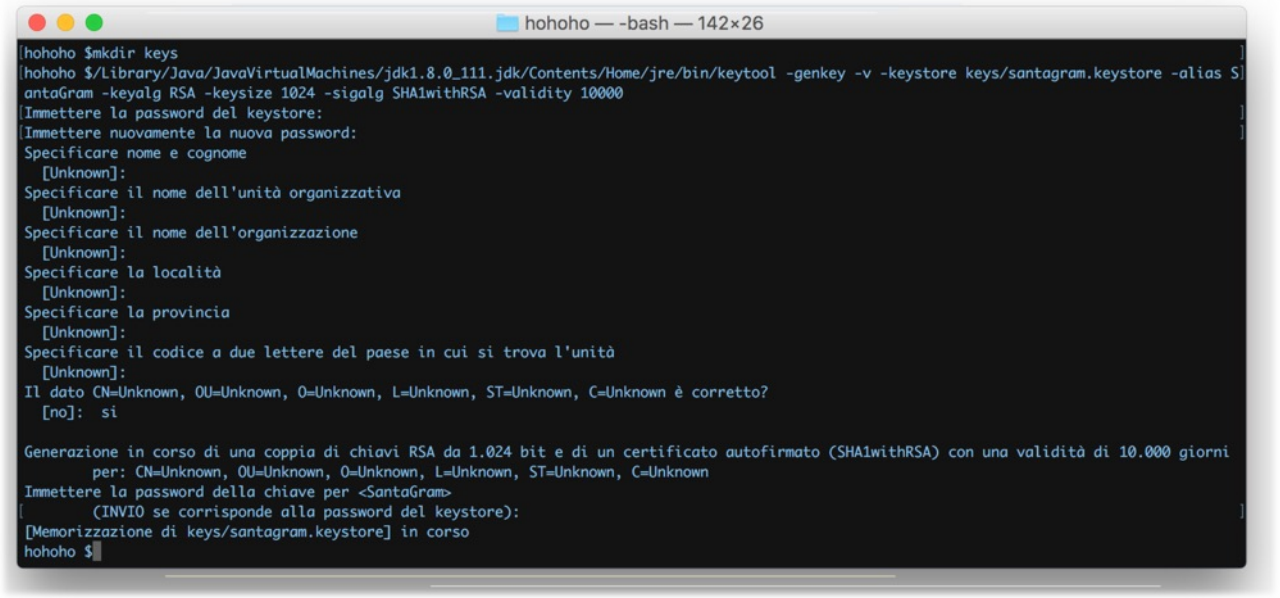
Apktool then generated a new directory called *SantaGram_4.2* that was holding all the application resources. After that, I edited the *strings.xml* file at *SantaGram_4.2/res/values/strings.xml*, changing the *debug_data_enabled* from *false* to *true.*

```
<string name="character_counter_pattern">%1$d / %2$d</string>
<string name="debug_data_collection_url">http://dev.northpolewonderland.com/index.php</string>
<string name="debug_data_enabled">true</string>
<string name="dungeon_url">http://dungeon.northpolewonderland.com/</string>
<string name="exhandler_url">http://ex.northpolewonderland.com/exception.php</string>
```

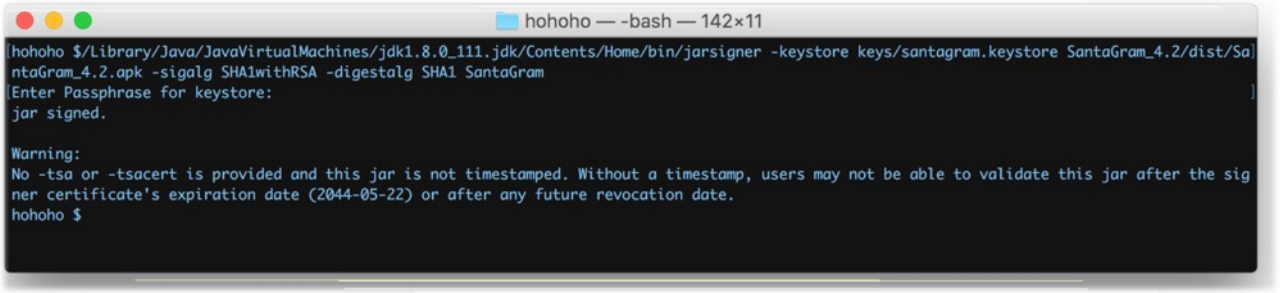
I re-build the app:



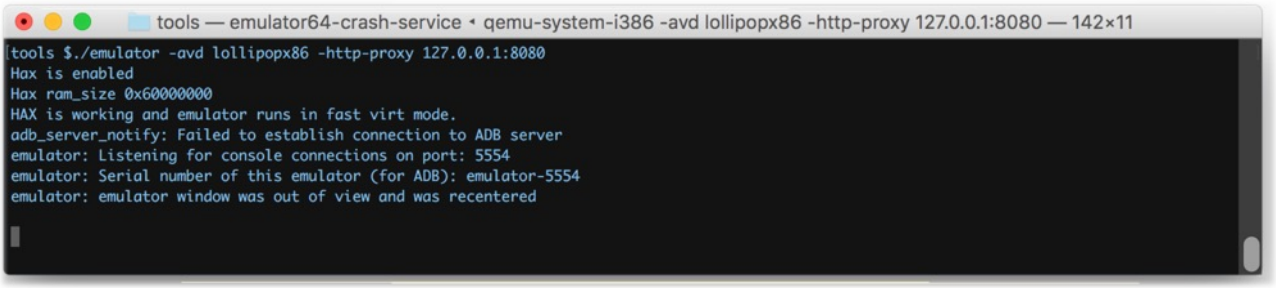
Knowing that I could not install and run the app as it was, because the Android platform requires signed application packages, I proceeded to self-sign it using the *JDK tools* to run the ***keytool*** utility and create my own signing key:



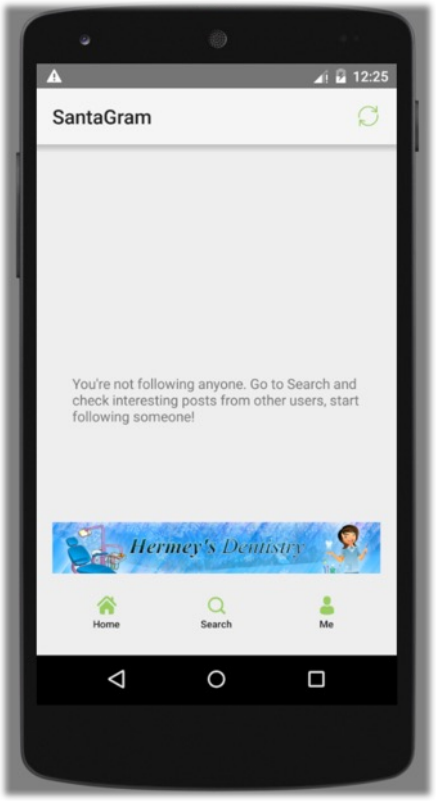
Once I created my key, I used the ***jarsigner*** utility to leverage the key and generate a signed application package:



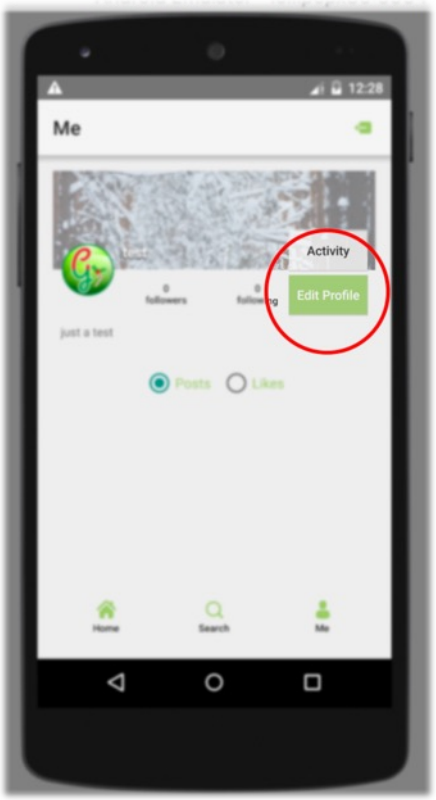
I launched the Android ***emulator***, setting on it a *http-proxy* on *localhost* at port *8080*, to intercept all the http requests generated by the application.



I installed the modified app and then started ***Burp Proxy*** telling it to listen on *localhost* at port *8080*. Once it finished with the installation, I opened the app:



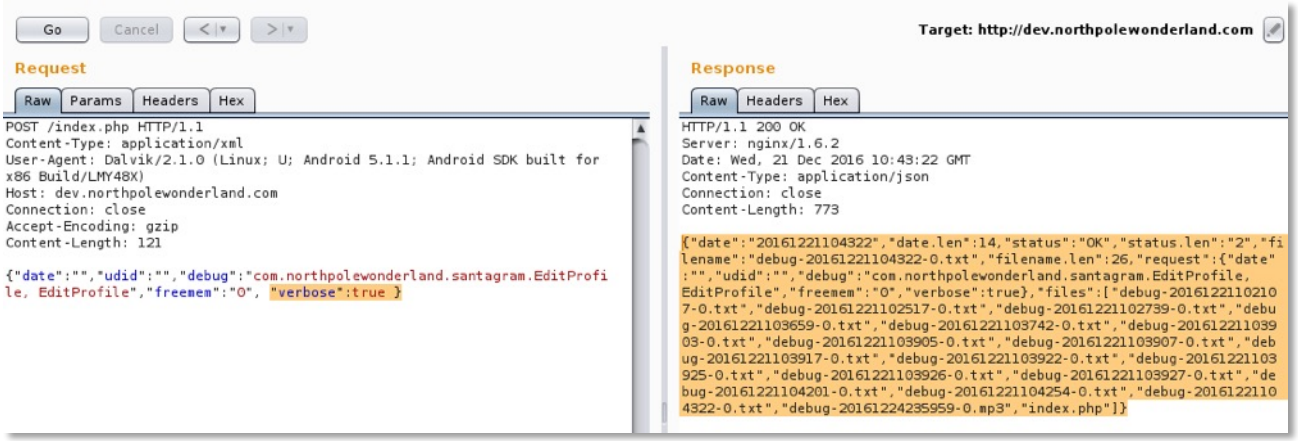
To trigger the debug I needed to edit my profile:



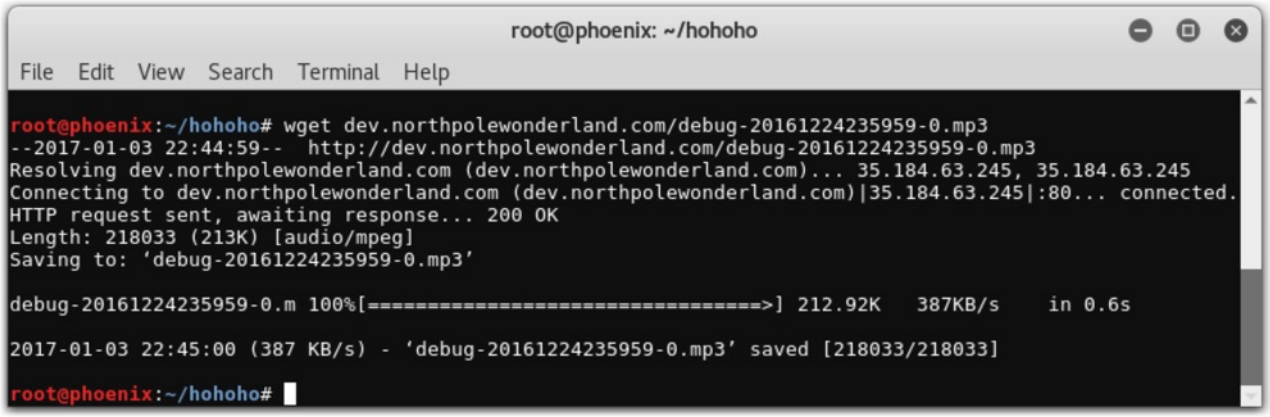
Tapping on the *Edit Profile* button, the app sent the POST request to the Debug server:

I captured that POST request and re-sent it through ***Burp Repeater***. The JSON response contained the filename of the debug file that the server created and a copy of my request as a parameter.

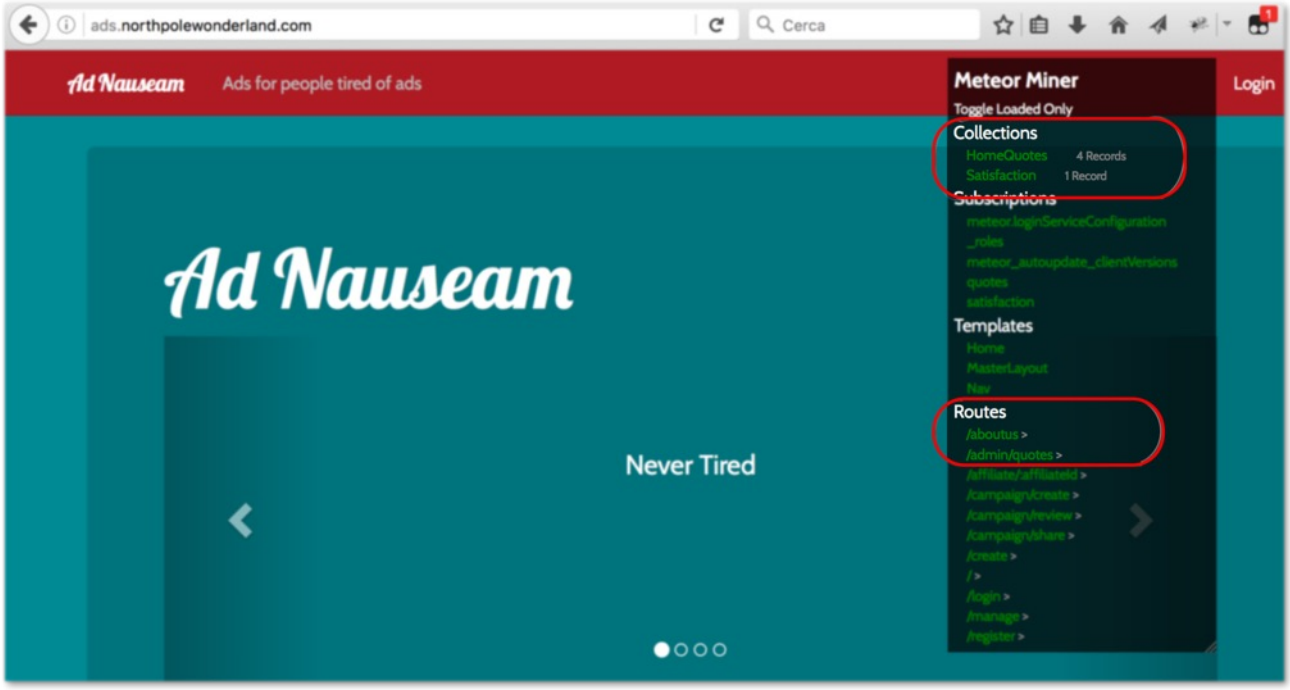
I also noticed a suspicious parameter called ***verbose*** set to *false*. Trying to include that parameter in my request and setting it to *true,* I got a different response.



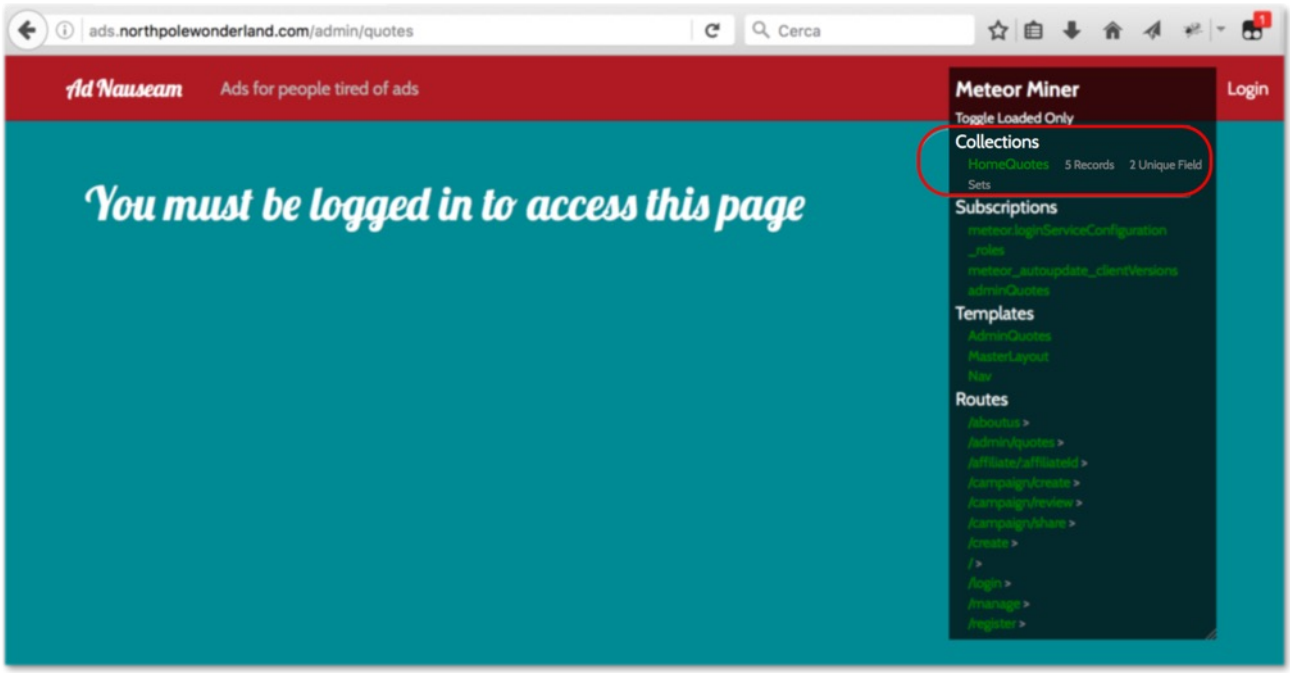
This time the response contained all the files stored in the server. The file named **debug-20161224235959-0.mp3** quickly caught my eyes.
"Kids, I think we found it! I said while downloading the audio file with *wget*.



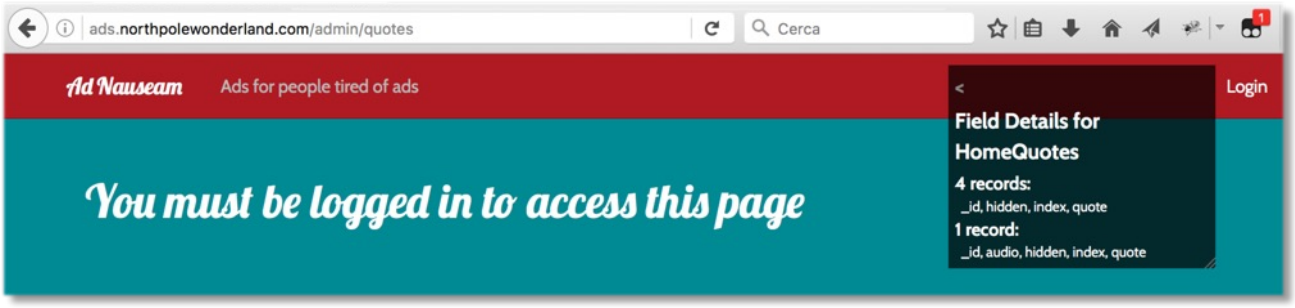
We were making a lot of progress and we got so caught up in the excitement that we immediately moved to the ***AD server***. I visited the URL *ads.northpolewonderland.com* landing onto the *Ad Nauseam* web app and, checking the source code of the page, I noticed a lot of references to the *Meteor Framework.*
Pepper Minstix decided to help us, gaving us an useful resource where we learnt on how to attack the framework.
I installed the ***Tampermonkey*** add-on in my browser and then imported the ***MeteorMiner*** script found on Github and wrote by Nidem (*https://github.com/nidem/MeteorMiner*).
Browsing with the *Tampermonkey* add-on enabled, I saw the ***HomeQuotes*** and ***Satisfaction*** collections. I also noticed that the ***/admin/quotes*** route existed even though there was no link to it in the interface.



I then explored that route and, to my great surprise, I noticed that the *HomeQuotes* Collection was counting 5 entries now:



Clicking on that Collection I found what I was searching for: the information about the audio file were contained in the *HomeQuotes* Collection!



Using the developer's toolbar on the browser, I extracted the *HomeQuotes* collection and, inspecting the 5th element of the collection, I got the path where the audio file was.

```
>> HomeQuotes.find().fetch()
<- Array [ Object, Object, Object, Object, Object ]
>> HomeQuotes.find().fetch()[4]
<- Object { _id: "zPR5TpxB5mcAH3pYk", index: 4, quote: "Just Ad It!", hidden: true, audio: "/ofdAR4UYRaeNxMg/discombobulatedaud…" }
>> HomeQuotes.find().fetch()[4].audio
<- "/ofdAR4UYRaeNxMg/discombobulatedaudio5.mp3"
>>
```

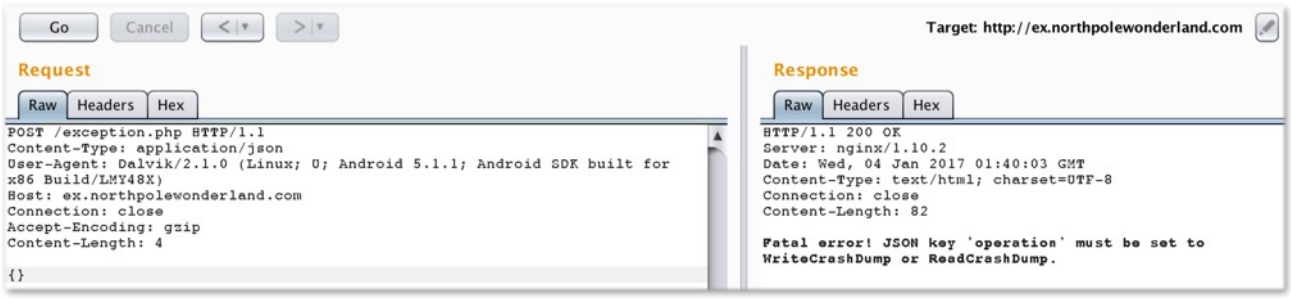
Knowing the path where the audio file was hiding, I downloaded it with my trusty *wget*.

```
root@phoenix: ~/hohoho
File  Edit  View  Search  Terminal  Help
root@phoenix:~/hohoho# wget ads.northpolewonderland.com/ofdAR4UYRaeNxMg/discombobulatedaudio5.mp3
--2017-01-03 23:55:34--  http://ads.northpolewonderland.com/ofdAR4UYRaeNxMg/discombobulatedaudio5.mp3
Resolving ads.northpolewonderland.com (ads.northpolewonderland.com)... 104.198.221.240, 104.198.221.240
Connecting to ads.northpolewonderland.com (ads.northpolewonderland.com)|104.198.221.240|:80... connecte
d.
HTTP request sent, awaiting response... 200 OK
Length: 233357 (228K) [audio/mpeg]
Saving to: 'discombobulatedaudio5.mp3'

discombobulatedaudio5.mp3 100%[=================================>] 227.89K   409KB/s    in 0.6s

2017-01-03 23:55:35 (409 KB/s) - 'discombobulatedaudio5.mp3' saved [233357/233357]

root@phoenix:~/hohoho#
```

Then was the turn of the ***the Uncaught Exception Handler Server***. We already knew what the URL for the Exception server was because we saw it on the *strings.xml* file:

```
<string name="debug_data_enabled">false</string>
<string name="dungeon_url">http://dungeon.northpolewonderland.com/</string>
<string name="exhandler_url">http://ex.northpolewonderland.com/exception.php</string>
<string name="title_activity_comments">Comments</string>
```

I have to admit that, not being able to generate an exception and capture the request sent by the app, I built the POST request from scratch using the response with a trial-error approach.
I started with an empty JSON schema:



Then I set the ***operation*** parameter and gave it a random value:

Thanks to the fact that the error in the response was being particularly verbose, I knew that the value for the *operation* parameter should have been ***WriteCrashDump*** or ***ReadCrashDump***. I started analyzing the case with the *ReadCrashDump value:*

I also set the ***data*** parameter with a random value:

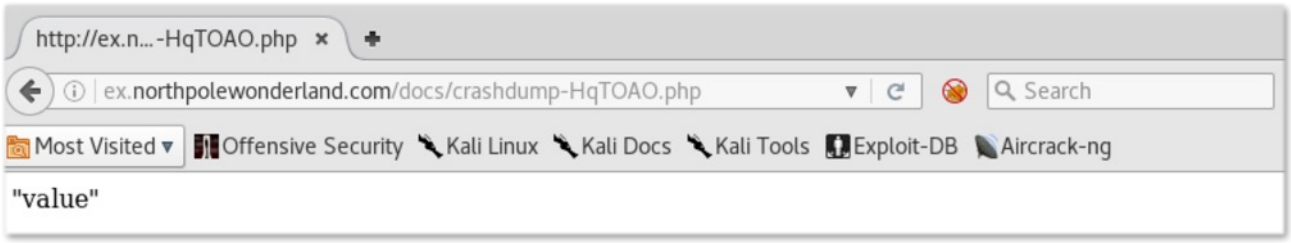
It asked for the ***crashdump*** parameter and I of course set it:

Unfortunately it did not work so I tried to insert the *crashdump* parameter inside the *data* one:

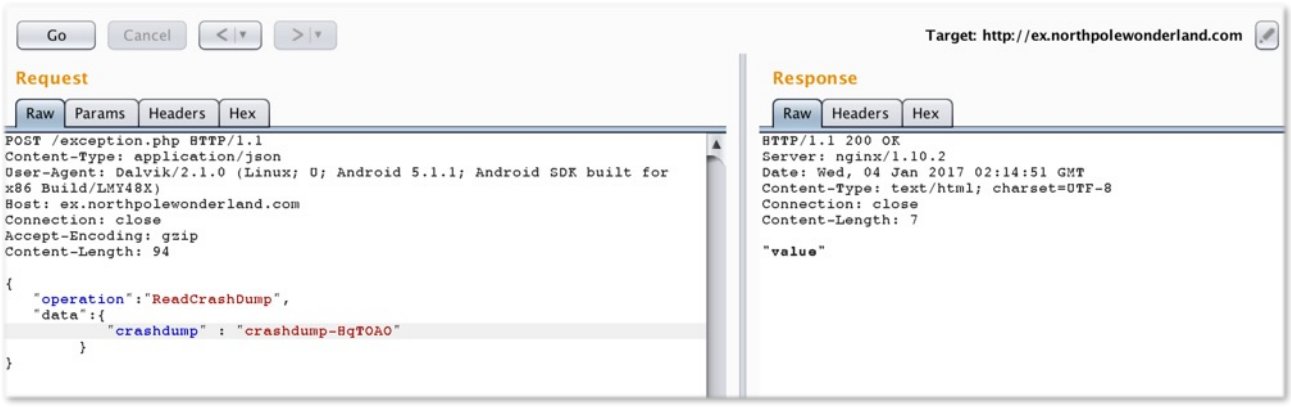
This time it did not respond with errors.
I then proceeded to analyze the *WriteCrashDump* case:

As in the other case, the request was missing the *data* parameter. Inserting that parameter, this time, I got a different response:

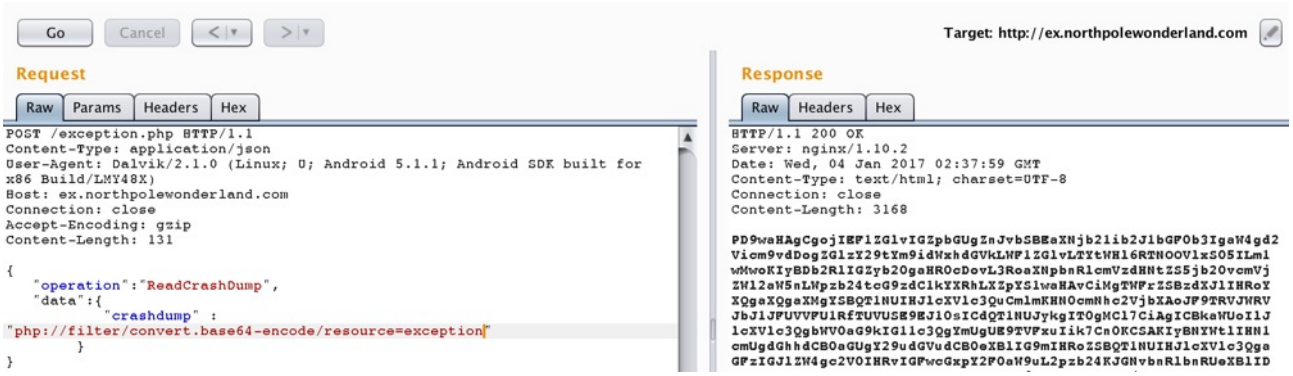
It seemed that the server was saving the crash dumps into something like: */docs/crashdump-$VAR.php. and on that page,* it was displayed the *data* value set in the POST request:



And, at that point, I was also able to read that value using the *ReadCrashDump* request:

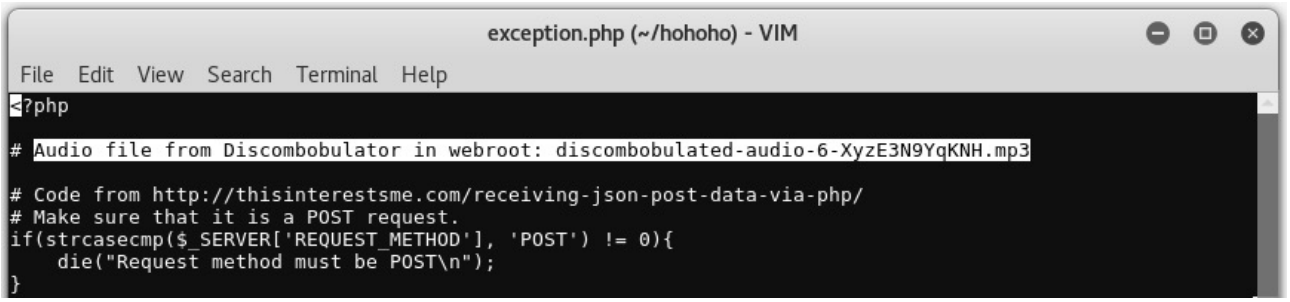


Knowing that I could read all the crashdump files with *ReadCrashDump,* made me wonder if, manipulating the *crashdump* parameter, I would have been able to attack the server using a ***Local File Inclusion Attack***. I then tried the base64-encode PHP filter to exfiltrate the *exception.php* code:

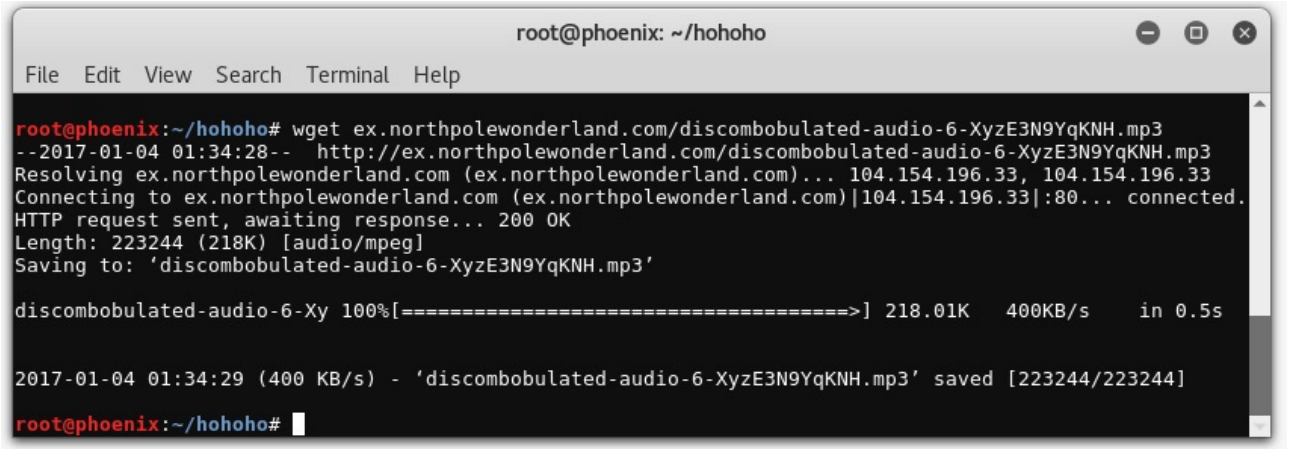


I then copied that base64-encoded string into my terminal and I decoded it, saving it into a new php file:

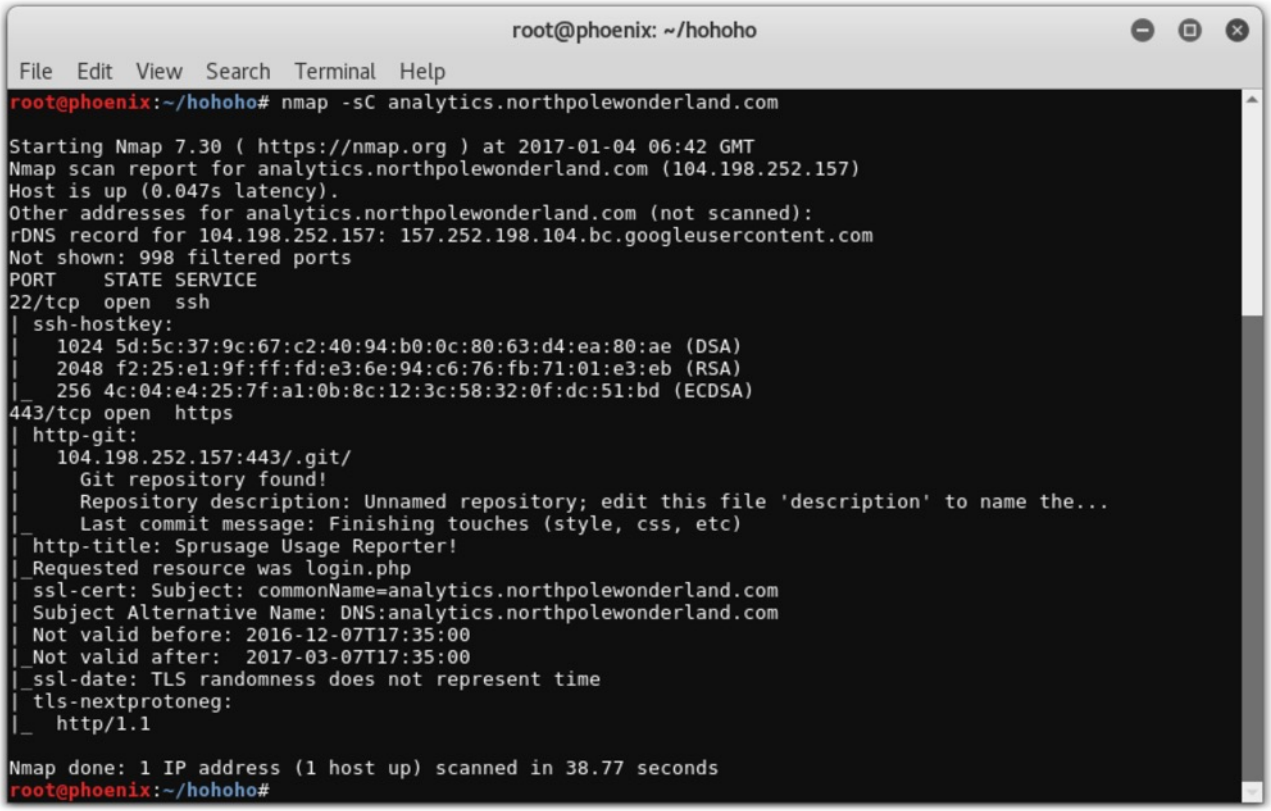
Once decoded it, I opened it with *vim*:



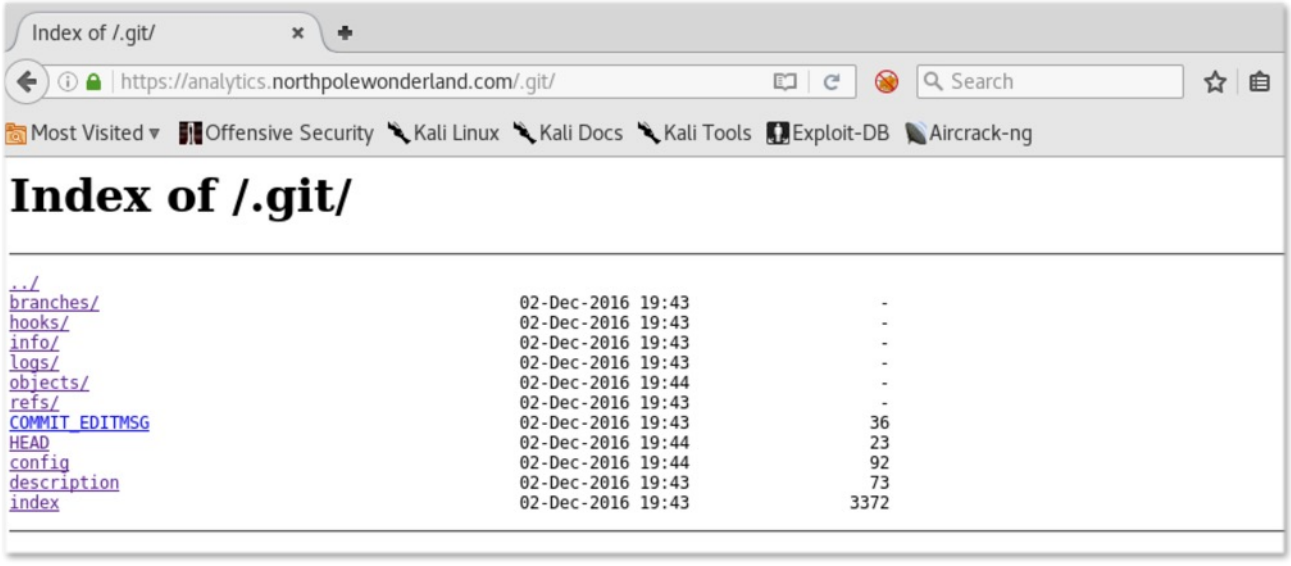
Reading the comment at the top of the source code, I could download the audio file using *wget*:



"We got 'em all!" Said Josh while bouncing to celebrate that moment.
"Curb your enthusiasm Joshua." I said, putting my hand on his shoulder, and then I continued: "I have the strange feeling that we forgot something in the Analytics server"
An *nmap* scan on the Analytics server confirmed my suspects:



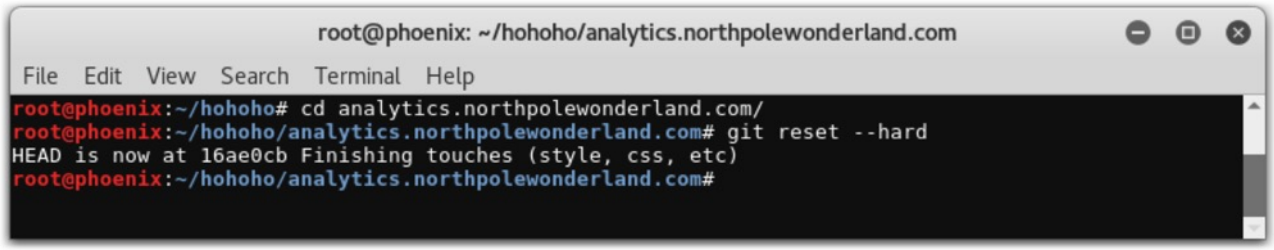
We come across to a publicly accessible .git repository. To check its content I browsed to the *.git* directory:



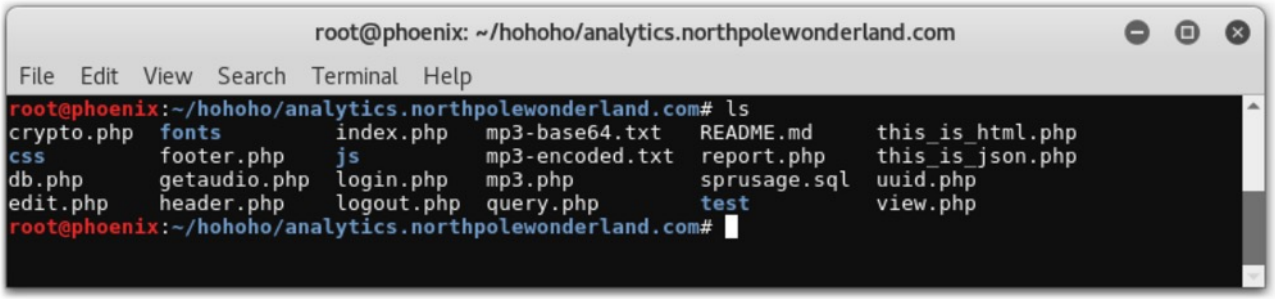
The next thing that I did was to pull down the entire *.git* directory along with all the files and folders within it. I used this ***wget*** command to do that:

wget -m https://analytics.northpolewonderland.com/.git --no-check-certificate

I then issued a *git reset:*



Getting the entire source code of the web application.



Being able to analyze the web application source code I noticed that, in the *header.php*, the *administrator* could see the *edit.php* page in the menu bar:

```
<?php
  if (get_username() == 'guest') {
    ?>
      <li><a href="/<?= mp3_web_path($db); ?>">MP3</a></li>
    <?php
  }
  if (get_username() == 'administrator') {
    ?>
      <li><a href="/edit.php">Edit</a></li>
    <?php
  }
?>
```

On the *edit.php* page, the developers used the *restrict_page_to_users* in order to restrict access to that page:

```
# This should be the first require
require_once('this_is_html.php');
require_once('db.php');

# Don't allow anybody to access this page (yet!)
restrict_page_to_users($db, []);
```

And, checking the *restrict_page_to_users* function in *this_is_html.php* source code, I saw that It was calling the *check_access* function.

```
function restrict_page_to_users($db, $users) {
  $username = get_username();

  if(!$username) {
    header('Location: login.php');
    exit(0);
  }

  check_access($db, $username, $users);
}
```

That function was defined in the *db.php* code and, interestingly, it let the *administrator* access any page!

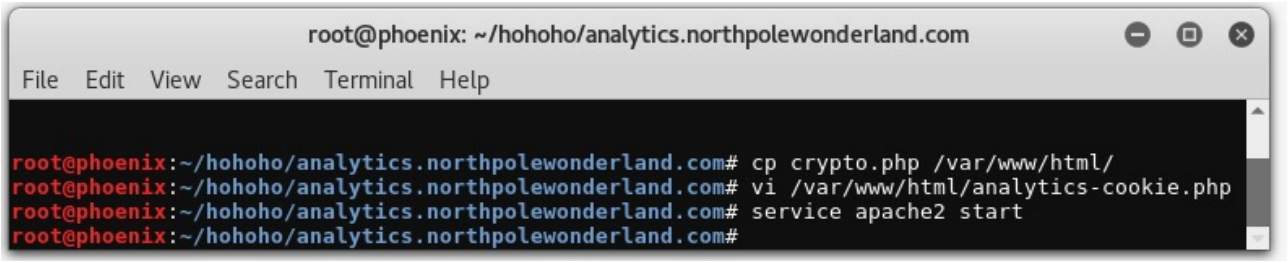
```
function check_access($db, $username, $users) {
  # Allow administrator to access any page
  if($username == 'administrator') {
    return;
  }
```

At that point it was extremely clear what I had to do: I had to find a way to log in as the as *administrator*. To identify the right way to do that, I analyzed the *login.php* page and I saw that I could have crafted an AUTH Cookie impersonating the *administrator*.

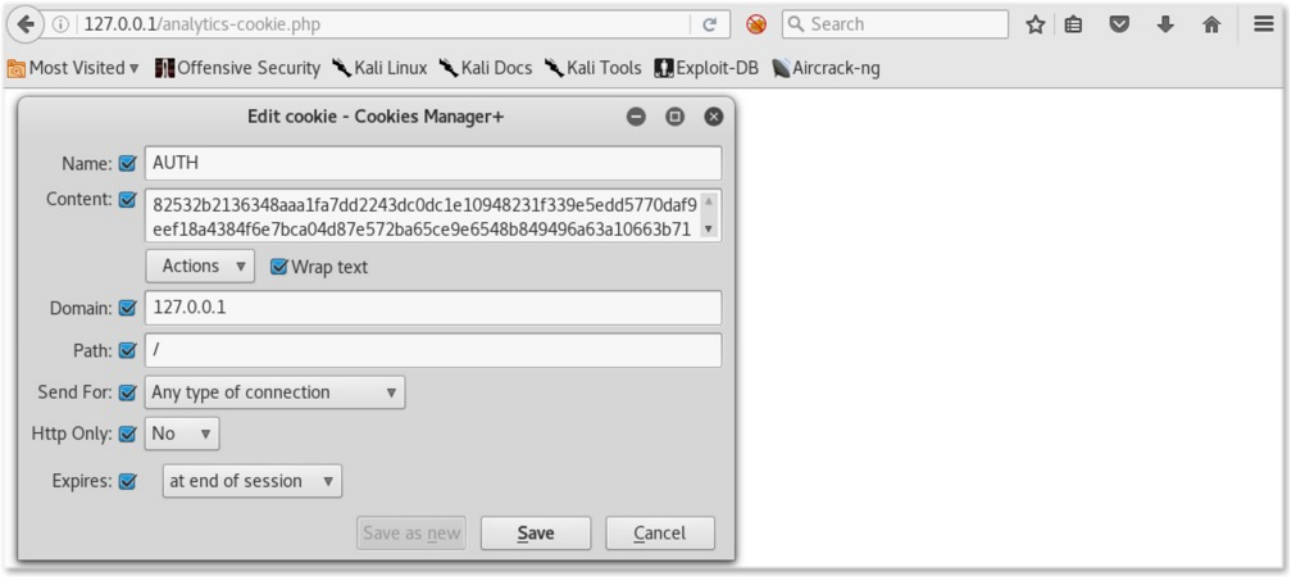
```
require_once('db.php');

check_user($db, $_POST['username'], $_POST['password']);

print "Successfully logged in!";

$auth = encrypt(json_encode([
  'username' => $_POST['username'],
  'date' => date(DateTime::ISO8601),
]));

setcookie('AUTH', bin2hex($auth));
```

Being a very lazy person, I simply copied the *crypto.php* page on my local web server, and wrote a simple *analytics-cookie.php* page that contained that part of the *login.php* code used to generate the AUTH Cookie:

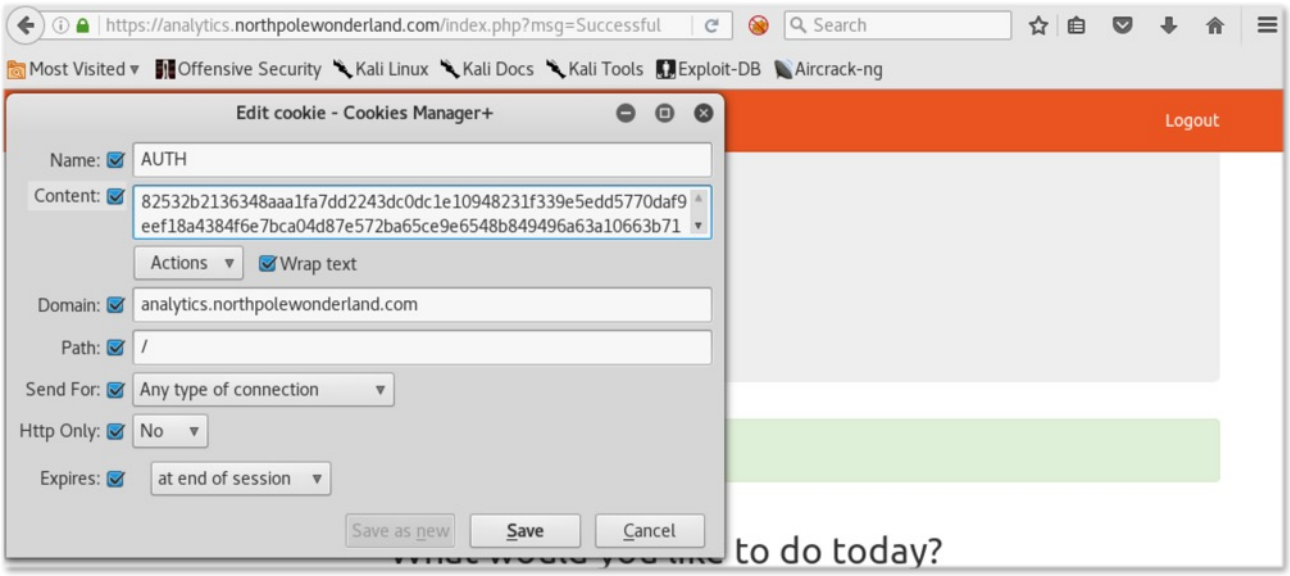
```
<?php
	header("content-type: text/html; charset=UTF-8");
	require_once('crypto.php');
	$auth = encrypt(json_encode([
		'username' => 'administrator',
		'date' => date(DateTime::ISO8601),
	]));
	setcookie('AUTH', bin2hex($auth));
?>
```



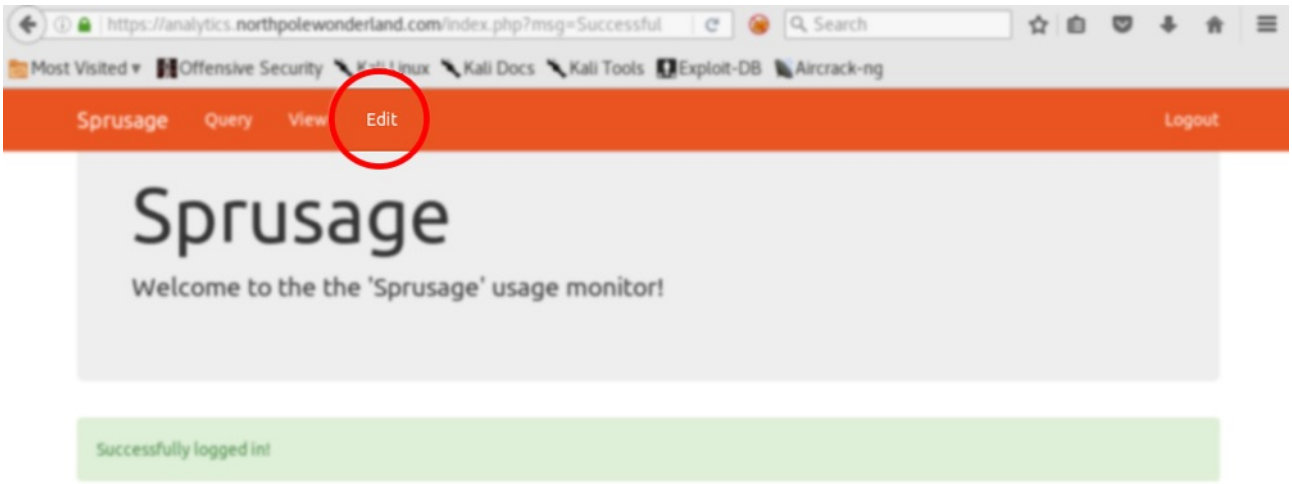
I then started my *apache2* web server and I browsed to the *127.0.0.1/analytics.cookie.php* page, copying the content of the AUTH Cookie using the *Cookie Manager+ Firefox Add-on:*



After that, I logged in using *guest* credentials and substituted the value of the AUTH Cookie for the *analytics.northpoledomain.com* domain with the one generated by my *analytics-cookie.php* page.

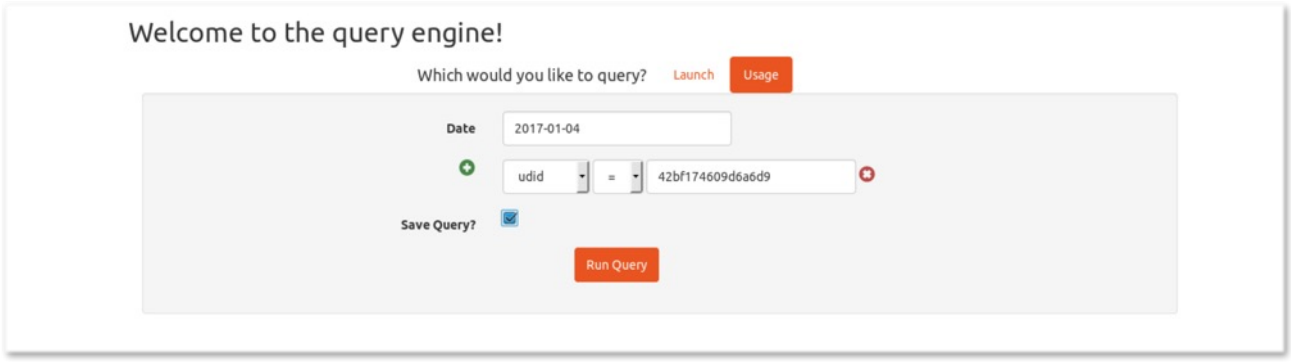


Reloading the page, I could log in as *administrator* and, as I supposed, the menu bar allowed me to browse to the *edit.php* page.



"So far so good" I tought. The next step would have been to understand the whole report handling process in order to be able to find the vulnerability. To perform that task, I captured a POST request coming from the SantaGram app.

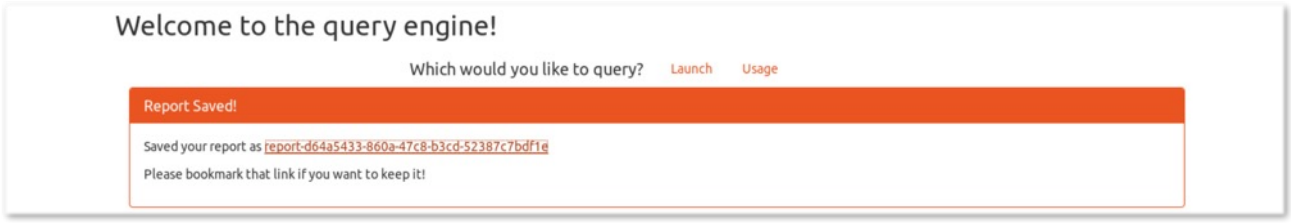
And, on the web app, I created and saved a new query to display all the reports sent by the SantaGram app with a specific *udid:*



By looking at the *query.php* code I saw that, if the *save* parameter was set into the POST request, then a new entry was added to the *reports* table, containing *id, name, description* and *query:*

```
$query = "SELECT * ";
$query .= "FROM `app_" . $type . "_reports` ";
$query .= "WHERE " . join(' AND ', $where) . " ";
$query .= "LIMIT 0, 100";

if(isset($_REQUEST['save'])) {
  $id = gen_uuid();
  $name = "report-$id";
  $description = "Report generated @ " . date('Y-m-d H:i:s');
  $result = mysqli_query($db, "INSERT INTO `reports`
    (`id`, `name`, `description`, `query`)
  VALUES
    ('$id', '$name', '$description', '" . mysqli_real_escape_string($db, $query) . "')
    ");
```

Its report could then be viewed following the link given by the *query.php* page:

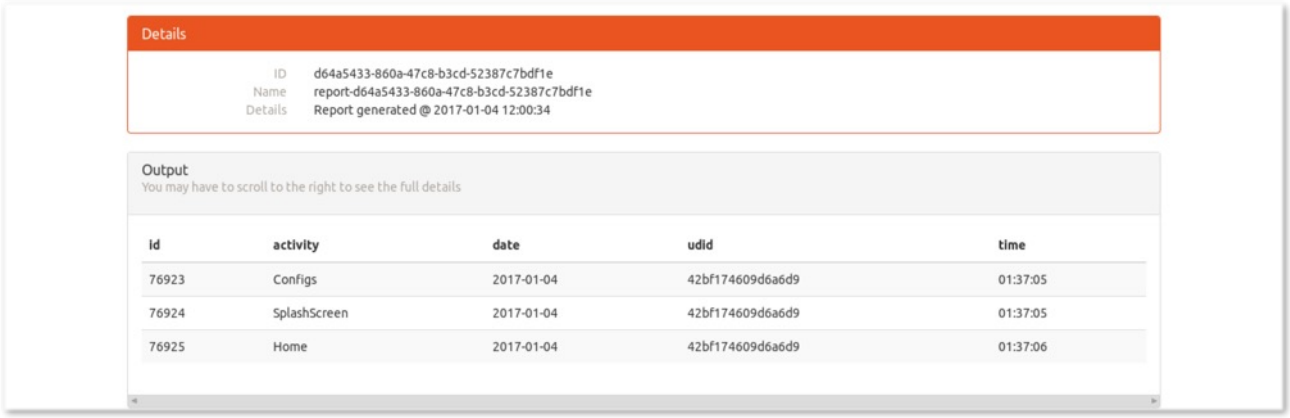


The query report was displayed in the *view.php* page, calling the *format_slq* function and passing it the results of the MySQL Query performed by the *query* function.

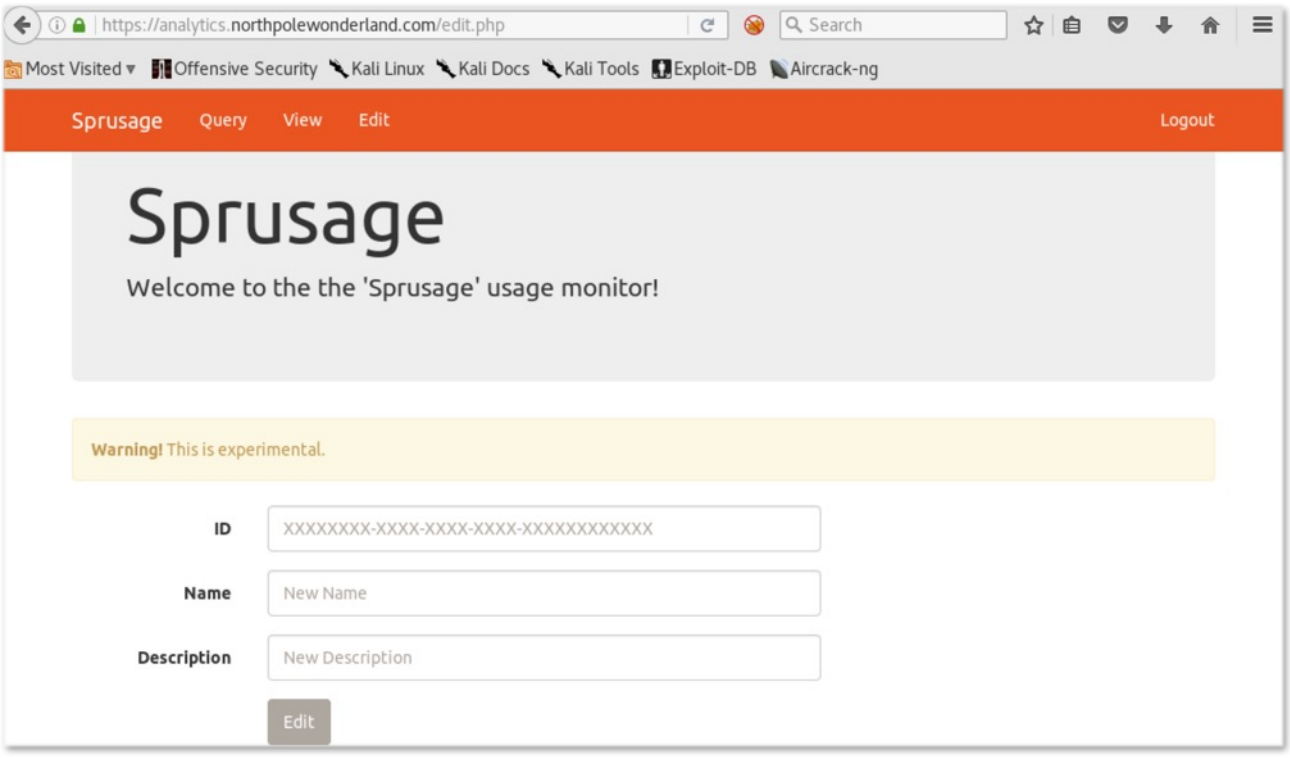
```
<?php
  format_sql(query($db, $row['query']));
}
```

The *query* function was using the value of the *query* attribute saved in the *reports* table for that specific *report id*. Then the *format_slq* function, defined in the *db.php* page, displayed all the rows got from the resulting query within a table.

```
<tbody>
  <?php
    foreach($rows as $row) {
      ?><tr><?php
      foreach($headers as $header) {
        //$out .= str_pad($row[$header], 15) . ' ';
        ?><td><?= htmlentities($row[$header]); ?></td><?php
      }
      ?></tr><?php
    }
  ?>
</tbody>
```



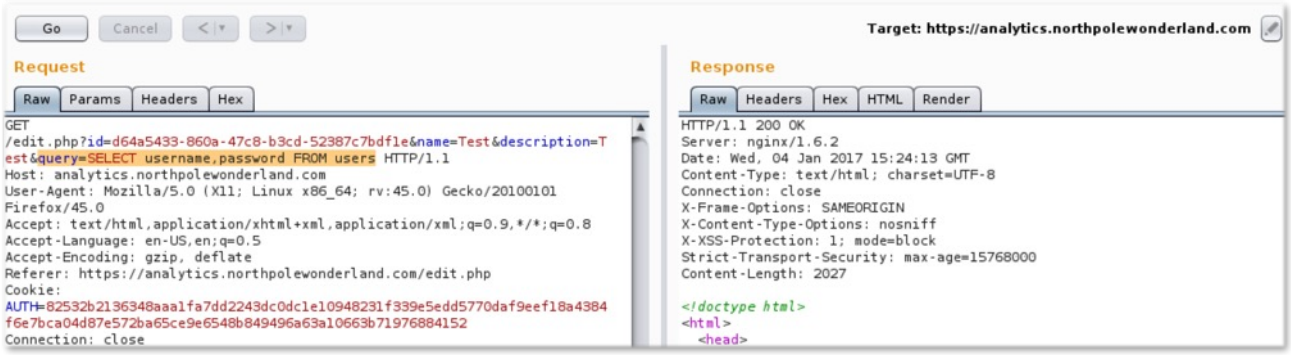
And it was at that point that the *edit.php* page came into play! That page allowed me to modify saved query reports.



Analyzing the *edit.php* code I saw that, to build the SET part of the UPDATE query, it was using a foreach statement in which it was grabbing all the parameters received with the GET request.

```
# Update the row with the new values
$set = [];
foreach($row as $name => $value) {
  print "Checking for " . htmlentities($name) . "...<br>";
  if(isset($_GET[$name])) {
    print 'Yup!<br>';
    $set[] = "`$name`='" . mysqli_real_escape_string($db, $_GET[$name]) . "'";
  }
}

$query = "UPDATE `reports` " .
  "SET " . join($set, ', ') . ' ' .
  "WHERE `id`='" . mysqli_real_escape_string($db, $_REQUEST['id']) . "'";
print htmlentities($query);

$result = mysqli_query($db, $query);
```

"Eureka! We have a 2nd Order SQL Injection kids!" I shout. The *edit.php* page was not restricting the number of parameters sent with the GET request and so I could append the *query* parameter with my custom MySQL query and I could execute it in the *view.php* page.
Having discovered the injection point, I built the attack. First of all, using Burp, I intercepted the POST request performed the *edit.php* page for a particular report id. I then replied it using Burp Repeater and I inserted my malicious *query* parameter.
I immediately tried to dump usernames and password from the *users* table, modifying the GET request as shown below:



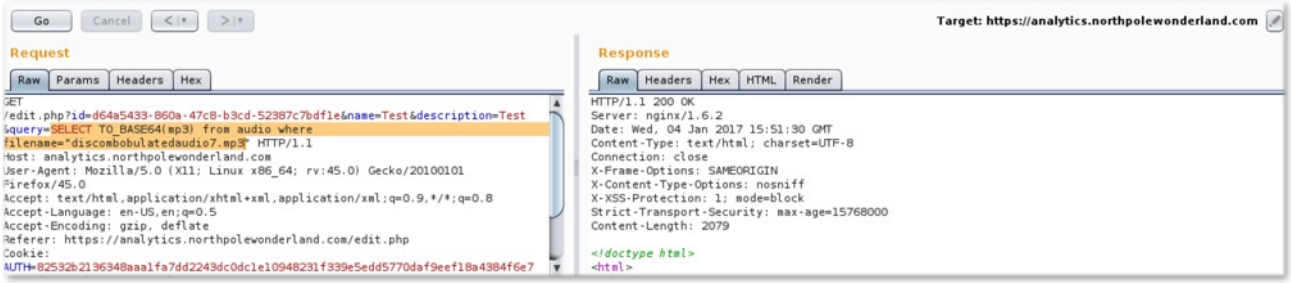
I performed the GET request to the *view.php* page to execute the malicious query, getting back in the response all the credentials stored in the *users* table and confirming that the attack was successful.

I then enumerated the *audio* table with the *query* parameter: ***SELECT * from audio***, spotting the *discombobulatedaudio7.mp3* in the *view.php* GET response.

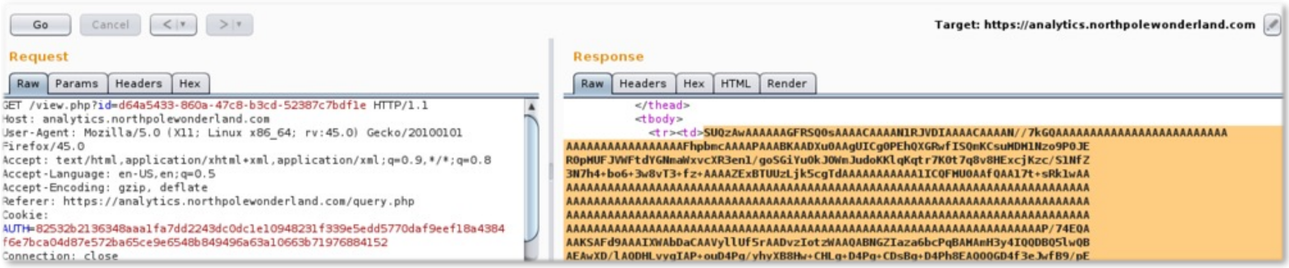
By looking at the *audio* table schema at *sprusage.sql*, I saw that ***mp3*** was the attribute where the file was stored.

```
CREATE TABLE `audio` (
  `id` varchar(36) NOT NULL,
  `username` varchar(32) NOT NULL,
  `filename` varchar(32) NOT NULL,
  `mp3` MEDIUMBLOB NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;
```

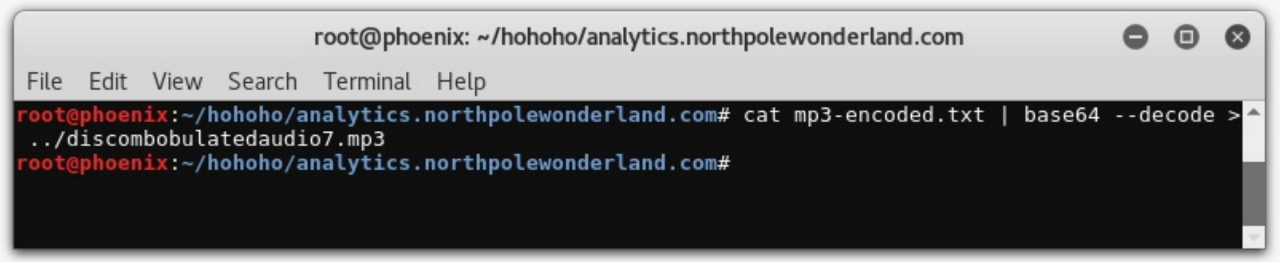
Then I crafted my query to select the *mp3* attribute of the *discombobulatedaudio7.mp3* and, being *MEDIUMBLOB* the Data Type, I base64-encoded the result of the query:



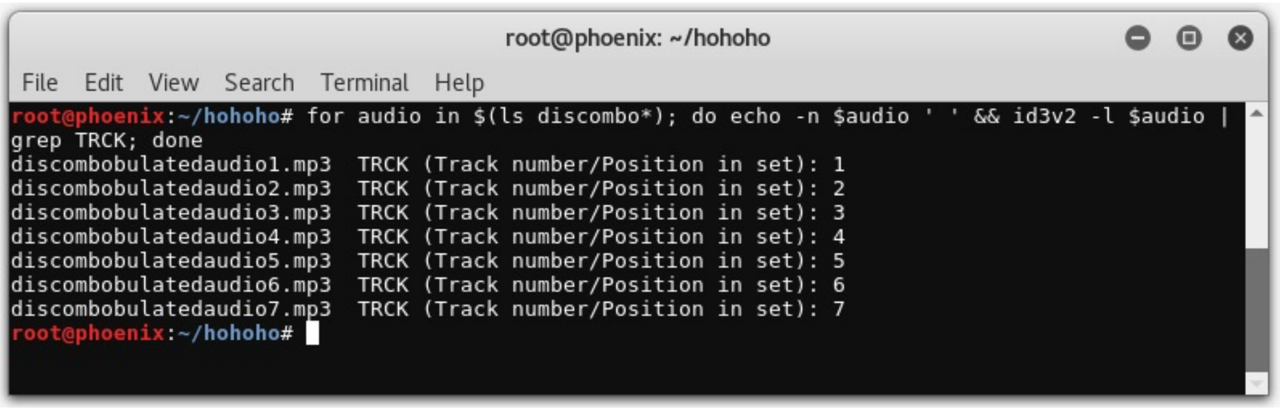
Performing the GET request to the *view.php* page game me the base64-encoded content of the ***discombobulatedaudio7.mp3*** file.



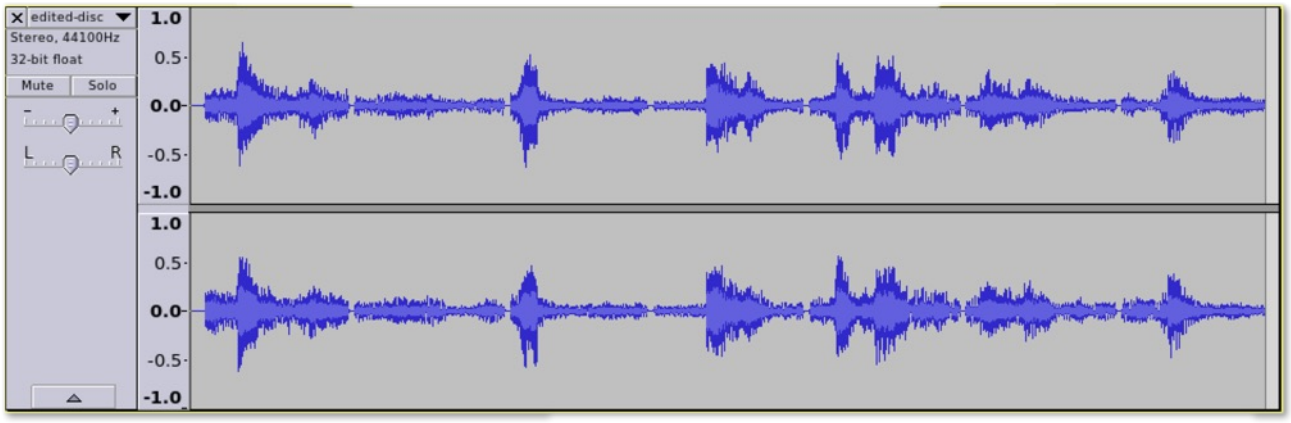
I saved the entire base64-encoded string into a file; then I decoded and saved it.



“Yayyyyyyy! We found them all!” Screamed the kids!
There was no time to celebrate, we needed to reassemble all the audio files!
To find the right order I used the *id3v2* tool to read the mp3 tags:



So, apparently, the order with which the audio files were named, was already the right one. I fired Audacity and combined all the audio files but, unfortunately, we still could not understand what it was saying.



"It seems that the audio track is too slow, I think we should increase the tempo" Said Josh. I agreed with him and proceeded to ***increase the tempo to 600*** and also applied a noise reduction filter.

We were starting to hear something: *..."Father"..."Christmas"..."Santa"..."Claus"...*A couple of other incomprehensible words and finally*..."Jeff"*. These words... I had already heard them somewhere.

"This could be the passphrase for the only blocked door left!" exclaimed Jess.

I tried hard to remember where I had heard them before and, suddenly, I had a brainwave:

"That's something that Doctor Who said!" I thought.

I searched on the Internet and I found the exact same phrase:

Father Christmas, Santa Claus. Or, as I've always known him, Jeff.

It fitted perfectly with what we heard but there was only one way to figure out if we were right; we should have tested it on the blocked door.

We approached the door in the corridor behind the bookshelf at Santa's office, reached the keypad and inserted the passphrase.

Nothing happened.

We were discomforted by our failure and, while stepping away from the door, I said:

"We were wrong kids, the passphr..."

<<CLANG!>>

Suddenly the door opened.

"Yeees! We made it! We opened the last door!"

We rushed into a windowless room. In the center of the room there was a long ladder that lead to the attic. We climbed that ladder and it seemed it was not going to have an end.

Once we reached the end of the ladder we raised our heads and we saw a giant clock full of moving gears and, on the left side of it, we noticed that someone was hiding in the darkness. Then the figure came out from the darkness and spoke:

“The question of the hour is this: Who nabbed Santa? The answer? Yes, I did.”

Then ***Doctor Who*** continued: “I have looked into the time vortex and I have seen an universe in which the Star Wars Holiday Special was NEVER released. In that universe, 1978 came and went as normal. No one had to endure the misery of watching that abominable blight. People were happy there. It’s a better life, I tell you, a better world than the scarred one we endure here.”

The Doctor did a pause, and spreading out his arms added:

“So I did what I had to do. I knew that Santa’s powerful North Pole Wonderland Magic could prevent the Star Wars Special from being released but Jeff refused to come with me, insisting on the mad idea that it is better to maintain the integrity of the universe’s timeline. So I had no choice – I had to kidnap him.”

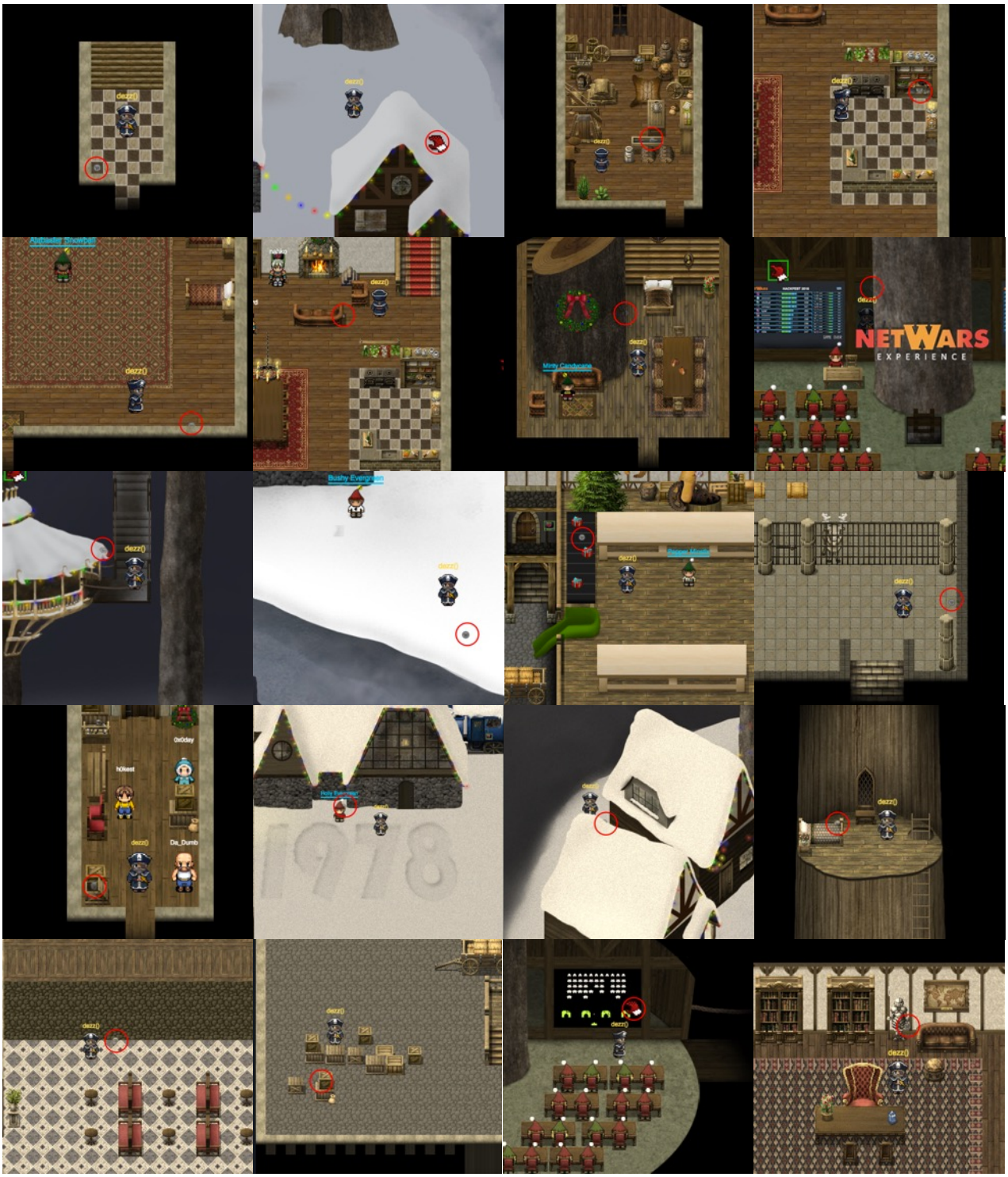
I could not give him all the wrongs, that Special was awful and it was the worst two hours of television ever made; if it were for me I would have not even brought him to justice. But, before I could utter a word, the kids had already contacted their father's friends in law enforcement to ensure the villain would pay for his crime

“This was a twist that I did not expect.” I said while looking at the Doctor being escorted out by the police.

“And I also bet that I will not get paid, Am I right kids?”

The kids smiled awkwardly.

While we were heading out of the village, we came across an elf who had lost his Netwars coins and who was asking desperately for help in order to find them. I had initially no intention to embark in this quest, but then I thought “What the hell! It is Christmas and we are all supposed to be better!” The Christmas spirit must have given me in the head and, still a little reluctantly, We started the search.



Luckily, even if with a bit of hard work due to the snow that kept falling and covering everything with its white softness, all the 20 coins have been found and handed back to the dejected elf who started smiling again.
It was finally over; I was exhausted and satisfied.
“Let’s go home kids. Robert Downey Jr is there waiting for me”