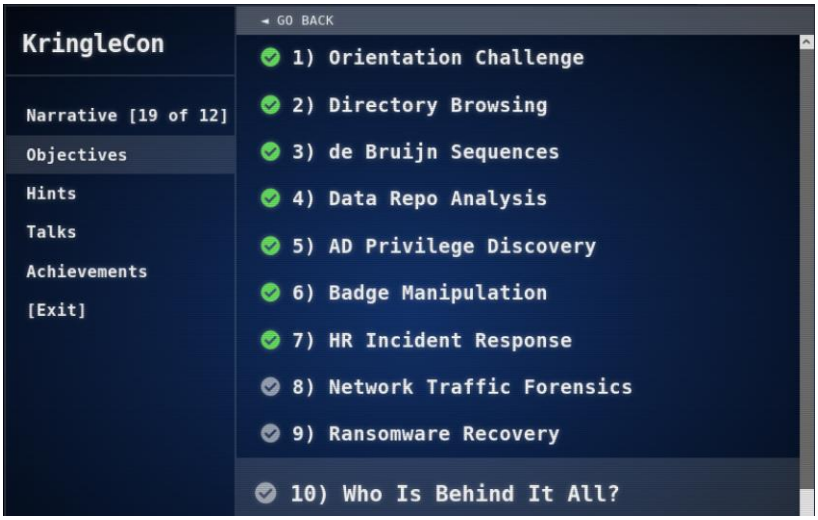
KringleCon 2018 Walkthrough

By: Akvile Kiskis

KringleCon is a free, online security conference hosted by SANS. In addition to the conference, there is a Holiday Hack Challenge that correlates with the talks given at the conference. You can create a character once you login and actually navigate throughout Santa's castle to complete all of the activities (see screenshot below).

I originally started the challenge on the day it opened, but the server was so overloaded, I kept getting kicked out. I only managed to complete the first challenge before getting irritated with the disconnects. I figured I would just wait until it died down and then somehow January 13th rolled around (they ask us to submit our write ups for a contest on the 14th) and I still had nothing done. Long story short, I marathoned as much as I could on that Sunday and part of Monday to meet the deadline. (Don't

do what I did.) Please, actually take the time to enjoy the conference and the awesomeness of what they created.
At this present moment, I have 7 challenges done, but if the servers stay open, I'll keep working on them and updating this write up.



Proof that I'm not just some shmuck who says they did this

Since this was a hefty list of objectives, I created a table of contents in case if anyone wants to look at certain objectives and not have to scroll on forever. That's also why I made this write up on a Google doc instead of directly on to my website.

Table of Contents

1) Orientation Challenge
2) Directory Browsing
3) de Bruijn Sequences
4) Data Repo Analysis
5) AD Privilege Discovery
6) Badge Manipulation
7) HR Incident Response

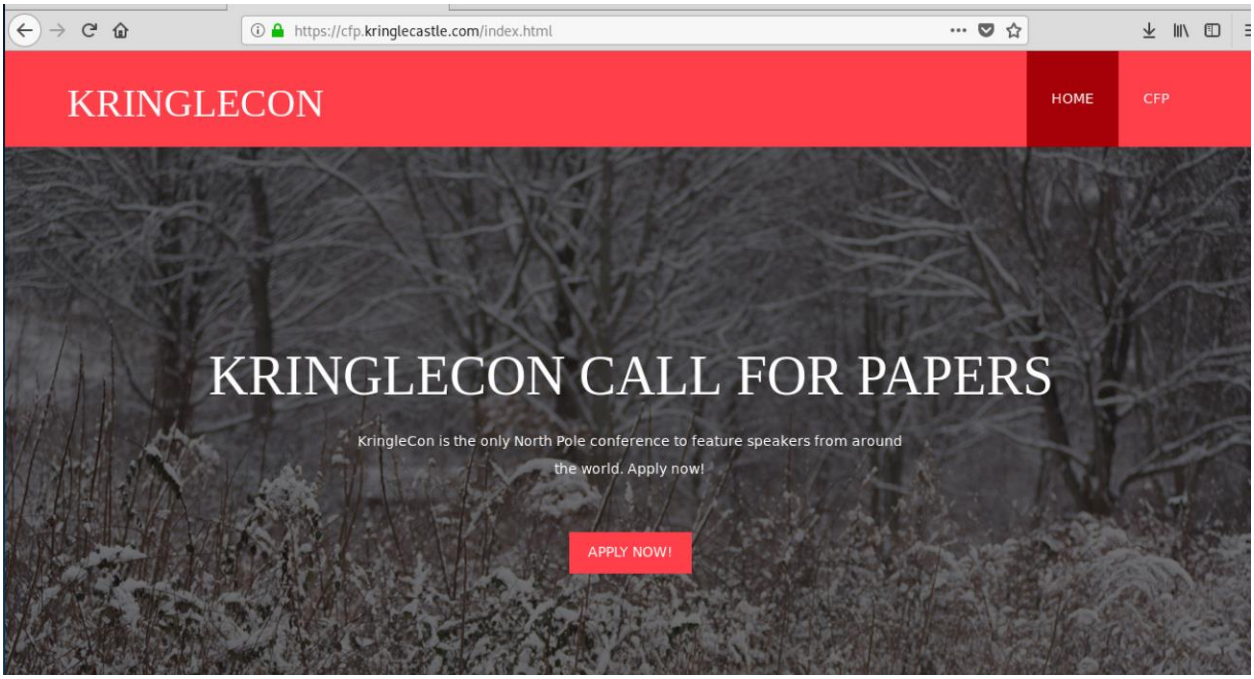
Note: All of the objectives have an optional elf challenge associated with it. My understanding is that the elf challenges were there to give you clues on how to complete the objective at hand. I did not complete all of the elf challenges, but for the ones I did, I separated each of the objectives into the elf challenge portion and flag portion so it would be easier to read.

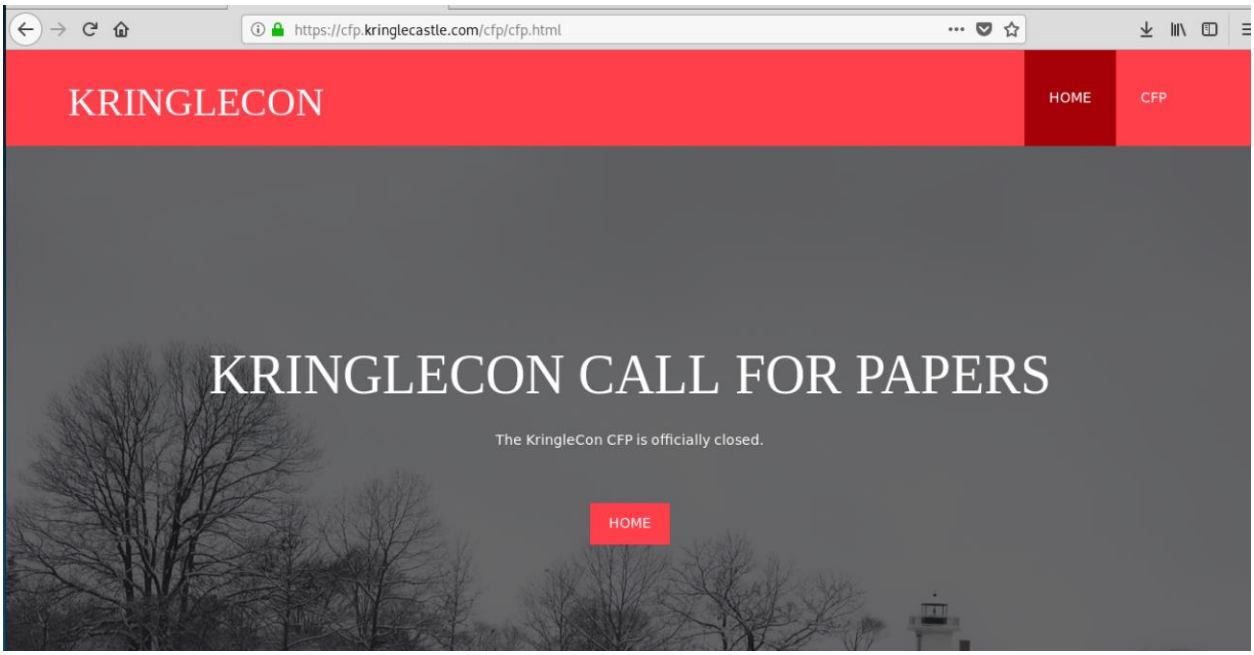
Orientation Challenge

This one was the least technical out of all of them (which makes sense for an orientation) so I won't go into this one too much. Basically, there are several questions that you need to answer based off of Ed Skoudis's talk to get the flag. There is also an intro challenge demonstrating how to use their cranberry pi terminals that you see next to all of the elves in the castle. The elf (whose name escapes me) said she was stuck in vim and didn't know how to exit, so I had to exit vim to complete the challenge. This is as simple as entering **:q!** which is the command to exit vim without saving. Trust me, the rest of the challenges aren't this easy.

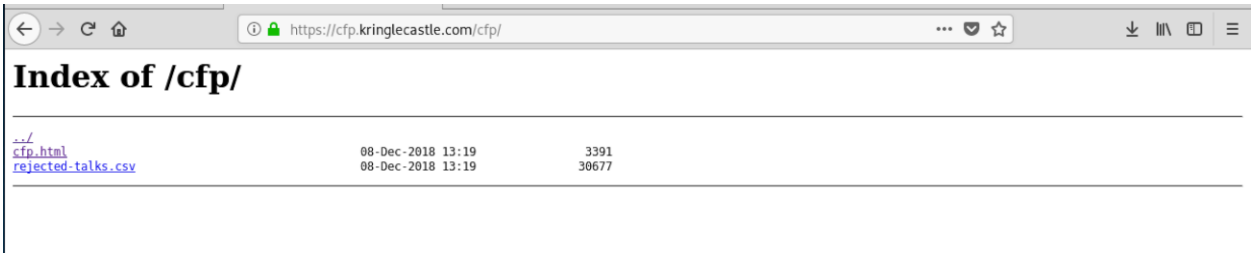
Directory Browsing

The purpose of this challenge is to find a csv file hidden somewhere on this web page to get the flag.





The first screenshot is the page that you're greeted with and the second page is what happens when you click *"Apply Now"*. There isn't much wiggle room for anything in terms of navigating elsewhere. If you look at the URL on the second web page, you can see that there is a */cfp/* directory. I was curious to see what happened if I navigated here.



Well, that was easy, the csv is right there. I must be a l337 hax0r now (I'm being sarcastic).

Flag: John McClane

Jump to table of contents

de Bruijn Sequences

Elf Challenge

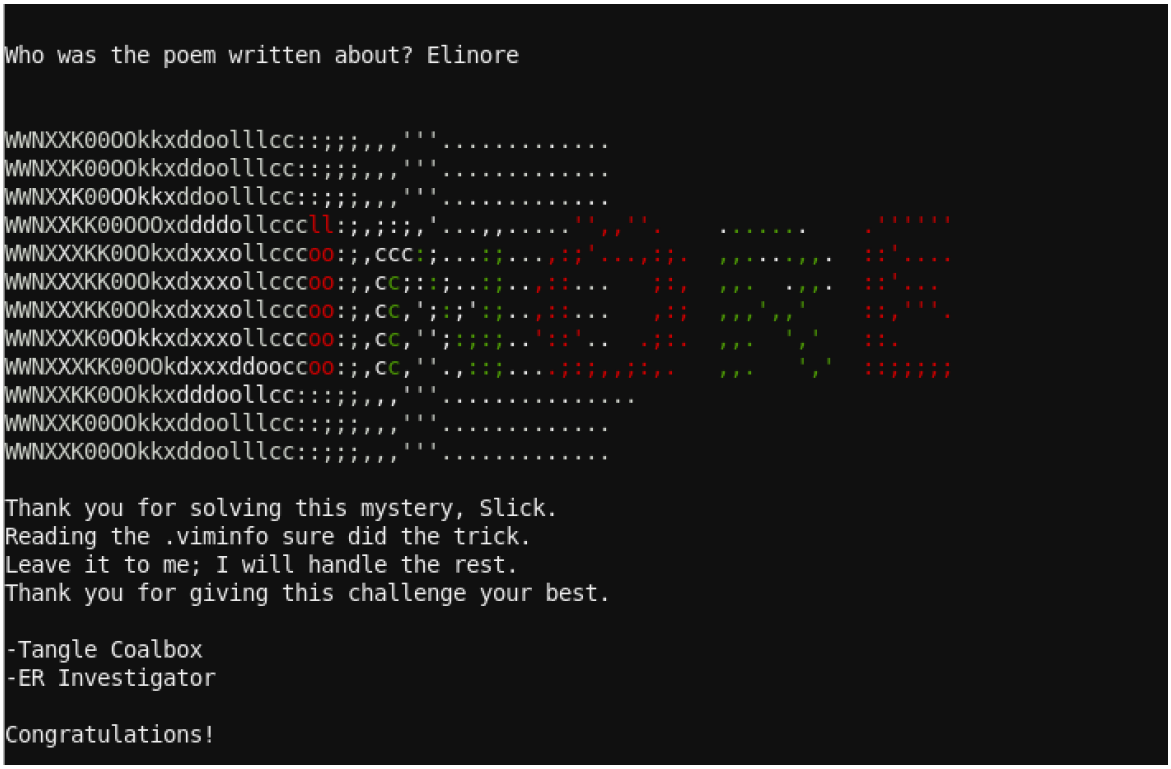
This was another vim challenge. An elf wrote a love poem to someone on this machine and we need to find out who it was. Naturally, my first command was **history | vim -** to see what commands were entered on the machine.

```
  1  set -o history
  2  whoami
  3  echo "No, really...  /-:"
  4  mkdir -p .secrets/her/
  5  firefox https://www.google.com/search?q=love+poetry
  6  vim
  7  ls -lAR
  8  exit
  9  set -o history
 10  df -h
 11  who
 12  firefox https://www.google.com/search?q=replacing+strings+in+vim
 13  time vim
 14  ls -lAR
 15  exit
 16  set -o history
 17  vim
 18  exit
 19  ls -lA
 20  cat .bash_history
 21  echo "" >> .bash_history
 22  firefox https://www.google.com/search?q=turn+off+bash+history
 23  set +o history
 24  set +o history
 25  ls
 26  file runtoanswer
 27  cd runtoanswer
```

First, I can see where the file is because the elf created a directory called **./secrets/her/** and then there was a firefox search for love poetry right after. He also kept deleting his history by using those **set** commands (sneaky). When I navigated to the poem, the name wasn't there (obviously, that would be way too easy), but there is a way to see what commands were used recently on vim. All you have to do is press the up key and you can see the previous commands.

```
Smile, she did, when he suggested that their future surely rested,
Up in flight above their cohort flying high like ne'er before!
  So he harnessed two young reindeer, bold and fresh and bearing no fear.
In they jumped and seated so near, off they flew - broke through the door!
Up and up climbed team and humor, Morcel being so adored,
  By his lovely NEVERMORE!

-Morcel Nougat

:%s/Elinore/NEVERMORE/g
```

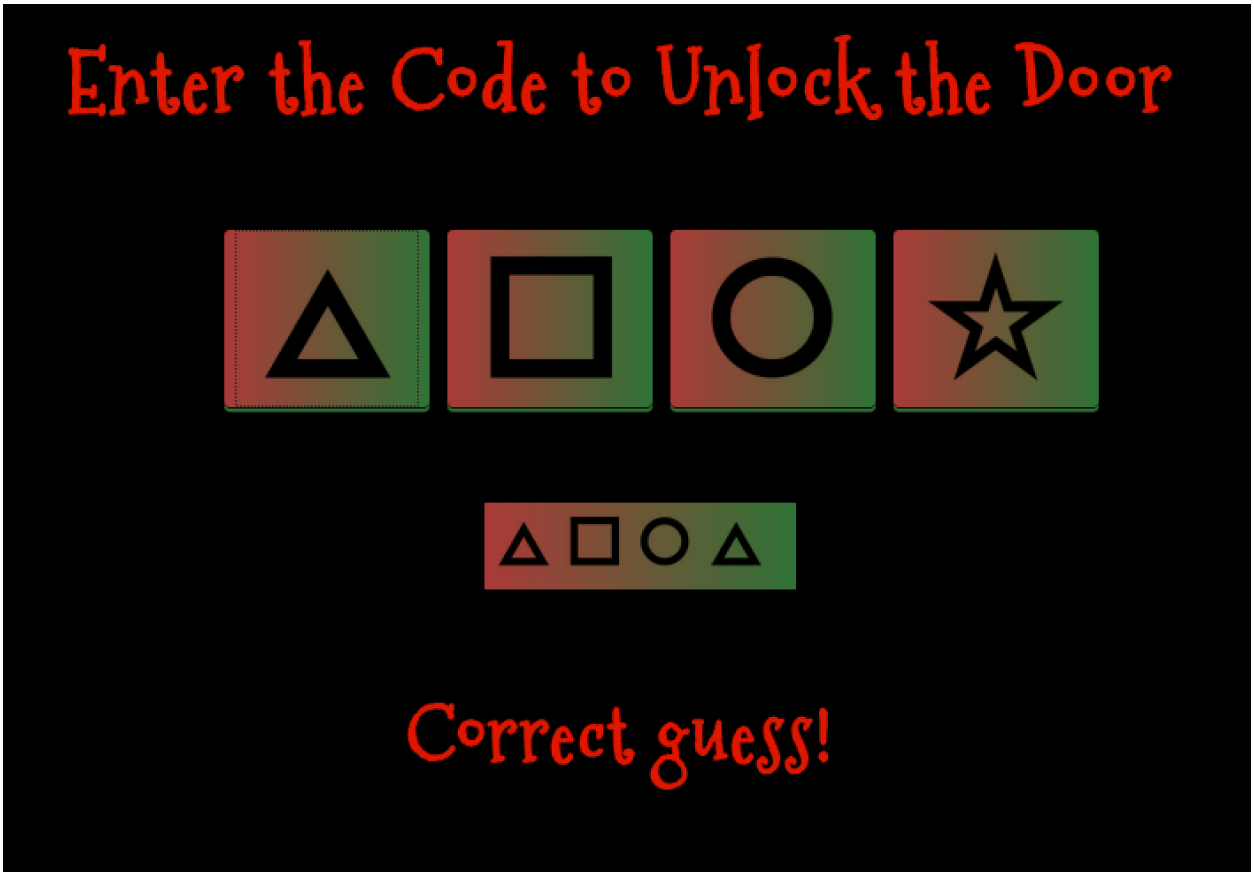
As you can see, he replaced “**Elinore**” with “**NEVERMORE**” on the last line. We got our answer.



Flag

This challenge required you to unlock a door and speak to someone in the room to get the flag. They give you 4 shapes and you need to enter them in a specific order to unlock the door.

One of the elves hinted that there is something called a de Bruijn sequence which can help you brute force something strategically. [I used this website] and set both the **k** and **n** values to 4 (since we have 4 shapes and the pin is 4 characters long) and used the table to work through each sequence accordingly until I unlocked the door.



Flag: Welcome unprepared speaker!

Jump to table of contents

Data Repo Analysis

Elf Challenge

In this one, the elf needed to upload their report to a Samba server and forgot their password. I needed to look through the terminal to find out what that password was. There was a hint left that basically told me how to find it. It turns out that you can see commands typed in that are associated with certain system processes. The way to do this is with the **ps aux | more**

command. To clarify, **ps aux** will give you detailed information on each process running on the machine; the **more** adds even more detail to the output.

```
elf@959418e6abba:~$ ps aux | more
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.1  0.0  17952  2888 pts/0    Ss   22:53   0:00 /bin/bash /sbin/init
root        10  0.0  0.0  45320  3176 pts/0    S    22:53   0:00 sudo -u manager /home/manager/
samba-wrapper.sh --verbosity=none --no-check-certificate --extraneous-command-argument --do-not
-run-as-tyler --accept-sage-advice -a 42 -d~ --ignore-sw-holiday-special --suppress --suppress
//localhost/report-upload/ directreindeerflatterystable -U report-upload
root        11  0.0  0.0  49532  3292 pts/0    S    22:53   0:00 sudo -E -u manager /usr/bin/py
thon /home/manager/report-check.py
manager     15  0.0  0.0  33848  7952 pts/0    S    22:53   0:00 /usr/bin/python /home/manager/
report-check.py
root        16  0.0  0.0  45320  3164 pts/0    S    22:53   0:00 sudo -u elf /bin/bash
elf         17  0.0  0.0  18204  3296 pts/0    S    22:53   0:00 /bin/bash
manager     18  0.0  0.0   9500  2616 pts/0    S    22:53   0:00 /bin/bash /home/manager/samba-
wrapper.sh --verbosity=none --no-check-certificate --extraneous-command-argument --do-not-run-a
s-tyler --accept-sage-advice -a 42 -d~ --ignore-sw-holiday-special --suppress --suppress //loca
lhost/report-upload/ directreindeerflatterystable -U report-upload
manager     20  0.0  0.0   4196   716 pts/0    S    22:53   0:00 sleep 60
root        23  0.0  0.0 316664 15616 ?        Ss   22:53   0:00 /usr/sbin/smbd
root        24  0.0  0.0 308372  5860 ?        S    22:53   0:00 /usr/sbin/smbd
root        25  0.0  0.0 308364  4504 ?        S    22:53   0:00 /usr/sbin/smbd
root        27  0.0  0.0 316664  6044 ?        S    22:53   0:00 /usr/sbin/smbd
```

As you can see, the manager uploaded a file by using “**directreindeerflatterystable**” as the password. Looks like we got it, let’s try it.

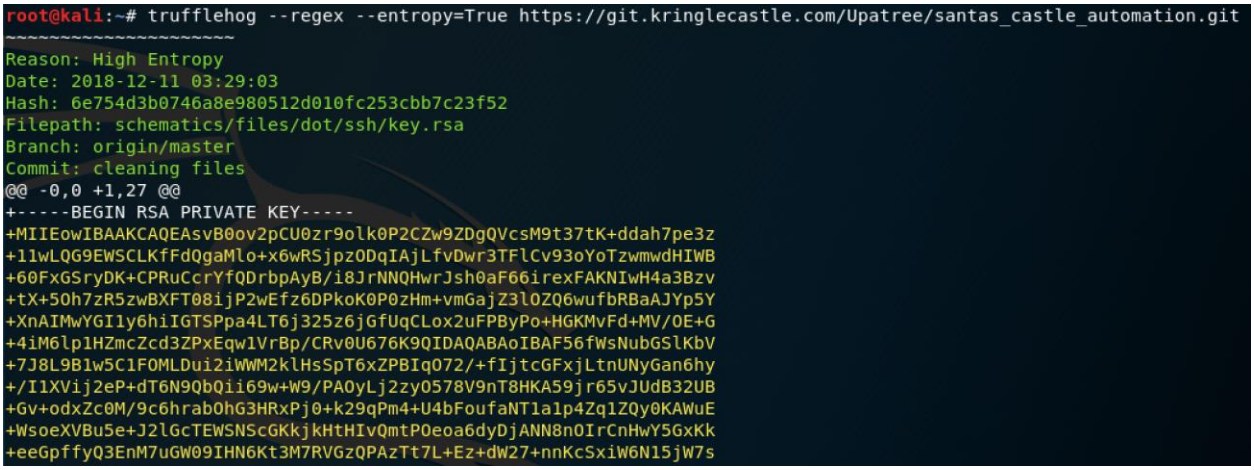
```
elf@959418e6abba:~$ smbclient //localhost/report-upload/ -c 'put report.txt' -U report-upload
WARNING: The "syslog" option is deprecated
Enter report-upload's password:
Domain=[WORKGROUP] OS=[Windows 6.1] Server=[Samba 4.5.12-Debian]
putting file report.txt as \report.txt (500.9 kb/s) (average 501.0 kb/s)
```

Nice! To clarify the command above, the **-U** parameter is for the username and the **-c** parameter is what command you wanted to run on the server. In this case, I used the “**put**” command to put the file on the server.

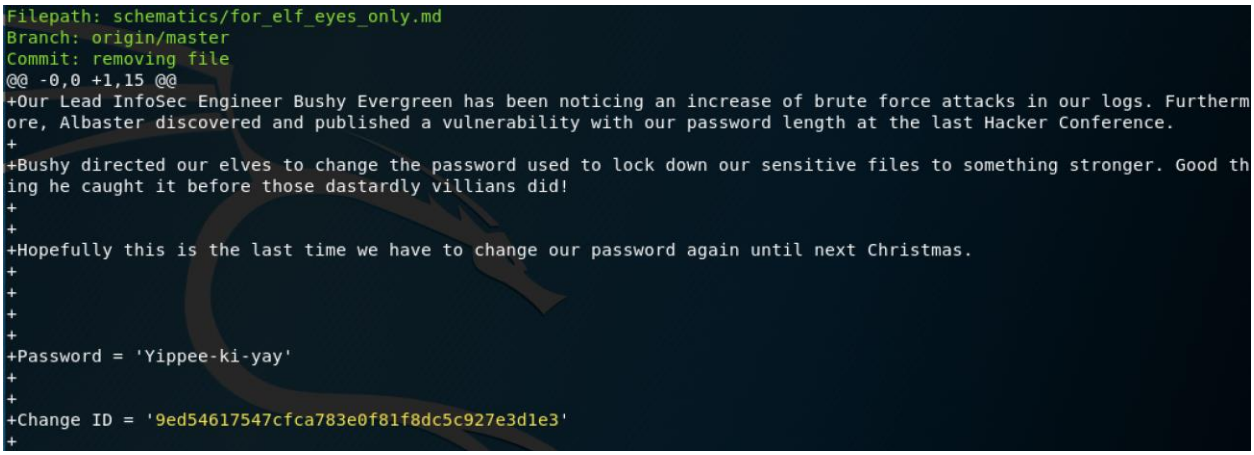
Flag

The purpose of this challenge was to find the password to unzip a file on Santa’s github. One of the elves mentioned a tool called truffleHog which will search through git repositories for

secrets by looking through the commit history and branches. I installed this on to my Kali box and ran the tool.



I added the **--entropy=True** to the command because the elf mentioned that it would be a good idea to use that. You can see that truffleHog retrieved RSA private keys from the repository (amongst other goodies).



I found the commit I was looking for after some scrolling and it appears that we found the flag.

Flag: Yippee-ki-yay

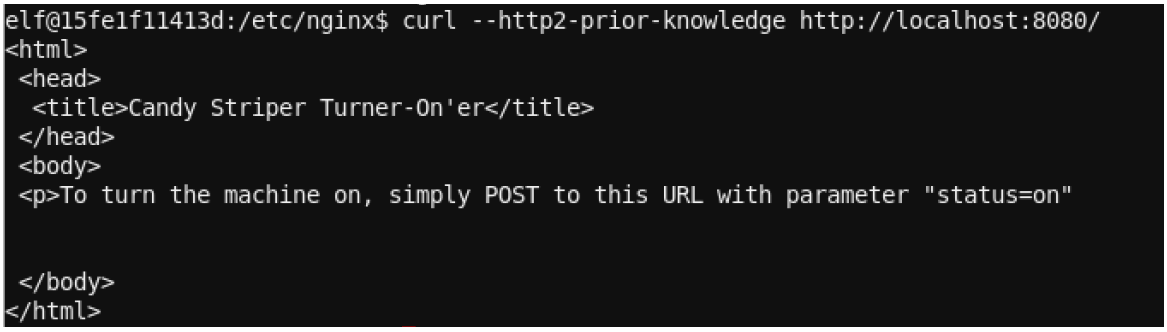
Jump to table of contents

AD Privilege Discovery

Elf Challenge

The elf is having trouble turning on to the server and her normal curl commands aren't working. Her brother put some weird configuration on the server and she can't figure it out. I took a look at the */etc/nginx.conf* file and sure enough, the server is configured to use http2.

This isn't a big issue though, curl can deal with http2, we just need special parameters to make it work.

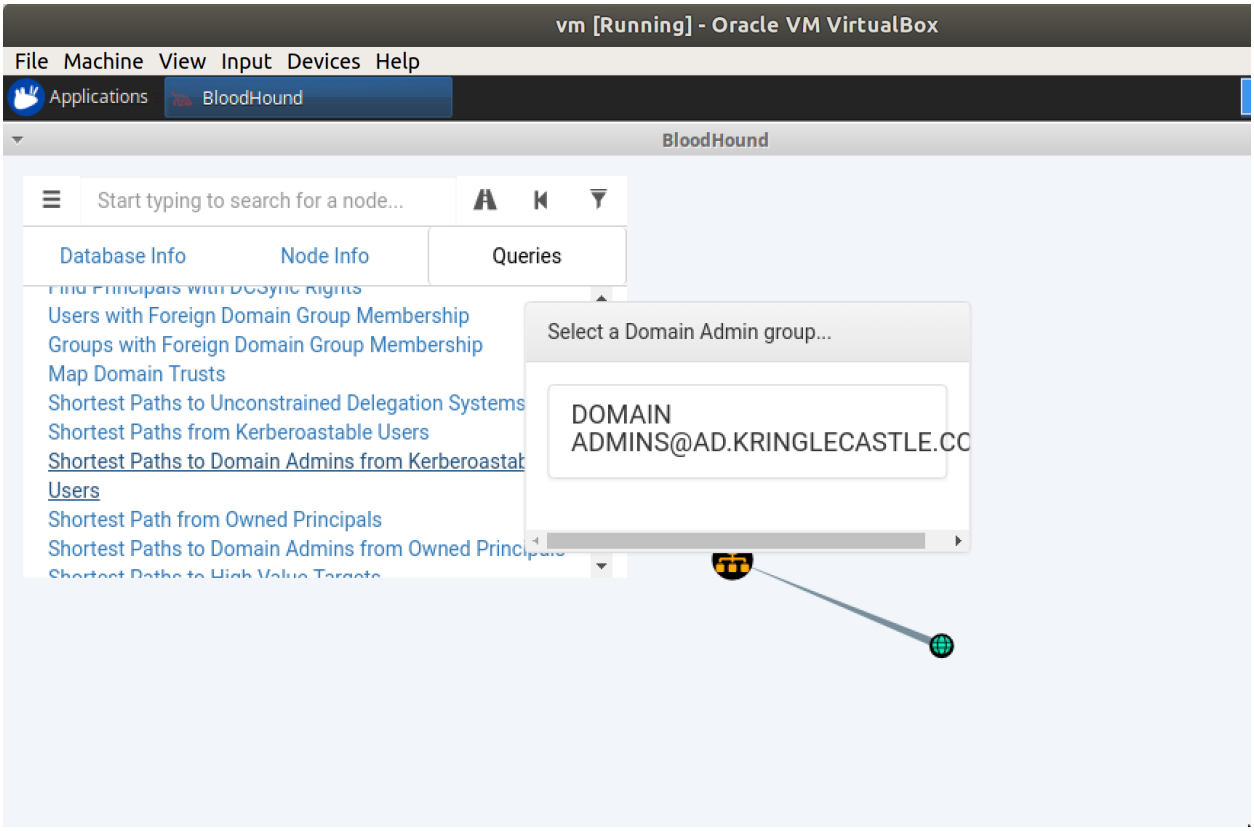


The parameter needed was **--http2-prior-knowledge.** The webpage gives instructions on how to turn the server on.

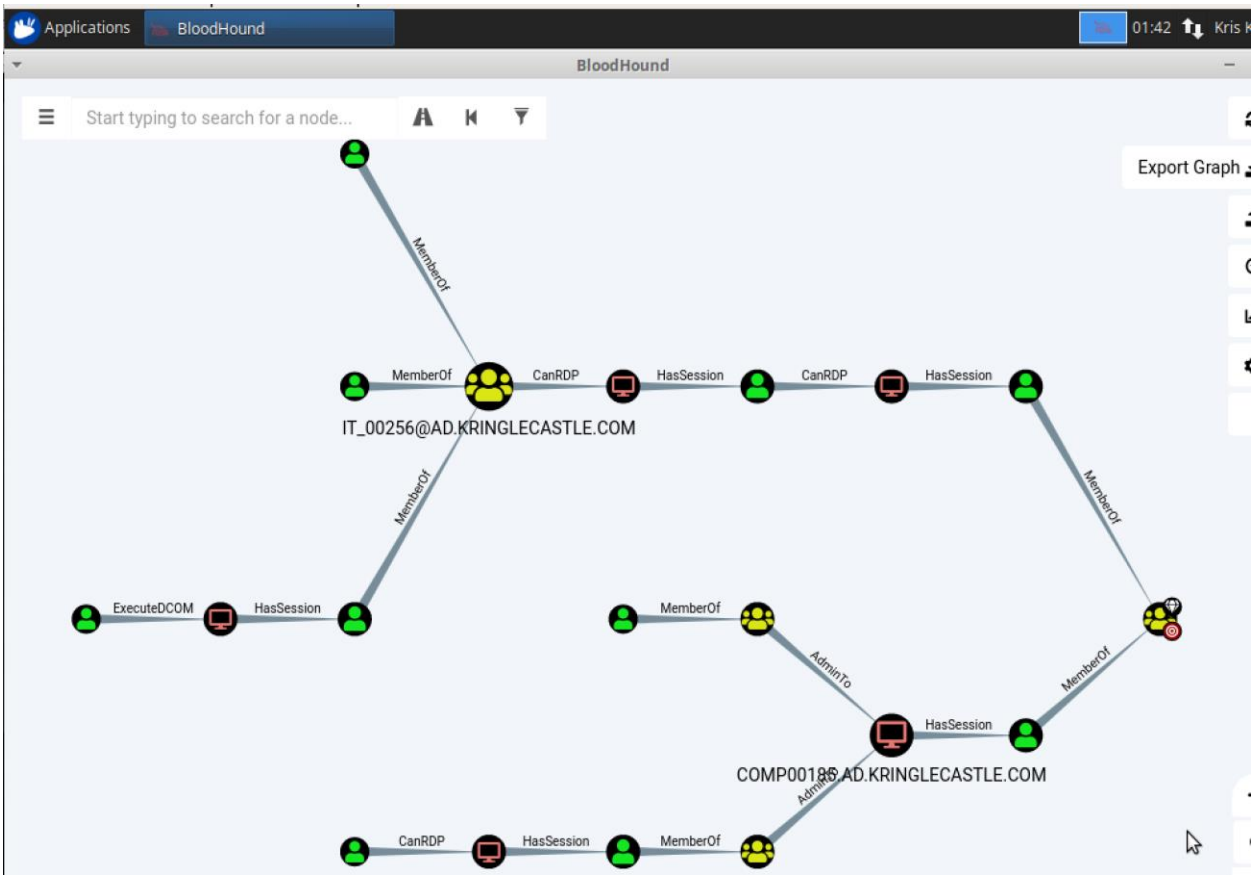
I used “**curl –http2-prior-knowledge POST -d “status=on” http://localhost:8080/**” and got on to the server.

Flag

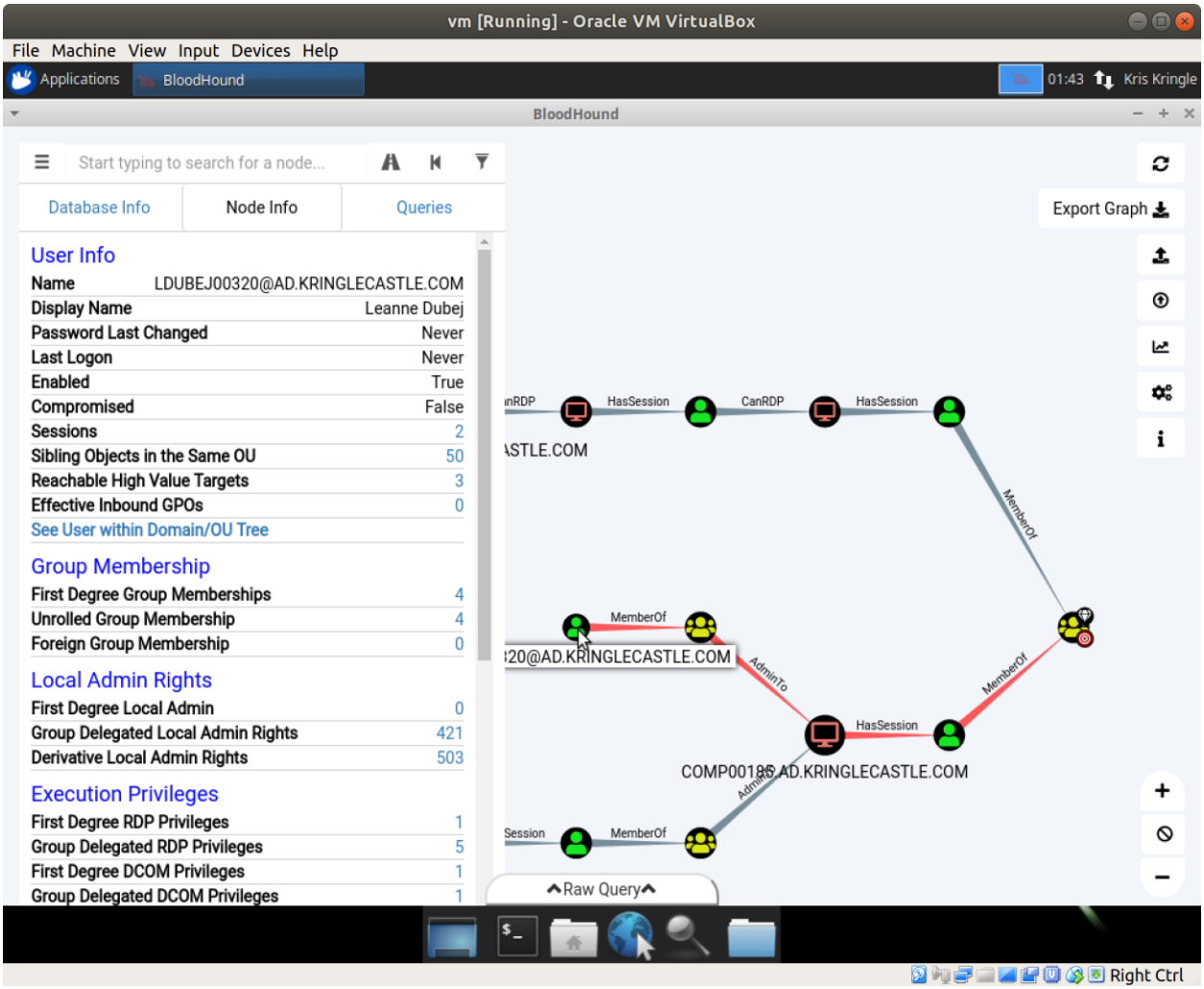
This challenge requires you to find a reliable path from a Kerberoastable user to the Domain Admins group by using the data set contained in the Slingshot Linux image that is provided. I had trouble powering up the VM, but it turns out that I just needed to change the VM to a 64-bit image to make it work. One of the elves gave me information about bloodhound which maps out an AD environment for you. Once I turned on the VM, there was a bloodhound shortcut already on the desktop so I opened it.



Lucky for us, there's already an existing query that will search for paths from domain admins to Kerberoastable users. Once I put that in, I can look for the path needed to get the flag.



It was noted in the challenge to avoid any paths that use RDP, which narrowed down our results pretty quickly.



Looks like Leanne was our winner.

Flag: LDUBEJ00320@AD.KRINGLECASTLE.COM

[Jump to table of contents]

Badge Manipulation

Elf Challenge

Pepper said that she had been a victim of password spraying, which is an attack where the hacker attempts to match multiple usernames to one password to try and get on to the system. Pepper had a log file she wanted me to look over so I could find whose account was compromised. She also mentioned that it was an evtx dump which can be difficult to read, but she has a python script that will make it XML so it's easier on the eyes. The XML file also has quotes in it which is a pain for me when I'm trying to use **grep** to search for things so I used the **sed** command below to make a new log file that without quotes in it so it would be easier to **grep**.

```
elf@7670722957b6:~$ ls
evtx_dump.py  ho-ho-no.evtx  runtoanswer
elf@7670722957b6:~$ python evtx_dump.py ho-ho-no.evtx > log.txt; sed 's/\"//g' > newlog.txt
```

I would be lying if I said I didn't spend a long time on this; I'm not used to these types of logs and my **grep** skills are subpar as is, so I was definitely struggling. I didn't even know what I was looking for at first, so I looked up how to find password spraying in a evtx log file. This gave me some more insight, specifically:

1. *TargetUserName* – this is the target user that we're looking for
2. *Status* – this is a failure code that will give you an idea on why the login attempt failed
3. *SubStatus* – this gives you a more specific failure code
4. *IpAddress* – the IP address that the connection was coming from

With this new information, I tried this command:

```
elf@714b88d1b6b4:~$ cat newlog.txt | grep -n -e "Status>0xc000006c" -e "SubStatus>0x000006a" -e
 "IpAddress>172.31.254.101" -e "TargetUserName>" | grep -v "TargetUserName>HealthMail" | grep -
v "TargetUserName>WIN"
```

To break down this monstrosity:

cat newlog.txt – displays the results of the log file

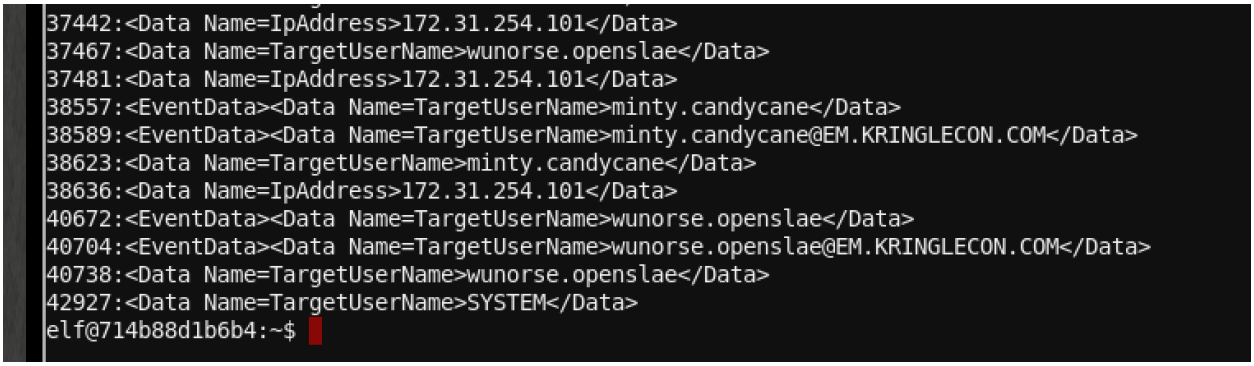
| - those are called pipe commands, these allow you to run multiple commands at once

grep -n -e “Status>0xc000006” -e “SubStatus>0x000006a” -e “IpAddress>172.31.254.101 -e “TargetUserName>” - this one seems like a lot, but is pretty self explanatory.
The **-n** parameter allows you to see the line numbers of where your results are in the text file (I was using those as a reference in case if I wanted to look at that specific event as a whole).
The **-e** parameter allows you to look for a specific string, so my first one was the “**Status**” that was looking for the status code that shows that there was a failure due to a bad username or password at login, the “**SubStatus**” was looking for the status code that show that there was a failure specifically due to a bad password and good username.
“**IpAddress**” was searching for the source IP address the request (I found this by combing through the log file and seeing that this IP address had some of the most requests, so I figured something weird was going on), and the “**TargetUserName**” was just looking for any targets associated with the above parameters.

grep -v “TargetUserName>HealthMail”
grep -v “TargetUserName>WIN” - the **-v** parameter will tell grep to exclude all of the results with this specific pattern. I added **HealthMail** and **WIN** because they were the beginning of some

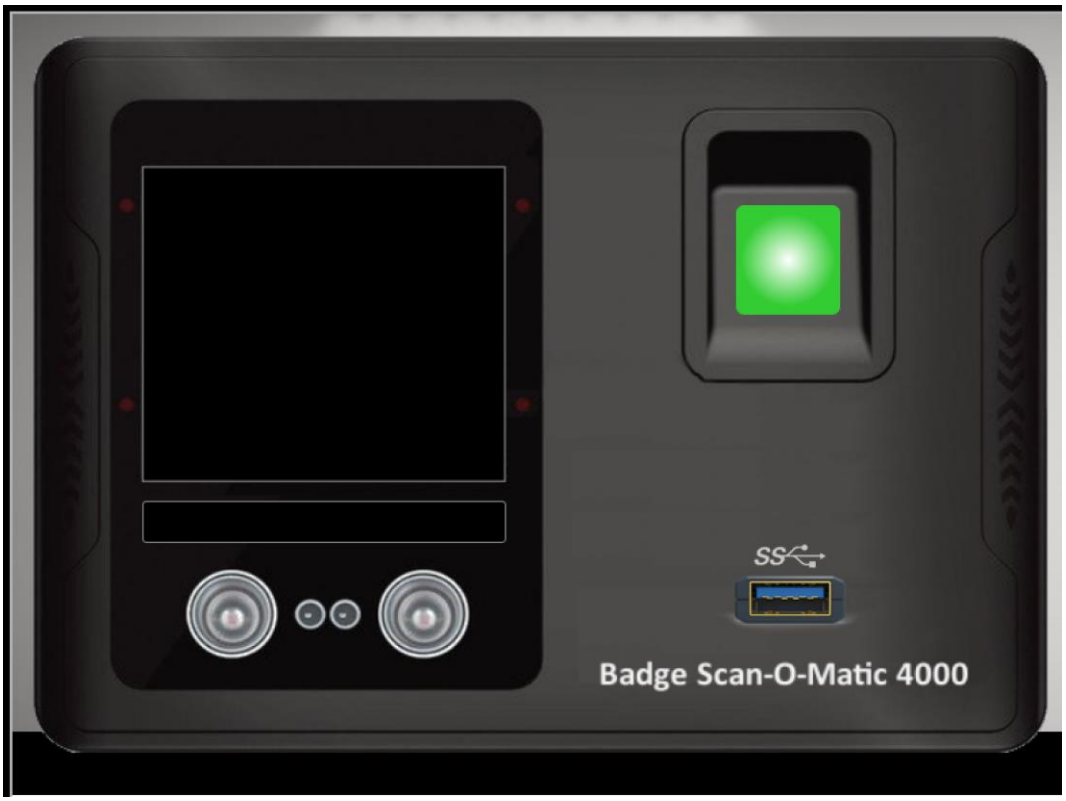
service name accounts that would clog up my results, so I didn't want them in my output.

The above command displayed a lot of usernames and I honestly wasn't sure how to filter them further, so I totally guessed on this one. I tried **wunorse.openslae** and it didn't work, but **minty.candycane** did so...yay? I really got lucky on this one which is why I feel like it was karma for the upcoming SQL injection objective to be the worst and most painful challenge for me.



Before I get into the pain, I'll give some context on how this challenge was laid out. The purpose of this challenge was to bypass the authentication mechanism associated with the room near Pepper Minstix (one of the elves). I was given a sample badge that showed me that the employees use QR codes to scan in.

The badge scanner looks like this:



I tried the sample badge on it, but it said that it only accepts PNG files (boo!), so I just saved it as a PNG instead. The scanner displayed the error *"Authorized user account has been disabled!"* Nice one Alabaster. Pepper did mention that the scanner uses SQL so before even attempting to inject anything, I had to throw an exception to see how the commands were laid out. I generated a QR code with some dummy text (I'm pretty sure you can put literally anything and it'll throw the exception) and got this:

"EXCEPTION AT (LINE 96 "user_info = query("SELECT first_name,last_name,enabled FROM employees WHERE authorized = 1 AND uid = '{}' LIMIT 1".format(uid))"): (1064, u"You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '' LIMIT 1' at line 1"
So this is a MariaDB server and the command should look something like:

user_info = query("SELECT first_name,last_name,enabled FROM employees WHERE authorized = 1 AND uid = '{}' LIMIT 1".format(uid))")

I'm pretty trash at SQL, but even my sloth self was able to deduce that the injection should be in the brackets after the uid. The question was, what goes in there? What constitutes as a valid user? It looks like the user needs to be both authorized and enabled for the badge to scan. I definitely needed some help so I used this link and got some pointers from the discord channel to come up with this:

user_info = query("SELECT first_name,last_name,enabled FROM employees WHERE authorized = 1 AND uid = '' 1' or '1'='1' and enabled=1 # LIMIT 1 ".format(uid))")

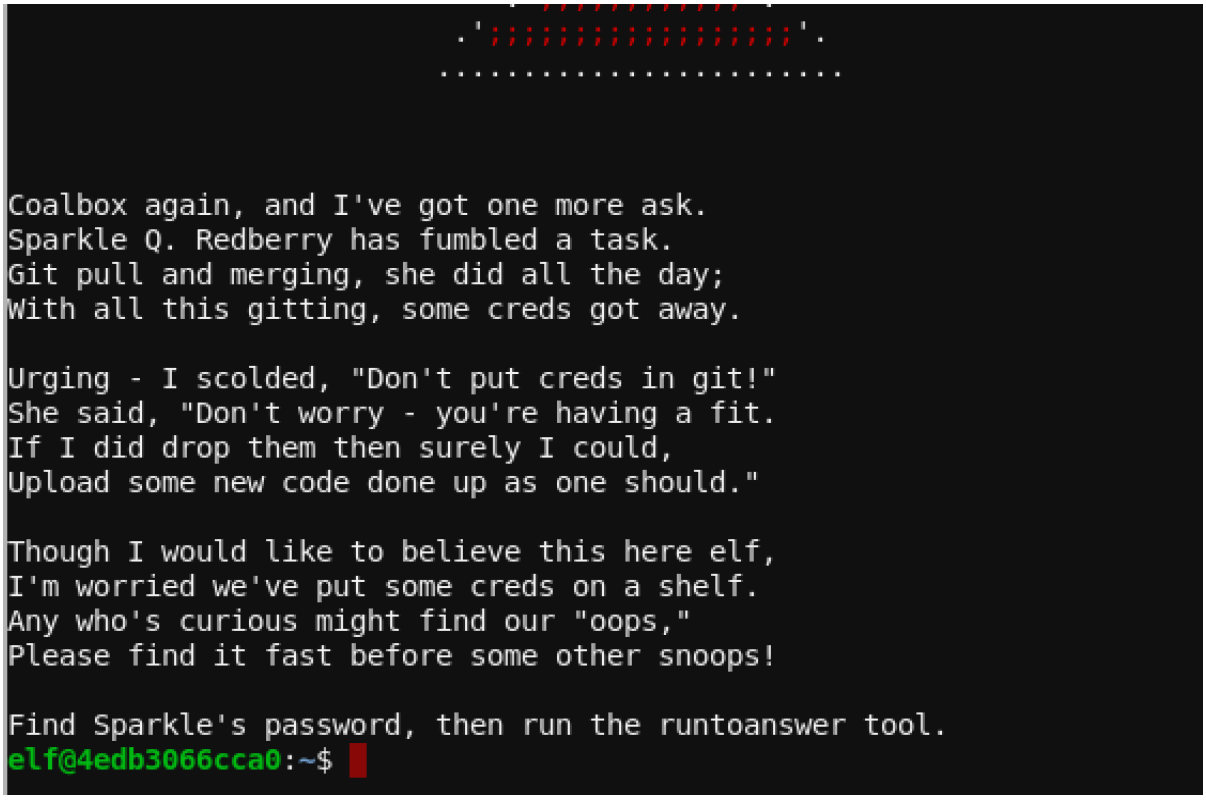
There was so much pain, so much trial and error, but finally, I was in. So many hours burnt on this, but at least I learned something new.

Flag: It's an access control number that flashes on the screen after you scan in, but I'm not sure if it's the same for everyone or different.

Jump to table of contents

HR Incident Response

Elf Challenge



As you can see, Sparkle forgot her password, but it's located in one of her commits in this git repository. First, I had to look at the commits themselves and this was done by using the **git log** command.

```
commit a6449287cf9ed9151d94fb747f6904158c2c4d71
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Fri Nov 9 14:08:04 2018 -0500

    Add passport middleware for user auth

commit 60a2ffea7520ee980a5fc60177ff4d0633f2516b
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Thu Nov 8 21:11:03 2018 -0500

    Per @tcoalbox admonishment, removed username/password from config.js, default settings in c
onfig.js.def need to be updated before use
```

After scrolling, it seems that I found the commit I needed since the comment says *"removed username/password from config.js."* Now, I can use the **git show** command which allows me to see the differences for a specific git commit.

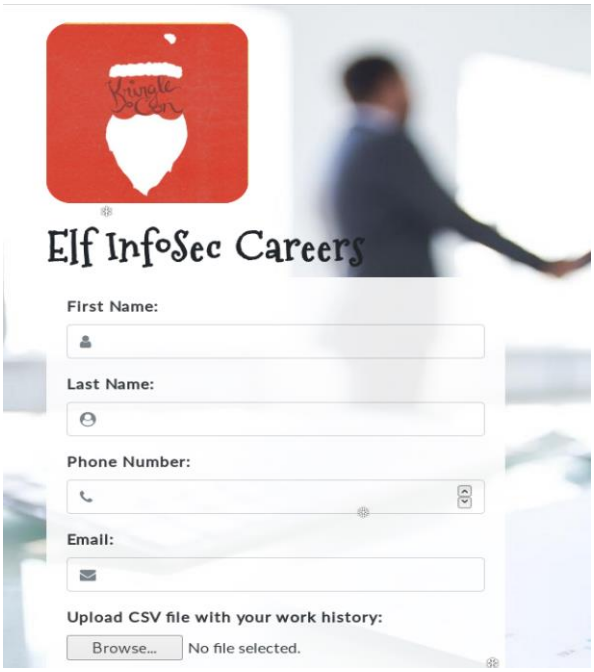
```
elf@4edb3066cca0:~/kcconfmgmt$ git show 60a2ffea7520ee980a5fc60177ff4d0633f2516b
commit 60a2ffea7520ee980a5fc60177ff4d0633f2516b
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Thu Nov 8 21:11:03 2018 -0500

    Per @tcoalbox admonishment, removed username/password from config.js, default settings in c
onfig.js.def need to be updated before use

diff --git a/server/config/config.js b/server/config/config.js
deleted file mode 100644
index 25be269..0000000
--- a/server/config/config.js
+++ /dev/null
@@ -1,4 +0,0 @@
-// Database URL
-module.exports = {
-    'url' : 'mongodb://sredberry:twinkletwinkletwinkle@127.0.0.1:27017/node-api'
-};
diff --git a/server/config/config.js.def b/server/config/config.js.def
new file mode 100644
index 0000000..740eba5
--- /dev/null
+++ b/server/config/config.js.def
@@ -0,0 +1,4 @@
+// Database URL
+module.exports = {
+    'url' : 'mongodb://username:password@127.0.0.1:27017/node-api'
+};
```

Sure enough, in the red, we found our answer: **twinkletwinkletwinkle**.

```
elf@4edb3066cca0:~/kcconfmgmt$ runtoanswer
Loading, please wait......



Enter Sparkle Redberry's password: twinkletwinkletwinkle


This ain't "I told you so" time, but it's true:
I shake my head at the goofs we go through.
Everyone knows that the gits aren't the place;
Store your credentials in some safer space.

Congratulations!
```

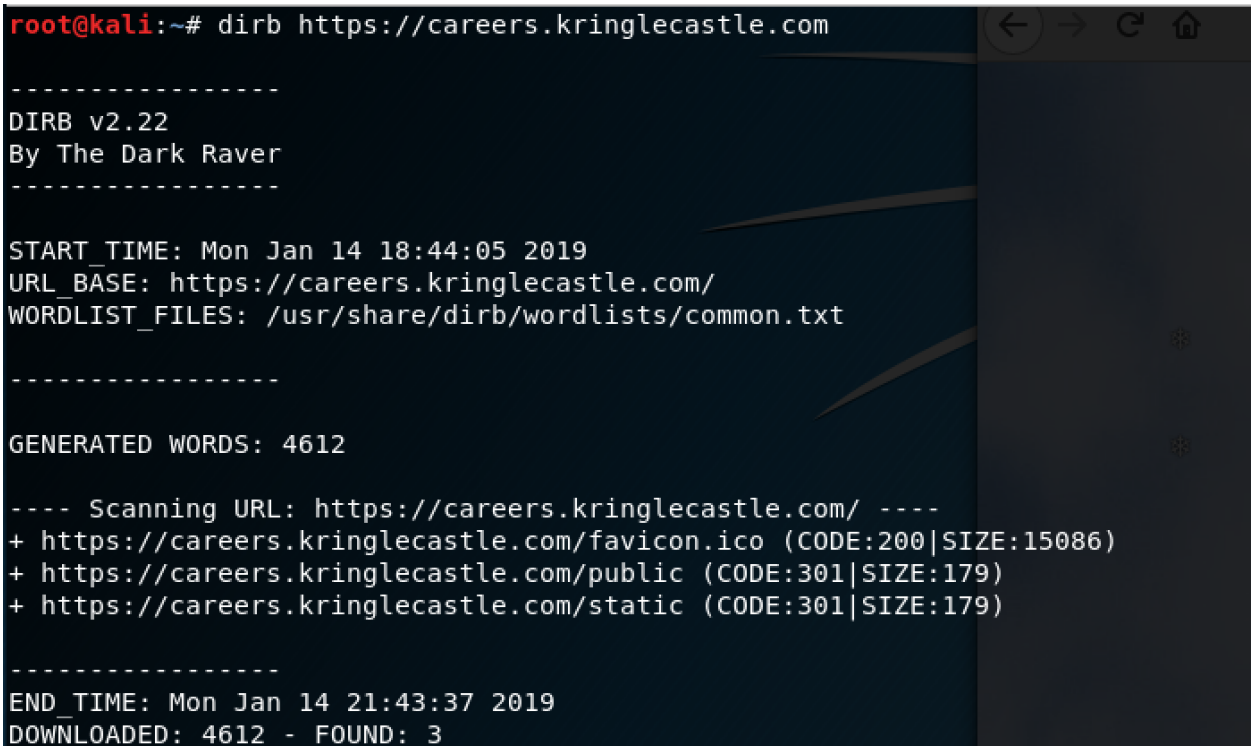
Flag

The purpose of this challenge was to use csv injection to get information off of a word document file located on a web server. One of the elves told me that there was a talk at KringleCon about csv injection, so I went to check that out and get some more insight. The concept itself seems pretty straightforward, I just had to figure out how I was going to get a hold of that file. The website where we upload the file looks like this.

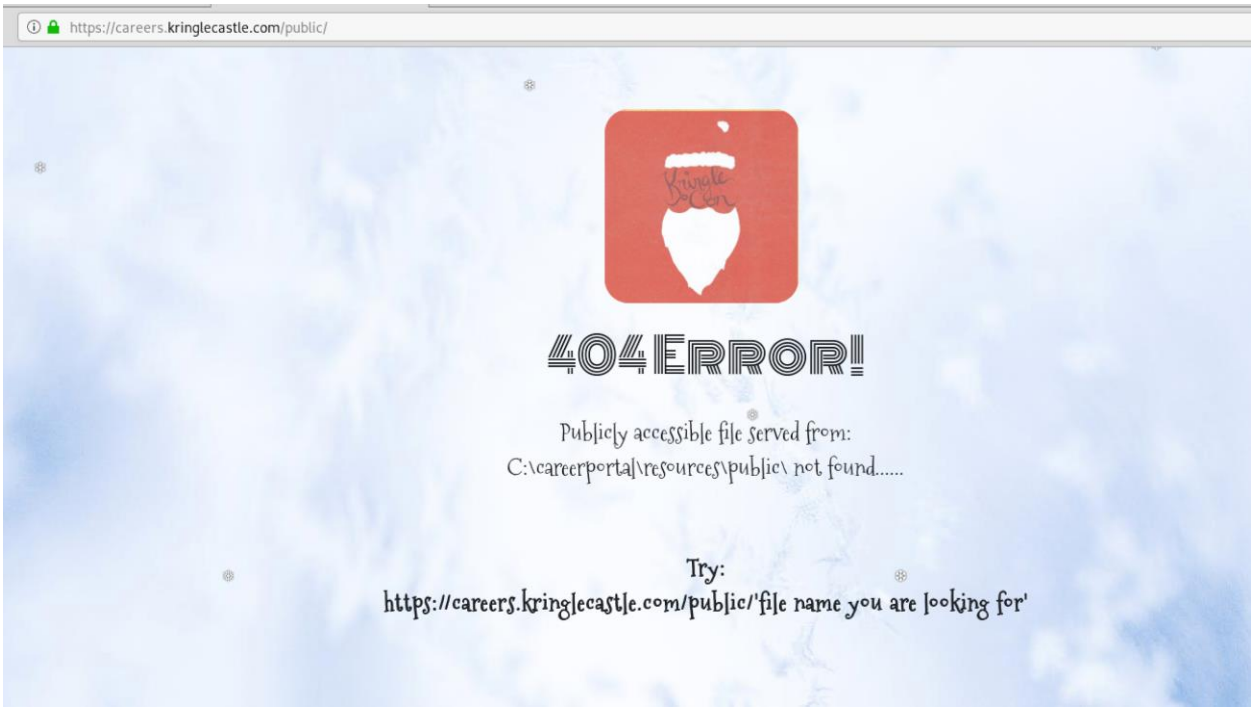


I also knew that the file we wanted was located on *C:\candidate_evaluation.docx* because that was explained in the objective.

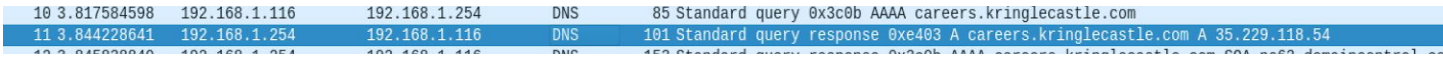
I wanted to get a map of how the site was laid out to get some clues from there. Dirb is a great tool for doing this.



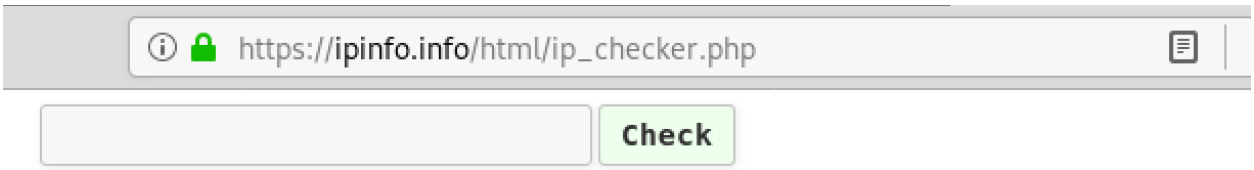
There are two pages, although the */static/* directory holds the js files, so */public/* was the only one I was interested in. This does give a 301 code which means it's being redirected, which probably means that we'll get a 404 error and won't be able to load the page.



Sure enough, we got an error. This may seem like a bad thing, but if I wanted to copy the file to a directory where I could access it (which is what I ended up doing), this would be the directory to do that in. Before trying that, I still was curious about my reverse shell option. I've only ever done these with a system that is on my network, so that's a lot easier, but never on a site located elsewhere. I wanted to find out what the IP address of the web page was to do some more digging. I opted to use Wireshark to sniff the traffic when I uploaded a csv file.



Based on this snippet, it seems that **35.229.118.54** should be our address. I checked multiple IP address lookup websites to double check and confirm this.



Checking Domain Name

Domain Name: careers.kringlecastle.com

Top Level Domain: COM (Commercial TLD)

DNS Lookup

IP Address: 35.229.118.54

Geolocation: US (United States), VA, Virginia, N/A N/A - Google Maps

Reverse DNS: 54.118.229.35.bc.googleusercontent.com

Just out of curiosity, I wanted to try entering the address in my web page to see what would happen.

Bad gateway and an nginx server, interesting. So there is something there! Let’s try nmap to see what happens.

```
root@kali:~# nmap -sV -p 1-65535 -T4 -Pn 33.229.118.54
Starting Nmap 7.70 ( https://nmap.org ) at 2019-01-14 22:40 EST
Nmap scan report for 33.229.118.54
Host is up (0.078s latency).
All 65535 scanned ports on 33.229.118.54 are filtered

Service detection performed. Please report any incorrect results at https://nmap
.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 162.61 seconds
```

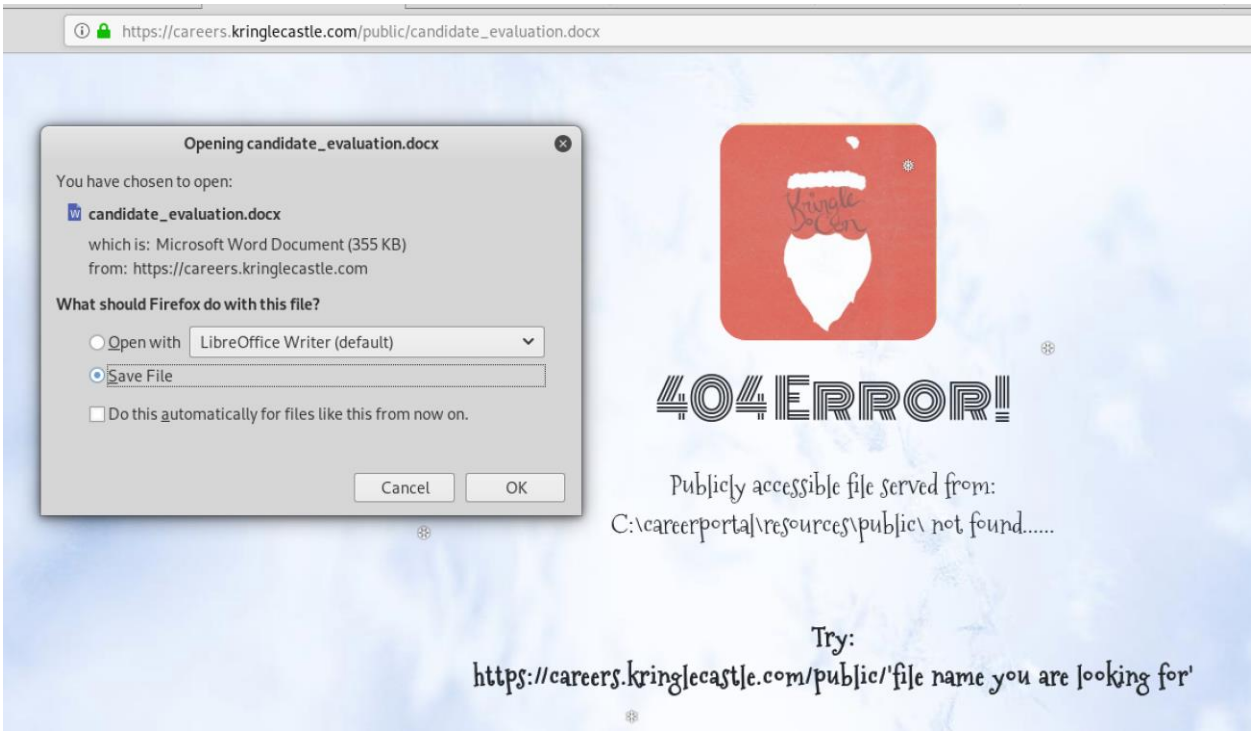
To clarify this command:

-sV - prints out the versions of the services on each port
-p - specifies what port numbers I want to scan (default only scans the most popular ones)
-T4 - this is the timing template parameter, which ranges from (0-5) and will specify how aggressively the nmap scan will be
-Pn - treat this host as online and skip the host discovery (I did my first scan without this parameter and it said the host was down, but recommended I try this parameter to get more answers)

So the host is up, but all of the ports are filtered and seem pretty locked down. It seems that our best bet is to indeed copy that file to the */public/* directory and extract it that way. This is as simple as executing a **copy** command to put it in a place where we can reach it. I found the easiest way to make this work was to not create an Excel or LibreOffice Calc file, but make the csv file in a text editor and save it as a csv file because the other programs were messing with my injection payload.

```
=cmd|'/c copy C:\candidate_evaluation.docx C:\careerportal\resources\public\candidate_evaluation.docx'!A1
```

Note: I knew that the file was on *C:\careerportal\resources\public* because it's listed on the 404 error on the webpage.



Eureka, we got the file with the flag in it.

Flag: Fancy Beaver

[Jump to table of contents]

Conclusion

This is easily the longest write up I have ever made (of the few write ups I've done so far) and I hope it was informative and relatively easy to understand. Make sure to tune in every year around Christmas time so you can participate in the Holiday Hack Challenge as well!