SANS HolidayHack 2012

Brian King
Independent Security Consultant
Columbus, Ohio, USA
bbk@pobox.com

ABSTRACT

In this paper, I describe my solution to the SANS 2012 Holiday Challenge, answering the objective questions in order. After the formal responses, I continue with a discussion of the principles, tools and techniques that proved useful in this engagement.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Access Controls, Authentication, Cryptographic controls, Verification.

D.2.3 [**Coding Tools and Techniques**]: HTTP Traffic During Redirects, PHP Language.

E.3 [**Data Encryption**]: Code breaking, key re-use, known-plaintext attacks, steganography, strong passwords.

E.5 [**Files**]: Backup/Recovery, secure storage, leaks.

Ref: The ACM Computing Classification Scheme: http://www.acm.org/class/1998/

General Terms

Algorithms, Management, Design, Reliability, Security, Human Factors

Keywords

SANS, HolidayHack, 2012, @sn0wm1s3r, @h34tm1s3r, @edskoudis, @timmedin, PHP, Information Disclosure, Subversion Repositories, Raspberry Pi, SHA1, curl, steghide, xxd, exiftool, watch-what-you-tweet

1.INTRODUCTION

In this section, I respond strictly and briefly to the formal questions, numbered 1 through 5. Here, no attempt is made to explain the methods behind the approach, or the clues discovered along the way. The section on question 6 contains much more background information and details, including identification of clues, thought processes involved, tools used and screenshots.

The final section is a table listing the flags discovered at each level for both Snow Miser and Heat Miser.

Abbreviations used in this text include:

SM = Snow Miser

HM = Heat Miser

L1, L2 ... = Level 1, Level 2, etc

1.1 Where did you find the remainder of Snow Miser's Zone 1 URL?

Snow Miser posted a photograph to his Twitter stream, showing a book and a glass of water in front of the base of a computer monitor. Part of the monitor's display, which contains the missing part of the URL, is reflected in the surface of the water in the glass.

1.2 What is the key you used with steghide to extract Snow Miser's Zone 2 URL? Where did you find the key?

The key was IceIceBaby! which was found in the 'comments' area of the 'off.jpg' file from the Zone 1 page.

1.3 On Snow Miser's Zone 3 page, why is using the same key multiple times a bad idea?

The Zone 3 page gave both the plaintext and ciphertext for a single message. By xor-ing the bytes of those two strings, the keystream data can be discovered. If the same key is used on another message, then the same keystream will also be used. Simply xor-ing the ciphertext of that other message with the derived keystream bytes will result in the plaintext of the unknown ciphertext.

1.4 What was the coding error in Zone 4 of Heat Miser's site that allowed you to find the URL for Zone 5?

The error is that the PHP code which adds the HTTP 302 'redirect' header fails to call exit() before outputting the rest of the HTTP response. This causes the full HTML to be sent to the browser. In normal use, the browser will follow the redirect and not display that content, but since it has been sent, a watchful visitor can read that HTML by, for example, using an intercepting HTTP proxy.

1.5 How did you manipulate the cookie to get to Zone 5 of Heat Miser's Control System?

I recognized the cookie as a likely MD5 hash, and discovered that it was a hash of '1001'. I tried using common "super-user" numbers such as zero and 501 with no success. On setting up a tool to brute force the first 1000 base-10 numbers, I got a hit on the MD5 hash of the number '1' which was the solution.

2. QUESTION 6: Please briefly describe the process, steps, and tools you used to conquer each zone, including all of the flags hidden in the comments of each zone page.

This section will walk step by step through each challenge, identifying the clues discovered, the methods and tools used to exploit those clues, and the solutions to each level.

2.1 Reconnaissance

The first step in any engagement is to understand the scope, and investigate what resources are available to guide one's efforts. For this engagement, the starting point was http://pen-testing.sans.org/holiday-challenge/2012 which described the context around the challenge, the individuals involved, the objective, and the rules of engagement. In the real world, the rules of engagement (ROE)

define the scope of what systems may be probed and what techniques may be used. In a practice challenge like this, they also provide clues as to what types of attacks are likely to be dead-ends. In either case, paying close attention to the ROE will save the tester time and effort, and will help the tester avoid inadvertently breaking laws or causing damage.

2.1.1 The Questions In the Challenge Page

The questions themselves reveal some information about how to approach some of the levels. I will come back to each of them in turn as I walk through each level.

2.1.2 The Players & Information about Them

The two main characters are the Snow Miser and the Heat Miser. Their mother also makes an appearance.

* Each "Miser" has a Twitter feed, which they use as normal brothers would: to talk about what they're doing, post ephemeral photos, and taunt each other. It is easy to inadvertently disclose sensitive information on Twitter. These feeds will prove quite valuable.
* Each Miser has a publicly-visible page on his Industrial Control System. Loading the root directory of each results in an HTTP-level redirect to very similar-looking URLs. Each system follows the same pattern of URL.
* Mother Nature has a Twitter feed, but no photographs or links. As it happens, nothing necessary for solving the puzzles is to be found here.
* Ed Skoudis and Tim Medin, who created the challenge, also have Twitter feeds. They may offer additional hints, intentional or not.
* All clues start with one of these:
    * http://pen-testing.sans.org/holiday-challenge/2012
    * https://twitter.com/sn0w_m1s3r
    * https://twitter.com/h34t_m1s3r
    * http://twitter.com/edskoudis
    * http://twitter.com/timmedin
    * https://twitter.com/m0th3r_n4tur3

3.Snow Miser

3.1 Snow Miser, Getting To Level 1

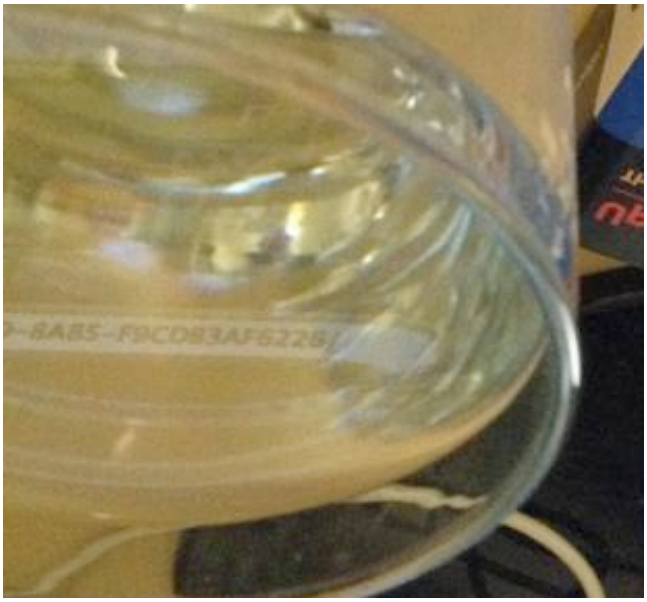
The first clue to Snow Miser Level 1 is in the first question on the page that introduces the challenge "*Where did you find the remainder of the Snow Miser's Zone 1 URL?*"

This tells us it's probably just a hunting game. There is also a clue in the text of the Zone 0 page, which is accessible to any anonymous visitor:

```
Those of you with proper access, the URL you need starts with the following:
zone-1-D2E31380-50E6-4869-8A85-XXXXXXXXXXXX
```

If we look at Snow Miser's Twitter feed, we find one image posted, with the caption, "We're chillin' here with some ice cold drinks." In the image, there is a drink, and in the surface reflection, there is some text, presumably from Snow Miser's computer display, the base of which is in the photo.

Using an image editor, I flipped and reversed that part of the image to find that it is part of a GUID-looking string, and it starts with "8A85" which is how the given portion on the web page ends.



Pasting the given part of the URL into my browser's address bar, and replacing the XXXXXXXXXXXX with F9CDB3AF6226 lands me on the Zone 1 Controller page.

Snow Miser Level 1 Flag: `38bef0b61ba8edda377b626fe6708bfa`

3.2 Snow Miser, Getting To Level 2

I noticed that clicking on the "Disable" button changed the image on the page, and that the "off" image was a JPEG, where the "on" image was a PNG. Looking back on the given questions, number 2 is:

What is the key you used with steghide to extract Snow Miser's Zone 2 URL? Where did you find the key?

I know that steghide works on JPEG files, but not PNG files, so let's look at 'off.jpg'. I know that JPEG files can contain comments before the image data, so let's look there, using xxd to show the bytes of the file as hex - we're looking for anything "interesting":

```
00000e0: 4153 4349 4900 0000 4963 6549 6365 4261  ASCII...IceIceBa
00000f0: 6279 2100 ffe0 0010 4a46 4946 0001 0101  by!.....JFIF....
```

"IceIceBaby!" ?? Looking again with exiftool, we see that is a "comment" embedded in the file:

```
User Comment                    : IceIceBaby!
```

Let's try that as the password for the file itself with steghide

```
root@bt:~/Desktop# steghide extract -sf off.jpg
Enter passphrase:
wrote extracted data to "tmpfile.txt".
root@bt:~/Desktop# cat tmpfile.txt
zone-2-6D46A633-25D7-42C8-AF94-8E786142A3E3
root@bt:~/Desktop#
```

The string it produces is the Zone 2 URL

`zone-2-6D46A633-25D7-42C8-AF94-8E786142A3E3`

Snow Miser Level 2 Flag: `b8231c2bac801b54f732cfbdcd7e47b7`

3.3 Snow Miser, Getting To Level 3

This is where Snow Miser's vulnerabilities took a turn. The clues in the text of the page don't seem very promising - it says that the same person who "messed up the Zone 2 link also messed up the Zone 3 link" - but the images on this page are both PNG files, and steghide won't work with PNG files.

Looking back at the information available to us, we return to Heat Miser's Twitter stream, and find this tweet:

I download a copy of this file and extract it. I notice an SSH client, which looks interesting, but the ROE puts any SSH system out of scope. I run nmap against snowmiser.counterhack.com anyhow, but port 22 is closed. Just a quick port scan isn't much of a violation, is it?

The browser cache proves to be more useful. Running 'strings' against this file:

`data/data/com.android.browser/cache`

...returns some URLs that look promising:

```
wgjf
http://snowmiser.counterhack.com/zone-3-eab6b031-4efa-49f1-b542-30ebe9eb3962L
http://snowmiser.counterhack.com/zone-3-eab6b031-4efa-49f1-b542-30ebe9eb3962
404 Not Found
http://snowmiser.counterhack.com/zone-3-eab6b031-4efa-49f1-b542-30ebe9eb3962/M
http://snowmiser.counterhack.com/zone-3-eab6b031-4efa-49f1-b542-30ebe9eb3962/
404 Not Found
http://snowmiser.counterhack.com/zone-3-EAB6B031-4EFA-49F1-B542-30EBE9EB3962/M
http://snowmiser.counterhack.com/zone-3-EAB6B031-4EFA-49F1-B542-30EBE9EB3962/F
Snow Miser SnowTalk HMI for the Global Chiller Control System - Zone 3
```

That last one is what we want, but the "/F" is not part of it:

`zone-3-EAB6B031-4EFA-49F1-B542-30EBE9EB3962`

Snow Miser Level 3 Flag: `08ba610172aade5d1c8ea738013a2e99`

3.4 Snow Miser, Getting To Level 4

Getting to level four is a cryptography challenge. The page gives you the plaintext and corresponding ciphertext for the current URL, and it gives you the ciphertext for the URL of the next level, and tells you that they all use the same key.

That's all you need to know.

In a cryptosystem, the key is used to generate a keystream, and that keystream is xor-ed with the plaintext to produce the ciphertext. Because the URL we want to decipher won't be any longer than the URL whose ciphertext we have, we don't need to know the actual key, or even what crypto algorithm is in use, here. All we need is xor.

(plaintext) xor (keystream) = ciphertext.

(ciphertext) xor (keystream) = plaintext therefore...

(ciphertext) xor (plaintext) = keystream

So, if we xor each byte of the given plaintext with the same byte of the given Zone 4 URL's ciphertext, we'll derive the keystream byte for that position. Then, all we need to do is xor that with the corresponding byte of the Zone 5 URL's ciphertext to obtain the plaintext byte.

Here's a walk-thru of the first byte:

Given plaintext:

`zone-4-F7677DA8-3D77-11E2-BB65-E4BF6188709B`

Given ciphertext:

`20d916c6c29ee53c30ea1effc63b1c72147eb86b998a25c0cf1bf66939e8621b3132d83abb1683df619238`

The ciphertext is twice as long as the plaintext because each letter in the plaintext is a one-byte character - if it were shown in hexadecimal bytes like the ciphertext, they would be the same length. We could translate it, but it's easier to let Python do it for us:

```
>>> hex(ord('z') ^ 0x20)
'0x5a'
```

The first byte of the keystream is 0x5a. We don't actually need the first seven bytes, since they're identical in both ciphertexts (and we know that a Zone 4 URL will start with "zone-4-") but for completeness, I decided to do the whole thing.

The result is this:

`zone-4-9D469367-B60E-4E08-BDF1-FED7CC74AF33`

Snow Miser Level 4 Flag: `de32b158f102a60aba7de3ee8d5d265a`

3.5 Snow Miser, Getting To Level 5

The text on the Level 4 page tells us that the password to get to level 5 is a SHA-1 hash, and that Subversion 1.7 is used to store the source code. Just to see what happens, type anything into the password field and click "authenticate" - this gives you the URL for Zone 5, but says "Access Denied."

Looking back again at our resources, I notice that Heat Miser gave a hint related to Subversion:

Skimming the article, I find it ends with this paragraph:

Of course, just having all the source code means someone could completely copy the site, but we can also search the code for inline SQL, database credentials, encryption keys, or other sensitive information. Cool! And Hot.

This is definitely what we need. Actually, it's exactly what we need. Simply following along after "Here is an example scenario" gives us what we need in order to find the solution. His first step is to get the Subversion database:

wget http://www.sometarget.tgt/.svn/wc.db

Sure enough, if I add "/.svn/wc.db" after the Zone 5 URL I got when I entered a bad password, I get a SQLite database. I use curl instead of wget, but the result is the same:

`curl -O http://snowmiser.counterhack.com/zone-5-89DE9B26-CF7D-4B07-88DE-7A2F0A7B16FE/.svn/wc.db`

Then, I just follow along the instructions from Tim's article, and end up with a copy of two PHP files: noaccess.php (which is what I got when I entered a bad password) and index.php (which is presumably what validates my password)

```
$ sqlite3 wc.db 'select local_relpath, ".svn/pristine/" || substr(checksum,7,2) || "/" || substr(checksum,7) || ".svn-base" as alpha from NODES;'

|

n o a c c e s s . p h p | . s v n / p r i s t i n e / 41/4134e0e954d144ed932fd639b5a897f9ad47fff9.svn-base

i n d e x . p h p | . s v n / p r i s t i n e / 7 d / 7d63810b0da679648fc20b4f1c84680ac08ec872.svn-base
```

Download those files with curl and open them in a text editor. This reveals the logic that validates the password:

```
<?php
function generate_otp($time) {
    $pass = sha1("$time 7998f77a7dc74f182a76219d7ee58db38be3841c");
    return($pass);
}

function verify_otp($inpass) {
    // passwords are valid for up to 3 minutes
    // don't forget to use the server time (see the noaccess.php page)
    $validstamps = array(
        date('Y-m-d H:i', strtotime('+1 minute')), // added just in cas
        date('Y-m-d H:i'),
        date('Y-m-d H:i', strtotime('-1 minute')),
        date('Y-m-d H:i', strtotime('-2 minute')),
    );
```

From this we can tell that the password is a SHA-1 hash of the current timestamp and a salt (a random-looking string) that starts with "[space]7998f". And we see that the timestamp can be any of previous two minutes, the current minute, or the next minute. We also see that "noaccess.php" will tell us the current server time, so we don't need to worry about synchronizing our clocks.

Since my Mac doesn't have sha1sum on it, and I got Raspberry Pi for Christmas that I'm looking for something to do with, I'll use that for this step. I'll do this with environment variables, to save typing in case I need several tries.

First, get the current server's time, and the proper format for it, from the noaccess.php page:

```
49          <h1>Access Denied</h1>
50 <!-- current server time is 2012-12-26 13:10 -->
51      </div>
```

Next, set an environment variable to hold the salt, and another to hold the time.

Then, echo them together (with that space between them) (and with the -n switch to avoid adding a newline character) and pipe that into sha1sum to get what should be the password:

```
pi@raspberrypi ~ $ export SALT=7998f77a7dc74f182a76219d7ee58db38be3841c
pi@raspberrypi ~ $ export TMS='2012-12-26 13:10'
pi@raspberrypi ~ $ echo -n $TMS $SALT | sha1sum
dd20c9af621fe73c31a3f7fd9654383b7a7fac00  -
pi@raspberrypi ~ $
```

Now, go back and enter that SHA1 hash (but not the hyphen after it) as the password, and you're in. Quickly click "disable" before the clock ticks past the lifetime of your hash.

`zone-5-89DE9B26-CF7D-4B07-88DE-7A2F0A7B16FE`

Snow Miser Level 5 Flag: `3ab1c5fa327343721bc798f116be8dc6`

Snow Miser's been hacked. The North Pole begins to thaw...

4.Heat Miser

4.1 Heat Miser, Getting To Level 1

Looking at the Level zero page, there's this "warning" in the text: *"We had a security concern where the Zone 1 URL ended up in search engine results. We added a file to prevent the search engines from caching these pages."*

The file that prevents a (well-behaved) search engine from indexing pages is /robots.txt - looking there, I find this:

```
User-agent: *
Disallow: /zone-1-E919DBF1-E4FA-4141-97C4-3F38693D2161
Disallow: /zone-2-*
Disallow: /zone-3-*
Disallow: /zone-4-*
Disallow: /zone-5-*
```

`/zone-1-E919DBF1-E4FA-4141-97C4-3F38693D2161`

Zone 1 flag is: `d8c94233daef256c42bb95bd61382e02`

4.2 Heat Miser, Getting To Level 2

The text here says they had to "temporarily remove the link" to Zone 2. When a developer has to "temporarily remove" something, they often just comment it out. Checking the HTML source for this page, I see that's what's been done here:

```
57 <p>We had an issue with
58 <!-- redacted, too many people clicked on the link and took it offline
59 <a href="/zone-2-761EBBCF-099F-4DB0-B63F-9ADC61825D49">Zone 2</a>
60 -->
61 Zone 2 and we had to temporarily remove the link. It is now back and in
```

That's the Zone 2 URL right there in the comments:

`/zone-2-761EBBCF-099F-4DB0-B63F-9ADC61825D49`

Zone 2 Flag: `ef963731de7e886226fe4a6a6c2971f1`

4.3 Heat Miser, Getting To Level 3

This page has some text that catches my attention:

The new zone 3 link starts with zone-3-83FEE8BE-B1C6-4395-A56A-XXXXXXXXXXXX.

That's a lot like Snow Miser's Level 1 puzzle. Let's go back to our resources. Heat Miser has posted a single image in his Twitter stream as well, but it's a screenshot of Metasploit. Metasploit is out of scope for this engagement, so that's not the clue.

I find this in Snow Miser's Twitter history:

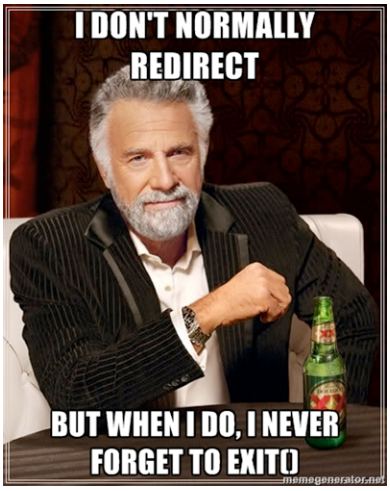
Adjusting the color and contrast in the image, I see that it's similar to the Snow Miser image - in a window behind the Metasploit window, quite faint and blurry, there's the missing part of the URL, with 4395-A56A- from the given text in the page:

zone-3-83FEE8BE-B1C6-4395-A56A-BF933FC85254/

Zone 3 flag: 0d524fb8d8f9f88eb9da5b286661a824

4.4 Heat Miser, Getting To Level 4

This page has a direct link to Zone 4, but when I click it, I get a page saying "Access Denied". Looking at our resources again, I see Snow Miser taunting Heat Miser with this meme image:



I'm not all that familiar with PHP, so it's off to Google to search for php and redirect and exit().

The top hit is this page: http://stackoverflow.com/questions/2747791/why-i-have-to-call-exit-after-redirection-through-headerlocation-in-php which points out that if you don't call exit() after setting the HTTP 302 and Location header, PHP will execute the rest of the script.

Sure enough, looking at this sequence in an intercepting HTTP proxy, I see the full HTML comes down in the same response as the HTTP 302. My browser automatically goes to the "/noaccess.php" page, but I can see what I need in the HTML body:

/zone-4-0F2EA639-19BF-40DD-A38D-635E1344C02B/

Flag for Zone 4: e3ae414e6d428c3b0c7cff03783e305f

Link to Zone 5:

/zone-5-15614E3A-CEA7-4A28-A85A-D688CC418287/

4.5 Heat Miser, Getting To Level 5

The link to Zone 5 was found in level 4, but visiting it gives a similar "No Access" message as was seen before. In this case, though, it's the immediate response, not a redirect.

Looking back at our resources, I notice one of the questions is, "*How did you manipulate the cookie to get to Zone 5 of Heat Miser's Control System?*" - so let's look at that cookie. None of the previous stages have used a cookie at all.

The cookie is called UID and contains a 32-character long hexidecimal string. Could be an MD5 hash. We'll use Google as a kind of rainbow table - search for the string 'b8c37e33defde51cf91e1e03e51657da' and the first response says it's a hash of the string '1001'

Snow Miser has a related tweet:

UID implies "user ID" so I tried zero (the UID of the Linux root user) and 501 (Windows admin) but got nowhere. I set up a script to iterate from 1 to 1000, and when I tested it, I found that I got a totally different - and much longer - response when I sent the hash of 1 than when I sent the hash of 2 or 3 or anything else.

The md5 hash of '1' is c4ca4238a0b923820dcc509a6f75849b. When I set the cookie to that value, I get to the Zone 5 Controller where I can click the "Disable" button.

Zone 5 flag: f478c549e37fa33467241d847f862e6f

Heat Miser is hacked, and flakes begin to fall in Southtown.

5.Summary: Levels and Flags

Table 1. Flags Discovered at Each Zone for SM and HM

Level	Flag
SM L1	38bef0b61ba8edda377b626fe6708bfa
SM L2	b8231c2bac801b54f732cfbdcd7e47b7
SM L3	08ba610172aade5d1c8ea738013a2e99
SM L4	de32b158f102a60aba7de3ee8d5d265a
SM L5	3ab1c5fa327343721bc798f116be8dc6
HM L1	d8c94233daef256c42bb95bd61382e02
HM L2	ef963731de7e886226fe4a6a6c2971f1
HM L3	0d524fb8d8f9f88eb9da5b286661a824
HM L4	3ae414e6d428c3b0c7cff03783e305f
HM L5	f478c549e37fa33467241d847f862e6f

5.1 ACKNOWLEDGMENTS

My thanks to Mark Elliott (@mge007) for his subtle hint at how to analyze the Android dump, which confirmed I was on the right track, but showed an easier route.

Thanks also to SANS, Ed Skoudis, Tim Medin and all involved in creating and running the puzzle.

Thanks also, to everyone who solved this before I did but didn't share the answers. Thank you for no spoilers!

6.REFERENCES

For further study, these references point to information about the tools and other factors used in this exercise.

[1] Steghide: tool for hiding information by steganography in image files: http://steghide.sourceforge.net

[2] xxd: creates a hex dump of a given file or standard input. It can also convert a hex dump back to its original binary form.: http://linuxcommand.org/man_pages/xxd1.html

[3] PHP exit() function call - terminate a script early: http://us3.php.net/manual/en/function.exit.php

[4] robots.txt: file to ask automated user agents to ignore certain files or directories: http://www.robotstxt.org/robotstxt.html

[5] sha1sum: command line utility to calculate the SHA1 hash of arbitrary data: http://en.wikipedia.org/wiki/Sha1sum

[6] md5sum: command line utility to calculate the MD5 hash of arbitrary data: http://en.wikipedia.org/wiki/Md5sum

[7] BurpSuite: intercepting HTTP proxy allows analysis and manipulation of HTTP traffic between a browser and a web server. http://www.portswigger.net/burp/

[8] curl: command line utility to interact with HTTP servers. Similar to wget. http://curl.haxx.se

[9] wget: command line utility to interact with HTTP servers. Similar to curl: http://www.gnu.org/software/wget/

[10] exiftool: command line utility to view and edit EXIF data in image files: http://www.sno.phy.queensu.ca/~phil/exiftool/

[11] Raspberry Pi: a fully-functional computer the size of a deck of cards, costing under $50. Default operating system includes some of the tools used in this exercise. http://www.raspberrypi.org

[12] xor "encryption" explained: http://www.devshed.com/c/a/Security/A-Sequel-to-Cryptography/2/

[13] Discussion of ways to do xor in Python: http://stackoverflow.com/questions/11119632/bitwise-xor-of-hex-numbers-in-python

[14] strings: Linux command to extract readable strings from arbitrary files: http://linux.die.net/man/1/strings

[15] BackTrack Linux: full operating system with security-related tools. When run as a virtual machine alongside other operating systems, offers great flexibility and a many useful tools preconfigured: http://www.backtrack-linux.org