SANS Holiday Hack

Helping Mrs Claus save Christmas!

Adrián Bravo Navarro

22/12/2012

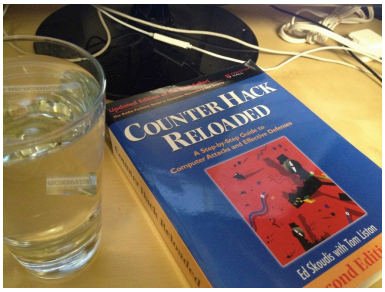
Table of Contents

Adrian Bravo Navarro

SANS Holiday Hack

Helping Mrs Claus save Christmas!

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

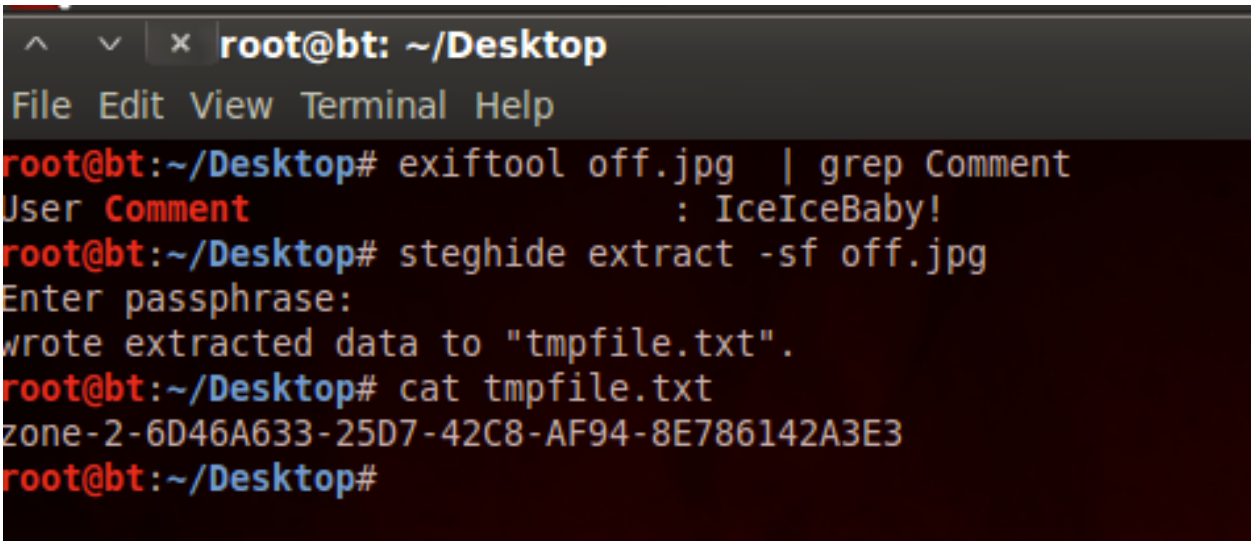


Sn0w M1s3r leaking info!

Snow Miser's zone URL was leaked in the twitter image on the right. There is a reflection on the glass of water that with a little bit of Photoshop treatment becomes perfectly sharp as shown below.

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

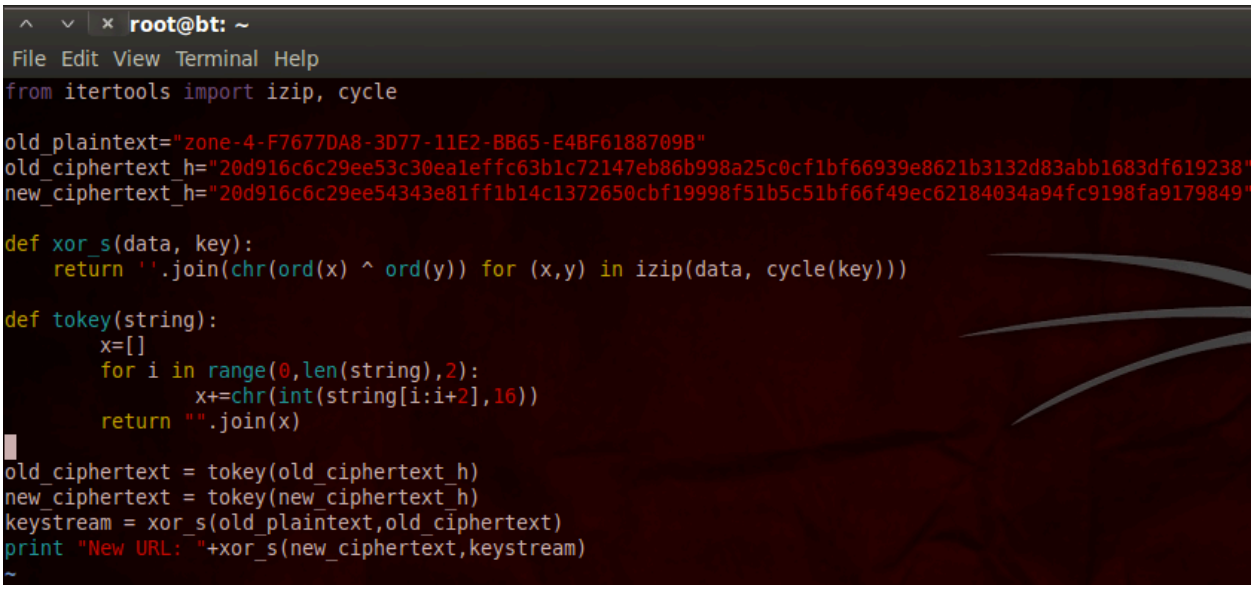
The key used to extract Snow Miser's Zone 2 URL is "IceIceBaby!". It was hidden as a comment on the JPEG image off.jpeg displayed when the "Disabled" button was hit. I recovered the key with exiftool and then used it with steghide as shown below.



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

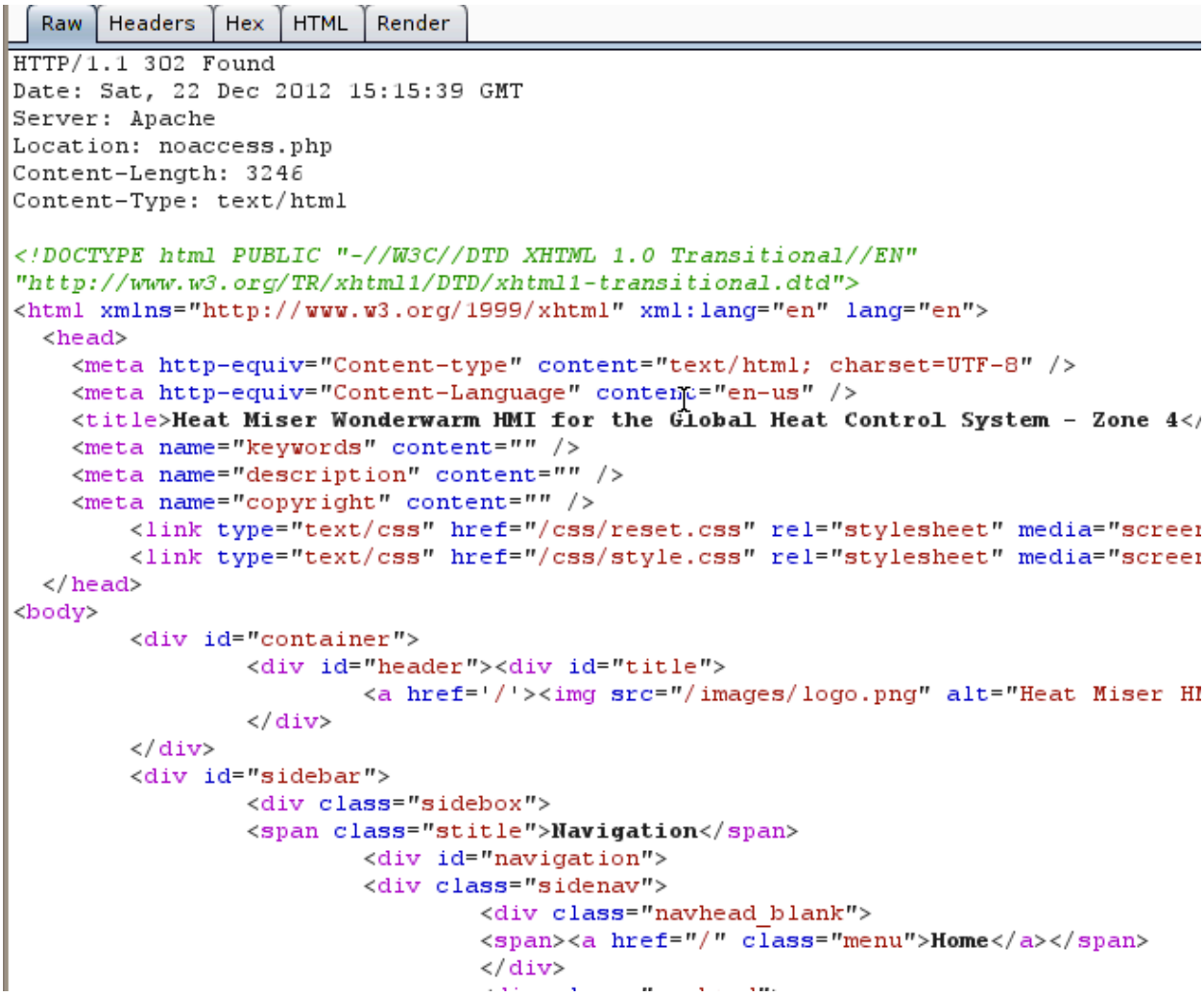
They are using a stream cipher to protect Zone 4's URL. The "law of stream ciphers" states that you must never repeat the key. The reason is that given a known plaintext and ciphertext pair, the attacker can obtain a keystream that can be used to decrypt other ciphertexts.

That's the case with Zone 4's URL. The ciphertext is displayed as hexadecimal so it can be printed to the screen. We need to revert it back to the binary representation, and the XOR it with the old URL's plaintext to obtain the keystream. Then we XOR this keystream with the binary representation of the new URL's ciphertext to obtain the new URL's plaintext. The following python code was used:



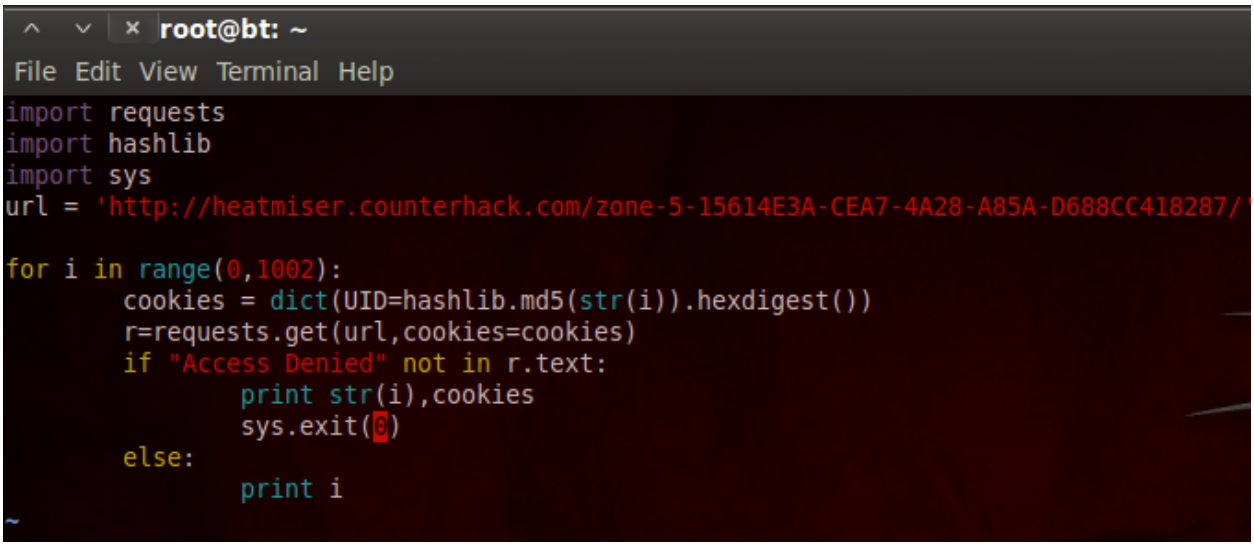
What was the coding error in Zone 4 of Heat Miser's site that allowed you to find the URL for Zone 5?

Zone 4 redirects via 302 to noaccess.php but they forgot to close the connection right after redirecting, so the rest of Zone 4's code executes and we can see the Zone 5 URL in the 302 response. I changed in Burp the 302 to a 200 in order to see the site more comfortably.



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

Seeing the cookie, I made an educated guess that it was the md5 hash of some key. Given that the cookie is a UID it was likely a number. Based on Snow Miser's twitter hint, I checked the value md5(1001) and verified that was exactly the value used by the application. Based on those facts, I did a small python script to connect to Zone 5 sending the md5 hashed values of the numbers from 0 to 1000, stopping once the right value was found. The sought value turned out to be 1.



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.

NOTE: For all levels below, Burp proxy was used to inspect traffic.

Heat Miser

Zone 0:

Flag: 1732bcff12e6550ff9ea44d594001418

The robots.txt file contains the URL to zone 1. Based on the text on the page, I did a quick Google search with the site operand to no avail. Then I looked at the robots.txt file.

Zone 1:

Flag: d8c94233daef256c42bb95bd61382e02

Peeking at the HTML source there's a comment with the URL to zone 2

Zone 2:

Flag: ef963731de7e886226fe4a6a6c2971f1

Based on the text on the page, it looks like we're after some out of band information leak. Reading the tweets, Snow Miser scolds his brother about his terminal being transparent… Downloading the image and using Photoshop on it, it's possible to see the URL to Zone 3.

Zone 3:

Flag: 0d524fb8d8f9f88eb9da5b286661a824

URL to Zone 4 is visible, however when clicking on it, we get a 302 to noaccess.php. However, a closer inspection at the response reveals that they forgot to close/die the connection after redirecting, thus the script continues execution and the HTML for Zone 4 is visible. I changed the 302 for a 200 OK to look at the page in a more comfortable way.

Zone 4:

Flag: e3ae414e6d428c3b0c7cff03783e305f

When submitting the form we are taken to Zone 5 URL, but before that, the application issues a cookie named UID. When we reach the Zone 5 page with that cookie, we land in noaccess.php. A close inspection at the cookie and an educated guess later, we can verify that the hashed value is actually md5(1001). A handful of Python lines later, using Requests and Hashlib, we found out that the cookie granting access to Zone 5 is md5(1). The code is shown in one of the above questions.

Zone 5:

Flag: f478c549e37fa33467241d847f862e6f

Snow Miser:

Zone 0:

Flag: 3b5a630fc67251aa5555f4979787c93f

Based on the page text, it looks like there's nothing to do on it, so we're out to find something on the tweets of any of the brothers. A closer inspection to the image in Snow Miser's timeline, picturing a glass of water, reveals that the last two parts of the URL are slightly visible on the reflection in the glass of water. A minute of Photoshop later we see the clear URL value.

Zone 1:

Flag: 38bef0b61ba8edda377b626fe6708bfa

The off image on this zone is JPEG image, whereas all the others' have been PNG so far. That's suspicious and as such, I used exiftool to extract any possible metadata within it. There is a user comment reading "IceIceBaby!". Looking for steganography tools included in BackTrack as a first shot (apt-cache search steganography) the list's best looking tool was steghide. Using it on the off.jpeg image, with the key present on the JPEG comment returns the URL to Zone 2.

Zone 2:

Flag: b8231c2bac801b54f732cfbdcd7e47b7

After looking at the page to no avail, Heat Miser mentioned something about Snow Miser's phone being found and the data extracted. I downloaded the file. Went straight to the browser's cache with my hopes lying on Snow Miser browsing his own Zones. Within data/data/com.android.browser/cache file, using strings command we see several URL's, the one returning 200 OK is our URL to Zone 3.

Zone 3:

Flag: 08ba610172aade5d1c8ea738013a2e99

Measuring the lengths of both ciphertext and plaintext, it's obvious that the length of the ciphertext is twice the length of the plaintext. The length of the ciphertext is not multiple of 8,16,32, so it's unlikely they used a block cipher. I then realized the ciphertext must have been converted to hex representation to be printable, and based on that, both plaintext and ciphertext would be same size, so the stream cipher theory gains strength. Using a python script shown in the questions above I turned the ciphertext back to binary format and XORed it with the old URL to obtain a keystream that I used to XOR with the new URL ciphertext to obtain Zone 4's URL.

Zone 4:

Flag: de32b158f102a60aba7de3ee8d5d265a

Clicking on the authenticate button, we are taken to Zone 5. Unfortunately, without the right value on the input field, we are denied access. We know we need to provide a valid OTP. Based on the tweeter of the Miser's brothers, we know there's a .svn folder lingering around Zone 5. Following Tim's guidance (http://pen-testing.sans.org/blog/pen-testing/2012/12/06/all-your-svn-are-belong-to-us you) we downloaded the wc.db file using the browser. Using sqlite3 client I queried the database to obtain the files contained within it. Once I got the svn file's name, I downloaded both index.php and noaccess.php. Reading the code I found out the OTP is created by hashing the current server date, a whitespace, and a hardcoded value present on index.php code. Finding the current server date is easy as it is displayed on a HTML comment inside noaccess.php output. The code reveals the OTP keys are valid during a 3 minute window. A small python line returns a valid OTP:

```
hashlib.sha1('2012-12-22 06:45' + ' ' + '7998f77a7dc74f182a76219d7ee58db38be3841c'
).hexdigest()
```

Zone 5:

Flag: 3ab1c5fa327343721bc798f116be8dc6