```
    s.send('bullshit')
    data = s.recv(1024)
    if data:
        print("ANSWER: " + data)
    s.close()
```

This eventually returns the flag

```
output:
    trying: cat flag;DEXotxgPoDkrkHpgaVubVrFGOvTOPWgreL
    TocJHJmZGpLeLgChPKeLCjLYDFacUyxXVcdrUqmdRPVLIAwtgKj
    JsRPK1mhRTCTOamDyaqTFQfRFtfNWEbpcMIfupVOzMkejNAXYhW
    ujNGWAbMVOGfUdotGMjFOfhzLKjzoKTQXEFTCtwBqfndBvDmdWI
    pJOXauQitisLLouNDksWcOcISPqPenZSdsfoOHvglTqWIEjCvAb
    IyqpRjWHw
ANSWER: bkp{and you did not even have to break sha1}
```

Prudentialv2

Category: web

Posted on March 07, 2017

This challenge took you to a web page with two input boxes, username and password. Upon examining the webpage's source, we see this in the PHP.

```
require 'flag.php';

if (isset($_GET['name']) and isset($_GET['password'])) {
    $name = (string)$_GET['name'];
    $password = (string)$_GET['password'];

    if ($name == $password) {
        print 'Your password can not be your name.';
    } else if (sha1($name) === sha1($password)) {
      die('Flag: '.$flag);
    } else {
        print '

  Invalid password.


';
    }
}
```

It appears to print out the flag if the two strings given to it are not equal, while their SHA1 hashes are. This is commonly called a "collision". However, hash functions are typically constructed so that it is practically impossible to find a collision.

Luckily, if you're even remotely involved in the tech community, you would have heard the big news several days before this CTF.

The first SHA1 collision has been found. (https://shattered.it/)

They've even conveniently provided 2 files that produce a collision! Thus, we immediately tried to submit these PDF's as text files (through Python and the urllib.parse.quote function). Unluckily, we get back a "414 Request-URI Too Large" error.

Clearly, we cannot encode the entire PDF's. After some googling, we found a tool for creating your own SHA1 collisions (https://alf.nu/SHA1). Huh? I thought creating your own collision took a ton of time and was impractical? There must be some method that we can use to create collisions.

Browsing through the associated HN thread, we found a nice explanation (https://news.ycombinator.com/item?id=13728294) of what was going on.

"Yeah, I think that's pretty much the case. The first 320 bytes of the two PDFs released by Google result in the same SHA-1 state. Once you're at that point as long as you append identical data to each of the files you're going to get identical hashes. This is just taking those same 320 bytes and appending the combined images of your choice."

Therefore, only the first 320 bytes matter. Now, we can submit merely the first 320 bytes of each PDF.

```
import hashlib
import requests
import urllib
import base64

hasher = hashlib.sha1()
BLOCKSIZE = 320
buf1 =''
with open('a.pdf', 'rb') as afile:
    buf1 = afile.read(BLOCKSIZE)
    hasher.update(buf1)
print(hasher.hexdigest())
hasher = hashlib.sha1()
buf2=''
with open('b.pdf', 'rb') as afile:
    buf2 = afile.read(BLOCKSIZE)
    hasher.update(buf2)
print(hasher.hexdigest())

name = urllib.parse.quote(buf1)
password = urllib.parse.quote(buf2)
url = 'http://54.202.82.13/?name='+name+'&password='+password

r = requests.get(url)
print(r.content)
#FLAG{AfterThursdayWeHadToReduceThePointValue}
```

Sponge

Category: Crypto

Posted on March 07, 2017

The problem for this challenge was

> “ I've written a hash function. Come up with a string that collides with "I love using sponges for crypto".

The clue linked to to a python script (https://gist.github.com/tjade273/e771894ba9ade41aa6857072ffdda208#file-hash-py) containing the code for the hash validation server.

Reading through the code, we see that the hash function does the following: