Eddie Goynes
HW4 Network Security
931719883

ECE 478 Network Security: Homework #4

Disclamer

This submission reflects my own understanding of the homework and solutions. All of the ideas are my own, unless I explicitly acknowledge otherwise.

1. Describe clearly the format & contents of the cookie set by the server. How does the page authenticate a user's cookie?

   The format and contents of the Cookie set by the server is described by this code.

```
($username, $timestamp, $mac) = split /\./, $dat;
```

   So the cookie will look something along the lines of
   username.230820142214.mac

   Of course this will be hashed and hex encoded. So will look like a string of hex characters to the browser.

   The last part of the cookie some long hex value is created with the following process.

   1. First dumb_mac is called.

```
sub dumb_mac {
   my ($data) = @_;
   return hash( substr( hash($data), 0, $TRUNCATION_LENGTH) . $KEY );
}
```

   It hashes the data from my pipe concatenates the key to the end then hashes the whole function again. The only way to break a system like this will be to see if we can create hash collisions with the birthday bounds idea, we can do this using md5 encoding. This collision will tell us a username and timestamp that has the same hash output as admin.anytimestamp.

2. When you create an account and get an associated token, what parts of the token can you control completely? What parts can you predict?

   We can control the account name and predict the timestamp. Since we know when we create the account. We cannot control the mac created from the dumb_mac function. We can possibly predict the value of the mac. It's a hash of the account.timestamp value, concatenated with some key value then hashed again. Then this value is hex encoded. The timestamp we can predict because it's in the form
   ## current time in "ddmmyyyyhhmm", e.g., august 23 @ 10:14pm => "230820142214"

Since it only goes down to the minute we pick some minute in the future that we want to make our account to duplicate the hash.

3. What must a token contain in order to confer admin access? What parts of the token don't matter so much?

   We don't care about the timestamp since that is not checked for validation. The username is checked for validation. So we want the username to be admin. We need to have the correct mac value to make the hash correct. This is why we need to get a time stamp that is in the future that we can have combined with the admin username. In the code it doesn't explicitly compare admin to the account name that is passed in, it compares the hashed and hex encoded output.

4. Explain how a collision under MD5trunc could be used to gain a valid admin token. How much effort should it take to find collisions under MD5trunc?

   Since we only need to check the first 4 bytes of the hash to find out if there is a collision match. The equation is 2^(n/2) which would be 2^(32 / 2) . This value comes out to be 65536 different values that we will have to check. Each value will be checked in a negligible amount of time, so this may only take a few minutes. I will have to remember when choosing a timestamp to choose approximately 10 minutes in the future to give me plenty of time to computer the hash and login.

5. OK, now find a collision under MD5trunc satisfying the properties that you described in the previous questions. Describe your approach for finding the collision (provide your code if it's not too long, but a short description/pseudocode will be enough), and also show the actual values that collide.

   The general idea of this assignment is to find out what username and timestamp create a hex encoded hash that matches the hex encoded hash of admin.anytimestamp. In the code I use the hashlib to do md5 hashing. The idea is to make a long list of admin.randomtimestamp hash values. Then create a long list of randomusername.timestampinthefuture values. We then compare the hex encoded hashes of each of these values in every combination. If any of these match then we have a collision, and a possible username and timestamp to hack into the website that the professor has provided.

With the code below I actually was unable to find any collisions. It ran for almost an hour. Don't know if I was taking the right approach. Based on what I think I should be doing the code seems like it should be correct.

```
import os
import sys
import hashlib
import string
import random



def main():

    for j in range(65000):
        admin = hashlib.md5("admin." + str(random.uniform(100000000000,
900000000000)) ).hexdigest()[0:8:1]
        for i in range(100000):

            test1 = (''.join(random.choice(string.ascii_lowercase) for
i in range(4)))

            if admin == hashlib.md5(test1).hexdigest()[0:8:1]:
                print test1

if __name__ == "__main__":
    main()
```

6. Describe the mechanics of your final attack to gain admin access. What data did you send to the form? What did you get back in response? How did you manipulate it to get admin access?

   If I was able to find a collision I would be able to take the username that I found and during the minute that I had set the timestamp to I would create the account with that username. One I had created the account I would be able to take the cookie created and use that to log in as the Admin.

7. Let's find out how sensitive this problem is to the fact that we truncated the inner MD5 hash to 4 bytes. I have set up the page so that when you access it via hash-hw.cgi?5, hash-hw.cgi?6, etc, it will truncate the inner MD5 hash to 5 bytes, 6 bytes, etc, instead of 4 bytes. Authentication tokens generated with one truncation-length will not be compatible with other lengths. Also, the secret admin messages are different for the different truncation-lengths. How many of these truncation-lengths can you compromise to gain admin access? For each one that you were able to break (including length=4), list the hash collision and the secret admin message.

   Since I was unable to get the admin cookie from hashing for the 4 byte length I don't think I will be able to get the value for higher byte truncation lengths. Hopefully I just had made some mistake in my code that doesn't hash or hex the values exactly how was needed for this homework assignment.