# Extracting and saving data from accessible cloud account – research

## Ephraim Berkovitch, Aug 2019

*The task is to extract and save data from an accessible cloud account (Gmail, Google Drive),  
assuming we have a standard logged on user (our own google cookie or API token).*  
Required:

- a description of the research steps you took.  
- a description of the different security measures you observed.  
- a description of the possible ways to bypass or get around those security measures.  
- a description of Google’s behavior - how exactly can you get the relevant information? how is the data extracted using automatic methods different than data accessible with a standard interactive method?  
- blockage from the service can be irreversible - you have to stay undetected

Preliminary research and understanding the problem:  
1) let’s pay an attention, that we have google cookie or API token, and not user name + password (plain credentials). This forwards our research in certain direction – we do not have to study Google API, relying on user’s plain credentials and API key, got from Google (or any other cloud provider), but exploit somehow reached cookie and/or security tokens for a user.   
2) For the same reason, we do not have to (and may be, can not) get a Google developer API key, in order not to be recognized by service provider

3) how can we get user’s token? this is not necessary for this specific exercise, as it is assumed to be given in advance,   
but we should try at least to figure out the main steps for getting it – snooping, exploiting not-secured (HTTP) protocols, etc.

4) where and how do we store the received data? RDBMS, No-SQL? Where do we store assets? AWS S3 / Google cloud storage? Some simpler provider, like Digital Ocean, or just our own servers, not relying on any external provider?  
5) scalability of this service - not required here, but has to be figured out at least in main steps – how can we painlessly grow from obtaining data to a few users to dozens of simultaneous users?  
6) where the code is running? – client’s computer / cloud EC2-like instance / our own servers?  
7) is it scheduled to run every X minutes? or running in an infinite loop?  
8) does it create an instance / process / thread for each user, we are scraping his data?  
9) what are we interested to extract:  
 1) emails  
 2) contacts  
 3) calendar  
 4) Google drive items  
 5) tasks  
 6) passwords to external services - FB, Twitter, etc.?  
10) does google recognise and block rate limiting?  
11) how often the security token is refreshed, and which other mechanisms we are relying on, how often are they being refreshed?

12) does Google report failed authentication requests, and accordingly blocks an incoming IP after a number of consequent failed requests?

First Attempt

The simplest Python mechanism to get data by simulating functionality SPAs, like gmail / google cloud, is to use the urllib module, or better, the ‘requests’ module, encapsulating it:

import requests

resp = requests.post(url=url, data=payload, headers=headers, cookies=cookies)

print(resp.json())

* **urls** are varying according to the specific requested service (can be inspected in Google Chrome Developer Tools -> Network, for example)
* **data** is sent for POST requests, in Python dictionary format
* **headers** are important esp. for authentication purpose
* some web sites make heavy usage of **cookies**, besides headers and payload, for better security (they can somehow contribute indeed to security being encrypted and frequently refreshed)

Pic 1 – finding gmail endpoints for querying data

A screenshot of a social media post

Description automatically generated

Pic 2 – inspecting headers. I noticed the message about “provisional headers”, which explanation I found for example, below:

<https://stackoverflow.com/questions/21177387/caution-provisional-headers-are-shown-in-chrome-debugger>

A picture containing indoor

Description automatically generated

As a result of this behavior, I would consider using another web browser than Google Chrome, for further figuring out, how to simulate gmail / google drive functionality, may be, Opera, Firefox, or something built-in in Kali Linux

Pic 3 – inspecting query strings and POST payload:

A screenshot of a cell phone

Description automatically generated

Pic 4 – cookies, along with their expiration dates:

Here I was looking for cookies expiring soon, esp. today, though there is an option that cookie expiration date can be intentionally misleading, and even cookies, not expiring soon, are being refreshed frequently, such invalidating the requests.

I have some suspicion, that frequent google background calls to endpoints, like /logs, /s etc. can be intended to refresh tokens and cookies.

(<https://play.google.com/log?format=json&hasfast=true&authuser=0>, for example)

A screenshot of a cell phone

Description automatically generated

Pic 5

Local and session storage should be investigated, as well. I did not see any usage of them, made by google, although I might be wrong, and it worth further investigation

A screenshot of a cell phone

Description automatically generated

Pic 6 – we see, that in many cases OPTIONS is first invoked, and only then GET

A screenshot of a cell phone

Description automatically generated

Pic 7 – calls to Google Drive seem to be simpler, than to /tasks, /calendar, /mail, etc. – it looks like a simple GET request, with pretty long list of query parameters. It worth further investigation, though

A screenshot of a social media post

Description automatically generated

Above is some preliminary technical research to try to query Google. I did not succeed to figure out all the necessary details – exact list of headers, cookies, local storage items, frequency of their refresh, etc., needed in order to avoid the “unauthorized” response, I kept getting from google.

Preliminary answers to other questions:

4) where and how do we store the received data? RDBMS, No-SQL? Where do we store assets? AWS S3 / Google cloud storage? Some simpler provider, like Digital Ocean, or just our own servers, not relying on any external provider?

- calendar items can be better stored in tabular form, in DB like Postgres for better querying

- emails can not be simply structured, so document DB, like MongoDB, looks like better solution for them;

- contacts could be better stored in graph DB, like Neo4J

- assets – google drive files – should be stored in some storage, as AWS S3, with just a reference to them on some DB, We should better choose this way, and not story any BLOBs in our DBs, which might complicate usage and downgrade performance of our DB, without any significant contribution to querying these items

5) scalability of this service - not required here, but has to be figured out at least in main steps – how can we painlessly grow from obtaining data to a few users to dozens of simultaneous users?

- functionality of this service is not too heavy (perform some url requests and store results somewhere), after figuring out, what actually has to be done, so some solution, similar to AWS Lambda or Google functions look like a good fit and mostly scalable

6) where the code is running? – client’s computer / cloud EC2-like instance / our own servers?

- Most comfortable deployment would be to some cloud solution, like AWS Lambda or Google Functions, if the security and confidentiality permits that. If not, we have to look for some in-house Lambda-like solution for scalability (see <https://binaris.com/>

, for example)

7) is it scheduled to run every X minutes? or running in an infinite loop?

- I would prefer trigger-based solution – upon each acceptance of security token, invoke the process (lambda function) to get as much data as possible, till it expires;

- scheduling seems less relevant here, as it has to continuously poll for unused available security tokens, although it would certainly work, esp. if trigger-based deployment is not available

- infinite loop seems less relevant, because of scalability issues and recovery difficulties (what happens, if the infinite process crashes? How are we going to monitor these crashes and launch the process again?)

8) does it create an instance / process / thread for each user, we are scraping his data?

- if we choose serverless (lambda-like solution), this question is irrelevant. If not, creating a new worker (process) by supervisord, for example, looks more robust solution, as anyway processing two different security tokens does not involve shared memory, from which we enjoy using threads, and independent processes are more powerful and much easier to control and scale – both vertically and horizontally

**Bibliography** that can be used – I did not use it for this research, but I definitely would, if I had more time ☺

1. Justin Seitz, Black Hat Python
2. Justin Seitz, Gray Hat Python
3. Jon Erickson, Hacking: The Art of Exploitation