Content for API page. Have text centered, double spaced. Do that for paragraphs. Then steal someone’s settings for code setting apart. Have that be a div with class code, and have CSS do that as the set apart code. No run buttons, except for at the end.

Pages,

Introduction

Should have what the webpage is, the link to the API documentation, and a basic explanation of what the purpose is.

A tale of two APIs,

Link the article talking about both of the APIS. Show code to do the first api, the simple get url request for the first API. Talk about how the first API is deprecated and that we should use the second because it is richer.

V2 API,

Link to the good documentation. Explain a bit more in depth the three different ways that one can use the API. Go in to detail on the one he suggests and show your code.

Custom API for lists of email addresses. Check which of your friends is a cheater.

Here go in to detail on the API that I created. It should take a post request because posts don’t have length problems. The POST requests takes out the email addresses and makes get requests. Saves them in a JSON object. Return the JSON object from the POST request when you are out of email addresses. Suggest that it could be an Android app where you upload your address book and get a table of people with email addresses linked in Ashley Madison. Make sure to make a table that lists all of the email addresses provided by the JSON and indicates yes or no.

Introduction

The website haveibeenpwned.com ((HIBP) is a website that collects the username/password/any other available data from leaks that have occurred from various companies/websites. The information is quite detailed, and is available to search particular addresses.

The point of the website is that you can see if any of the accounts that you have created have been compromised. If passwords were leaked, it will tell you that as well. If a password was leaked, you should change any other account you might use that password on.

HIBP has an API that is publically available. It is on the simpler side of API’s, but can take a bit to get set up. You can get their documentation at <https://haveibeenpwned.com/API/v2>

Very Lax API Rules

HIBP has very lax API usage rules. No API key is necessary, there is no limiting on the number of requests. You must use SSL, but it will just convert your request if you do not. Cross-Origin Resource Sharing is also fully allowed.

Version 1 of the API is much simpler, and does not require an API key or any specification of who is making the request. Version 2 of the API still does not require an API key or registration, but does require that there be a User-Agent specified in the request header. More on both of these APIs on the next page. This data is all available elsewhere on the internet in bulk form so the owner of the webpage is not overly concerned with someone abusing the service.

A Tale of Two APIs

The older version of the API (version 1) was extraordinarily simple. The user merely had to send a get request to [https://haveibeenpwned.com/api/breachedaccount/{account}](https://haveibeenpwned.com/api/breachedaccount/%7baccount%7d) substituting the account information of course. No key or request headers necessary. JavaScript code that accomplishes that trivial task is below.

INSERT SIMPLE CALL

The second version of the API is still pretty simple, but actually has three methods that a person can use to access it. On top of the 3 methods, there are now more possible things to request from the API. V1 did just a JSON list of accounts that have been hacked for a particular email address. In V2 you can use the API to get breaches for an account, but also for all breached sites that they have record of, or all records for a single breached site, or for others.

The three methods to access the V2 API are interestingly discussed in “Your API versioning is wrong, which is why I decided to do it 3 different wrong ways”. In that article the owner of haveIbeenpwned.com discussed the general philosophy with his dilemma in designing the second iteration of the API for this website.

The problem comes with how to designate it is the 2nd version of the API. You are permitted to merely put v2 in the url, however that method is not preferred because the url should not be changing. The other two methods use the constant url, and either specify a custom request header called api-version, or use the same url and use content negotiation. With all methods you must specify the user id. I am going to use the static url and content negotiation method in the implementation on the next page. You can feel free to use any of the others should you so desire.

Implementation of the 2nd version of the HIBP API using content negotiation

The best option to use in my opinion with the 2nd version of the API is to specify in the request header the version, and to also specify in the request header the user agent. The user agent is the service that is requesting the information, it is required, but you can just make one up, you do not need to have registered it or anything like that. So to use the first version of the API, you merely use a get request with the url. To use the 2nd version of the API the best method is to use the url:

GET [https://haveibeenpwned.com/api/{service}/{parameter}](https://haveibeenpwned.com/api/%7bservice%7d/%7bparameter%7d)

And to specify in the request header:

Accept=application/vnd.haveibeenpwned.v2+json

You must also specify in the accept header. In the request below, I create a simple webpage that makes a request to HIBP using the 2nd version of the API. I have an old email of mine that was leaked hardcoded in this example, you could add your own, or make it user supplied very easily. These three lines set that up:

req.open('GET', sendAddress, true);

req.setRequestHeader("Accept", "application/vnd.haveibeenpwned.v2+json");

req.setRequestHeader("User-Agent", 'OSUTest');

The setRequestHeader lines set the content negotiation options (that dictate that we are using version 2 of the API) and also set the user agent, which is required by the API. If you do not send a User-Agent you will get a 403 error, no user agent specified. In this HTML page I do an AJAX request using that hard coded email. The response received is printed out to console, use Chrome and control+shift+J to view what is returned. You can easily integrate this into your webpage as well.

Custom API implementation

I set out originally to create an interface that would leverage the Ashley Madison hack data. Ashley Madison was supposedly a website where people involved in relationships would register presumably to facilitate cheating on their spouse. There was an enormous leak of account information on Ashley Madison and it is on HIBP. The intent of this customer interface was to allow a person to send a POST request containing a list of email addresses. The page would then do get requests using version 1 of the API to see if the email was leaked as part of the Ashley Madison hack. Unfortunately, (or really probably fortunately, this is sort of sensitive information) the creator of HIBP has anticipated uses such as this, and does not release the Ashley Madison data through the API, or to users except it will reveal if an email was in that leak if the user confirms that email. Of course that renders the planned example impossible.

I have built something similar, but not quite as useful. With this page, it accepts a POST request in application/x-www-form-urlencoded format. You would want to use the format:

[Email1=andrewjohnsonlaw@gmail.com&email2=jeff@jeff.com](mailto:Email1=andrewjohnsonlaw@gmail.com&email2=jeff@jeff.com)

For example. It will take each of those email addresses, and it will print to console the different hacks that those email addresses were a part of. Some problems exist with this implementation.

First, I read in all of the email addresses with the for(var email in req.body) line, and then send each in a get request using version 1 of the API. Since those are asynchronous calls, they do not show up right away. The result is returned therefore before the list can be sent back. I tried very hard to find a different way to do that, but I could not figure out a great way to hold up the returning until all of the requests were done.

Second, I was not able to pair the email address with each leak information. I tried many different ways, assuming I needed to create a closure, but it did not work. Consequently. This implementation will log to console all of the different leaks every email address appeared in.

INSERT CODE HERE.

This code could definitely be improved, but it is an example of how to implement a server sided call to the HIBP API. The advantage of using a POST through this method is the greater length allowed and the security of not having the addresses in the URL. This implementation did not use a hard coded url, but adds the email addresses as it goes.

Finally the Else if statement that is looking for error 403, in the first version of the API, 403 is returned if there are no records. This merely ignores that possibility and logs nothing when that happens.