<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to GraphQL</td>
<td>1</td>
</tr>
<tr>
<td>Type System</td>
<td>1</td>
</tr>
<tr>
<td>Object and Field Types</td>
<td>1</td>
</tr>
<tr>
<td>Query and Mutation Types</td>
<td>2</td>
</tr>
<tr>
<td>Get API Token</td>
<td>3</td>
</tr>
<tr>
<td>Get Account ID</td>
<td>4</td>
</tr>
<tr>
<td>Create a Security Profile</td>
<td>7</td>
</tr>
<tr>
<td>Get Security Profiles</td>
<td>10</td>
</tr>
<tr>
<td>Get Global Threat Analytics</td>
<td>13</td>
</tr>
<tr>
<td>Search for Global threats</td>
<td>23</td>
</tr>
<tr>
<td>Get an NSS ID’s Details</td>
<td>29</td>
</tr>
<tr>
<td>Downloading HTTP Traffic, PCAP Files, Shell Code and Malware Payload</td>
<td>34</td>
</tr>
<tr>
<td>Get Profile Details</td>
<td>38</td>
</tr>
<tr>
<td>GET List of Devices</td>
<td>43</td>
</tr>
<tr>
<td>Get List of Supported Applications, Browsers and Platform Packages</td>
<td>46</td>
</tr>
<tr>
<td>Submit a URL for Scan</td>
<td>50</td>
</tr>
<tr>
<td>Submit a File for Scan</td>
<td>53</td>
</tr>
<tr>
<td>Get URL/File Scan Status</td>
<td>56</td>
</tr>
<tr>
<td>Get Exploits/Malware Bypassing a Profile</td>
<td>60</td>
</tr>
<tr>
<td>API Reference Tables for URL and File Scan</td>
<td>62</td>
</tr>
<tr>
<td>Application Package Reference Table</td>
<td>62</td>
</tr>
<tr>
<td>Browser Package Reference Table</td>
<td>63</td>
</tr>
<tr>
<td>Platform Package Reference Table</td>
<td>63</td>
</tr>
</tbody>
</table>
INTRODUCTION TO GRAPHQL

GraphQL is a query language for your API, and a server-side runtime for executing queries by using a type system you define for your data. GraphQL is not tied to any specific database or storage engine and is instead backed by your existing code and data.

Type System

The GraphQL query language is about selecting fields on objects. So, for example, in the following query:

```graphql
{
  threatAnalytics {
    applicationThreats {
      count
    }
  }
}
```

1. Start with a special "root" object.
2. Select the `threatAnalytics` field.
3. For the object returned by `hero`, select the `applicationThreats` field, which has a `count` field.

Because the shape of a GraphQL query closely matches the result, you can predict what the query will return without knowing that much about the server. It is useful to have an exact description of the data we can ask for - what fields can we select? What kinds of objects might they return? What fields are available on those sub-objects? That's where the schema comes in. Every GraphQL service defines a set of types which completely describe the set of possible data you can query on that service. Then, when queries come in, they are validated and executed against that schema.

Object and Field Types

The most basic components of a GraphQL schema are object types, which represent a kind of object you can fetch from a service, and what fields it has. In the GraphQL schema language, we might represent it like this:

```graphql
type ExploitInfo {
  nsiid: String!
  discoveryDate: String!
  type: ExploitType!
  processCount: Int
  relatedProcesses: Int
  fileCount: Int
  connectionCount: Int
}
```

- **ExploitInfo** is a GraphQL Object Type, meaning it's a type with some fields. Most of the types in your schema will be object types
- **nsiid** through **connectionCount** are fields on the **ExploitInfo** type. That means that they are the only fields that can appear in any part of a GraphQL query that operates on the **ExploitInfo** type
- **String** is one of the built-in scalar types. These are types that resolve to a single scalar object, and can't have sub-selections in the query
- **String!** means that the field is non-nullable, meaning that the GraphQL service promises to always give you a value when you query this field. In the type language, we'll represent those with an exclamation mark.

- `[ExploitType]!` represents an array of `ExploitType` objects. Since it is also non-nullable, you can always expect an array (with zero or more items) when you query the `type` field.

**Query and Mutation Types**

Most types in a schema will just be normal object types, but there are two types that are special within a schema. Every service has a query type and may or may not have a mutation type. A query is a type that defines a data set that can be selected. A mutation is a type that defines an operation that creates or updates a data set. Here is an example of a query type called `threatInfo` that takes two parameters:

```graphql
{ threatInfo(start: String!, end: String!) {
  exploits {
    exploit,
    discoveryDate,
    type,
    productIdent,
    relatedExploits
  }
}
}
```
API Token

Every request sent to the CAWS 3 API requires an authorization header with a API Token as a value. This allows the API to ensure you are authenticated and allowed to send the request. The API Token can be generated by logging to CAWS UI and navigating to the Organizations -> License section.

How to generate the API Key:

- Login to CAWS UI [https://caws3.nsslabs.com](https://caws3.nsslabs.com) using your credentials
- Navigate to ORGANIZATIONS section by selecting ORGANIZATIONS from drop down menu
- Click on LICENSE to navigate to license tab
- Click on NEW API KEY button to generate a API Key
- Copy and save the API Key and use it with all the API Calls as defined in the examples

The screenshot below shows an example API KEY created using CAWS UI.

Example API Key:

1. MWM1LwV1NTQtNGUE_5_PoQHr1SIF1JBNd72s93q4Q28BMIe - hf62lpgw.........................

The API Key has been truncated for the convenience of documentation.
ACCOUNT ID

This query allows you get a collection of all organizations and accounts you have access to. In CAWS, the entitlement model allows a single organization to have multiple child accounts. And as a user, you may have access to multiple accounts. An account is the entry point to all operations you can perform using the API. As such, most API calls require an account that the API operation will be performed against. For example, to query for a collection of profiles, you need to provide the account whose profiles you want to query. This is a new query that has no equivalent in the CAWS 2 API.

Parameters

None

Request and Response Structure

See the online documentation(https://caws3.nsslabs.com/docs/securityprofileoverview.doc.html) for the complete request and response JSON structure for this query. As is the case for all GraphQL queries, only the fields you select in request JSON structure will be returned in the response JSON structure.

Curl Example API Call to Get Account ID

```
1. curl - X POST https://caws3.nsslabs.com/graphql -d query='query { accounts { id name organization { id name } } }' -H 'authorization: bearer <API Token>'
```

Example Output for Get Account Query

```
1. {
2.   "data": {
3.     "accounts": [{
4.       "id": "1",
5.       "name": "Account 1",
6.       "organization": {
7.         "id": "1",
8.         "name": "NSS 1"
9.       }
10.   }, {
11.     "id": "2",
12.     "name": "Account 2",
13.     "organization": {
14.       "id": "2",
15.       "name": "NSS 2"
16.     }
17.   }]
18. }]
19. }
```

Here is an example using Python. The example defines a function that you can reuse in a Python program. The code comments are pretty much self explanatory:

```
import json
import requests
```
# ...  
# Function to Get Your Organization and Account(s) Information.  
def get_organization(token):
    # Define Query to Send to Server.
    # The query that is sent to the server is either a GraphQL Query or a
    # GraphQL Mutation. In this example because we want to get data, a we are
    # sending a GraphQL Query.
    # Remember, GraphQL is graph based and we have to explicitly specify the
    # fields that we want returned in the response.
    # For an online reference to all the different GraphQL Queries and
    # Mutations supported by CAWS, including their entire response data
    # structures, the online documentation at https://caws3.nsslabs.com/docs
    # is your best friend.

    query = ""
    query Organizations{
        organizations{
            id
            name
            accounts{
                id,
                name
            }
        }
    }
    ""

    # Define Headers to Send to Server.
    # The most important header to send to the server is the "Authorization"
    # header that includes your authentication token acquired from a previous
    # call to the "authenticateUser" Mutation. See this manual for an example
    # on how to perform authentication.
    headers = {
        'Authorization': 'bearer {}'.format(token)
    }

    # Define Request Data to Send to Server.
    # This is both the query and any variables in requires. Remember, GraphQL
    # is JSON based so the variables parameter in the request has to be a JSON
    # string!
    data = {
        'query': query
    }

    # Send Request to Server.
    # Unlike traditional REST APIs, in GraphQL, there is always a single
# endpoint you send the request to. The query you send as part of the # request determines what the server actually does.
# The response body returned by the server is always formatted as JSON!
response = requests.post('https://caws3.nsslabs.com/graphql',
headers=headers, data=data)
results = response.json()

# Get Data from Results.
# A successful response from the server will have a top most object in the # JSON with a "data" key. "data" will then have an object with a key named # for the Query or Mutation you requested. The structure of that object # will be the return value of the Query or Mutation you requested. # Remember, the online documentation at https://caws3.nsslabs.com/docs is # your best friend!
data = results.get('data')
if data:
    # ...
    # We specified this above!
    results = data.get('organizations')
    return results
else:
    # Get Errors from Results.
    # A failed response from the server will have a top most array of # objects in the JSON with an "errors" key.
    errors = results.get('errors')
    for error in errors:
        errorMessage = error.get('message', 'No Message in Error.')
        print(errorMessage)

exit('Query Failed.')
CREATE SECURITY PROFILE

This mutation allows you to create a new profile. This is a new mutation that has no equivalent in the CAWS 2 API.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>int</td>
<td>Your account's id. You can only pass your account's id. Passing any other value will result in an error. Example: 1</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>The name to associate with the profile. This is used for future reference</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>A description to associate with the profile</td>
</tr>
<tr>
<td>applications</td>
<td>int[], array of integers</td>
<td>An array of application ids you want to associate with the profile. Example: [1,2,3]</td>
</tr>
<tr>
<td>device</td>
<td>int</td>
<td>A platform id you want to associate with the profile. Example: 1</td>
</tr>
<tr>
<td>platforms</td>
<td>int[], array of integers</td>
<td>An array of platform ids you want to associate with the profile. Example: [1,2,3]</td>
</tr>
<tr>
<td>location</td>
<td>string</td>
<td>An optional location to associate with the profile</td>
</tr>
<tr>
<td>notificationEnabled</td>
<td>Boolean</td>
<td>Flag to enable/disable notifications for a profile.</td>
</tr>
</tbody>
</table>

Request and Response Structures

See the online documentation(https://caws3.nsslabs.com/docs/securityprofiles-summary.doc.html) for the complete request and response JSON structure for this query. As is the case for all GraphQL queries, only the fields you select in the request JSON structure will be returned in the response JSON structure.

```python
import json
import requests

def create_profile(account, name, description, device, applications, platforms, token):
    # Define Query to Send to Server.
    # The query that is sent to the server is either a GraphQL Query or a GraphQL Mutation. In this example because we want to get data, a we are
```
# sending a GraphQL Query.
# Remember, GraphQL is graph based and we have to explicitly specify the
# fields that we want returned in the response.
# For an online reference to all the different GraphQL Queries and
# Mutations supported by CAWS, including their entire response data
# structures, the online documentation at https://caws3.nsslabs.com/docs
# is your best friend.
query = """mutations CreateSecurityProfile
{
  $account: ID!, $name: String!, $description: String!,
  $applications: [ID!]!,
  $notificationEnabled: $notiricationEnabled,
  $device: ID!, $platforms: [ID!]!
}
{
  createSecurityProfile
  (account: $account, name: $name, description: $description,
   applications: $applications,
   notificationEnabled: $notificationEnabled, device: $device,
   platforms: $platforms
  )
  {
    id
  }
}""

# Define Query Variables.
# Variables are a cool and useful feature of GraphQL that help avoid ugly
# string concatenation when constructing queries that are sent to the
# server. We recommend you use variables whenever possible.
variables = {
  'account': account,
  'name': account,
  'description': description,
  'applications': applications,
  'device': device,
  'platforms': platforms
}

# Define Headers to Send to Server.
# The most important header to send to the server is the "Authorization"
# header that includes your authentication token acquired from a previous
# call to the "authenticateUser" Mutation. See this manual for an example
# on how to perform authentication.
handlers = {
  'Authorization': 'bearer {}'.format(token)
# Define Request Data to Send to Server.
# This is both the query and any variables in requires. Remember, GraphQL
# is JSON based so the variables parameter in the request has to be a JSON
# string!
data = {
    'query': query,
    'variables': json.dumps(variables)
}

# Send Request to Server.
# Unlike traditional REST APIs, in GraphQL, there is always a single
# endpoint you send the request to. The query you send as part of the
# request determines what the server actually does.
# The response body returned by the server is always formatted as JSON!
response = requests.post('https://caws3.nsslabs.com/graphql',
headers=headers, data=data)
results = response.json()

# Get Data from Results.
# A successful response from the server will have a top most object in the
# JSON with a "data" key. "data" will then have an object with a key named
# for the Query or Mutation you requested. The structure of that object
# will be the return value of the Query or Mutation you requested.
# Remember, the online documentation at https://caws3.nsslabs.com/docs is
# your best friend!
data = results.get('data')
if data:
    # ...
    # We specified this above!
    results = data.get('createSecurityProfile')
    return results
else:
    # Get Errors from Results.
    # A failed response from the server will have a top most array of
    # objects in the JSON with an "errors" key.
    errors = results.get('errors')
    for error in errors:
        errorMessage = error.get('message', 'No Message in Error.')
        print(errorMessage)

    exit('Query Failed.')
GET SECURITY PROFILES

This query allows you to retrieve a list of all the profiles associated with your account. You will primarily need this query to get a list of your profile ids so that you can further use them to query for details pertaining to each profile. This is a new query that has no equivalent in the CAWS 2 API.

Here is an CURL example to get profile IDs.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>int</td>
<td>Your account's id. You can only pass your account's id. Passing any other value will result in an error. Example: 1</td>
</tr>
</tbody>
</table>

Request and Response Structure

See the online documentation (https://caws3.nsslabs.com/docs/securityprofileoverview.doc.html) for the complete request and response JSON structure for this query. As is the case for all GraphQL queries, only the fields you select in request JSON structure will be returned in the response JSON structure.

Get Security Profiles Curl API Call Example

1. `curl -X POST https://caws3.nsslabs.com/graphql -d query='query { securityProfiles(account: "1") { id name }}' -H 'authorization: bearer <API Token>'`

Example Output for Get Profile ID


Get Security Profiles Python API Call Example:

```python
import json
import requests

def get_profiles(account, token):
    # Define Query to Send to Server.
```
# The query that is sent to the server is either a GraphQL Query or a GraphQL Mutation. In this example because we want to get data, a we are sending a GraphQL Query.
# Remember, GraphQL is graph based and we have to explicitly specify the fields that we want returned in the response.
# For an online reference to all the different GraphQL Queries and Mutations supported by CAWS, including their entire response data structures, the online documentation at https://caws3.nsslabs.com/docs is your best friend.

```graphql
query SecurityProfiles($account: ID!){
  securityProfiles(account: $account){
    id
    name
  }
}
```

# Define Query Variables.
# Variables are a cool and useful feature of GraphQL that help avoid ugly string concatenation when constructing queries that are sent to the server. We recommend you use variables whenever possible.
```python
variables = {
    'account': account
}
```

# Define Headers to Send to Server.
# The most important header to send to the server is the "Authorization" header that includes your authentication token acquired from a previous call to the "authenticateUser" Mutation. See this manual for an example on how to perform authentication.
```python
headers = {
    'Authorization': 'bearer {}'.format(token)
}
```

# Define Request Data to Send to Server.
# This is both the query and any variables in requires. Remember, GraphQL is JSON based so the variables parameter in the request has to be a JSON string!
```python
data = {
    'query': query,
    'variables': json.dumps(variables)
}
```

# Send Request to Server.
# Unlike traditional REST APIs, in GraphQL, there is always a single
# endpoint you send the request to. The query you send as part of the
# request determines what the server actually does.

# The response body returned by the server is always formatted as JSON!
response = requests.post('https://caws3.nsslabs.com/graphql',
headers=headers, data=data)
results = response.json()

# Get Data from Results.
#
# A successful response from the server will have a top most object in the
# JSON with a "data" key. "data" will then have an object with a key named
# for the Query or Mutation you requested. The structure of that object
# will be the return value of the Query or Mutation you requested.
# Remember, the online documentation at https://caws3.nsslabs.com/docs is
# your best friend!
data = results.get('data')
if data:
    # ...
    # We specified this above!
    results = data.get('securityProfiles')
    return results
else:
    # Get Errors from Results.
    # A failed response from the server will have a top most array of
    # objects in the JSON with an "errors" key.
errors = results.get('errors')
for error in errors:
    errorMessage = error.get('message', 'No Message in Error.')
    print(errorMessage)

exit('Query Failed.')
GET GLOBAL THREAT ANALYTICS

This query allows you to get global analytics, not tied to a specific profile, pertaining to drive-by and file exploits that have been detected by CAWS. This is a new query that has no equivalent in the CAWS 2 API.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>string</td>
<td>The start date you want to retrieve data from, in ISO-8601 format. Example: &quot;2017-03-01&quot;</td>
</tr>
<tr>
<td>end</td>
<td>string</td>
<td>The end date you want to retrieve data to, in ISO-8601 format. Example: &quot;2017-04-01&quot;</td>
</tr>
<tr>
<td>applications</td>
<td>int[], array of integers</td>
<td>An optional array of application ids you want to restrict data for. Example: [1,2,3]</td>
</tr>
<tr>
<td>platforms</td>
<td>int[], array of integers</td>
<td>An optional array of platform ids you want to restrict data for. Example: [1,2,3]</td>
</tr>
<tr>
<td>countries</td>
<td>string[], array of strings</td>
<td>An optional array of country codes, in ISO-3166 format, you want to restrict data for. Example: [&quot;US&quot;]</td>
</tr>
</tbody>
</table>

Request and Response Structure

See the online documentation (https://caws3.nsslabs.com/docs/securityprofileoverview.doc.html) for the complete request and response JSON structure for this query. As is the case for all GraphQL queries, only the fields you select in the request JSON structure will be returned in the response JSON structure.

Example Curl API call to get Global Threat Analytics for given date range

```
1. curl https://caws3.nsslabs.com/graphql -d 'query=query {
2.    threatAnalytics(start: "2017-06-17", end: "2017-06-17") {
3.        applicationThreatCounts {
4.            application {
5.                id         name         family         vendor  {
6.                    id           name
7.                }
8.            } count
9.        }
10.    } countryThreatCounts {
11.        countryCode count
12.    }
13.    platformThreatCounts {
14.        platform {
15.            id         name         family         vendor  {
16.                id           name
17.            }
18.        }
19.    }
20.    metrics {
21. }
```
exploits {
    count
}
scannedFiles {
    count
}
scannedUrls {
    count
}
}
'}

`authorization: bearer <API Token>'

**JSON Response**

```json
{
  "data": {
    "threatAnalytics": {
      "applicationThreatCounts": [
        {
          "application": {
            "id": "176",
            "name": "Adobe Flash Player 18.0.0.268",
            "family": "Adobe Flash Player",
            "vendor": {
              "id": "1",
              "name": "Adobe"
            }
          },
          "count": 1
        },
        {
          "application": {
            "id": "178",
            "name": "Adobe Flash Player 18.0.0.160",
            "family": "Adobe Flash Player",
            "vendor": {
              "id": "1",
              "name": "Adobe"
            }
          },
          "count": 2
        }
      ],
      "countryThreatCounts": [
        {
          "countryCode": "US",
          "count": 19
        }
      ],
      "platformThreatCounts": [
        {
          "platform": {
            "id": "2",
            "name": "Windows 7 SP1",
            "family": "Windows",
            "vendor": {
              "id": "18",
              "name": "Microsoft"
            }
          }
        },
        {
          "platform": {
            "id": "4",
            "name": "Windows 8.1",
            "family": "Windows",
            "vendor": {
              "id": "4",
              "name": "Microsoft"
            }
          }
        }
      ]
    }
  }
}
```
"vendor": {
  "id": "18",
  "name": "Microsoft"
}
"metrics": {
  "exploits": {
    "count": 19
  },
  "scannedFiles": {
    "count": 0
  },
  "scannedUrls": {
    "count": 143800
  }
}

Example API call to get Global Threat Analytics for given date range and application

```bash
1. curl https://caws3.nsslabs.com/graphql -d 'query=query {
2.   threatAnalytics(start: "2017-06-17", end: "2017-06-17", applications: [176]) {
3.     applicationThreatCounts {
4.       application {
5.         id         name         family         vendor  {
6.           id           name
7.         } }
8.         count
9.       } }
10.   countryThreatCounts {
11.     countryCode   count
12.   }
13.   platformThreatCounts {
14.     platform {
15.       id         name         family         vendor  {
16.         id           name
17.       } }
18.   }
19.   }
20.   metrics {
21.     exploits {
22.       count
23.     }
24.   }
25.   }
26.   }
27.   }
28.   }
29.   }
30.   }
31.   }
32. }
33. }' -H 'authorization: bearer <API Token>'
```
Example API call to get Global Threat Analytics for given date range, Application, Platform and Country

```
1. curl https://caws3.nsslabs.com/graphql -d 'query=query {
2.   threatAnalytics(start: "2017-06-17", end: "2017-06-17", applications: [176], platforms: [2], countries: ["US"])
3.   applicationThreatCounts {
4.     application {
5.       id         name         family         vendor {
```
```
6.       id   name
7.   }
8. }
9.   count
10. }
11. countryThreatCounts {
12.     countryCode   count
13. }
14. platformThreatCounts {
15.     platform {
16.       id   name   family   vendor {
17.         id   name
18.     }
19.   }
20. } metrics {
21.     exploits {
22.       count
23.   }
24.   }
25.   scannedFiles {
26.       count
27.   }
28.   scannedUrls {
29.     count
30.   }
31. }
32. }
33. } 

' -H 'authorization: bearer <API Token>''

JSON Response
```
```
1. {
2.   "data": {
3.     "threatAnalytics": {
4.       "applicationThreatCounts": [{
5.         "application": {
6.           "id": "176",
7.           "name": "Adobe Flash Player 18.0.0.160",
8.           "family": "Adobe Flash Player",
9.           "vendor": {
10.              "id": "1",
11.              "name": "Adobe"
12.           }
13.         },
14.         "count": 2
15.       }],
16.       "countryThreatCounts": [{
17.         "countryCode": "US",
18.         "count": 2
19.       }],
20.       "platformThreatCounts": [{
21.         "platform": {
22.           "id": "2",
23.           "name": "Windows 7 SP1",
24.           "family": "Windows",
25.           "vendor": {
26.              "id": "18",
27.              "name": "Microsoft"
28.           }
29.         }
30.   }
31. }
32. }
33. }
```
Example API call to get Malware scoring

```
curl -X POST https://caws-dev.nsslabs.com/graphql -d 'query=query
  securityProfileAnalysis
    (start: "2013-08-02", end: "2017-08-31", account: 4, profile: 18
      ) {
        id
        name
        pagedThreats
          (excludeBlocked: true, excludeBypassed: false, sortBy: testDate, sortDescending: true
            ) {
            pagingInfo {
              total
              pageCount
              currentPage
            }
            nodes {
              nssid
testDate
discoveryDate
device {
              id
              name
              type
              vendor {
                id
                name
              }
            }
            exploitBlocked
captureDroppedMalware
            malwareBlocked
```
41. processCount
42. maliciousProcessCount
43. fileCount
44. connectionCount
45. }
46. }
47. }
48. }
49. ' -H ' authorization: bearer < API Token > ' 

**JSON Response**

```json
1. {
2.   "data": {
3.     "securityProfileAnalysis": {
4.       "id": "18",
5.       "name": "Windows Defender-allapp-allplat",
6.       "pagedThreats": {
7.         "pagingInfo": {
8.           "pageCount": 1,
9.           "currentPage": 1
10.       },
11.         "nodes": [
12.           {
13.             "nssid": "NSS-2017-3TXTG5",
14.             "testDate": "2017-08-30T01:27:45.000Z",
15.             "discoveryDate": "2017-08-30T00:11:27.000Z",
16.             "device": {
17.               "id": "165",
18.               "name": "Windows Defender",
19.               "type": "Endpoint Protection",
20.               "vendor": {
21.                 "id": "18",
22.                 "name": "Microsoft"
23.               },
24.               "exploitBlocked": false,
25.               "captureDroppedMalware": false,
26.               "malwareBlocked": null,
27.               "processCount": 5,
28.               "maliciousProcessCount": 0,
29.               "fileCount": 0,
30.               "connectionCount": 0
31.             },
32.           },
33.           {
34.             "nssid": "NSS-2017-3TPKM6",
35.             "testDate": "2017-08-30T01:27:53.000Z",
36.             "discoveryDate": "2017-08-29T02:05:45.000Z",
37.             "device": {
38.               "id": "165",
39.               "name": "Windows Defender",
40.               "type": "Endpoint Protection",
41.               "vendor": {
42.                 "id": "18",
43.                 "name": "Microsoft"
44.               },
45.             }
46.         }
47.     }
48.   }
49. }
```
Example Python API call to get Global Threat Analytics for given date range:

```python
import json
import requests

# ... 

# Function to Get Global Analytics (Capture Data).
def get_global_analytics(start, end, token):
    # Define Query to Send to Server.
    # The query that is sent to the server is either a GraphQL Query or a
    # GraphQL Mutation. In this example because we want to get data, a we are
    # sending a GraphQL Query.
    # Remember, GraphQL is graph based and we have to explicitly specify the
    # fields that we want returned in the response.
    # For an online reference to all the different GraphQL Queries and
    # Mutations supported by CAWS, including their entire response data
    # structures, the online documentation at https://caws3.nsslabs.com/docs
    # is your best friend.
    query = ""
    query ThreatAnalytics($start: String!, $end: String!){
    threatAnalytics(start: $start, end: $end){
      applicationThreatCounts{
        application{
          id
          name
          family
          vendor{
            id
            name
          }
        }
      }
      count
    }
    countryThreatCounts{
      countryCode
      count
    }
    platformThreatCounts{
      platform{
        id
        name
        family
        vendor{
          id
          name
        }
      }
    }
```
# Define Query Variables.
#
# Variables are a cool and useful feature of GraphQL that help avoid ugly
# string concatenation when constructing queries that are sent to the
# server. We recommend you use variables whenever possible.
variables = {
    'start': start,
    'end': end
}

# Define Headers to Send to Server.
#
# The most important header to send to the server is the "Authorization"
# header that includes your authentication token acquired from a previous
# call to the "authenticateUser" Mutation. See this manual for an example
# on how to perform authentication.
headers = {
    'Authorization': 'bearer {}'.format(token)
}

# Define Request Data to Send to Server.
#
# This is both the query and any variables in requires. Remember, GraphQL
# is JSON based so the variables parameter in the request has to be a JSON
# string!
data = {
    'query': query,
    'variables': json.dumps(variables)
}

# Send Request to Server.
#
# Unlike traditional REST APIs, in GraphQL, there is always a single
# endpoint you send the request to. The query you send as part of the
# request determines what the server actually does.
# The response body returned by the server is always formatted as JSON!
response = requests.post('https://caws3.nsslabs.com/graphql',
headers=headers, data=data)
results = response.json()

# Get Data from Results.
#
# A successful response from the server will have a top most object in the
# JSON with a "data" key. "data" will then have an object with a key named
# for the Query or Mutation you requested. The structure of that object
# will be the return value of the Query or Mutation you requested.
# Remember, the online documentation at https://caws3.nsslabs.com/docs is
# your best friend!
data = results.get('data')
if data:
    # ...
    # We specified this above!
    results = data.get('threatAnalytics')
    return results
else:
    # Get Errors from Results.
    #
    # A failed response from the server will have a top most array of
    # objects in the JSON with an "errors" key.
    errors = results.get('errors')
    for error in errors:
        errorMessage = error.get('message', 'No Message in Error.')
        print(errorMessage)
    exit('Query Failed.')
This query allows you to search for global threats, not tied to a specific profile, pertaining to drive-by-exploits that have been detected by CAWS. There is currently no support to search for file exploits. Results returned by this query are batched in pages. Pagination information is returned in the result and client code is expected to examine that information to iterate through all the pages in the result. This query is equivalent to the /Captures/List endpoint in the CAWS 2 API.

## Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>string</td>
<td>The start date you want to retrieve data from, in ISO-8601 format. Example: &quot;2017-03-01&quot;</td>
</tr>
<tr>
<td>end</td>
<td>string</td>
<td>The end date you want to retrieve data to, in ISO-8601 format. Example: &quot;2017-04-01&quot;</td>
</tr>
<tr>
<td>applications</td>
<td>int[], array of integers</td>
<td>An optional array of application ids you want to restrict data for. Example: [1,2,3]</td>
</tr>
<tr>
<td>platforms</td>
<td>int[], array of integers</td>
<td>An optional array of platform ids you want to restrict data for. Example: [1,2,3]</td>
</tr>
<tr>
<td>countries</td>
<td>string[], array of strings</td>
<td>An optional array of country codes, in ISO-3166 format, you want to restrict data for. Example: [&quot;US&quot;]</td>
</tr>
<tr>
<td>ipAddresses</td>
<td>string[], array of strings</td>
<td>an optional array of IP addresses you want to restrict data for. Example: [&quot;66.127.0.1&quot;]</td>
</tr>
<tr>
<td>processes</td>
<td>string[], array of strings</td>
<td>An optional array of MD5 hashes of files dropped by exploits and executed as processes you want to restrict data for. Example: [&quot;ABC&quot;]</td>
</tr>
</tbody>
</table>

## Request and Response Structure

See the online documentation (https://caws3.nsslabs.com/docs/securityprofileoverview.doc.html) for the complete request and response JSON structure for this query. As is the case for all GraphQL queries, only the fields you select in request JSON structure will be returned in the response JSON structure.

**Example API call to get Search for Global Threat by date range**

```
1. curl https://caws3.nsslabs.com/graphql -d 'query=query {
2.   threatInfo(start: "2017-06-17", end: "2017-06-17") {
3.     exploits
4.       { nssid
discoveryDate
processCount
maliciousProcessCount
connectionCount
5.         },
6.         connections
7.         {
8.           ipAddress,
```

- 23 -
3. port,
6. },
1. processes
{ processName,
    md5Hash,
    sha1Hash,
    isMalicious
7. }
8. } ' -H 'authorization: bearer <API Token>'

JSON Response
1. {
2. "data": {
3. "threatInfo": {
4. "exploits": [{
5. "nssid": "NSS-2017-38ZSFT",
6. "discoveryDate": "2017-06-17T00:05:10.000Z",
7. "processCount": 8,
8. "maliciousProcessCount": 0,
9. "connectionCount": 0,
10. }, {
12. "discoveryDate": "2017-06-17T12:33:08.000Z",
13. "processCount": 8,
14. "maliciousProcessCount": 1,
15. "connectionCount": 1,
16. }, {
17. "ipAddress": "104.28.12.38",
18. "port": 80,
19. }, {
20. "ipAddress": "69.50.130.106",
21. "port": 80,
22. }],
23. "processes": null,
24. },
25. "errors": [{
26. "message": "Cannot read property 'length' of undefined",
27. "status": "400",
28. "code": "",
29. "path": ["threatInfo", "processes"],
30. "meta": {}
31. }]
32. }
33. }

Example API Call to search for Global Threats by Date Range and IP Address.
1. curl https://caws3.nsslabs.com/graphql -d 'query=query {
2. threatInfo(start: "2017-06-17", end: "2017-06-17", ipAddresses: ["104.28.12.38"], platforms: [2, 3]) {
3. exploits {
1. nssid
2. discoveryDate
3. processCount
4. maliciousProcessCount
5. connectionCount
4. },
5. connections {
6. }'
Example API Call to search for Global Threats by Date Range, IP Address and Platforms

```
curl https://caws3.nsslabs.com/graphql -d 'query=query {
  threatInfo(start:  "2017-06-1", end:  "2017-06-17", ipAddresses: ["104.28.12.38"], platforms: [2, 3]) {
    exploits {
      nssid       discoveryDate       processCount       maliciousProcessCount       connectionCount
      "NSS-2017-38XNJ3", "2017-06-16T18:20:49.000Z", 8, 0, 0
    },
    connections {
      ipAddress,       port,
      "104.28.12.38", 80
    },
    processes {
      processName,       md5Hash,       sha1Hash,       isMalicious
      "processName",       "md5Hash",       "sha1Hash",       false
    }
  }
}

' -H 'authorization: bearer <API Token>'
```

JSON Response

```
{
  "data": {
    "threatInfo": {
      "exploits": [{
        "nssid": "NSS-2017-38XNJ3",
        "discoveryDate": "2017-06-16T18:20:49.000Z",
        "processCount": 8,
        "maliciousProcessCount": 0,
        "connectionCount": 0
      }],
      "connections": [{
        "ipAddress": "104.28.12.38",
        "port": 80
      }],
      "processes": []
    }
  }
}
```

Example API Call to search for Global Threats by Date Range, IP Address and Platforms

```
curl https://caws3.nsslabs.com/graphql -d 'query=query {
  threatInfo(start:  "2017-06-1", end:  "2017-06-17", ipAddresses: ["104.28.12.38"], platforms: [2, 3]) {
    exploits {
      nssid       discoveryDate       processCount       maliciousProcessCount       connectionCount
      "NSS-2017-38ZSFT", "2017-06-17T00:05:10.000Z", 8, 0, 0
    },
    connections {
      ipAddress,       port,
      "104.28.12.38", 80
    },
    processes {
      processName,       md5Hash,       sha1Hash,       isMalicious
      "processName",       "md5Hash",       "sha1Hash",       false
    }
  }
}

' -H 'authorization: bearer <API Token>'
```

JSON Response

```
{
  "data": {
    "threatInfo": {
      "exploits": [{
        "nssid": "NSS-2017-38ZSFT",
        "discoveryDate": "2017-06-17T00:05:10.000Z",
        "processCount": 8,
        "maliciousProcessCount": 0,
        "connectionCount": 0
      }],
      "connections": [{
        "ipAddress": "104.28.12.38",
        "port": 80
      }],
      "processes": []
    }
  }
}
```
Here is an example using Python. The example defines a function that you can reuse in a Python program. The code comments are pretty much self explanatory:

```python
import json
import requests

# ...

# Function to Search for Global Threats (Capture Data).
def find_global_threats(start, end, page, token):
    # Define Query to Send to Server.
    # The query that is sent to the server is either a GraphQL Query or a GraphQL Mutation. In this example because we want to get data, we are sending a GraphQL Query.
    # Remember, GraphQL is graph based and we have to explicitly specify the fields that we want returned in the response.
    # For an online reference to all the different GraphQL Queries and Mutations supported by CAWS, including their entire response data structures, the online documentation at https://caws3.nsslabs.com/docs is your best friend.
    query = ""
    query ThreatInfo($start: String!, $end: String!, $page: Int!){
        threatInfo(start: $start, end: $end){
            pagedExploits(page: $page){
                nodes{
                    nssid
discoveryDate
processCount
maliciousProcessCount
connectionCount
                }
            pagingInfo{
                currentPage
total
hasMore
            }
        }
        pagedConnections(page: $page){
```

```
node{
ipAddress
port
}
pagingInfo{
currentPage
total
hasMore
}
}
pagedProcesses(page: $page){
nodes{
processName
md5Hash
sha1Hash
isMalicious
}
pagingInfo{
currentPage
total
hasMore
}
}

""

# Define Query Variables.
#
# Variables are a cool and useful feature of GraphQL that help avoid ugly
# string concatenation when constructing queries that are sent to the
# server. We recommend you use variables whenever possible.
variables = {
    'start': start,
    'end': end,
    'page': page
}

# Define Headers to Send to Server.
#
# The most important header to send to the server is the "Authorization"
# header that includes your authentication token acquired from a previous
# call to the "authenticateUser" Mutation. See this manual for an example
# on how to perform authentication.
headers = {
    'Authorization': 'bearer {}'.format(token)
}

# Define Request Data to Send to Server.
#
# This is both the query and any variables in requires. Remember, GraphQL
# is JSON based so the variables parameter in the request has to be a JSON
# string!
data = {

'query': query,
'variables': json.dumps(variables)
}

# Send Request to Server.
#
# Unlike traditional REST APIs, in GraphQL, there is always a single
# endpoint you send the request to. The query you send as part of the
# request determines what the server actually does.
# The response body returned by the server is always formatted as JSON!
response = requests.post('https://caws3.nsslabs.com/graphql',
headers=headers, data=data)
results = response.json()

# Get Data from Results.
#
# A successful response from the server will have a top most object in the
# JSON with a "data" key. "data" will then have an object with a key named
# for the Query or Mutation you requested. The structure of that object
# will be the return value of the Query or Mutation you requested.
# Remember, the online documentation at https://caws3.nsslabs.com/docs is
# your best friend!
data = results.get('data')
if data:
    # ...
    # We specified this above!
    results = data.get('threatInfo')
    return results
else:
    # Get Errors from Results.
    #
    # A failed response from the server will have a top most array of
    # objects in the JSON with an "errors" key.
    errors = results.get('errors')
    for error in errors:
        errorMessage = error.get('message', 'No Message in Error.')
        print(errorMessage)

    exit('Query Failed.')
GET AN NSS ID’S DETAILS

This query allows you to retrieve information about a specific threat, identified by an NSS ID, that has been detected by CAWS. This query is equivalent to the /Captures/Get/{NSSId} endpoint in the CAWS 2 API.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>string</td>
<td>The start date you want to calculate metrics for the NSS ID relative to, in ISO-8601 format. This does not have to be the date the NSS ID was discovered. Example: &quot;2017-03-01&quot;</td>
</tr>
<tr>
<td>end</td>
<td>string</td>
<td>The end date you want to calculate metrics for the NSS ID relative to, in ISO-8601 format. This does not have to be the date the NSS ID was discovered. Example: &quot;2017-04-01&quot;</td>
</tr>
<tr>
<td>nssid</td>
<td>string</td>
<td>The NSS ID you want to retrieve data for. Example: &quot;NSS-2017-38ZSFT&quot;</td>
</tr>
<tr>
<td>reversingLabs</td>
<td>Json</td>
<td>Malware classification data.</td>
</tr>
</tbody>
</table>

Request and Response Structure

See the online documentation (https://caws3.nsslabs.com/docs/securityprofileoverview.doc.html) for the complete request and response JSON structure for this query. As is the case for all GraphQL queries, only the fields you select in request JSON structure will be returned in the response JSON structure.

Example API Call to GET NSS ID Details

```
1. curl https://caws3.nsslabs.com/graphql -d "query=query {
3.      nssid     details {
4.        exploitType        discoveryDate        ipAddress        url        application {
5.            application {
6.              id            name            family            vendor {
7.                  id  name
8.                  }
9.              }
10.         }
11.         platform {
12.            platform {
13.              id            name            family            vendor {
14.                  id  name
15.                  }
16.              }
17.              }
```

- 29 -
Response Structure

```json
{
  "data": {
    "threatIntel": {
      "nssid": "NSS-2017-3ZSFT",
      "details": {
        "exploitType": "url",
        "discoveryDate": "2017-06-17T00:05:10Z",
        "ipAddress": "104.28.12.38",
        "url": "http://ihackedit.com",
        "application": {
          "id": "176",
          "name": "Adobe Flash Player 18.0.0.160",
          "family": "Adobe Flash Player",
          "vendor": {
            "id": "1",
            "name": "Adobe"
          }
        },
        "platform": {
          "id": "2",
          "name": "Windows 7 SP1",
          "family": "Windows",
          "vendor": {
            "id": "18",
            "name": "Microsoft"
          }
        },
        "browser": {
          "id": "63",
          "name": "Internet Explorer 9",
          "family": "InternetExplorer",
          "vendor": {
            "id": "18",
            "name": "Microsoft"
          }
        }
      }
    }
  }
}
```

Here is an example using Python. The example defines a function that you can reuse in a Python program. The code comments are pretty much self explanatory:

```python
import json
import requests

# ...
#
# Function to Get an NSS ID's Details (Capture Data)
def get_nssid(nssid, token):
    # Define Query to Send to Server.
    # The query that is sent to the server is either a GraphQL Query or a
    # GraphQL Mutation. In this example because we want to get data, we are
    # sending a GraphQL Query.
    # Remember, GraphQL is graph based and we have to explicitly specify the
    # fields that we want returned in the response.
    # For an online reference to all the different GraphQL Queries and
    # Mutations supported by CAWS, including their entire response data
    # structures, the online documentation at https://caws3.nsslabs.com/docs
    # is your best friend.
    query = ""
    query ThreatIntel($nssid: String!){
        threatIntel(nssid: $nssid){
            nssid
            details{
                exploitType
discoveryDate
ipAddress
url
application{
    application{
        id
name
family
vendor{
            id
name
        }
    }
}
```


```graphql

# Define Query Variables.
# Variables are a cool and useful feature of GraphQL that help avoid ugly
# string concatenation when constructing queries that are sent to the
# server. We recommend you use variables whenever possible.
variables = {
    'nssid': nssid
}

# Define Headers to Send to Server.
# The most important header to send to the server is the "Authorization"
# header that includes your authentication token acquired from a previous
```

- 32 -
# call to the "authenticateUser" Mutation. See this manual for an example
# on how to perform authentication.
headers = {
    'Authorization': 'bearer {}'.format(token)
}

# Define Request Data to Send to Server.
#
# This is both the query and any variables in requires. Remember, GraphQL
# is JSON based so the variables parameter in the request has to be a JSON
# string!
data = {
    'query': query,
    'variables': json.dumps(variables)
}

# Send Request to Server.
#
# Unlike traditional REST APIs, in GraphQL, there is always a single
# endpoint you send the request to. The query you send as part of the
# request determines what the server actually does.
#
# The response body returned by the server is always formatted as JSON!
response = requests.post('https://caws3.nsslabs.com/graphql',
headers=headers, data=data)
results = response.json()

# Get Data from Results.
#
# A successful response from the server will have a top most object in the
# JSON with a "data" key. "data" will then have an object with a key named
# for the Query or Mutation you requested. The structure of that object
# will be the return value of the Query or Mutation you requested.
# Remember, the online documentation at https://caws3.nsslabs.com/docs is
# your best friend!
data = results.get('data')
if data:
    # ...
    # We specified this above!
    results = data.get('threatIntel')
    return results
else:
    # Get Errors from Results.
    # A failed response from the server will have a top most array of
    # objects in the JSON with an "errors" key.
    errors = results.get('errors')
    for error in errors:
        errorMessage = error.get('message', 'No Message in Error.')
        print(errorMessage)
    exit('Query Failed.')
DOWNLOADING HTTP TRAFFIC, PCAP FILES, SHELL CODE AND MALWARE PAYLOAD FILES

For any given NSS ID, you can download its associated HTTP Traffic, PCAP, and Shellcode files. Please note that not every NSS ID will have these files available.

Example API Calls to Download HTTP, PCAP and SAZ Files

<table>
<thead>
<tr>
<th>Action</th>
<th>Method</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Download HTTP Traffic File</td>
<td>GET</td>
<td>/graphql/downloads?type=SAZ&amp;nssid={NSSId}</td>
</tr>
<tr>
<td>Download PCAP</td>
<td>GET</td>
<td>/graphql/downloads?type=PCAP&amp;nssid={NSSId}</td>
</tr>
<tr>
<td>Download Shellcode</td>
<td>GET</td>
<td>/graphql/downloads?type=SHELL&amp;nssid={NSSId}</td>
</tr>
<tr>
<td>Download Malware</td>
<td>GET</td>
<td>/graphql/downloads?type=PAYLOAD&amp;hash={MD5Hash}</td>
</tr>
</tbody>
</table>

Request and Response Structure

See the online documentation(https://caws3.nsslabs.com/docs/threatintel.doc.html) for the complete request and response JSON structure for this query. As is the case for all GraphQL queries, only the fields you select in request JSON structure will be returned in the response JSON structure.

Example Get Query to Download Shell Code


Example Query to Download SAZ File


Example Query to Download PCAP File


Example Query to Download Exploit Payload File


Here are examples using Python to download HTTP Traffic, PCAP and Shellcode, and malware payload files. The example defines functions that you can reuse in a Python program. The code comments are pretty much self explanatory:

```python
import json
import requests
```
# Function Download HTTP Traffic File.
def download_traffic_file(nss_id, token):
    # Define Download URL.
    # Unlike other requests to the server, to download a file using the CAWS API, you just need to send an HTTP request to the following endpoint and indicate what kind of file you want.

    # Define Headers to Send to Server.
    # The most important header to send to the server is the "Authorization" header that includes your authentication token acquired from a previous call to the "authenticateUser" Mutation. See this manual for an example on how to perform authentication.
    headers = {
        'Authorization': 'bearer {}'.format(token)
    }

    # Send Request to Server.
    # The response body returned by the server is a binary stream! You can choose to handle it however you want, like saving it to a local flat file for example.
    response = requests.get(api_url, headers=headers)
    return response.content

# Function Download PCAP File.
def download_pcap_file(nss_id, token):
    # Define Download URL.
    # Unlike other requests to the server, to download a file using the CAWS API, you just need to send an HTTP request to the following endpoint and indicate what kind of file you want.

    # Define Headers to Send to Server.
    # The most important header to send to the server is the "Authorization" header that includes your authentication token acquired from a previous call to the "authenticateUser" Mutation. See this manual for an example on how to perform authentication.
    headers = {
        'Authorization': 'bearer {}'.format(token)
    }

    # Send Request to Server.
    # The response body returned by the server is a binary stream! You can
# Function Download Shellcode File.
def download_shellcode_file(nss_id, token):
    # Define Download URL.
    # Unlike other requests to the server, to download a file using the CAWS
    # API, you just need to send an HTTP request to the following endpoint and
    # indicate what kind of file you want.
    # Define Headers to Send to Server.
    # The most important header to send to the server is the "Authorization"
    # header that includes your authentication token acquired from a previous
    # call to the "authenticateUser" Mutation. See this manual for an example
    # on how to perform authentication.
    headers = {
        'Authorization': 'bearer {}'.format(token)
    }
    # Send Request to Server.
    # The response body returned by the server is a binary stream! You can
    # choose to handle it however you want, like saving it to a local flat
    # file for example.
    response = requests.get(api_url, headers=headers)
    return response.content

# ...
headers = {
    'Authorization': 'bearer {}'.format(token)
}

# Send Request to Server.
# The response body returned by the server is a binary stream! You can
# choose to handle it however you want, like saving it to a local flat
# file for example.
response = requests.get(api_url, headers=headers)
return response.content
**Get Profile Details**

This query allows you to retrieve information about a specific profile. This includes all threats found to be targeting the applications in that profile, and all threats that are tested against the security products in that profile. This query is equivalent to the /Replays/List endpoint in the CAWS 2 API. Starting with CAWS 3, you can only retrieve information pertaining to security products relative to a specific profile associated with your account. This is because most metrics pertaining to security products are now computed relative to a specific profile. This allows you to retrieve information about a security product that is more relevant to your organization.

### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>int</td>
<td>Your account's id. You can only pass your account's id. Passing any other value will result in an error. Example: 1</td>
</tr>
<tr>
<td>profile</td>
<td>int</td>
<td>The id of the profile you want to retrieve data for. You can only pass an id for a profile associated with your account. Passing any other value will result in an error. You can get a list of profile ids associated with your account by invoking the <code>securityProfiles</code> query. Example: 62</td>
</tr>
<tr>
<td>start</td>
<td>string</td>
<td>The start date you want to calculate metrics for the profile ID relative to, in ISO-8601 format. Example: &quot;2017-03-01&quot;</td>
</tr>
<tr>
<td>end</td>
<td>string</td>
<td>The end date you want to calculate metrics for the profile relative to, in ISO-8601 format. Example: &quot;2017-04-01&quot;</td>
</tr>
</tbody>
</table>

### Request and Response Structure

See the online documentation ([https://caws3.nsslabs.com/docs/securityprofilethreat.doc.html](https://caws3.nsslabs.com/docs/securityprofilethreat.doc.html)) for the complete request and response JSON structure for this query. As is the case for all GraphQL queries, only the fields you select in request JSON structure will be returned in the response JSON structure.

**Example API call to Get Profile Details**

```bash
curl https://caws3.nsslabs.com/graphql --data-urlencode 'query=query {
    id
    securityProducts {
      securityProductResults {
        device {
          id
          name
        }
      }
    }
    blockedCount
    bypassingCount
    totalCount
    blockRate
  }
}
```
13. applications {
14.     targetedApplications {
15.         application {
16.             id           name           family           vendor {
17.                 id             name
18.             }
19.         }
20.         bypassingCount         totalCount
21.     }
22. }
23.
24. }
25. "-H 'authorization: bearer <API Token>'

JSON Response

1. {
2.     "data": {
3.         "securityProfileAnalysis": {
4.             "id": "62",
5.             "securityProducts": {
6.                 "securityProductResults": [
7.                     {
8.                         "device": {
9.                             "id": "165",
10.                            "name": "Windows Defender",
11.                           "type": "Endpoint Protection",
12.                         "vendor": {
13.                             "id": "18",
14.                             "name": "Microsoft"
15.                         }
16.                     }]
17.                     "blockedCount": 21,
18.                     "bypassingCount": 30,
19.                     "totalCount": 51,
20.                     "blockRate": 41.176
21.                 }
22.             },
23.             "applications": {
24.                 "targetedApplications": [
25.                     {
26.                         "application": {
27.                             "id": "159",
28.                             "name": "Internet Explorer 10",
29.                             "family": "Internet Explorer",
30.                             "vendor": {
31.                                 "id": "18",
32.                                 "name": "Microsoft"
33.                             }
34.                         },
35.                     "bypassingCount": 18,
36.                     "totalCount": 25
37.                 },
38.                 {
39.                     "application": {
40.                         "id": "53",
41.                         "name": "Internet Explorer 9",
42.                         "family": "Internet Explorer",
43.                         "vendor": {
44.                             "id": "18",
45.                             "name": "Microsoft"
46.                         }
47.                     },
48.                     "bypassingCount": 1,
49.                     "totalCount": 9
50.                 }
51.             }
52.         }
53.     }
54. }
55.}

Here is an example using Python. The example defines a function that you can reuse in a Python program. The code comments are pretty much self explanatory:

```python
import json
import requests

# Function to Get a Profile's Data (Replay Data).
def get_profile_data(start, end, account, profile, token):
    # Define Query to Send to Server.
    # The query that is sent to the server is either a GraphQL Query or a
    # GraphQL Mutation. In this example because we want to get data, a we are
    # sending a GraphQL Query.
    # Remember, GraphQL is graph based and we have to explicitly specify the
    # fields that we want returned in the response.
    # For an online reference to all the different GraphQL Queries and
    # Mutations supported by CAWS, including their entire response data
    # structures, the online documentation at https://caws3.nsslabs.com/docs
    # is your best friend.
    query = ""
        query SecurityProfileAnalysis($start: String!, $end: String!,
        $account: ID!, $profile: ID!){
            securityProfileAnalysis(start: $start, end: $end, account:
            $account, profile: $profile){
                id
                securityProducts{
                    securityProductResults{
                        device{
                            id
                            name
                            type
                            vendor{
                                id
                                name
                            }
                        }
                        blockedCount
                        bypassingCount
                        totalCount
                        blockRate
                    }
                }
            }
        }
    ")
```
targetedApplications{
  application{
    id
    name
    family
    vendor{
      id
      name
    }
  }
  bypassingCount
  totalCount
}
}

""

# Define Query Variables.
#
# Variables are a cool and useful feature of GraphQL that help avoid ugly
# string concatenation when constructing queries that are sent to the
# server. We recommend you use variables whenever possible.
variables = {
  'start': start,
  'end': end,
  'account': account,
  'profile': profile
}

# Define Headers to Send to Server.
#
# The most important header to send to the server is the "Authorization"
# header that includes your authentication token acquired from a previous
# call to the "authenticateUser" Mutation. See this manual for an example
# on how to perform authentication.
headers = {
  'Authorization': 'bearer {}'.format(token)
}

# Define Request Data to Send to Server.
#
# This is both the query and any variables in requires. Remember, GraphQL
# is JSON based so the variables parameter in the request has to be a JSON
# string!
data = {
  'query': query,
  'variables': json.dumps(variables)
}

# Send Request to Server.
#
# Unlike traditional REST APIs, in GraphQL, there is always a single
# endpoint you send the request to. The query you send as part of the
# request determines what the server actually does.
# The response body returned by the server is always formatted as JSON!
response = requests.post('https://caws3.nsslabs.com/graphql',
headers=headers, data=data)
results = response.json()

# Get Data from Results.
#
# A successful response from the server will have a top most object in the
# JSON with a "data" key. "data" will then have an object with a key named
# for the Query or Mutation you requested. The structure of that object
# will be the return value of the Query or Mutation you requested.
# Remember, the online documentation at https://caws3.nsslabs.com/docs is
# your best friend!
data = results.get('data')
if data:
    # ...
    # We specified this above!
    results = data.get('securityProfileAnalysis')
    return results
else:
    # Get Errors from Results.
    #
    # A failed response from the server will have a top most array of
    # objects in the JSON with an "errors" key.
    errors = results.get('errors')
    for error in errors:
        errorMessage = error.get('message', 'No Message in Error.')
        print(errorMessage)

    exit('Query Failed.')
GET LIST OF DEVICES

This query allows you to get list of security devices and IDs associated with the device names. The device IDs need to be used in the API calls to create security profiles etc. The response has been truncated for the convenience of documentation. The actual response for this query contains the list of security devices for which the account has access to.

GET Query to fetch list of devices:
1. curl https://caws3.nsslabs.com/graphql --data-urlencode 'query=query {
2.   devices(account: 1) {
3.     id, name
4.   }
5. }
6. ' -H 'authorization: bearer <API Token>'

Query Responet that contains list of devices:
1. {
2.   "data": {
3.     "devices": [{
4.       "id": "11",
5.       "name": "Emsisoft Internet Security 9.0"
6.     }, {
7.       "id": "12",
8.       "name": "ESET Smart Security 8.0"
9.     }]
10. }
11. }
**CREATE A PROFILE**

This query allows you to create a new security profile. You have to create a profile to understand the threats bypassing or blocked by a security product. Below is a curl example of the API Call.

**Post Query to Create a Profile**

```
2.   -H 'cache-control: no-cache' -H 'content-type: application/x-www-form-urlencoded' -
3.   d 'query=mutation {
4.     createSecurityProfile(account: 8, name: "Profile-1", description: "Profile 1 Description", applications: ["1", "2", "3"], device: "171", platforms: ["1", "2", "3"], location: "Profile 1 Location") {
5.       account name description applications {
6.         id name family __typename
7.       }
8.       device {
9.         id name type __typename
10.    }
11.   platforms {
12.     id name family __typename
13.   }
14. location __typename
15. }
```

**Query Response for Create a Profile**

```
1. {
2.   "data": {
3.     "createSecurityProfile": {
4.       "account": "8",
5.       "name": "Profile-1",
6.       "description": "Profile 1 Description",
7.       "applications": [{
8.         "id": "1",
9.         "name": "Adobe AIR 1.0.4990",
10.        "family": "Adobe AIR",
11.       } __typename": "Application"
12.     }, {
13.         "id": "2",
14.         "name": "Adobe AIR 13.0.0.83",
15.         "family": "Adobe AIR",
16.       } __typename": "Application"
17.     }, {
18.         "id": "3",
19.         "name": "Adobe AIR 2.0.2.12610",
20.        "family": "Adobe AIR",
21.       } __typename": "Application"
22.     }
23.     "device": {
24.         "id": "171",
25.         "name": "Cylance PROTECT",
26.        "type": "Advanced Endpoint Protection",
27.       } __typename": "Device"
28.     },
29.     "platforms": [{
30.         "id": "1",
31.         "name": "Windows 7",
```
<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>family</th>
<th>__typename</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Windows 2000</td>
<td>Windows</td>
<td>Platform</td>
</tr>
<tr>
<td>2</td>
<td>Windows 7 SP1</td>
<td>Windows</td>
<td>Platform</td>
</tr>
<tr>
<td>3</td>
<td>Windows 8</td>
<td>Windows</td>
<td>Platform</td>
</tr>
</tbody>
</table>

location: Profile 1 Location, __typename: SecurityProfileSummary
GET LIST OF SUPPORTED APPLICATIONS, BROWSERS AND PLATFORM PACKAGES

This query allows you to retrieve a list of supported application, browser, and platform packages supported for user submissions. You will primarily need this query to get a list of ids to pass as arguments to the appropriate parameters when you are submitting either a file or URL for scanning by CAWS. This is a new query that has no equivalent in the CAWS 2 API.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>int</td>
<td>Your account's id. You can only pass your account's id. Passing any other value will result in an error. Example: 1</td>
</tr>
<tr>
<td>infold</td>
<td>int</td>
<td>Pass the same value as the account parameter</td>
</tr>
</tbody>
</table>

Request and Response Structure

See the online documentation (Applications, Browsers, Platforms) for the complete request and response JSON structure for this query. As is the case for all GraphQL queries, only the fields you select in request JSON structure will be returned in the response JSON structure.

Examples

1. curl https://caws3.nslslabs.com/graphql --data-urlencode 'query=query {
2.   applications (accountId: 1, infoId: 1) {
3.     applications {
4.       id, name, supportedExtensions
5.     }, browsers {
6.       id, name
7.     }, platforms {
8.       id, name
9.     }
10. }
11. }
12. "-H 'authorization: bearer <API Token>'"

Response

1. {
2.   "data": {
3.     "submissions": {
4.       "applications": [
5.         { "id": 1,
6.         "name": "Adobe Reader DC 2015.007.20033",
7.         "supportedExtensions": ".pdf"
8.       },
9.         { "id": 2,
10.         "name": "Other application", ..."}]]
11.   }}}}
Here is an example using Python. The example defines a function that you can reuse in a Python program if you wish. The code comments are pretty much self explanatory:

```python
import json
import requests

# ...

# Function to Get a List of Supported Application, Browser, and Platform Packages for User Submissions.
def get_submissions(account):
    # Define Query to Send to Server.
    # The query that is sent to the server is either a GraphQL Query or a GraphQL Mutation. In this example because we want to get data, we are sending a GraphQL Query.
    # Remember, GraphQL is graph based and we have to explicitly specify the
```
# fields that we want returned in the response.
#
# For an online reference to all the different GraphQL Queries and
# Mutations supported by CAWS, including their entire response data
# structures, the online documentation at https://caws3.nsslabs.com/docs
# is your best friend.
query = ""
    query Submissions($accountId: ID!, $infoId: ID!){
        submissions(accountId: $accountId, infoId: $infoId){
            applications{
                id
                name
                supportedExtensions
            }
            browsers{
                id
                name
            }
            platforms{
                id
                name
            }
        }
    }
""

# Define Query Variables.
#
# Variables are a cool and useful feature of GraphQL that help avoid ugly
# string concatenation when constructing queries that are sent to the
# server. We recommend you use variables whenever possible.
variables = {
    'accountId': account,
    'infoId': account
}

# Define Headers to Send to Server.
#
# The most important header to send to the server is the "Authorization"
# header that includes your authentication token acquired from a previous
# call to the "authenticateUser" Mutation. See this manual for an example
# on how to perform authentication.
headers = {
    'Authorization': 'bearer {}'.format(token)
}

# Define Request Data to Send to Server.
#
# This is both the query and any variables in requires. Remember, GraphQL
# is JSON based so the variables parameter in the request has to be a JSON
# string!
data = {
    'query': query,
    'variables': json.dumps(variables)
}
# Send Request to Server.
# Unlike traditional REST APIs, in GraphQL, there is always a single
# endpoint you send the request to. The query you send as part of the
# request determines what the server actually does.
# The response body returned by the server is always formatted as JSON!
response = requests.post('https://caws3.nsslabs.com/graphql',
headers=headers, data=data)
results = response.json()

# Get Data from Results.
# A successful response from the server will have a top most object in the
# JSON with a "data" key. "data" will then have an object with a key named
# for the Query or Mutation you requested. The structure of that object
# will be the return value of the Query or Mutation you requested.
# Remember, the online documentation at https://caws3.nsslabs.com/docs is
# your best friend!
data = results.get('data')
if data:
    # ...
    # We specified this above!
    results = data.get('submissions')
    return results
else:
    # Get Errors from Results.
    # A failed response from the server will have a top most array of
    # objects in the JSON with an "errors" key.
    errors = results.get('errors')
    for error in errors:
        errorMessage = error.get('message', 'No Message in Error.')
        print(errorMessage)

    exit('Query Failed.')
SUBMIT A URL FOR SCAN

This mutation allows you to submit a URL to CAWS for scanning. When you submit a URL to CAWS for scanning, you will receive an email indicating that the scanning operation has been completed. If, however, you are submitting URLs from a script, you will need to poll the system to determine when the results are available. We recommend that you poll the system after 5 to 10 minutes, as that is how long the scanning operation usually takes, depending on load. This query is equivalent to the /Scan/Url endpoint in the CAWS 2 API.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>int</td>
<td>Your account's id. You can only pass your account's id. Passing any other value will result in an error. Example: 1</td>
</tr>
<tr>
<td>url</td>
<td>string</td>
<td>The URL you wish to submit for scanning. Example: &quot;<a href="http://www.google.com">http://www.google.com</a>&quot;</td>
</tr>
<tr>
<td>platforms</td>
<td>int[], array of integers</td>
<td>An array of platform package ids you want to scan on. You can get a list of platform package ids by invoking the submissions query. Example: [1,2,3]</td>
</tr>
<tr>
<td>browsers</td>
<td>int[], array of integers</td>
<td>An array of browser package ids you want to scan using. You can get a list of browser package ids by invoking the submissions query. Refer to Browser Package reference table to get the browser ID</td>
</tr>
</tbody>
</table>

Return Value

An id identifying the submission. You can use the id to poll the system and retrieve the results associated with the submission when they are available. To poll the system and retrieve results when they are available, use the submissions query.

```bash
1. curl https://caws3.nsslabs.com/graphql --data-urlencode 'query=mutation {
2.   uploadUrl(accountId: 1, url: "http://www.google.com", platforms: [1], browsers: [5])
3.     { id
4.       }
5.   }
6. } -H 'authorization: bearer <API Token>'

Response for URL Submissions Post Query

```

```json
<p>|</p>
<table>
<thead>
<tr>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>uploadURL: [&quot;AB671C35586C4CC29C00ED3D7F6231C5&quot;]</td>
</tr>
</tbody>
</table>
|```
Here is an example using Python. The example defines a function that you can reuse in a Python program. The code comments are pretty much self explanatory:

```python
import json
import requests
import requests_toolbelt

# ...
#
# Function to Submit a URL.
def submit_url(url, account_id, platforms, browsers, token):
    # Define Query to Send to Server.
    #
    # The query that is sent to the server is either a GraphQL Query or a
    # GraphQL Mutation. In this example because we want to get data, a we are
    # sending a GraphQL Query.
    #
    # Remember, GraphQL is graph based and we have to explicitly specify the
    # fields that we want returned in the response.
    #
    # For an online reference to all the different GraphQL Queries and
    # Mutations supported by CAWS, including their entire response data
    # structures, the online documentation at https://caws3.nsslabs.com/docs
    # is your best friend.
    query = ""
    mutation UploadUrl($account_id: ID!, $url: String!, $platforms:
    [Int]!, $browsers: [Int]!)
      {uploadUrl(accountId: $account_id, url: $url, platforms:
      $platforms, browsers: $browsers)
      }
    ""

    # Define Query Variables.
    #
    # Variables are a cool and useful feature of GraphQL that help avoid ugly
    # string concatenation when constructing queries that are sent to the
    # server. We recommend you use variables whenever possible.
    variables = {
        'account_id': account_id,
        'url': url,
        'platforms': platforms,
        'browsers': browsers
    }

    # Define Headers to Send to Server.
    #
    # The most important header to send to the server is the "Authorization"
    # header that includes your authentication token acquired from a previous
    # call to the "authenticateUser" Mutation. See this manual for an example
    # on how to perform authentication.
    headers = {
        'Authorization': 'bearer {}'.format(token)
    }
```
# Define Request Data to Send to Server.
#
# This is both the query and any variables in requires. Remember, GraphQL
# is JSON based so the variables parameter in the request has to be a JSON
# string!
data = {
    'query': query,
    'variables': json.dumps(variables)
}
# Send Request to Server.
#
# Unlike traditional REST APIs, in GraphQL, there is always a single
# endpoint you send the request to. The query you send as part of the
# request determines what the server actually does.
# The response body returned by the server is always formatted as JSON!
response = requests.post('https://caws3.nsslabs.com/graphql', data=data,
                          headers=headers)
results = response.json()

# Get Data from Results.
#
# A successful response from the server will have a top most object in the
# JSON with a "data" key. "data" will then have an object with a key named
# for the Query or Mutation you requested. The structure of that object
# will be the return value of the Query or Mutation you requested.
# Remember, the online documentation at https://caws3.nsslabs.com/docs is
# your best friend!
data = results.get('data')
if data:
    # ...
    # We specified this above!
    submission_tokens = data.get('uploadUrl')
    return submission_tokens
else:
    # Get Errors from Results.
    # A failed response from the server will have a top most array of
    # objects in the JSON with an "errors" key.
    errors = results.get('errors')
    for error in errors:
        errorMessage = error.get('message', 'No Message in Error.')
        print(errorMessage)

    exit('URL Submit Failed.')
SUBMIT A FILE FOR SCAN

Unlike other requests to the server, to download a file using the CAWS API, you just need to send an HTTP request to an endpoint. When you submit a file to CAWS for scanning, you will receive an email indicating that the scanning operation has been completed. If, however, you are submitting files from a script, you will need to poll the system to determine when the results are available. We recommend that you poll the system after 5 to 10 minutes, as that is how long the scanning operation usually takes, depending on load. This query is equivalent to the /Scan/File endpoint in the CAWS 2 API.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>int</td>
<td>Your account's id. You can only pass your account's id. Passing any other value will result in an error. Example: 1</td>
</tr>
<tr>
<td>file</td>
<td>string</td>
<td>Name of the file. Example: @testfile.pdf</td>
</tr>
<tr>
<td>platforms</td>
<td>int[], array of integers</td>
<td>An array of platform package ids you want to scan on. You can get a list of platform package ids by invoking the submissions query. Example: [1,2,3]</td>
</tr>
<tr>
<td>ApplicationPackage</td>
<td>int[], array of integers</td>
<td>The application to scan the file with. Use one of the values listed in the example and make sure to encode the value in the request. Refer to Application Package reference table to get the application ID</td>
</tr>
</tbody>
</table>

Return Value

An id identifying the submission. You can use the id to poll the system and retrieve the results associated with the submission when they are available. To poll the system and retrieve results when they are available, use the submissions query.

```
1. curl -X POST https://caws3.nsslabs.com/graphql
2.   -H 'authorization: bearer <API Token>'
3.   -H 'cache-control: no-cache'
4.   -H 'content-type: multipart/form-data; boundary=--WebKitFormBoundary7MA4YWxkTrZu0gW'
5.   -H 'postman-token: 5113 cac1-82 c8 - 0e35 - 9 f49 - 763e d5499501'
6.   -F 'platforms = [1]'
7.   -F 'applications = [2]'
8.   -F accountId=1
9.   -F 'operationName = "UploadFile"'
10.  -F 'file = @CAWS User Guide3.0_DRAFT_V1_amit1.pdf'
```
Here is an example using Python. The example defines a function that you can reuse in a Python program. The code comments are pretty much self explanatory:

```python
import json
import requests
import requests_toolbelt

def submit_file(file_name, file_path, file_mime_type, account_id, platforms, applications, token):
    m = requests_toolbelt.multipart.encoder.MultipartEncoder(
        [('files', (file_name, open(file_path, 'rb'), file_mime_type)),
         ('accountId', json.dumps(account_id)),
         ('platforms', json.dumps(platforms)),
         ('applications', json.dumps(applications))
    )

    # Define Headers to Send to Server.

    # The most important header to send to the server is the "Authorization"
    # header that includes your authentication token acquired from a previous
    # call to the "authenticateUser" Mutation. See this manual for an example
    # on how to perform authentication.
    headers = {
        'Authorization': 'bearer {}'.format(token),
        'Content-Type': m.content_type
    }

    # Send Request to Server.

    # Unlike other requests to the server, to submit a file using the CAWS
    # API, you just need to send an HTTP request to the following endpoint.
    # The response body returned by the server is always formatted as JSON!
    response = requests.post('https://caws3.nsslabs.com/graphql',
                              headers=headers,
                              data=m)
    results = response.json()
    if isinstance(results, list) and len(results) == 1:
        results = results[0]
    else:
```
exit('Submit File Failed: response="{}"'.format(str(results)))

data = results.get('data')
if data:
    submission_tokens = data.get('uploadFile')
    return submission_tokens
else:
    errors = results.get('errors')
    for error in errors:
        print(error.get('message', 'No Message in Error'))

    exit('Submit File Failed.')
**GET URL/FILE SCAN STATUS**

This query allows you to retrieve results for submitted files and URLs. You'll need to use this query in two ways. First, to poll the system to determine whether a submission is completed or not. Second, to retrieve the results for a completed submission. This query is equivalent to the Scan/Status/Url/{Token} in the CAWS 2 API.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>int</td>
<td>Your account's id. You can only pass your account's id. Passing any other value will result in an error. Example: 1</td>
</tr>
<tr>
<td>infold</td>
<td>Int</td>
<td>Pass the same value as the account parameter</td>
</tr>
</tbody>
</table>

**Request and Response Structure**

See the [online documentation](https://caws3.nsslabs.com/docs/securityprofilethreat.doc.html) for the complete request and response JSON structure for this query. As is the case for all GraphQL queries, only the fields you select in request JSON structure will be returned in the response JSON structure.

**Post Method to Get File Submissions Results**

1. `curl -X POST https://caws3.nsslabs.com/graphql`
2. `-H 'authorization: bearer <API Token>' -H 'cache-control: no-cache' -H 'content-type: application/json'`
3. `'
   {
   "query": "query fileDetailsInfo {submissions(accountId: 1, requestId: "7F613AA2FC9F4778A8B3491167886742") {fileDetailsInfo { id state displayName result hasExploit user platformName platformId applicationName applicationId utcSubmissionDate utcResponseDate }}}, "variables": null, "operationName": "fileDetailsInfo"
   }
   '``

**Response for File Scan Results**

1. `{{
2.   "data": {
3.     "submissions": {
4.       "fileDetailsInfo": {
5.         "id": 61,
6.         "state": 3,
7.         "displayName": "DOC_pdf_small_.pdf",
8.         "result": null,
9.         "hasExploit": null,
10.        "user": "apatel@nsslabs.com",
11.        "platformName": "Windows 7",
12.        "platformId": 1,
13.        "applicationName": "Adobe Reader 9.4",
14.        "applicationId": 20,
15.        "utcSubmissionDate": "Wed Jul 12 2017 16:36:25 GMT+0000 (UTC)",
16.        "utcResponseDate": null
17.     }
18.    }
19. }}`
Here is an example using Python. The example defines a function that you can reuse in a Python program. The code comments are pretty much self explanatory:

```python
import json
import requests

# Function to Get User Submission Status and Results.
def get_my_submissions(account):
    # Define Query to Send to Server.
    # The query that is sent to the server is either a GraphQL Query or a
    # GraphQL Mutation. In this example because we want to get data, a we are
    # sending a GraphQL Query.
    # Remember, GraphQL is graph based and we have to explicitly specify the
    # fields that we want returned in the response.
    # For an online reference to all the different GraphQL Queries and
    # Mutations supported by CAWS, including their entire response data
    # structures, the online documentation at https://caws3.nsslabs.com/docs
    # is your best friend.
    query = ""
    query MySubmissions($accountId: ID!, $infoId: ID!){
        submissions(accountId: $accountId, infoId: $infoId){
            pending{
                id
                displayName
                type
                typeName
                info
                platformName
            }
            urlInfo{
                id
                displayName
                result
                hasExploit
            }
            fileInfo{
                id
                displayName
                displayHash
                result
                hasExploit
            }
    }
```
# Define Query Variables.
# Variables are a cool and useful feature of GraphQL that help avoid ugly
# string concatenation when constructing queries that are sent to the
# server. We recommend you use variables whenever possible.
variables = {
    'accountId': account,
    'infoId': account
}

# Define Headers to Send to Server.
# The most important header to send to the server is the "Authorization"
# header that includes your authentication token acquired from a previous
# call to the "authenticateUser" Mutation. See this manual for an example
# on how to perform authentication.
headers = {
    'Authorization': 'bearer {}'.format(token)
}

# Define Request Data to Send to Server.
# This is both the query and any variables in requires. Remember, GraphQL
# is JSON based so the variables parameter in the request has to be a JSON
# string!
data = {
    'query': query,
    'variables': json.dumps(variables)
}

# Send Request to Server.
# Unlike traditional REST APIs, in GraphQL, there is always a single
# endpoint you send the request to. The query you send as part of the
# request determines what the server actually does.
# The response body returned by the server is always formatted as JSON!
response = requests.post('https://caws3.nsslabs.com/graphql',
headers=headers, data=data)
results = response.json()

# Get Data from Results.
# A successful response from the server will have a top most object in the
# JSON with a "data" key. "data" will then have an object with a key named
# for the Query or Mutation you requested. The structure of that object
# will be the return value of the Query or Mutation you requested.
# Remember, the online documentation at https://caws3.nsslabs.com/docs is
# your best friend!
data = results.get('data')
if data:
# ... 
# We specified this above!
results = data.get('submissions')
return results

else:
    # Get Errors from Results.
    #
    # A failed response from the server will have a top most array of
    # objects in the JSON with an "errors" key.
    errors = results.get('errors')
    for error in errors:
        errorMessage = error.get('message', 'No Message in Error.')
        print(errorMessage)

    exit('Query Failed.')
GET EXPLOITS/MALWARE Bypassing A Profile

This API call is used to get the list of Exploits blocked or bypassed by security products. The number of exploits bypassing one or more of your security products.

API CALL CURL EXAMPLES FOR PAGINATION AND BYPASS SCORING

```bash
1. curl -X POST https://caws3.nsslabs.com/graphql
2. -H 'authorization: bearer <API Token>'
3. -H 'content-type: application/x-www-form-urlencoded'
4. -d 'query=query {
  1. securityProfileAnalysis{
  2. start: "2017-08-30",
  3. end: "2017-08-30",
  4. account: 4,
  5. profile: 925
  6. {
  7. id  name  pagedThreats(excludeBlocked: true, excludeBypassed: false, page: 1, sortBy: testDate, sortDescending: true) {
  8. pagingInfo {
  9. total  pageCount  currentPage
  10. }  
  11. nodes {
  12. nssid  testDate  discoveryDate  device {
  13. id  name  
  14. }  
  15. exploitBlocked  captureDroppedMalware  malwareBlocked  processCount  maliciousProcessCount  
  16. fileCount  connectionCount
  17. }  
  18. }  
  19. }

json Response

1. {
  2. "data": {
  3. "securityProfileAnalysis": {
  4. "id": "925",
  5. "name": "Windows Defender-165",
  6. "pagedThreats": {
  7. "pagingInfo": {
  8. "total": 37,
  9. "pageCount": 1,
  10. "currentPage": 1
  11. },
  12. "nodes": [{
  13. "nssid": "NSS-2017-3TXTG5",
  14. "testDate": "2017-08-30T01:27:45.000Z",
  15. "discoveryDate": "2017-08-30T00:11:27.000Z",
  16. "device": {
  17. "id": "165",
  18. "name": "Windows Defender",
  19. "type": "Endpoint Protection",
  20. "vendor": {
```

- 60 -
"id": "18",
"name": "Microsoft",
"exploitBlocked": false,
"captureDroppedMalware": false,
"malwareBlocked": null,
"processCount": 5,
"maliciousProcessCount": 0,
"fileCount": 0,
"connectionCount": 0,
API Reference Tables for URL and File Scan

This section covers the tables that provide the IDs for Application Packages, Browser Packages and Platforms. In the API calls if Browser or platform or application is the variable the ID has to be passed.

Application Package Reference Table

This reference table lists the application package IDs you need to pass as a parameter when submitting files to CAWS for scanning. Each application package supports a specific extension and you are expected to pass an application package ID appropriate for the extension of the file you are submitting.

<table>
<thead>
<tr>
<th>Id</th>
<th>Application</th>
<th>Supported Extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Adobe Reader DC 2015.007.20033</td>
<td>.pdf</td>
</tr>
<tr>
<td>2</td>
<td>Adobe Reader 9.4</td>
<td>.pdf</td>
</tr>
<tr>
<td>3</td>
<td>Adobe Reader DC 2015.020.20039</td>
<td>.pdf</td>
</tr>
<tr>
<td>4</td>
<td>Quicktime 7.79</td>
<td>.mp3, .mp4, .m4a, .wav, .avi, .mov</td>
</tr>
<tr>
<td>5</td>
<td>Itunes 12.5.1</td>
<td>.mp3, .mp4, .m4a, .wav, .avi, .mov</td>
</tr>
<tr>
<td>6</td>
<td>Itunes 12.5.4</td>
<td>.mp3, .mp4, .m4a, .wav, .avi, .mov</td>
</tr>
<tr>
<td>7</td>
<td>Microsoft Office 2013</td>
<td>.doc, .docx, .xls, .xlsx, .csv, .ppt, .pptx</td>
</tr>
<tr>
<td>8</td>
<td>Microsoft Office 2016</td>
<td>.doc, .docx, .xls, .xlsx, .csv, .ppt, .pptx</td>
</tr>
<tr>
<td>9</td>
<td>Windows Media Player 12.0.7601</td>
<td>.mp3, .mp4, .m4a, .wav, .avi, .mov</td>
</tr>
<tr>
<td>10</td>
<td>Windows Media Player 12.0.10011</td>
<td>.mp3, .mp4, .m4a, .wav, .avi, .mov</td>
</tr>
<tr>
<td>11</td>
<td>VLC 2.2.3</td>
<td>.mp3, .mp4, .m4a, .wav, .avi, .mov</td>
</tr>
<tr>
<td>12</td>
<td>VLC 2.2.4</td>
<td>.mp3, .mp4, .m4a, .wav, .avi, .mov</td>
</tr>
</tbody>
</table>

Browser Package Reference Table

This reference table lists the browser package IDs you need to pass as a parameter when submitting URLs to CAWS for scanning.

<table>
<thead>
<tr>
<th>Id</th>
<th>Browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Firefox 50.0.1</td>
</tr>
<tr>
<td>2</td>
<td>Firefox 50.0.2</td>
</tr>
<tr>
<td>3</td>
<td>Google Chrome 53.0.2785.101</td>
</tr>
</tbody>
</table>
### Platform Package Reference Table

This reference table lists the platform package IDs you need to pass as a parameter when submitting files or URLs to CAWS for scanning.

<table>
<thead>
<tr>
<th>Id</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Windows 7 SP1</td>
</tr>
<tr>
<td>2</td>
<td>Windows 10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id</th>
<th>Browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Google Chrome 54.0.2840.59</td>
</tr>
<tr>
<td>5</td>
<td>Internet Explorer 9</td>
</tr>
<tr>
<td>6</td>
<td>Internet Explorer 11</td>
</tr>
</tbody>
</table>