Introduction

The purpose of this document is to aid you in the conversion of code that uses the StrikeTracker 2 API to use the new StrikeTracker API. The new API gives you much more flexibility, improved security, and many added features, and the new API exposes the full configuration system to you. This document walks you through the basics of authentication, walking the account tree, fetching analytics, viewing and changing configuration, managing users, and purging content on the CDN.

This guide assumes that you are using an administrator user to authenticate the API which has a minimum permission set of report read, configuration edit, account edit, and content edit. If your API user has fewer permissions, as you go through this guide you may experience 403 Permission Denied errors or 404 Not Found errors. Please adjust either your user's permissions or exclude sections from this guide when writing your own API integration.

This guide uses PHP in order to illustrate concepts. PHP is by no means required for your API integration. It just happens to be one of the most common programming languages used for API integration by our API users today. Furthermore, we use PHP 5.5. If you are using an older version of PHP, you may need to adjust the code accordingly.

Most of the API examples available on our github page (https://github.com/highwinds/CDNWS-examples) are written in Python using the requests library. You may use whatever programming language suits your purposes. In addition, it is common for many API integrations written in PHP to use cURL to interact with the API. In this guide, we use a third-party library called Guzzle, which tends to make the code easier to read, especially for developers not familiar with PHP. For more information on Guzzle, visit their website at guzzle.readthedocs.org. To create a client for use throughout the guide, simply instantiate the Client class:

```
<?php
$client = new GuzzleHttp\Client();
```

Authentication

The StrikeTracker 2 API allowed you to use HTTP Basic Authentication in order to authenticate with the API. There are known security issues with this method, not the least of which is the requirement to store your API user's username and password in the clear with your application. For improved security and added flexibility, authentication for the new StrikeTracker API uses oauth2, a security standard used by most public APIs, including Twitter and Facebook. Generic oauth2 libraries for your programming language may work with the new StrikeTracker API, making the authentication portion of your application easier to write. For the purposes of this guide, we do not use such a library, but we explicitly call the APIs, making it easier to translate to other programming languages besides PHP.

First, you must obtain an access token. This token becomes your session key for authenticating future requests to the API. The token is valid for one hour, after which you may use a refresh token to obtain a new access token. To get an access token, simply form post your credentials to the authentication endpoint. You can then set the Authorization header to be used in future requests to the API.

```
# Fetch an access token
$response = $client->post('https://striketracker.highwinds.com/auth/token', [
```

```
    'body' => [
        'grant_type' => 'password',
        'username' => $username,
        'password' => $password
    ]
]);
$ACCESS_TOKEN = $response->json()['access_token'];
echo "Access token: " . $ACCESS_TOKEN . "\n";

# Include Authorization header with all future requests
$client->setDefaultOption('headers/Authorization', 'Bearer ' . $ACCESS_TOKEN);
```

Account context

Every action in the new StrikeTracker API is done in an account context. When interacting with subaccounts, you must use the account context of the subaccount, not the parent account. Most URLs follow this format:

`/api/accounts/`**`[accountHash]`**`/`**`[entity]`**

For example, to fetch a list of hosts for the user's root account, use the **`users/me`** endpoint to fetch their root account hash, and then use that account context to fetch hosts:

```
# Fetch the current user's account context
$response = $client->get('https://striketracker.highwinds.com/api/users/me');
$ACCOUNT_HASH =  $response->json()['accountHash'];
echo "Account hash: " . $ACCOUNT_HASH . "\n";

# Fetch a list of hosts
$response = $client-
>get("https://striketracker.highwinds.com/api/accounts/$ACCOUNT_HASH/hosts");
$hosts = $response->json()['list'];
echo "Hosts:\n";
foreach($hosts as $host) {
    echo "  " . $host['hashCode'] . "\t" . $host['name'] . "\n";
}
```

If you have subaccounts, you can use the subaccounts API to fetch their account hashes for use in the account context:

```
# Fetch account information for one of your subaccounts
$response = $client-
>get("https://striketracker.highwinds.com/api/accounts/$ACCOUNT_HASH/subaccounts");
$subaccount = $response->json()['list'][0];
echo "One of your subaccounts:\n";
echo "  Name: " . $subaccount['accountName'] . "\n";
echo "  Hash: " . $subaccount['accountHash'] . "\n";
```

Fetching analytics

The StrikeTracker 2 API only allowed you to fetch analytics for one host and platform. The new StrikeTracker analytics API is much more flexible and provides many more options than were available in the StrikeTracker 2 API. This guide shows you how to fetch analytics for a single host and platform in order to convert existing analytics code as well as how to fetch analytics for your entire account grouped by host.

In StrikeTracker 2, an example request looked like this:

```
GET
https://striketracker2.highwinds.com/webservices/stats/{hashCode}/{platformCode}?inte
rval=1day&metrics=actualTransfer;attemptedTransfer;hits;duration&startDate=2014-12-
01T00:00:00Z&endDate=2014-12-31T23:55:00Z
<?xml version="1.0" encoding="UTF-8"?>
<dataPoints>
  <dataPoint>
    <timestamp>...</timestamp>
    <actualTransfer>...</actualTransfer>
    <attemptedTransfer>...</attemptedTransfer>
    <hits>...</hits>
    <duration>...</duration>
  </dataPoint>
</dataPoints>
```

Here is the corresponding request in the new StrikeTracker:

```
# Fetch analytics for a host on the user's root account
$hostHash = $hosts[0]['hashCode'];
$response = $client->get(

"https://striketracker.highwinds.com/api/accounts/$ACCOUNT_HASH/analytics/transfer" .
    "?hosts=$hostHash&platforms=CDS&granularity=P1D&startDate=2014-12-
01T00:00:00Z&endDate=2014-12-31T23:55:00Z");
$analytics = $response->json();
$timestamp =            array_search('usageTime',
$analytics['series'][0]['metrics']);
$actualTransfer =       array_search('xferUsedTotalMB',
$analytics['series'][0]['metrics']);
$attemptedTransfer =    array_search('xferAttemptedTotalMB',
$analytics['series'][0]['metrics']);
$hits =                 array_search('requestsCountTotal',
$analytics['series'][0]['metrics']);
$duration =             array_search('durationTotal',
$analytics['series'][0]['metrics']);
echo "Traffic for " . $hostHash . " for December 2014:\n";
```

```
if (count($analytics['series'][0]['data']) === 0) {
    echo "  No analytics recorded during this time period\n";
}
foreach ($analytics['series'][0]['data'] as $row) {
    echo "  " . date('Y-m-d', $row[$timestamp]/1000) . "\t";
    echo $row[$actualTransfer] . "\t";
    echo $row[$attemptedTransfer] . "\t";
    echo $row[$hits] . "\t";
    echo $row[$duration] . "\n";
}
```

If your goal is to fetch analytics for all of your hosts, simply leave off the hosts parameter and add a groupBy=HOST:

```
# Fetch analytics for all hosts on the user's root account
$response = $client->get(

"https://striketracker.highwinds.com/api/accounts/$ACCOUNT_HASH/analytics/transfer" .
    "?groupBy=HOST&platforms=CDS&granularity=P1D&startDate=2014-12-
01T00:00:00Z&endDate=2014-12-31T23:55:00Z");
$analytics = $response->json();
$timestamp =            array_search('usageTime',
$analytics['series'][0]['metrics']);
$actualTransfer =       array_search('xferUsedTotalMB',
$analytics['series'][0]['metrics']);
$attemptedTransfer =    array_search('xferAttemptedTotalMB',
$analytics['series'][0]['metrics']);
$hits =                 array_search('requestsCountTotal',
$analytics['series'][0]['metrics']);
$duration =             array_search('durationTotal',
$analytics['series'][0]['metrics']);
foreach ($analytics['series'] as $series) {
    echo "Traffic for " . $series['key'] . " for December 2014:\n";
    if (count($series['data']) === 0) {
        echo "  No analytics recorded during this time period\n";
    }
    foreach ($series['data'] as $row) {
        echo "  " . date('Y-m-d', $row[$timestamp]/1000) . "\t";
        echo $row[$actualTransfer] . "\t";
        echo $row[$attemptedTransfer] . "\t";
        echo $row[$hits] . "\t";
        echo $row[$duration] . "\n";
    }
}
```

Viewing and changing configuration

The configuration system in the new StrikeTracker is much more flexible and powerful than the StrikeTracker 2 configuration API, which mostly just allowed you to view and set a handful of content protection policies. The new StrikeTracker configuration API exposes the full power of the CDN's configuration system, and as such, requires a fundamentally different approach when setting configuration. To begin, first fetch the host's information. For our purposes in this guide, we saved the first host from the hosts list call earlier in the guide. A host looks something like this:

```
{
    "name": "www.example.com",
    "hashCode": "a1b2c3d4",
    "services": [
        {
            "id": 40,
            "name": "HTTP Caching (CDS)",
            "description": "NA\/EU"
        }
    ],
    "scopes": [
        {
            "id": 1234,
            "platform": "CDS",
            "path": "\/"
        }
    ]
}
```

The StrikeTracker 2 API forced you to use a different endpoint for each configuration type that you wanted to set, providing the host hash and path as part of the request. For example, there is an HTTP Basic Authentication API that allows you to set this policy on a host. However, the most common use case is setting many different types of configuration on a single path on the CDN, which makes this kind of interaction with the StrikeTracker 2 API very chatty.

The new StrikeTracker introduces the concept of a configuration scope in order to ease the setting of many configuration options at once. In looking at the host above, the hashCode is the host hash we use in the URL in order to view and edit configuration. The scope id is what we use to view and edit configuration for a specific configuration scope. The scope presented in the services array above is the scope you probably want to use for setting most configuration options. This is the CDS root scope, which controls configuration for CDN delivery. You may have other delivery platforms or more deeply nested scopes in the scope list on your hosts. However, if you are just starting out with setting configuration on the CDN, this is the scope you should use.

Once you know the host and scope you wish to view, simply issue a GET with the appropriate parameters interpolated in the URL:

```
# Grab the configuration for the first host on the user's root account
$hostHash = $hosts[0]['hashCode'];
$scopeId = null;
foreach ($hosts[0]['scopes'] as $scope) {
```

```
    if ($scope['platform'] === 'CDS' && $scope['path'] === '/') {
        $scopeId = $scope['id'];
    }
}
if ($scopeId === null) {
    echo "Couldn't find CDS root scope!\n";
    exit();
}
$response = $client->get(

"https://striketracker.highwinds.com/api/accounts/$ACCOUNT_HASH/hosts/$hostHash/confi
guration/$scopeId");
echo "Configuration for the CDS root scope on $hostHash\n";
echo json_encode($response->json(), JSON_PRETTY_PRINT);
```

The configuration returned looks something like this:


```
{
    "cacheControl": [
        {
            "id": 2906471162,
            "maxAge": 86400
        }
    ],
    "originPullCacheExtension": {
        "id": 4172915425,
        "expiredCacheExtension": 86400
    },
    "originPullHost": {
        "id": 35748,
        "primary": 1234,
        "secondary": null,
        "path": null
    },
    "originPullPolicy": [
        {
            "id": 1848473332,
            "expirePolicy": "CACHE_CONTROL",
            "expireSeconds": 86400
        }
    ],
    "originPullShield": {
        "id": 435462008,
        "enabled": false
    },
    "scope": {
        "id": 72183,
        "platform": "CDS",
```

```
        "path": "\/"
    }
}
```

The Highwinds configuration system is a key/type/value system where some types are groupable (represented as arrays in the API response) and some are non-groupable (represented as objects in the API response). Each instance of a type represented in the API has an id. This represents the unique identifier for a policy on this scope. If you want to edit one of these policies, you should include the id in the request body sent to the API. This gives the API the information necessary to identify the change as an update instead of the creation of a new record. Including the id also allows the API to perform conflict checking for you, assuring that the policy has not been updated by anyone else in the interim between your GET and your PUT. If you wish to delete the existing policy and replace it with your own, simply leave off all of the ids, and the API performs a delete and insert.

When managing configuration through the API, you may alter the JSON response from the GET call and PUT it back to the API. However, the new StrikeTracker also supports partial updates on a per-type basis. You can simply pass a single policy you want to either add, update, or delete, and the API takes the appropriate action and return the new configuration.

Here is an example of setting an HTTP Basic Authentication policy on this configuration scope:

```
# Add an HTTP Basic Authentication policy here
$response = $client->put(

"https://striketracker.highwinds.com/api/accounts/$ACCOUNT_HASH/hosts/$hostHash/confi
guration/$scopeId", ['json' => [
    'authHttpBasic' => [
        'bindingPoint' => "http://www.example.com/auth",
        'realm' => "Authenticate",
        'ttl' => 60
    ]
]]);
echo "Configuration after applying basic authentication policy\n";
echo json_encode($response->json(), JSON_PRETTY_PRINT) . "\n";
```

To delete the policy, simply send an empty array:

```
# Delete the HTTP Basic Authentication policy here
$response = $client->put(

"https://striketracker.highwinds.com/api/accounts/$ACCOUNT_HASH/hosts/$hostHash/confi
guration/$scopeId", ['json' => [
    'authHttpBasic' => []
]]);
echo "Configuration after deleting basic authentication policy\n";
echo json_encode($response->json(), JSON_PRETTY_PRINT) . "\n";
```

The configuration API exposes a discovery document at `/api/configuration` that details all of the types and keys supported on the CDN. This document is used to automatically generate much of the configuration section of StrikeTracker. You can use this document to discover what configuration you can set on your hosts. It is updated whenever the CDN supports new functionality and is an exhaustive source of configuration information. Here is an example of fetching the configuration discovery document programmatically and listing out the supported types:

```
# List configuration types available on the CDN
$response = $client->get("https://striketracker.highwinds.com/api/configuration");
$configuration_spec = $response->json();
echo "Configuration types available on the CDN: ";
$types = [];
foreach ($configuration_spec as $category) {
    foreach($category as $type=>$typeDef) {
        array_push($types, $type);
    }
}
echo implode(', ', $types) . "\n";
```

Managing users

The new StrikeTracker users API is almost identical to the users API in StrikeTracker 2. If you use the id to manage users, only the user and response format changes. If you are using the email format, you need to instead use the users/list API in order to find the appropriate user and PUT back the user by id. For the purposes of this guide, we are selecting the first user in the list and updating their fax number.

First, fetch the user list for the current account context:

```
# Fetch a list of users
$response = $client->get(
    "https://striketracker.highwinds.com/api/accounts/$ACCOUNT_HASH/users");
echo "Users on your root account:\n";
$users = $response->json()['list'];
foreach ($users as $user) {
    echo "  " . $user['firstName'] . " " . $user['lastName'] . "\n";
}
```

Then update the fax number for the first user:

```
# Set a user's fax number
$user = $users[0];
$user['fax'] = '555-867-5309';
$response = $client->put(
    "https://striketracker.highwinds.com/api/accounts/$ACCOUNT_HASH/users/" .
$user['id'], ['json' => $user]);
echo "User after change:\n";
echo json_encode($response->json(), JSON_PRETTY_PRINT) . "\n";
```

The format of the user roles have changed but operate in much the same way. There are a list of permissions for each user role, with a value of EDIT, READ, or NONE, just like the StrikeTracker 2 API. In addition, a userType can be set that gives the user a set of meta permissions. For most use cases, either use an "Administrator" userType in order to give the user the ability to administer other users' permissions or "Normal" userType, which does not give the user any permissions for editing other users. For example, here is an example of some roles that could be set on a user:

```
{
    ...
    "roles": {
        "userAccount": {
            "report": "EDIT",
            "account": "EDIT",
            "content": "EDIT",
            "configuration": "EDIT"
        },
        "subaccounts": {
            "report": "EDIT",
            "account": "EDIT",
            "content": "EDIT",
            "configuration": "EDIT"
        }
    },
    "userType": "Administrator",
    ...
}
```

Purging content on the CDN

One of the most popular use cases for API integration is purging content on the CDN. This forces the CDN to invalidate the cache and reingest the asset on the next request. Many sites use long expiration times and programmatically purge content when it is updated so the CDN always has the latest asset in cache. The StrikeTracker 2 API was very limited in that it only allowed you to purge a single asset on a single configuration path on a single host. The new StrikeTracker purge API now supports account-level purge by hostname and provides a mechanism for batching multiple purges together. For example, here is an example of purging all CSS files on three different properties serving traffic through the CDN:

```
# Purge some content from the CDN
$response = $client->post(
    "https://striketracker.highwinds.com/api/accounts/$ACCOUNT_HASH/purge", ['json'
=> [
        'list' => [
            [
                'url' => 'http://cds.' . $hosts[0]['hashCode'] . '.hwcdn.net/css/',
                'recursive' => true
            ],
            [
```

```
                'url' => 'http://cds.' . $hosts[1]['hashCode'] . '.hwcdn.net/css/',
                'recursive' => true
            ],
            [
                'url' => 'http://cds.' . $hosts[2]['hashCode'] . '.hwcdn.net/css/',
                'recursive' => true
            ],
        ]
]]);
if ($response->getStatusCode() === 200) {
    echo "Purge has been successfully submitted to the CDN\n";
} else {
    echo "Purge request failed: " . $response->getBody() . "\n";
}
```

As you can see, you can provide a batch of purge requests from any hostname or CNAME target associated with your account. This makes it very easy and efficient to purge the cache when necessary. You may submit up to 2MB of encoded URLs in the request body.

A word on API rate limiting

To discourage API integration behaviors that negatively affect service performance on the new StrikeTracker API, we have enabled a fixed but generous API rate limit on the API. When your requests begin to exceed the rate limit, your requests are slowed and your concurrent connections limited so that you fall within the parameters of our existing rate limits. This allows you to burst to a temporarily larger number of requests at peak. However, if you continue to exceed the rate limits, the API automatically rejects your requests until you fall within the rate limit again. Please follow these best practices when interacting with the API to avoid rate limits:

- Only request a new auth token when needed and not before every interaction with the API. When available, use a caching service such as memcached to cache your access token so that multiple requests can take advantage of the same token. You may also use a refresh token to obtain a new access token once yours has expired without having access to the original username and password.
- Exceeding rate limits is especially common when purging content from the CDN. Take advantage of purge batches to dramatically reduce the number of total requests that are necessary to purge your content from the CDN. You may provide up to 2MB of URLs in a single purge batch.
- Cache data that does not change often, like user emails and account information. Requesting this information too often can cause you to exceed your rate limit.
- When writing applications that interact with analytics, please cache the results whenever possible. Analytics for previous days and months rarely change, and as such can be cached for long periods of time.