Attacking Secondary Contexts in Web Applications

Sam Curry

whoami

- Sam Curry
  (@samwcyo)
- Full time bug bounty hunter
  (3 years on-and-off)
- Passionate about application security/research
  (run blog @ samcurry.net)

How I previously thought all HTTP servers worked...

- Application files are stored/accessed in webserver folder
  - /var/www/html/
  - /usr/share/nginx/html/
  - … etc …
- GET /index.html
  - Tries to load in /webserver/index.html
- GET /folder/index.html
  - Tries to load in /webserver/folder/index.html
- Very straightforward and simple

Different ways web applications do routing

- Not actually dealing with stored files, rather using defined routes

```
4
5   const MainUserRouter = require("express").Router();
6
7   MainUserRouter.route("/activate")
8       .get(require("./show-activate-page.js"))
9       .post(require("activate.js"));
10
11  MainUserRouter.route("/deactivate")
12      .get(require("./show-deactivate-page.js"))
13      .post(require("deactivate.js"));
14
15  MainUserRouter.route("/register")
16      .get(require("./show-register-page.js"))
17      .post(require("register.js"));
18
19  module.exports = MainUserRouter;
```

```
const express = require('express' 4.17.1 )
const app = express()
const port = 3000

app.get('/', (req, res) => res.send('Hello World!'))

app.listen(port, () => console.log(`Example app listening on port ${port}!`))
```

Different ways web applications do routing

- Sent across middleware and proxies, sometimes through load balancers...

```
location /some/path/ {
    proxy_pass http://www.example.com/link/;
}
```

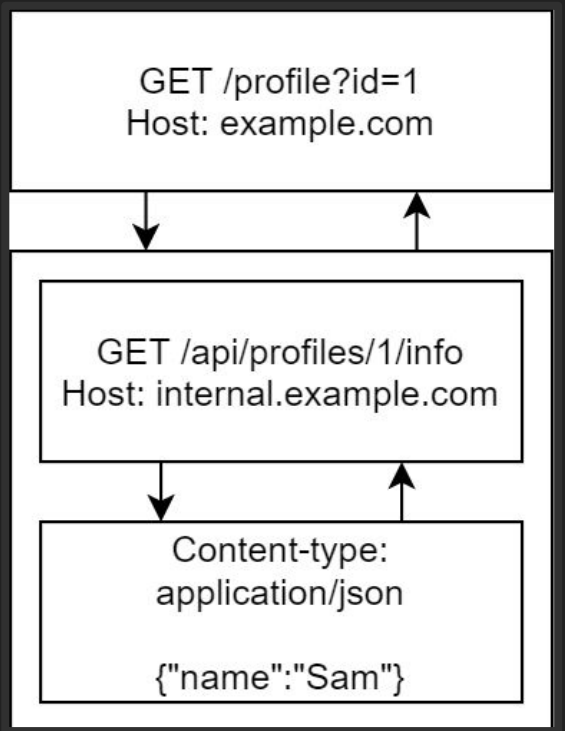
```
location ~ \.php {
    proxy_pass http://127.0.0.1:8000;
}
```

```
ProxyPass "/"  "http://www.example.com/"
ProxyPassReverse "/"  "http://www.example.com/"
```

```
ProxyPass "/images"  "http://www.example.com/"
ProxyPassReverse "/images"  "http://www.example.com/"
```

Different ways web applications do routing

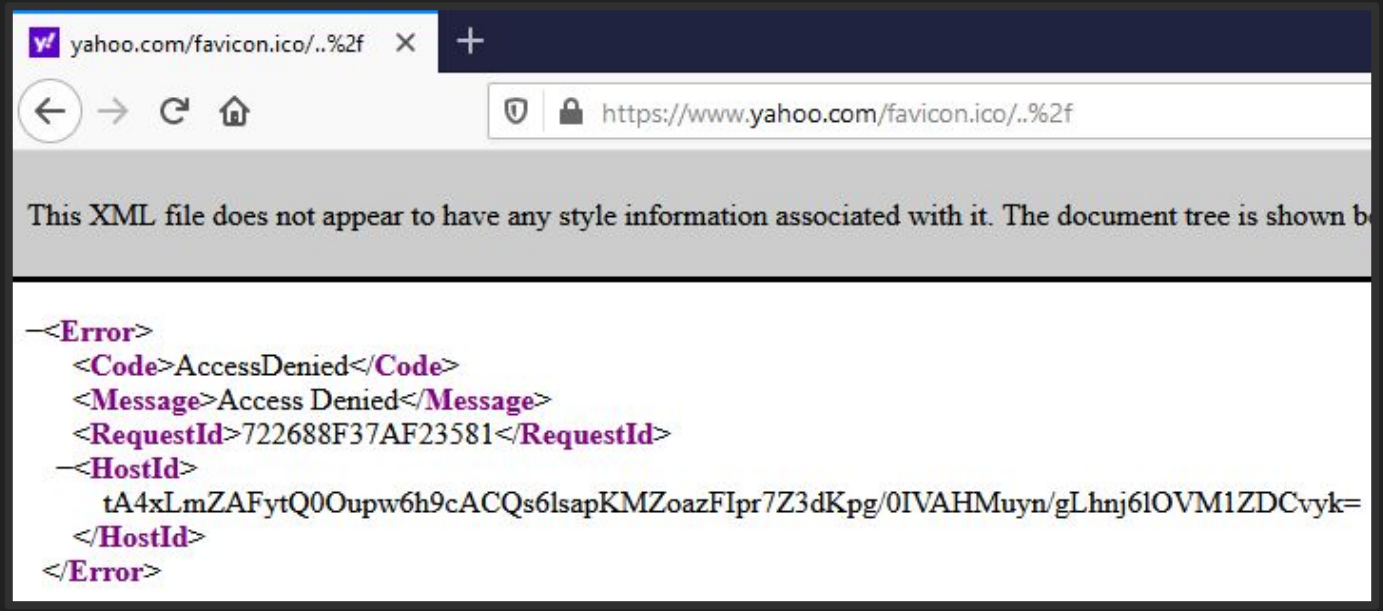
- Fetching content from APIs
  - Sending a 2nd HTTP request
  - Usually a different host
  - Common lack of input validation
- Sometimes carries auth info to API
  - Underlying authentication models
    - Sometimes not present…



Methods for identifying application routing

- Directory traversal
  - Does “/api/../” return something different than “/”?
- Fuzzing using control characters
  - %23 (#), %3f (?), %26 (&), %2e (.), %2f (/), %40 (@)
  - Double/triple URL encoding
- Does the behavior suddenly change for certain directories?
  - Why does “/images/” return different headers than “/”?
- Are there any nice bits of information we can catch?
  - “internal.company.com:8080 returned the following: ‘500 internal server error’”

Identifying application routing - Examples



- We can identify /favicon.ico* is being served through CloudFront
- What if this was being served through an S3 bucket?
  - GET /favicon.ico/..%2f..%2fattackersbucket%2fxss.html
  - (Proxied as https://s3.amazonaws.com/yahoo-bucket/favicon.ico/../../attackersbucket/xss.html)

Identifying application routing - Examples

- Requesting the webroot behaves totally normally
- Browsing to /api/v1/ reveals different behavior
  - Different headers, content-type, etc.
- We can confirm the routing is separate via traversing backwards to “/” on the API server via “/../../../”

```
GET /api/v1/groups/../../../ HTTP/1.1
Host: [redacted]
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Date: Fri, 20 Mar 2020 06:10:20 GMT
X-Yahoo-Serving-Host: [redacted]
Age: 0
Server: ATS
Referrer-Policy: no-referrer-when-downgrade
Connection: keep-alive
Strict-Transport-Security: max-age=15552000
Expect-CT: max-age=31536000,
report-uri="http://csp.yahoo.com/beacon/csp?src=yahoocom-expect-ct-repor
t-only"
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Content-Length: 1690

{"handlers":[{"id"[redacted]onfig.StatisticsRequestHandler",
"class":[redacted]fig.StatisticsRequestHandler","bundle":"
container-disc:5.50.9","serverBindings":["http://*/statistics/*","https:
//*/statistics/*"]},{"id":[redacted]ndler.observability.App
licationStatusHandler","class":[redacted]dler.observabilit
y.ApplicationStatusHandler","bundle":"container-search-and-docproc:5.50.
9","serverBindings":["http://*/ApplicationStatus","https://*/Application
Status"]},{"id"[redacted]ndler.VipStatusHandler","class":"
[redacted]dler.VipStatusHandler","bundle":"container-disc:5
.50.9","serverBindings":["http://*/status.html","https://*/status.html"]
},{"id":[redacted].VipStatusHandler"[redacted]
oups.gapi.VipStatusHandler","bundle":"gapi:1.0.0","serverBindings":["htt
p://*:4080/status.html"]},{"id":"[redacted]bility.BindingsO
verviewHandler","class":"[redacted]bility.BindingsOverviewH
```

Common issues with secondary contexts

- Data is being served across extra layers
  - Introduces translation issues like HTTP request smuggling
  - CRLF injection in weird places
- Developers do not expect users to be able to control parameters/paths
  - Functionality you would normally see in a development environment is accessible (?debug=1, /server-status,
- Information disclosure
  - Internal HTTP headers, access token
- SSRF and XSS via manipulating response content
  - Finding an open redirect in 2nd context = server issuing/potentially rendering arbitrary request

Identifying application routing - Examples

Request

Raw | Headers | Hex

```
GET /files/lol.png%23 HTTP/1.1
Host: [redacted]
Cookie:
SMIDENTITY=Z+AOJgtla9FaWgDUJpN3eJvDuB6lS3qeKu7qMEQM3f4M
4TzxITDPggJ9BHzSVagzku8N7v/qJRtl9Sadbq7/P7YWietG2fYSl+
grJMCSS6iJECJK4E0/CDvWNcUrc9jO867i4OuYlyuJJeriZDVahti9
pCkQa5geM7anggUwgD4+htE/NL5Jvr2vLmtLkQZlLpLP3PF6x9/HH5
```

Response

Raw | Headers | Hex | JSON Beautifier

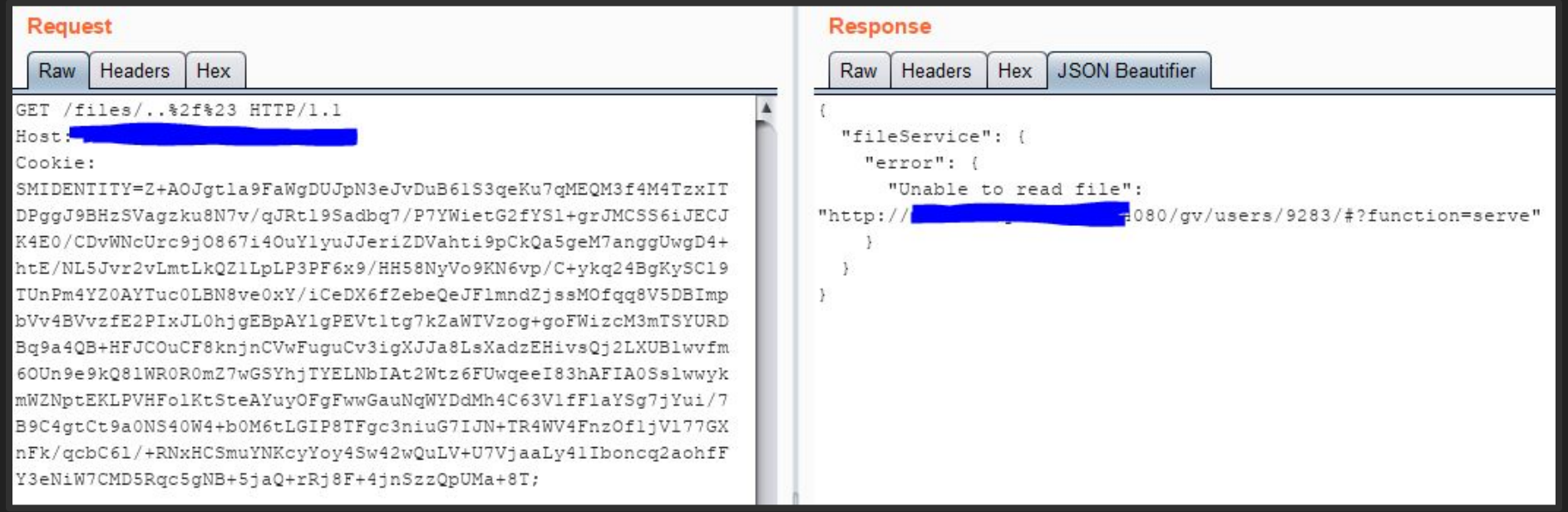
```
{
  "fileService": {
    "error": {
      "Unable to read file":
"http://[redacted]4080/gv/users/9283/uploads/lol.png#?function=serve"
    }
  }
}
```

- Passing in “%23” turns into “#” and makes the underlying request fail as the parameters are dropped

- What control do we have over the second request?

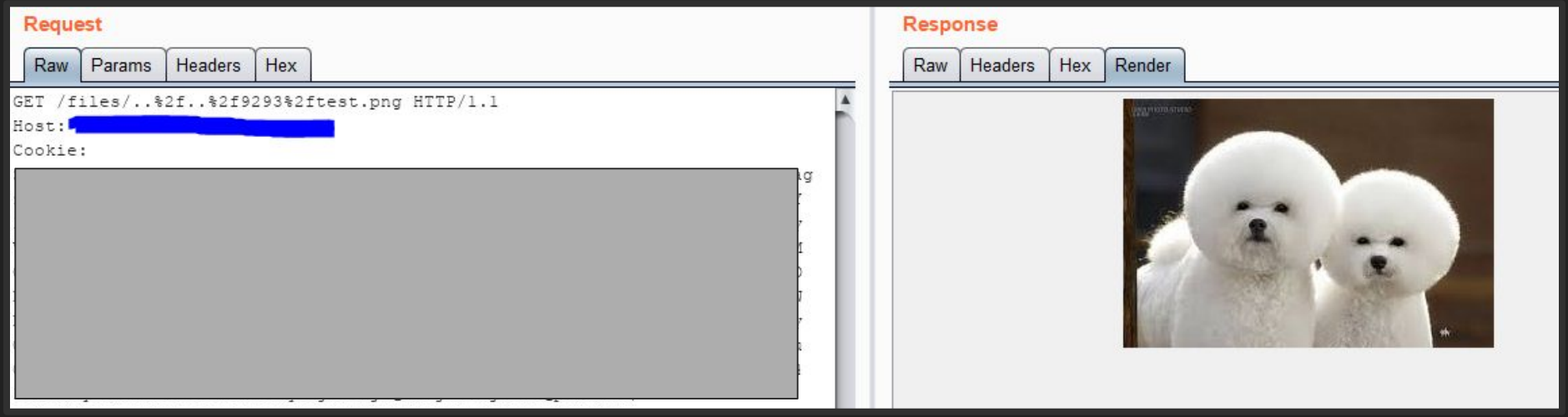
- How could this be exploited by an attacker?

Identifying application routing - Examples



- Traversing backwards allows us to overwrite the API paths
- Indexing for user ID is based on the session cookie

Identifying application routing - Examples

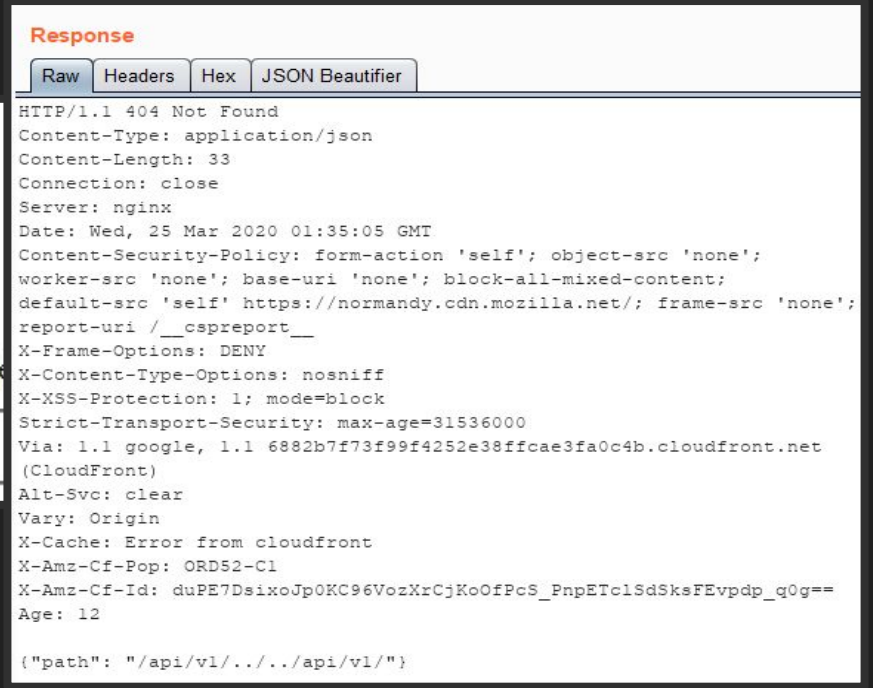


- We can traverse the internal API, overwrite the user ID, then read a victim's file
- All other API calls are also accessible

GET /files/..%2f..%2f + victim ID + %2f + victim filename

Common issues attacking secondary contexts

- APIs will oftentimes not normalize request URLs
  - Impossible to traverse API calls



Common issues attacking secondary contexts

- Underlying authentication makes access control issues impossible
  - Even if an API is internal, there isn't any benefit besides widened attack surface

The `ProxyPassReverseCookieDomain` directive has syntax:

1

```
ProxyPassReverseCookieDomain internal-domain public-domain [interpolate]
```

Just like in this example for `ProxyPassReverse`, the **order is reversed** (back-end first):

```
ProxyPass         "/mirror/foo/" "http://backend.example.com/"
ProxyPassReverse  "/mirror/foo/" "http://backend.example.com/"
ProxyPassReverseCookieDomain  "backend.example.com"  "public.example.com"
ProxyPassReverseCookiePath  "/"  "/mirror/foo/"
```

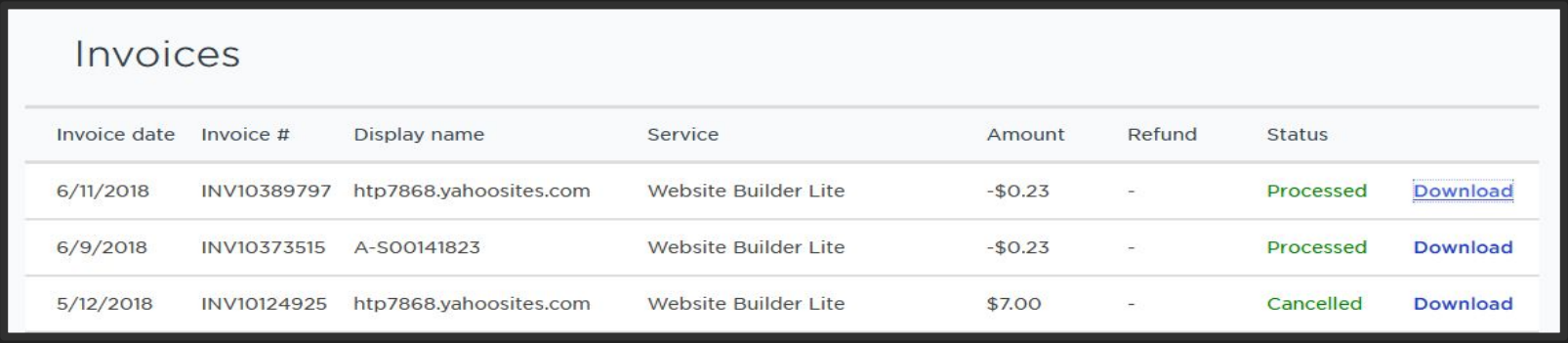
share improve this answer

answered Jul 8 '18 at 8:20

Esa Jokinen

27.8k ●2 ●43 ●73

Identifying application routing - Examples



Invoices

Invoice date	Invoice #	Display name	Service	Amount	Refund	Status	
6/11/2018	INV10389797	htp7868.yahoosites.com	Website Builder Lite	-$0.23	-	Processed	Download
6/9/2018	INV10373515	A-S00141823	Website Builder Lite	-$0.23	-	Processed	Download
5/12/2018	INV10124925	htp7868.yahoosites.com	Website Builder Lite	$7.00	-	Cancelled	Download

- HTTP request loads the specified invoice PDF
- IDOR doesn't work, returns 404 (somewhat interesting)
- Are they doing anything weird/exploitable here?

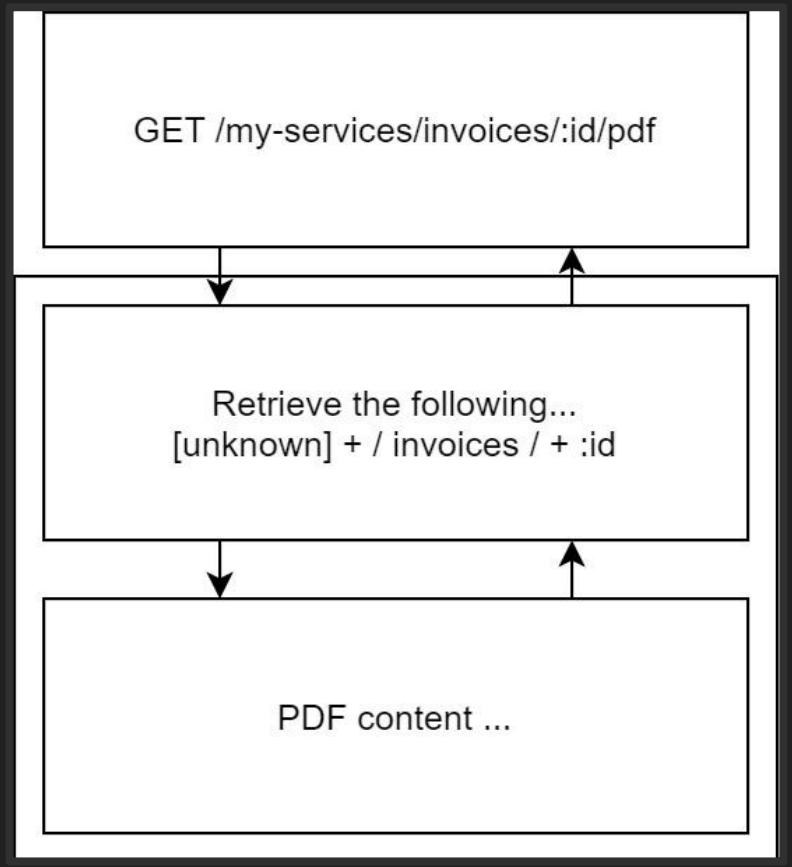
Identifying application routing - Examples

- GET /my-services/invoices/..%2finvoices%2fINV08179455/pdf
  - This works (200 with PDF content)
- GET /my-services/invoices/..%2f..%2fmy-services%2finvoices%2fINV08179455/pdf
  - This doesn't (404 without PDF content)
- This doesn't really prove anything, but it's interesting
  - If it were traversing on the same box/normally, it'd likely load both
  - This is probably worth at least investigating a little bit

```
Content-disposition: inline; filename=INV10389797.pdf
```

Identifying application routing - Examples

- There’s a possibility a directory before “/invoices/” is indexing our uploads (/:userid/invoices/:invoiceid)
- If we can guess this directory, we can potentially view other users invoices
- Lots of things to guess here...



Identifying application routing - Examples

- Intruder (0-1000000) not working
- Email not working
- Username not working

… but …

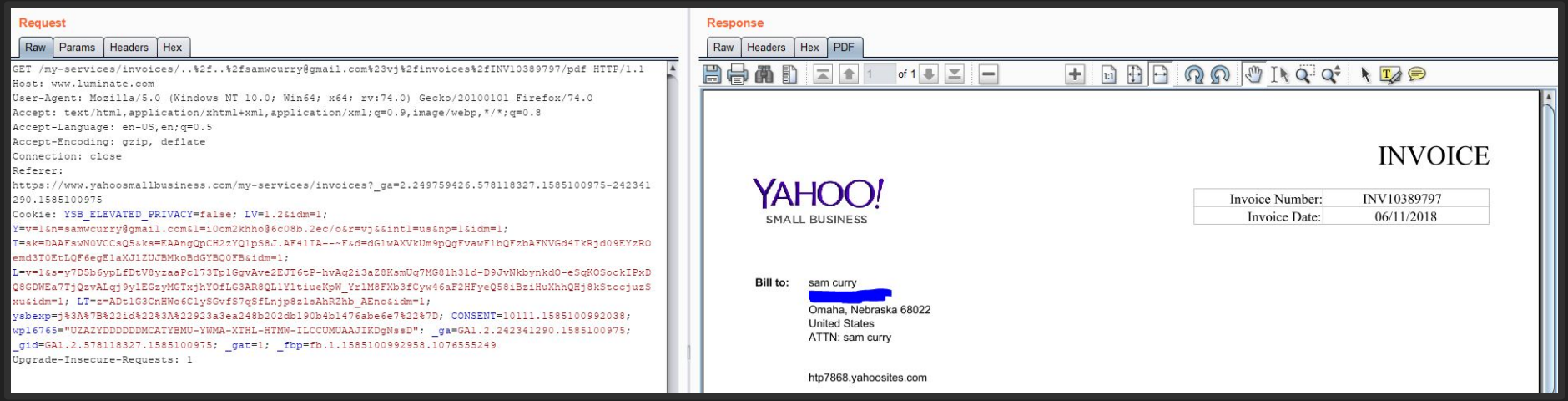
- Error message on another part of the app discloses the following…

```
{"error":"Id samwcurry@gmail.com#vj does not have permission to modify the domain example.com."}
```

- Moment of truth...

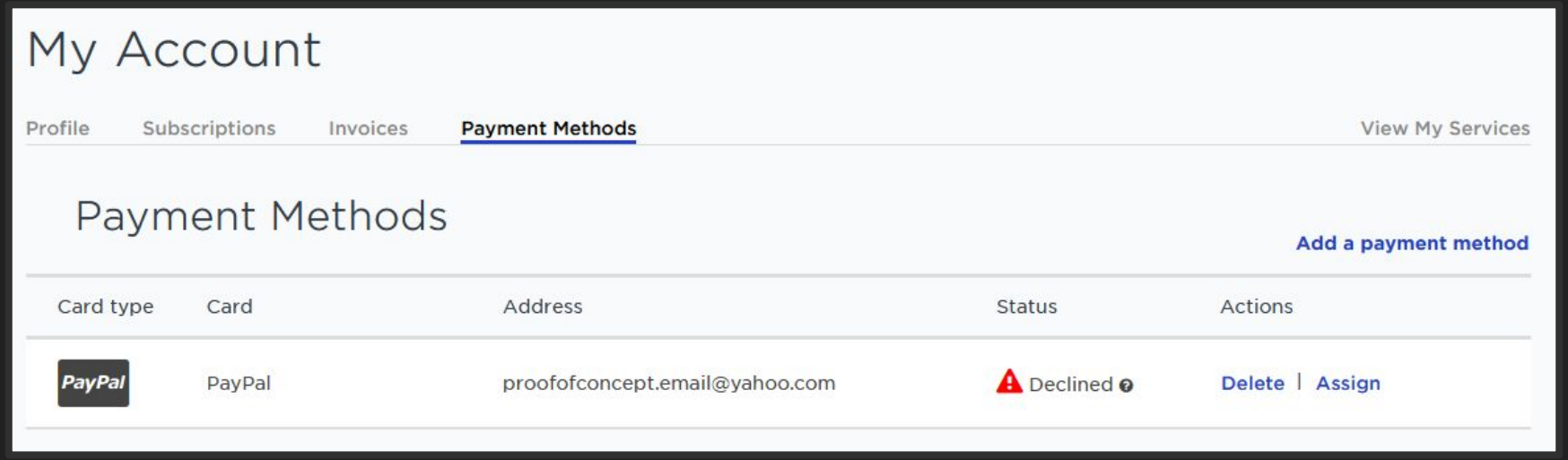
```
GET /my-services/invoices/..%2f..%2fsamwcurry@gmail.com%23vj%2finvoices%2fINV10389797/pdf HTTP/1.1
Host: www.luminate.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:74.0) Gecko/20100101 Firefox/74.0
```

Identifying application routing - Examples



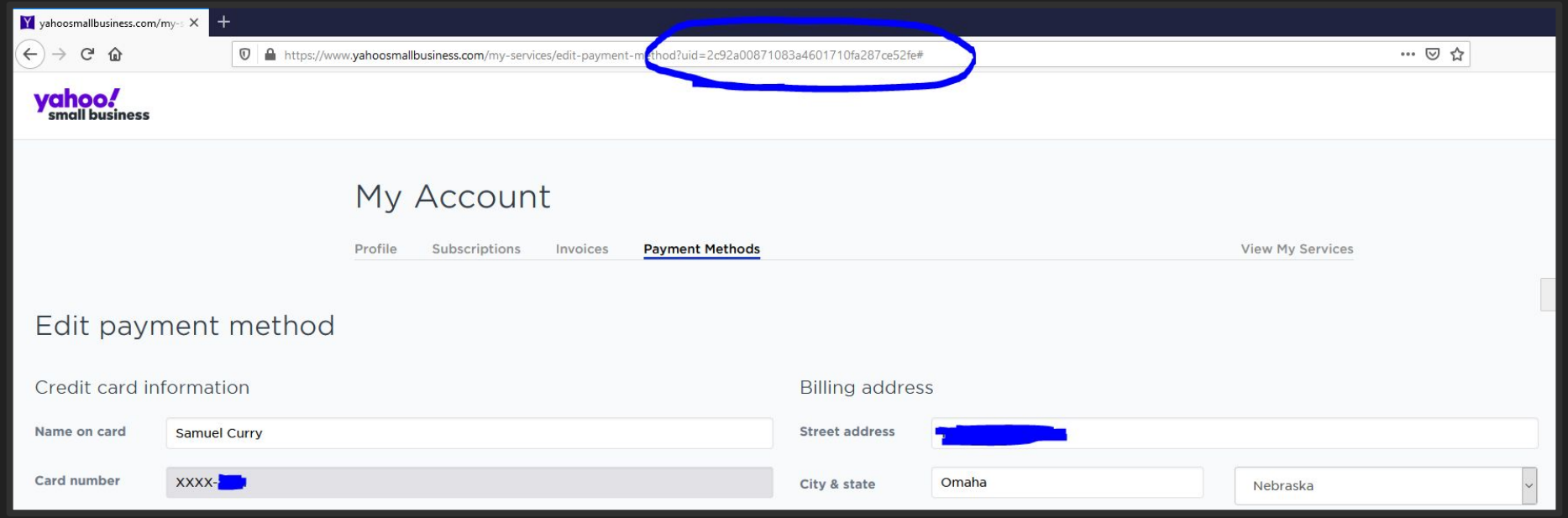
- Attacker can read anyones PDF if they know their…
  - Email address
  - Invoice number
- An alright bug… I guess....
- Is this behavior anywhere else on the app?

Identifying application routing - Examples



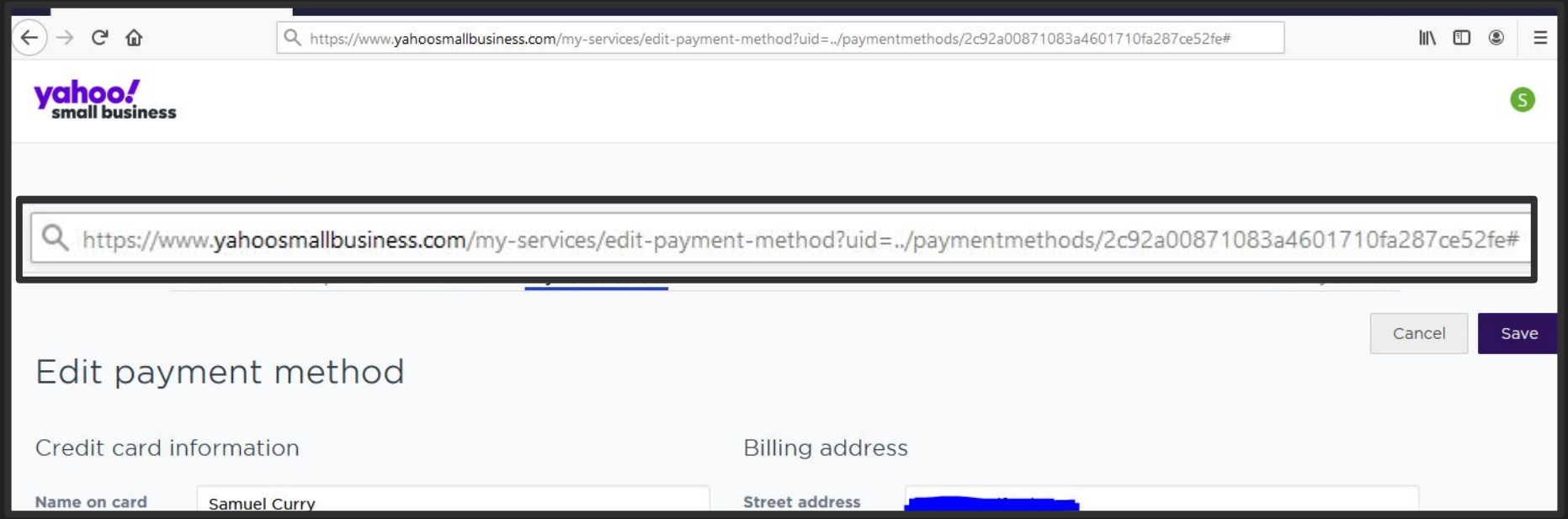
- Definitely a more interesting part of the website
- How is payment information fetched?

Identifying application routing - Examples



- Maybe this is stored the same way, but if so…
  - What is the directory name?
  - How can we retrieve that unique ID?

Identifying application routing - Examples



- Maybe this is stored the same way, but if so…
  - ~~What is the directory name?~~ (/paymentmethods/)
  - How can we retrieve that unique ID?

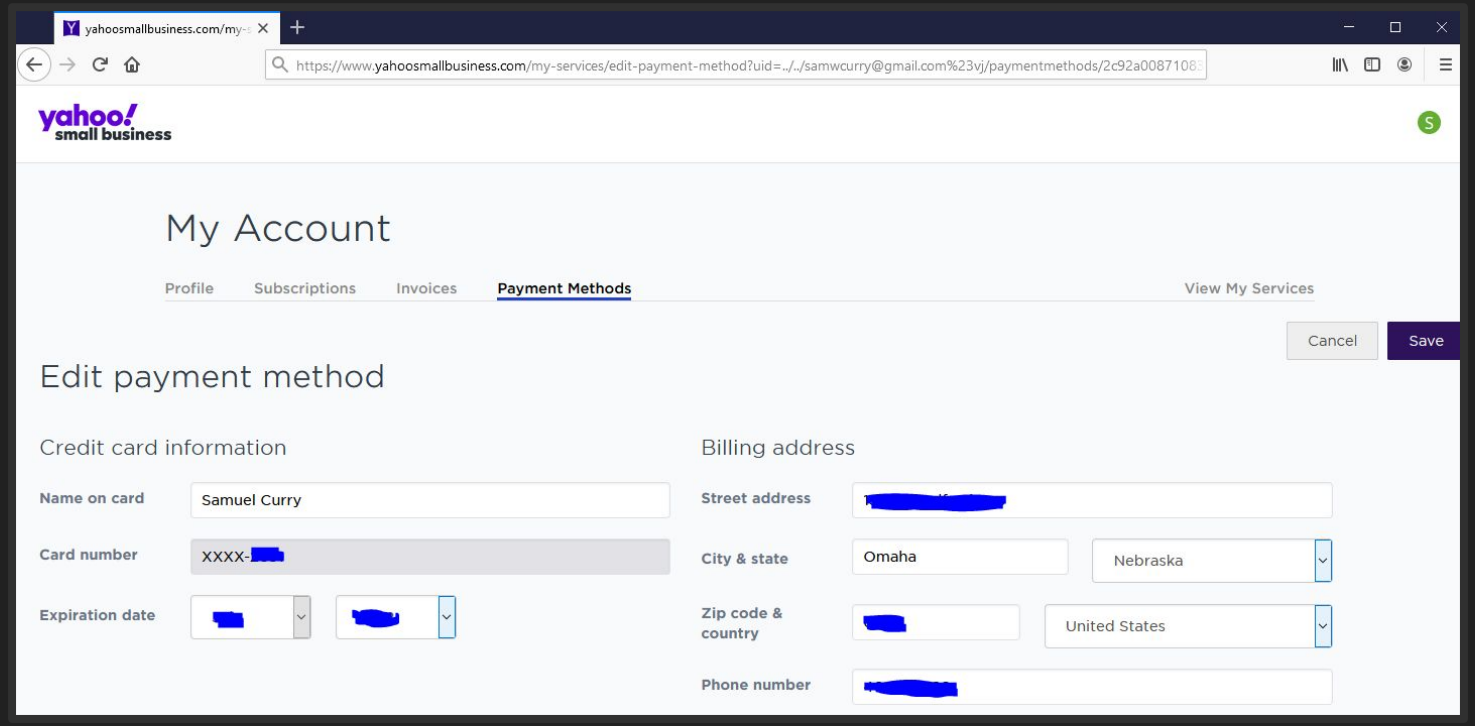
Identifying application routing - Examples

- GET /subscriptions/:id
  +
  Same trick from before
  =
  Traversing to view payment method IDs

https://www.luminate.com/subscriptions/..%2f..%2f + email + %2f + id

- Maybe this is stored the same way, but if so…
  - ~~What is the directory name?~~ (/paymentmethods/)
  - ~~How can we retrieve that unique ID?~~ (trick with /subscriptions/)

Identifying application routing - Examples



GET /my-services/edit-payment-method?uid=../../samwcurry@gmail.com%23vj/paymentmethods/2c92a00871083a4600fa287ce52fe

Identifying application routing - Examples

- Escalated severity from reading users invoices to reading payment information
- The only piece of information we need is the victim’s email address
  - The subscription ID can be brute forced
  - We obtain the payment ID from the subscription ID traversal

Exploring all possibilities

- Although directory traversal is useful for these types of bugs, it isn’t necessary for various attacks

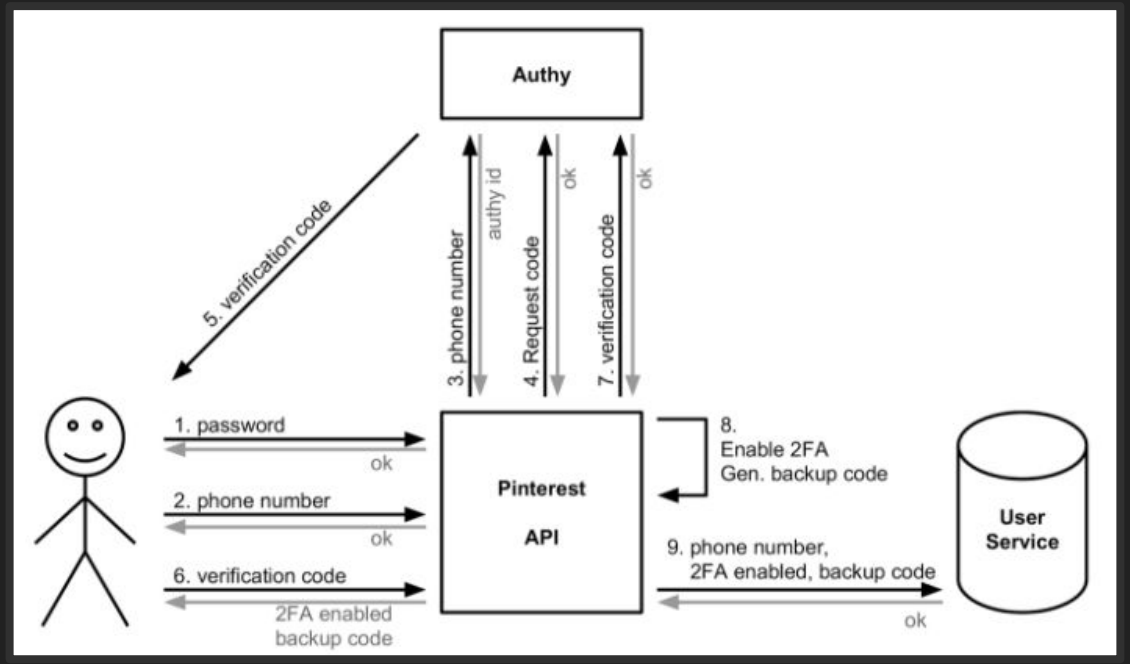
- In some cases, API calls behave similarly to a SQL query evaluating to true/false

Does https://internal.com/?code=1234 return 200?	Does SELECT * FROM `x` WHERE `id`=1234 return "True"?

- Impact of course varies per case, but there are lots of interesting possibilities

Case Study - Authy 2FA bypass

- Authy - 2FA service, installable library
- User -> [Client -> Authy]



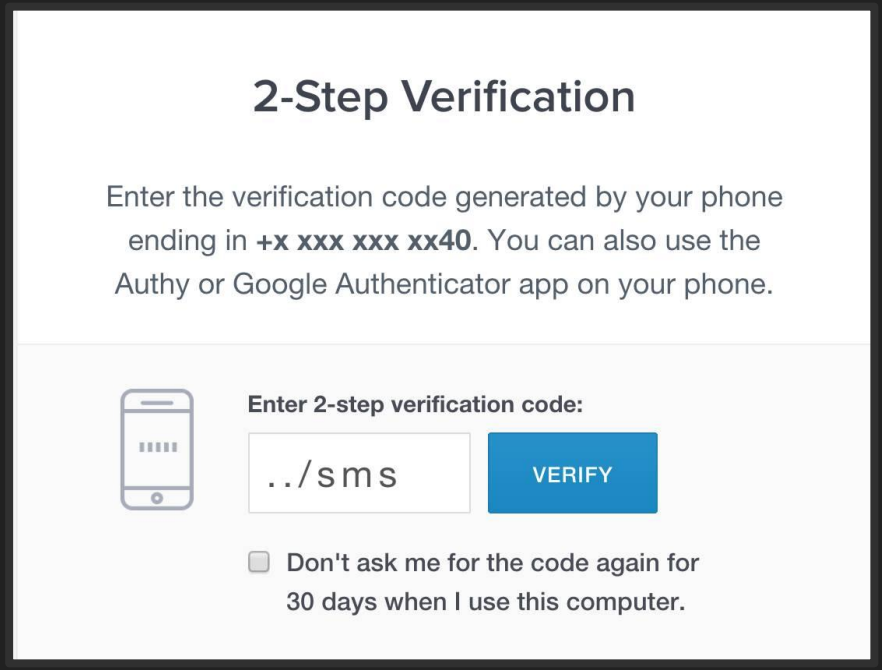
Case Study - Authy 2FA bypass

- When reading the response from Authy, the server only checked for…
  - JSON {“success”:true}
  - HTTP 200 OK
- How is the users token sent to Authy?

```
this._request("get", "/protected/json/verify/" + token + "/" + id, {}, callback, qs);
```

- GET /protected/json returns both 200 OK and JSON {“success”:true}
  - Is it really that simple?

Case Study - Authy 2FA bypass



Universal 2FA bypass for huge portion of Authy libraries
(credit: Egor Homakov, @homakov)

Review

- Lots of unique opportunities in attacking secondary contexts
  - Requests often sent internally
  - Often less restrictive environments
  - Authorization sometimes seemingly arbitrary (200 v.s. 403 when you control route)

- Very complicated problem for developers
  - Requests sent between servers with different behaviors
  - Hard to isolate internal APIs where user data isn't dangerous
  - Sanitizing for paths is relatively difficult 2-3 proxies deep

- Lots of new research relative to similar approaches
  - Using "Max-Forwards" header to figure out more information about your requests (https://www.agarri.fr/blog/archives/2011/11/12/traceroute-like_http_scanner/index.html)

Thank you Kernelcon!

- Questions? Maybe answers?

Sam Curry
@samwcyo