Who, What, Where, When, Wordlist

@TomNomNom

What's a wordlist?

- It's… A list of words.
- Usually used for brute-forcing* *something*

*Yes I am using this term incorrectly (:

Why are they useful?

- We *could* check all the combinations
- That would take a *really* long time
  - >4 years to cover [a-zA-Z_-]{6} if you can somehow maintain 500 requests per second
- Wordlists exist to save time and resources :)

Yes I really am explaining what wordlists are.

Where can they be used?

- Subdomain enumeration
- Path guessing
- Authentication guessing
- API method guessing
- Parameter guessing
- Header guessing
- ...you get the point.

Don't worry, it's going to get better soon...

Pre-baked lists

- https://github.com/danielmiessler/SecLists
  - Well organised
  - Well-known for a reason :)
- https://github.com/google/fuzzing/tree/master/dictionaries
  - Google's fuzzing dictionaries
  - Need a bit of prep if they're not in the format you need
- Others:
  - https://crackstation.net/crackstation-wordlist-password-cracking-dictionary.htm
  - /usr/share/dict/words (:
- There's more scattered all over the internet…

You know how to use Google, right?

So what's the problem?

- These wordlists are awesome
- They are genuinely useful
- But:
    - Everyone uses them
    - They're generally generic
    - They're often very large* (we're trying to save time and resources, remember?)

*Like me.

Custom lists

- You need custom wordlists if you want to find things others don't
- Every target has things specific to them
  - Generic wordlists probably won't contain *TargetName_SuperHappyAPIDocs*
- Specific use-cases call for specific wordlists
  - No point using a subdomain list when you're guessing headers*

*Well, probably anyway.

Manually curated

- My most used word-list was hand-writt^wtyped
- It's not very big
- But it's very useful
- I add to it when I read about something interesting

I'm sure you can figure this one out

Target-specific lists

- You will need:
  - A need
  - A source of data
  - A way to process the data
  - A way to use the wordlist
- Let's look at path-guessing as an example

No sticky-backed plastic required.

Getting path data

- We want lots of URLs for the target
- Your Burp Suite history is a good source
  - Target tab > Site map -> Right click a host -> 'Copy URLs in this host'
- Google dorks are handy but can be a bit tedious - I have a trick for that ;)
- gau (https://github.com/lc/gau) to get URLs from the Wayback Machine etc
- Sitemaps
- Remember to include paths from all of your target's domains!
  - Sometimes a boring path on one host is an interesting path on another :)

Running out of snarky comments, sorry.

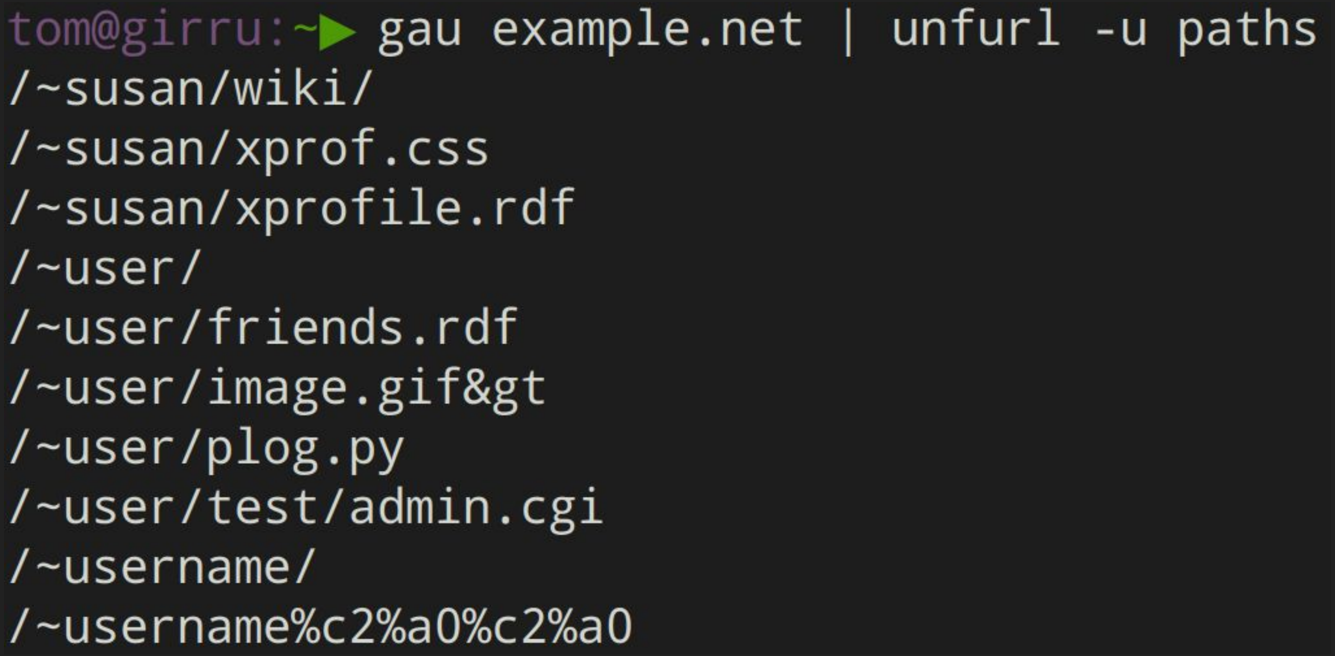
Google dorking

- Demo time :)

Hope it goes OK!

Processing path data

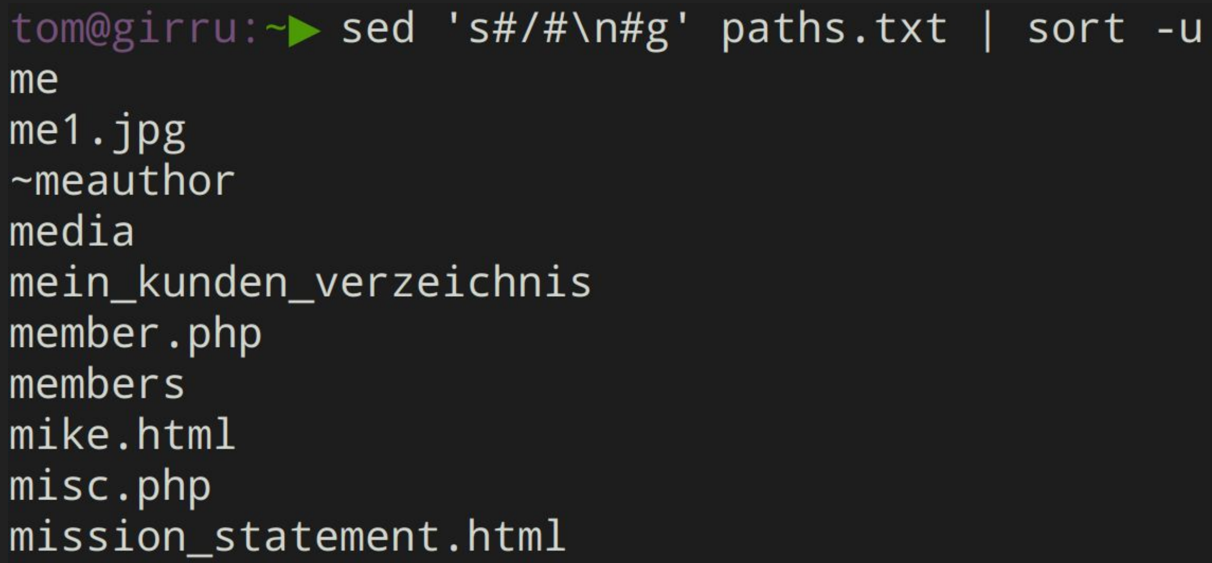
- I like unfurl (: (https://github.com/tomnomnom/unfurl)



```
tom@girru:~▶ gau example.net | unfurl -u paths
/~susan/wiki/
/~susan/xprof.css
/~susan/xprofile.rdf
/~user/
/~user/friends.rdf
/~user/image.gif&gt
/~user/plog.py
/~user/test/admin.cgi
/~username/
/~username%c2%a0%c2%a0
```

Extract all the parts

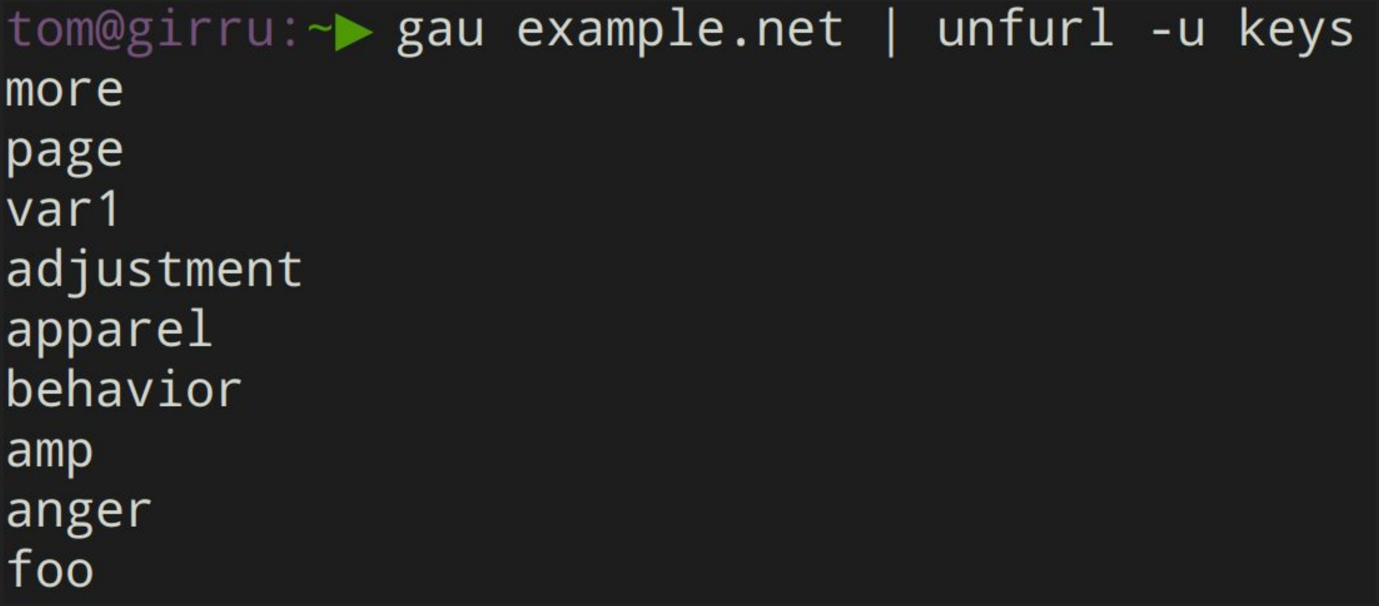
- Using complete paths can be fast, but it will miss things
- Pull the paths apart for greatest effect
  - Then re-combine them recursively :)



```
tom@girru:~▶ sed 's#/#\n#g' paths.txt | sort -u
me
me1.jpg
~meauthor
media
mein_kunden_verzeichnis
member.php
members
mike.html
misc.php
mission_statement.html
```

While we're here...

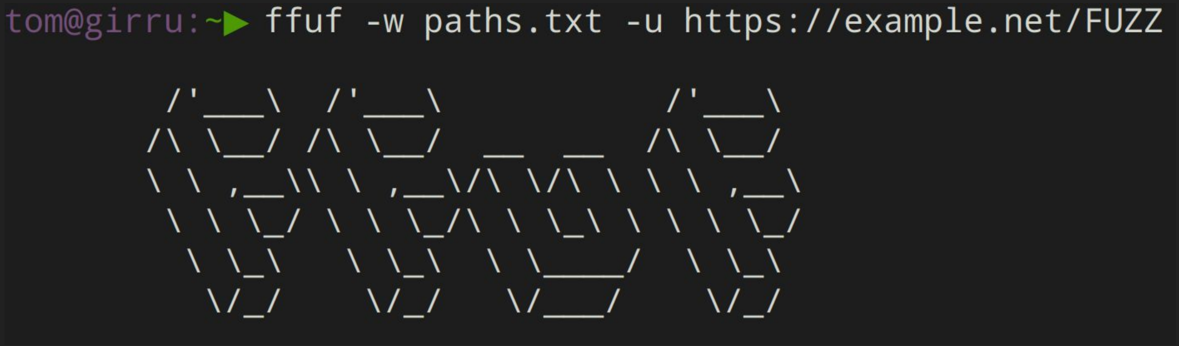
- Unfurl can extract lots of other things for you too
  - Query string keys and values, domains, fragments, ports...



```
tom@girru:~▶ gau example.net | unfurl -u keys
more
page
var1
adjustment
apparel
behavior
amp
anger
foo
```

Using the list

- We have a *lot* of options here
  - And you probably already have a favorite
- ffuf is good (https://github.com/ffuf/ffuf)
- Burp Suite's intruder (and turbo intruder) is good
- I have a few tools of my own too
  - meg, concurl, fff

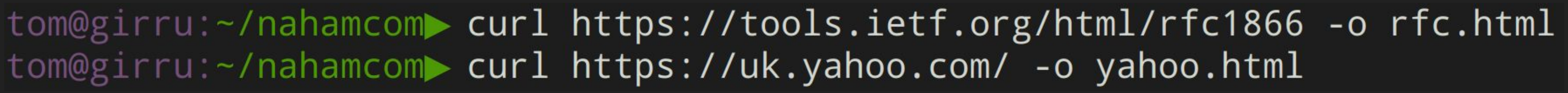


```
tom@girru:~▶ ffuf -w paths.txt -u https://example.net/FUZZ

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/
```

Finding words unique to a target

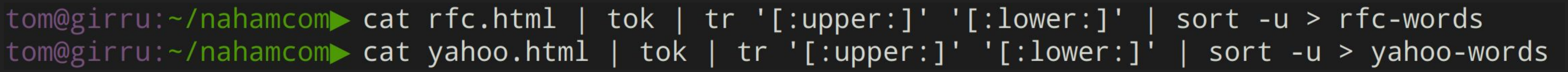
- It can be useful to have a list of (mostly) target-specific words
  - But how are you supposed to know which ones they are?
  - You can't, but you can remove some of the ones that probably aren't!
- The HTML RFC is a handy reference :D



```
tom@girru:~/nahamcom▶ curl https://tools.ietf.org/html/rfc1866 -o rfc.html
tom@girru:~/nahamcom▶ curl https://uk.yahoo.com/ -o yahoo.html
```

Tokenizing

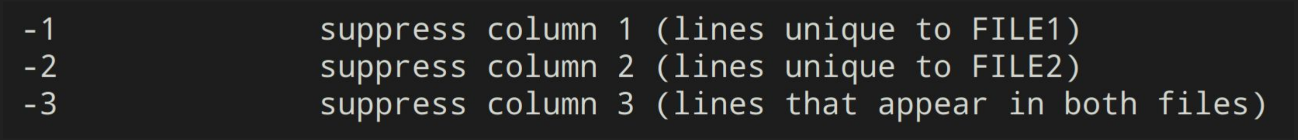
- I have a silly little alpha-quality tool called tok for doing this
  - https://github.com/tomnomnom/hacks/tree/master/tok
  - You can use grep or whatever you want instead :)
- Make sure both lists are sorted and normalised



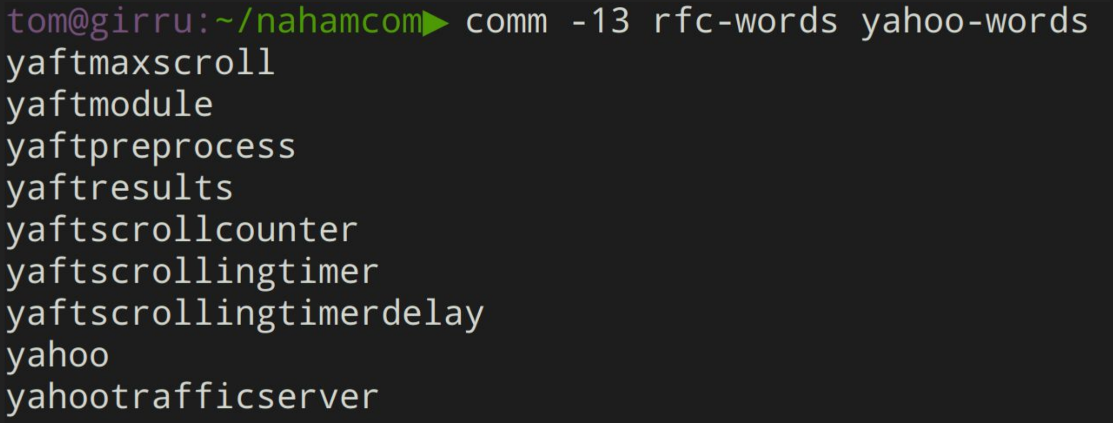
```
tom@girru:~/nahamcom▶ cat rfc.html | tok | tr '[:upper:]' '[:lower:]' | sort -u > rfc-words
tom@girru:~/nahamcom▶ cat yahoo.html | tok | tr '[:upper:]' '[:lower:]' | sort -u > yahoo-words
```

Commparing

- The comm tool compares sorted files
  - The options are a little confusing though
- A little manual curation to remove junk can be useful :)



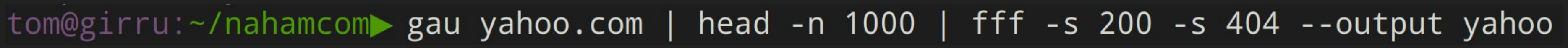
```
-1              suppress column 1 (lines unique to FILE1)
-2              suppress column 2 (lines unique to FILE2)
-3              suppress column 3 (lines that appear in both files)
```



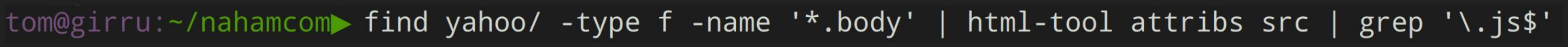
```
tom@girru:~/nahamcom▶ comm -13 rfc-words yahoo-words
yaftmaxscroll
yaftmodule
yaftpreprocess
yaftresults
yaftscrollcounter
yaftscrollingtimer
yaftscrollingtimerdelay
yahoo
yahootrafficserver
```

Fetch all the things

- I've found it useful to fetch lots of URLs for analysis
- You can extract parts of the responses to generate lists
- JavaScript files are especially helpful
  - You can use html-tool (https://github.com/tomnomnom/hacks/tree/master/html-tool) to extract script src values
  - Then fetch the JS files and tokenize them :)
  - Super handy for path guessing, and guessing API methods too



```
tom@girru:~/nahamcom▶ gau yahoo.com | head -n 1000 | fff -s 200 -s 404 --output yahoo
```



```
tom@girru:~/nahamcom▶ find yahoo/ -type f -name '*.body' | html-tool attribs src | grep '\.js$'
```

Custom but generic

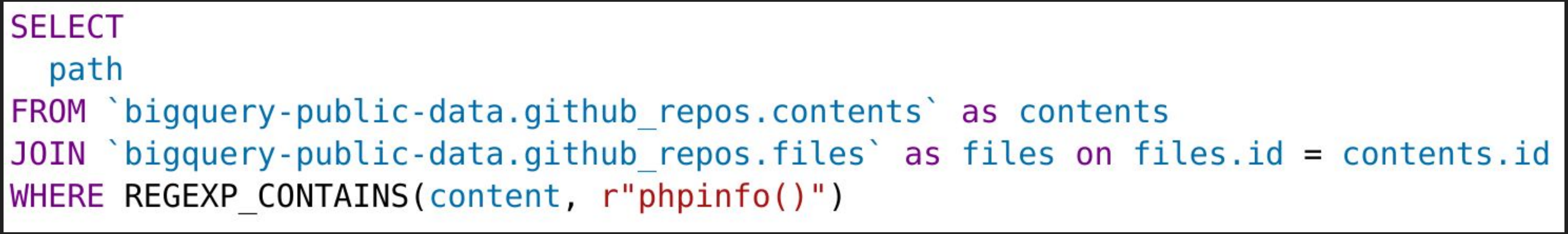
- Target-specific lists can be very useful
  - But there's lots they don't find
- Lots of people like to hunt on multiple targets at once
- It'd still be nice to have good wordlists
  - Especially if they include something that public lists do not
- Sometimes you want a list for a specific vulnerability or piece of software

Writing Big Queries

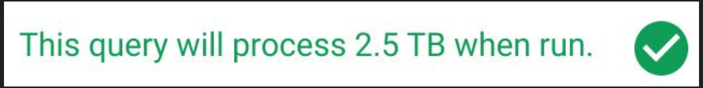
- WARNING: doing some of the things in the next few slides can cost you not-insignificant amounts of money
  - BigQuery does have some free usage included, but it's easy to go over that
- Google's BigQuery has some useful datasets
  - `bigquery-public-data.github_repos` is a favourite of mine
  - There's an httparchive dataset too but it might cost you > $30 *per query* depending on what you use it for

Finding paths

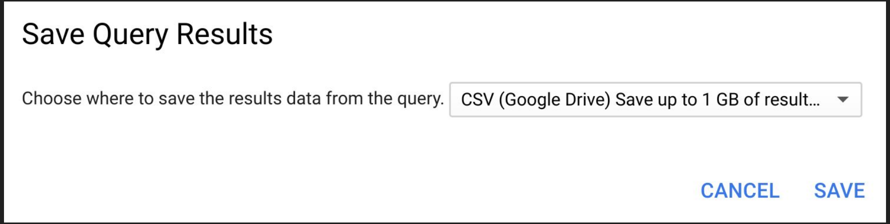
- You can match file contents to get a list of paths directly
  - This one is a bit expensive because it reads file contents
- Use the "Save to Google Drive" option to get up to 1GB of results



```
SELECT
  path
FROM `bigquery-public-data.github_repos.contents` as contents
JOIN `bigquery-public-data.github_repos.files` as files on files.id = contents.id
WHERE REGEXP_CONTAINS(content, r"phpinfo()")
```



This query will process 2.5 TB when run.

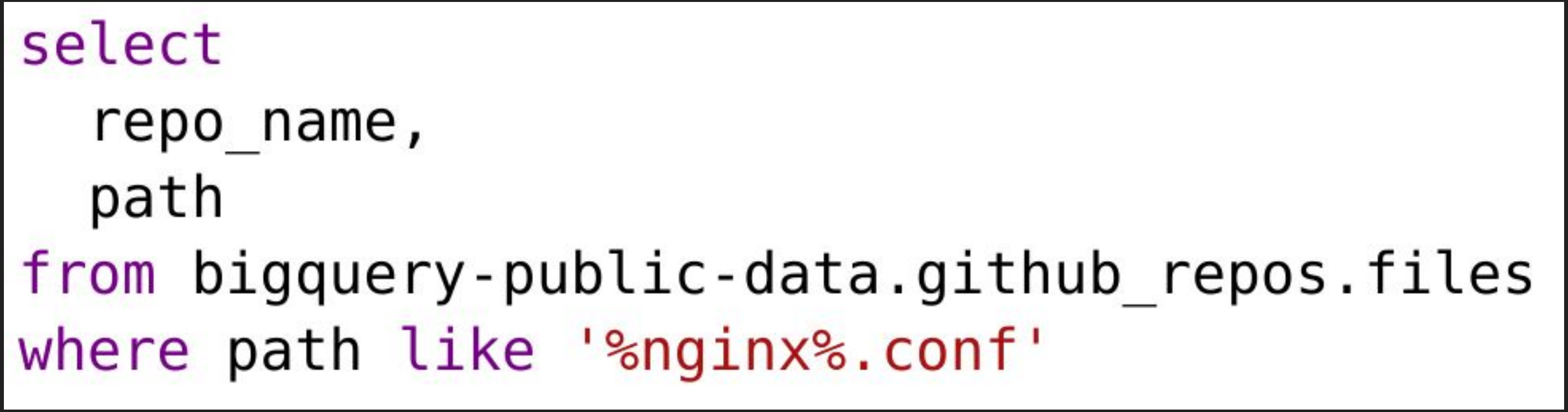


Save Query Results

Choose where to save the results data from the query. CSV (Google Drive) Save up to 1 GB of result...

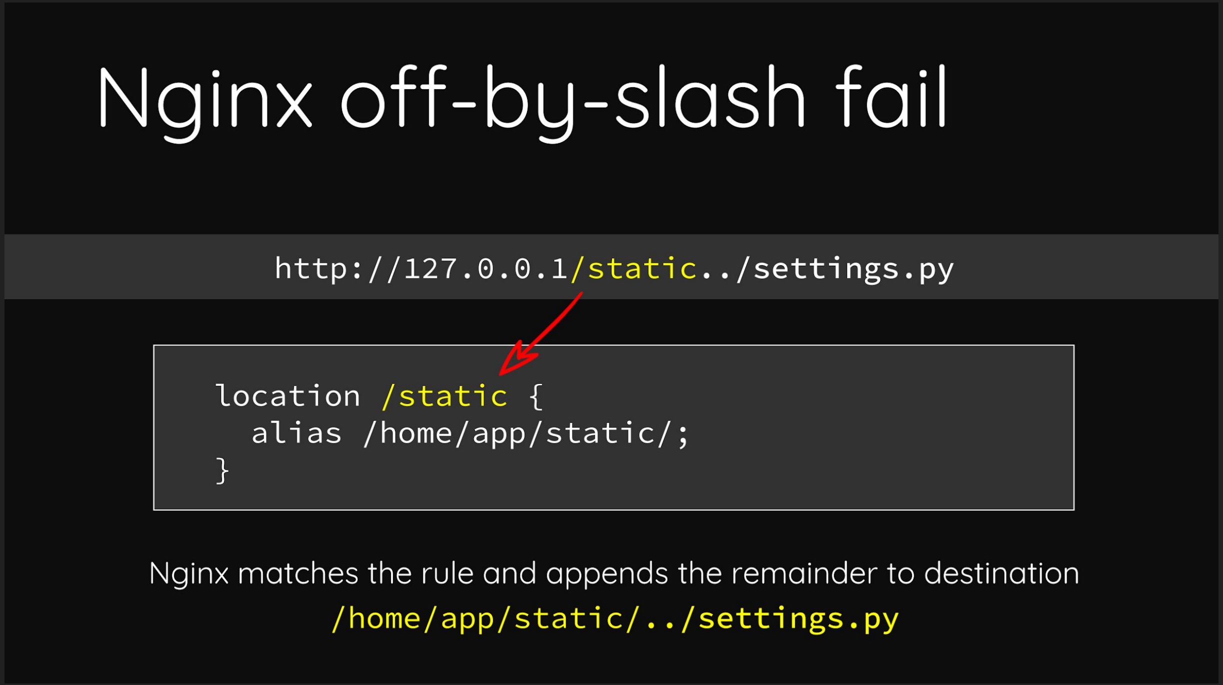
Using paths to find things instead

- The files table is smaller than the contents table so let's stick to that one
- Let's build a wordlist for the infamous nginx "off-by-slash" issue
- We need both the repo name and the path to the file:



```sql
select
  repo_name,
  path
from bigquery-public-data.github_repos.files
where path like '%nginx%.conf'
```

The bug



Nginx off-by-slash fail

```
http://127.0.0.1/static../settings.py
```

```
location /static {
  alias /home/app/static/;
}
```

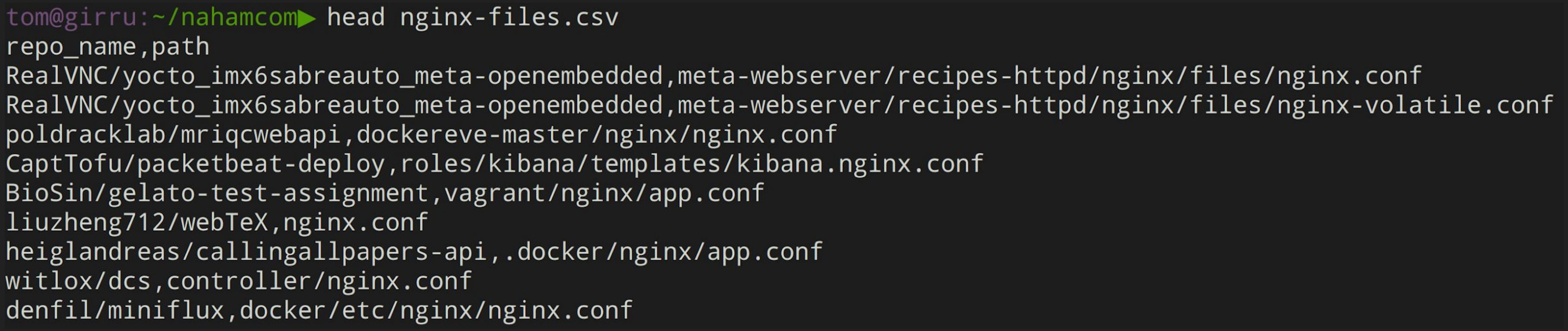
Nginx matches the rule and appends the remainder to destination

`/home/app/static/../settings.py`

https://i.blackhat.com/us-18/Wed-August-8/us-18-Orange-Tsai-Breaking-Parser-Logic-Take-Your-Path-Normalization-Off-And-Pop-0days-Out-2.pdf

The results

- The results file looks like this:

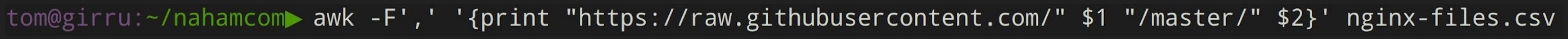


```
tom@girru:~/nahamcom▶ head nginx-files.csv
repo_name,path
RealVNC/yocto_imx6sabreauto_meta-openembedded,meta-webserver/recipes-httpd/nginx/files/nginx.conf
RealVNC/yocto_imx6sabreauto_meta-openembedded,meta-webserver/recipes-httpd/nginx/files/nginx-volatile.conf
poldracklab/mriqcwebapi,dockereve-master/nginx/nginx.conf
CaptTofu/packetbeat-deploy,roles/kibana/templates/kibana.nginx.conf
BioSin/gelato-test-assignment,vagrant/nginx/app.conf
liuzheng712/webTeX,nginx.conf
heiglandreas/callingallpapers-api,.docker/nginx/app.conf
witlox/dcs,controller/nginx.conf
denfil/miniflux,docker/etc/nginx/nginx.conf
```

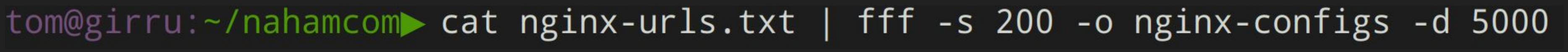
What are we supposed to do with that?!

<3 GitHub

- You can fetch files straight from raw.githubusercontent.com :)
  - Please use rate-limiting!*



```
tom@girru:~/nahamcom▶ awk -F',' '{print "https://raw.githubusercontent.com/" $1 "/master/" $2}' nginx-files.csv
```

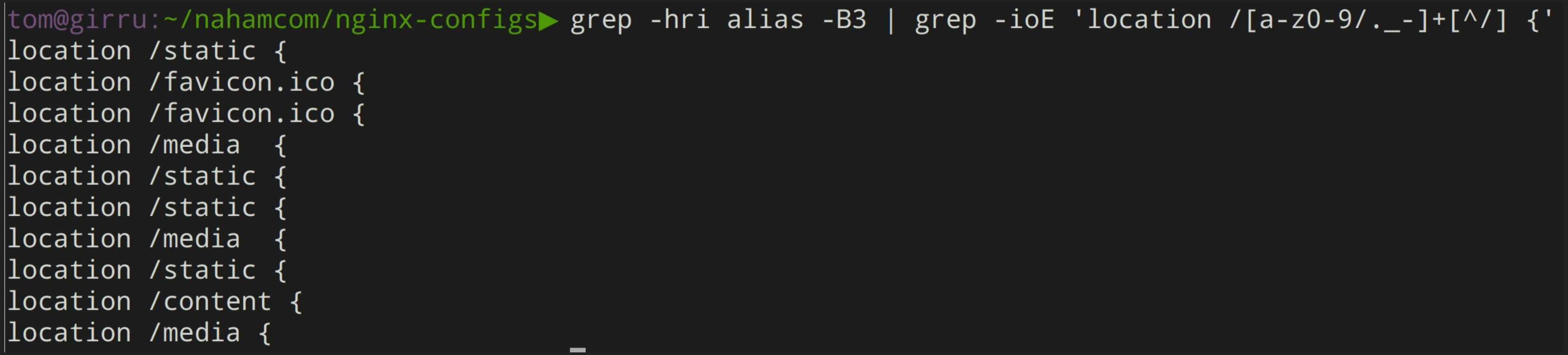


```
tom@girru:~/nahamcom▶ cat nginx-urls.txt | fff -s 200 -o nginx-configs -d 5000
```

*Please don't hate me, GitHub UwU

Processing the files

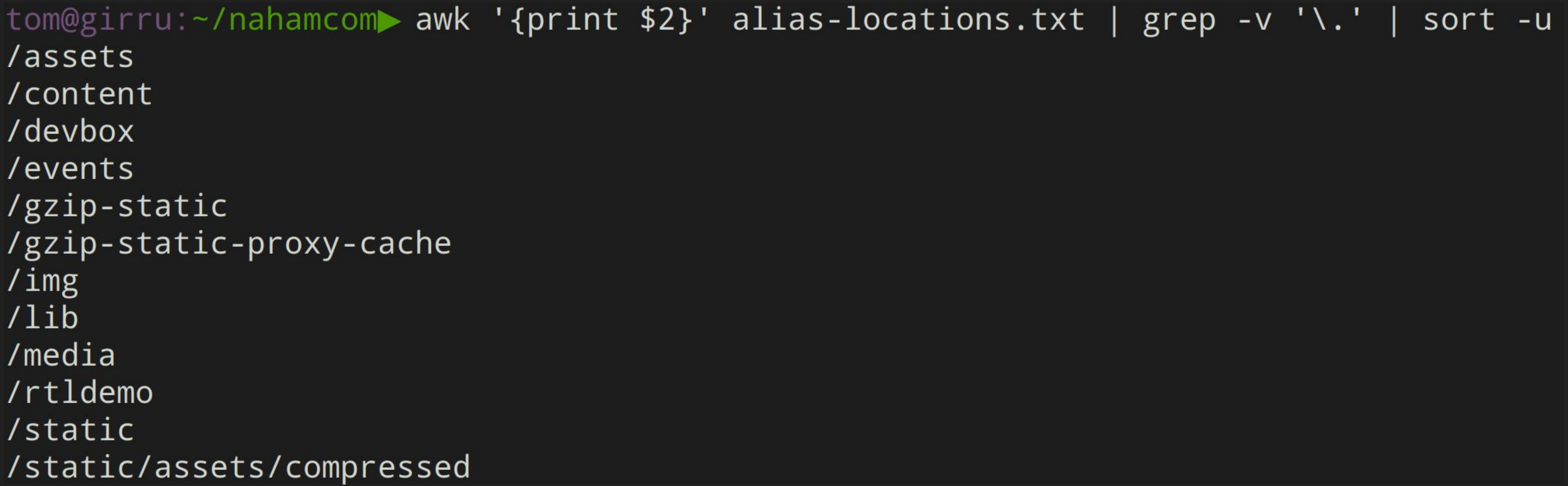
- If you were going to do this properly you'd want to parse the config files
- I'm going to hack it with grep and awk (:
  - For my purposes it doesn't have to be perfect, only *good enough*



```
tom@girru:~/nahamcom/nginx-configs▶ grep -hri alias -B3 | grep -ioE 'location /[a-z0-9/._-]+[^/] {'
location /static {
location /favicon.ico {
location /favicon.ico {
location /media  {
location /static {
location /static {
location /media  {
location /static {
location /content {
location /media {
```

Data cleaning

- This approach can leave you with messy data
- It's a good idea to clean it
  - Simple shell tools like grep, awk, and sed are super handy
  - Don't be afraid to do stuff manually in your editor though!



```
tom@girru:~/nahamcom▶ awk '{print $2}' alias-locations.txt | grep -v '\.' | sort -u
/assets
/content
/devbox
/events
/gzip-static
/gzip-static-proxy-cache
/img
/lib
/media
/rtldemo
/static
/static/assets/compressed
```

More files

- This technique extends to many other files with common names
  - robots.txt
  - .gitignore
  - Makefile
  - Rakefile
  - Fakefile

That last one is fake.

Thank you!

- Remember:
  - Keep a manually-curated list for things you read about or happen upon manually
  - Target-specific word lists will help you discover things that generic wordlists cannot
  - Google dorking is fun
  - Set billing limits in BigQuery!
- And most of all:
  - Stay safe and have fun :)