



The Bug Hunter's Methodology



Let's talk about goals...

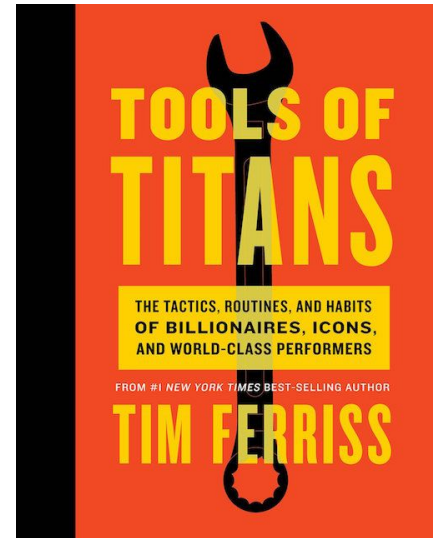
- Goal of this talk is to:
 - Outline and provide an actionable methodology for effectively and efficiently testing for, and finding security vulnerabilities in web applications
 - You probably already do a lot of these things...
 - Cover common vuln classes/types/categories from a high level
 - Provide useful tools and processes that you can take right out into the world to immediately improve your own bug hunting abilities
 - Hopefully everyone can learn at least one new thing from this...
 - Profit

\$whoami

- Hi, my name is Grant
- Manager of Solutions Architecture at Bugcrowd
 - Previously an Application Security Engineer
- Been around bug bounties a lot
 - In doing so, seen a lot of bugs, and the ways people find them
 - ASE, bug hunter, setting up/managing programs
- Music on the side

About this workshop

- Built off of Jason Haddix's *How To Shot Web* talk given at DefCon 23
- Further added to by Jason Haddix's *Bug Hunter's Methodology V2* talk at Bugcrowd's LevelUp 2017 online-conference
- I'll be adding a few things of my own...
- Consider this sort of a *Tools of Titans* for bug hunting
 - Incredible tools built by the community
 - Standing on the shoulders of giants
 - As best I can, I'll give appropriate credit



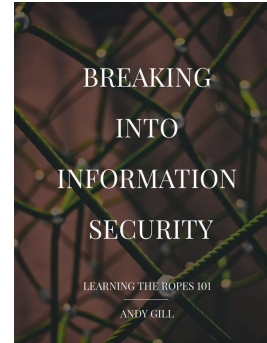
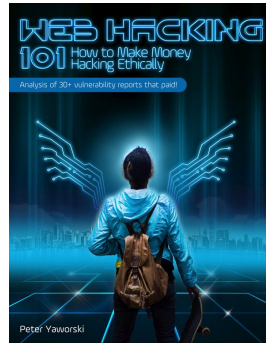
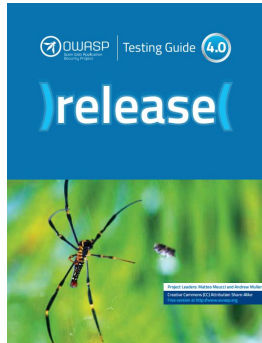
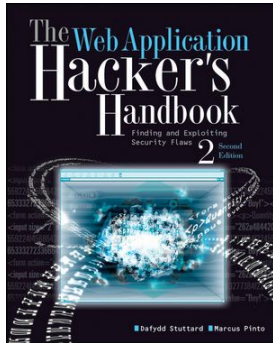
One step back... bug bounties?

- Bug bounties aren't new, but they are growing
 - Netscape; growing into new spaces
 - #equifax
 - There's an increasing incentive to find issues in the wild before they're exploited
- Part of a larger space called "crowdsourced security"
- Advantages to bug bounties AKA crowdsourced security
 - Competition #firstToFind
 - Pay based on findings, and impact
 - Incentivized to find innovative, unique issues
- But really, it's as simple as...
 - $2 > 1$; $500 > 2$; ...
 - More eyes tend to equal more results
 - Hone skills; have fun; hiring; #cash



Other...

- Common guides/methodologies
 - AKA suggested “light” reading...
 - Once again, not just limited to bounties/responsible disclosure



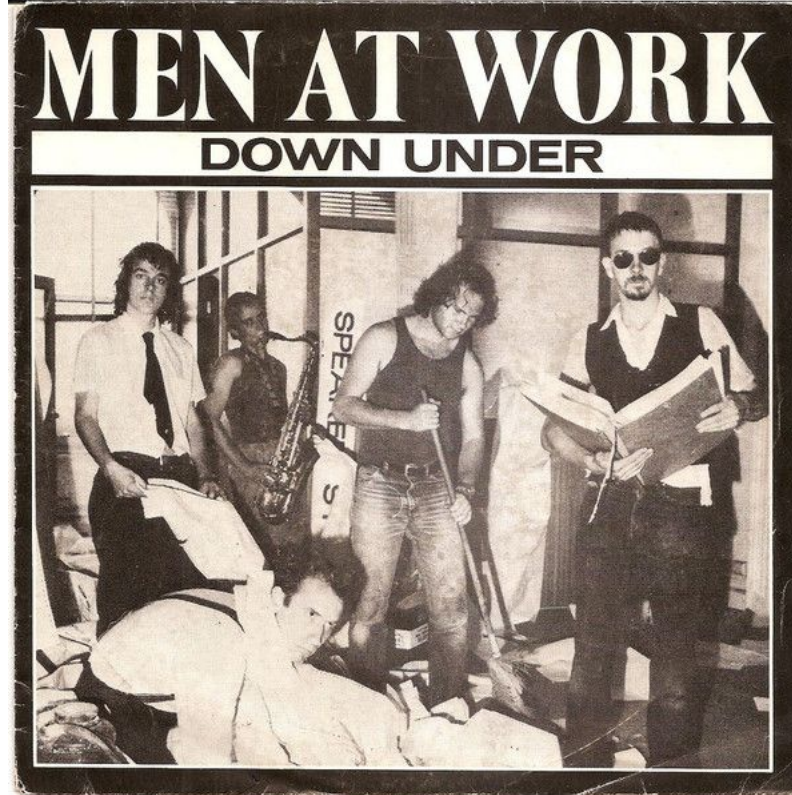
Our Journey (we've got a long road ahead...)

- Discovery
- Mapping
- Auth/Session
- XSS
- SQLi
- File upload/AFI/LFI
- CSRF
- IDOR
- SSRF
- And much, much more!!!



Let's get down to business

#nextSlide



Discovery

- Q: what's the value in discovery?
 - A: a lot.
 - Specifically, because we want to find things that are less tested than flagship site (fresher attack surface = easier to find issues = more critical issues = more profit)
- Scope
 - Today we're going to assume open scope...
 - Always check scope
- Where do we start?
 - *.tesla.com -- tons of amazing tools to help us here

Targets

*.tesla.com

A hardware product that you own or are authorized to test against (Vehicle/PowerWall/etc.)

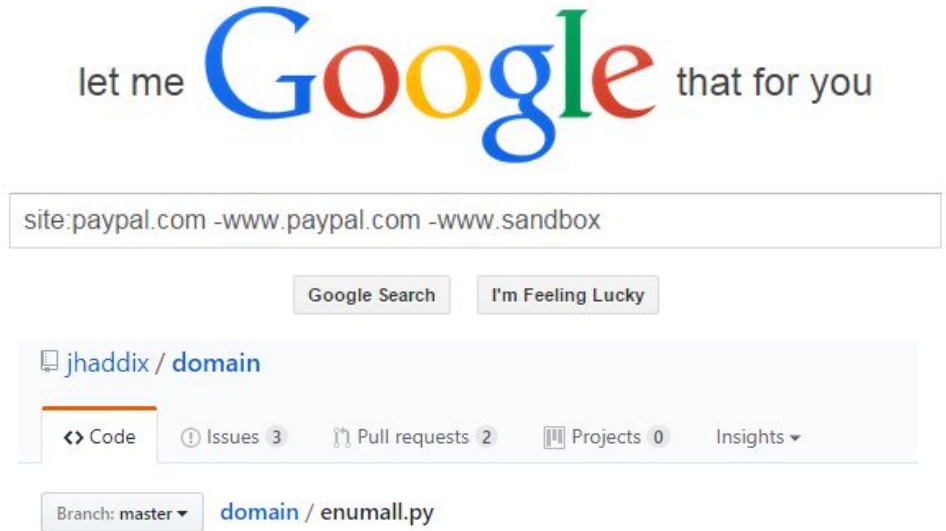
Mobile Applications ("Tesla Model S" on iOS and Android)

Any host verified to be owned by Tesla Motors Inc. (domains/IP space/etc.)

*.teslamotors.com

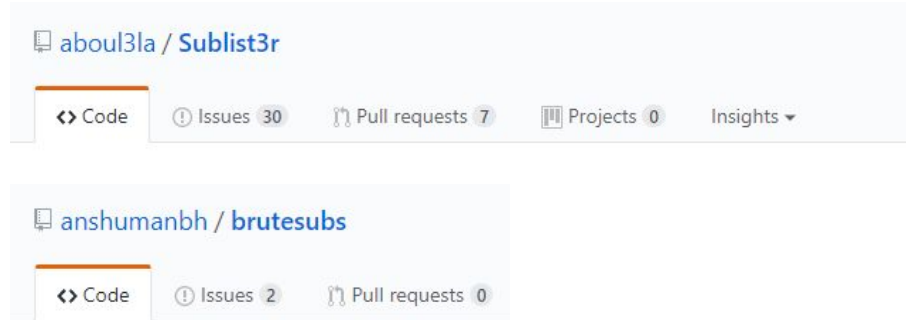
Using search engines...

- Use search engines to do the work for you...
 - (automated via recon-ng, etc)
- Even better: enumall.py
 - Wrapper around recon-ng
 - google/bing/baidu/netcraft
 - Also does brute force
 - More on this later!



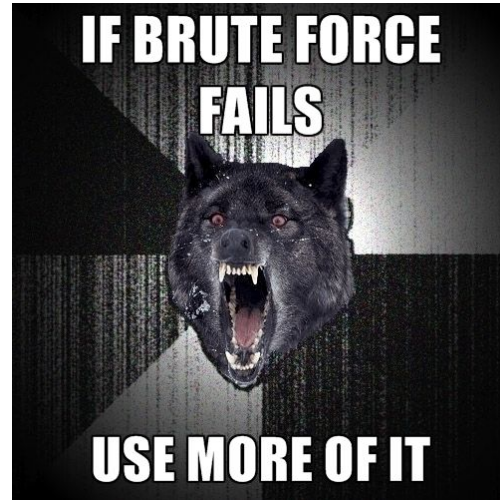
But wait, there's more!

- Sublist3r
 - Another subdomain finder/scrapper
 - Each tool has its own merits (hits some different search engines)
- Why not both?
 - Enter brutesubs by Anshuman
 - Sublist3r + enumall.py
 - + altdns
 - Some configuration required



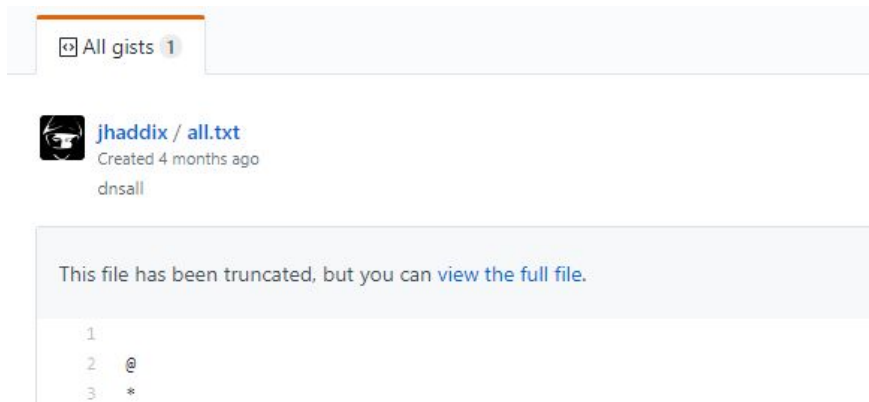
On subdomain bruteforcing

- A comparison of common subdomain brute forcing tools - using a 1,136,964 line subdomain dictionary
 - subbrute: errored out
 - gobuster: 21m15s; found 87
 - massdns: **1m24s; found 213**
 - dns-parallel-prober: 42m2s; found 43
 - blacksheepwall: 256m9s; found 61
 - Credit: @jhaddix
- Takeaway:
 - Massdns is pretty quick
 - Best tool for quickly getting a list of subdomains via brute force
 - Distributes resolvers (not all are reliable though)



That one million line file...

- all.txt
 - Created by @jhaddix
 - A marriage of virtually every subdomain list ever (bitquark's research, deep magic, etc)
 - THE definitive subdomain list
 - gist.github.com/jhaddix



Now that we have a ton of subdomains...

How do we work through them quickly?

- Screenshots!
- Quickly see what's on each host
 - Eyewitness
 - HTTPScreenshot
 - Aquatone
 - Can also do discovery
 - Relatively new
 - Only supported in Kali



Let's not forget about

- Mergers and acquisitions
 - (depending on scope; always check the scope -- ex: Google or Tesla)
 - Check:
 - Wikipedia
 - Crunchbase, etc
 - And iterate from there...
- ASNs
 - (again, scope)
- Reading up on disclosures
 - May be able to find these same issues on other parts of the app
 - See trends and ideas for what other people have found

[AS394161 Tesla Motors, Inc.](#)

[Home](#)
[Report](#)
[Export](#)
[Export](#)

AS Info

Graph v4

Prefixes v4

Peers v4

Whois

IRR

Prefix	Description
209.133.79.0/24	Tesla Motors, Inc.

Another discovery tool

- Intrigue.io
 - OSINT framework; simple to integrate
 - DNS subdomain bruteforce
 - Web spider
 - nmap
 - API; over 45 built in tasks



Don't forget port scanning

- Why?
 - Ex1: facebook had an open jenkins script console with no auth
 - Ex2: exposed tomcat or coldfusion admin panels with default creds
 - And so on...
- But what if there's A LOT to scan... e.g. 65k hosts?
 - Masscan
 - 11m4s to scan 65k hosts for top 1000 ports

Github

- Search Github for usernames/passwords/keys that developers might have left up.
 - Happens far more often than you'd think...

Mapping

- Directory brute forcing
 - All about the lists -- and GoBuster
 - RAFT lists (included in seclists brute force)
 - Robots disallowed ^
 - SVN Digger ^
 - Git Digger



On brute forcing directories...

After bruteforcing look for other status codes indicating you are denied or require auth then append list there to test for misconfigured access control.

Example:

GET http://www.acme.com - 200

GET http://www.acme.com/backlog/ - 404

GET http://www.acme.com/controlpanel/ - 401 hmm.. ok

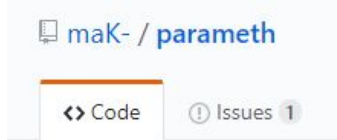
GET http://www.acme.com/controlpanel/[bruteforce past here now]

What about brute forcing parameters?

#whynot

- parameth

- tool with some heuristics to
- help discover params



```
parameth/maK# ./parameth.py -u https://makthepla.net/parameth/simpletest.php

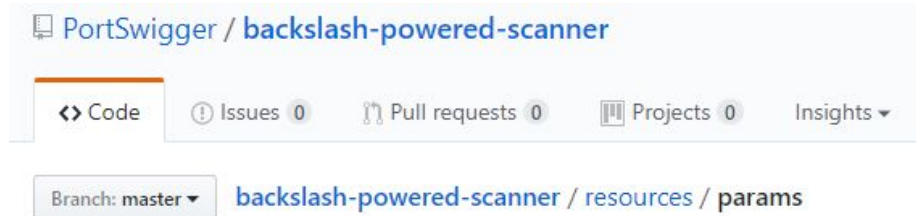
discobiscuits

parameth v1.0 - find parameters and craic rocks
Author: Ciaran McNally - https://makthepla.net

=====
Establishing base figures...
GET: content-length-> 22 status-> 200
POST: content-length-> 22 status-> 200
Scanning it like you own it...
GET(size): m | 22 ->36 ( https://makthepla.net/parameth/simpletest.php?m=discobiscuits )
POST(size): r | 22 ->42 ( https://makthepla.net/parameth/simpletest.php )
GET(status): redirect | 200->301 ( https://makthepla.net/parameth/simpletest.php?redirect=discobiscuits )
parameth/maK#
```

- backslash-powered-scanner by portswigger

- nice list of 2500 top alexa params



Mapping [2]

- Platform identification:
 - Waplyzer (chrome)
 - Extension; shows info on the tech stack for the site (based on headers, etc)
 - Builtwith (chrome)
 - ^
 - retire.js (command line or Burp)
 - Notes any outdated libraries
 - VulnScanner
 - github.com/vulnersCom/burp-vulners-scanner
- Auxiliary:
 - If you find they're using a CMS...
 - WPScan
 - CMSmap

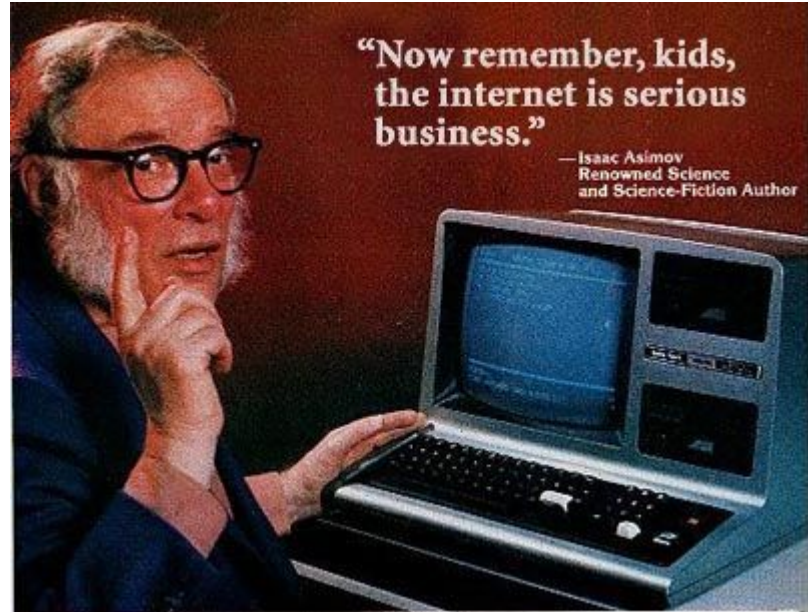
The image shows a screenshot of the Burp Suite application. On the left, the 'HTTP History' pane lists several requests, including 'http://localhost:8000' and 'https://safebrowsing-cache.google.com'. The right pane displays the details of a selected request to 'jquery-1.6.2.js'. Under the 'Advisory' tab, a warning icon is shown next to the text: 'The JavaScript file 'jquery-1.6.2.js' includes a vulnerable version of the library 'jquery''. Other tabs like 'Request' and 'Response' are visible but not selected.

The screenshot displays the Burp Suite interface with the following components:

- Left Panel (Request List):** A list of intercepted HTTP requests. The selected request is a GET to `https://vulnerable-xyz.com/` with status 200.
- Center Panel (Request Details):** A detailed view of the selected GET request. It shows the URL `https://vulnerable-xyz.com/`, method `GET`, status `200`, and various headers including `Server: Apache/2.4.18 (Ubuntu)` and `Content-Type: text/html`.
- Right Panel (Issues):** A list of detected vulnerabilities. The primary issue is "[Vulnrs] Vulnerable Software detected" for WordPress 4.7.5, which is marked as "Severe". Other issues include "Session token in URL" and "Password field with autocomplete enabled".

Remember...

- None of this, despite how great these tools are, replaces actually walking and understanding the app.



Vuln discovery

Still part of mapping

- For reasons we've touched on...
 - (could inform ideas and places to test)
- Resources:
 - xssed.com
 - [reddit xss /r/xss](https://reddit.com/r/xss)
 - [punkspider](https://punkspider.com)
 - xss.cx
 - xssposed.org
 - twitter searching (XSS + Tesla)

Auth and Session

- Being quick is important
 - Many are OOS (username enumeration, etc)
- Chaining...
 - Weak reset passwords (4 chars emailed) + login page brute force = complete account compromise
- Session/auth things to think about/look for include:
 - Session fixation
 - Insufficient session expiration
 - No password on account changes (e.g. password)
 - Most people will ask for current password, but not always on email change - which, if you can control the email, then you can do a regular password reset!
 - Not expiring reset tokens after email changed, etc

Tactical fuzzing...

#nextSlide



XSS

- Does the page display something to the users? Is it dynamic?
- Polyglot payloads...
 - Executes in multiple contexts, built-in filter evasion; blanket injection that can save time
 - You probably already use them...
- Some favorites:
 - #1(from Rsnake's XSS Cheat Sheet - now the OWASP XSS cheat sheet)
 - `'><SCRIPT>alert(String.fromCharCode(88,83,83))//';alert(String.fromCharCode(88,83,83))//";alert(String.fromCharCode(88,83,83))//";alert(String.fromCharCode(88,83,83))//--></SCRIPT>">'><SCRIPT>alert(String.fromCharCode(88,83,83)) </SCRIPT>`

More polyglots



- #2 (Ashar Javed XSS Research) (<http://slides.com/mscasherjaved/cross-site-scripting-my-love#/>)
 - ">><marquee></marquee>"
></plaintext>\></|\><plaintext/onmouseover=prompt(1)
><script>prompt(1)</script>@gmail.com<isindex formaction=javascript:alert(/XSS/) type=submit>'-->" ></script><script>alert(1)</script>"><img/id="confirm(1)"/alt="/"src="/"onerror=eval(id&%23x29;>">
- #3 (Mathias Karlsson) (<http://www.slideshare.net/MathiasKarlsson2/polyglot-payloads-in-practice-by-avlidienbrunn-at-hackpra>)
 - " onclick=alert(1)//<button ' onclick=alert(1)//> */ alert(1)//
- #4 0xsobky's "Ultimate XSS Polyglot" (<https://github.com/0xsobky/HackVault/wiki/Unleashing-an-Ultimate-XSS-Polyglot>)
 - jaVaScRipt:/*-/*`/*\`/*'/*"/**/(/* */oNcliCk=alert()
)//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert())//>\x3e

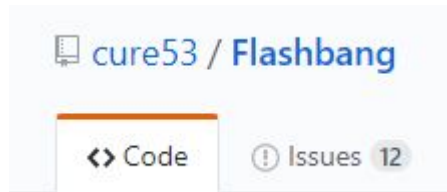
Finding XSS

- Common places XSS tends to show up...
 - Customizable themes/profiles
 - Event meetings/names
 - URI based (redirect=)
 - Content imported from a 3rd party
 - (not sanitizing other data it's using)
 - File upload names
 - Uploaded files (swf/HTML)
 - Custom error messages
 - (injected page was not found...)
 - Fake params (foo=">)
 - JSON responses; check content type (IE only)
 - Login/forgot password forms
 - (an email has been sent to...)



SWF XSS

- Flashvars
 - Common ones include:
 - onload
 - allowedDomain
 - movieplayer
 - xmlPath
 - ++
- Flashbang
 - Decompiles and parses the SWF file, and returns possible params to test



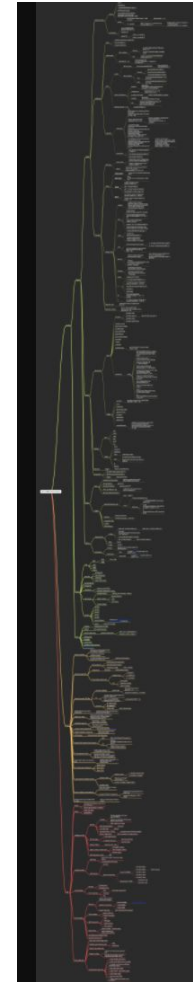
Blind XSS

- Stored, but XSS that you can't verify by hand
- Some frameworks (because you want to use a framework)
 - sleepy-puppy (netflix)
 - Good for campaigns
 - Xsshunter
 - Most commonly used (it seems)
 - Gives back a lot of great info when it fires
 - Screenshot; cookies; email!
 - Does require a domain, wildcard SSL cert, mailgun account, and some setup, but is pretty slick when configured
- Growing area; but there are questions about in/out of scope
 - Always check scope



One last resource on XSS

- Jackmasa's XSS mindmap
 - Massive dump of tons of varying contexts and injections

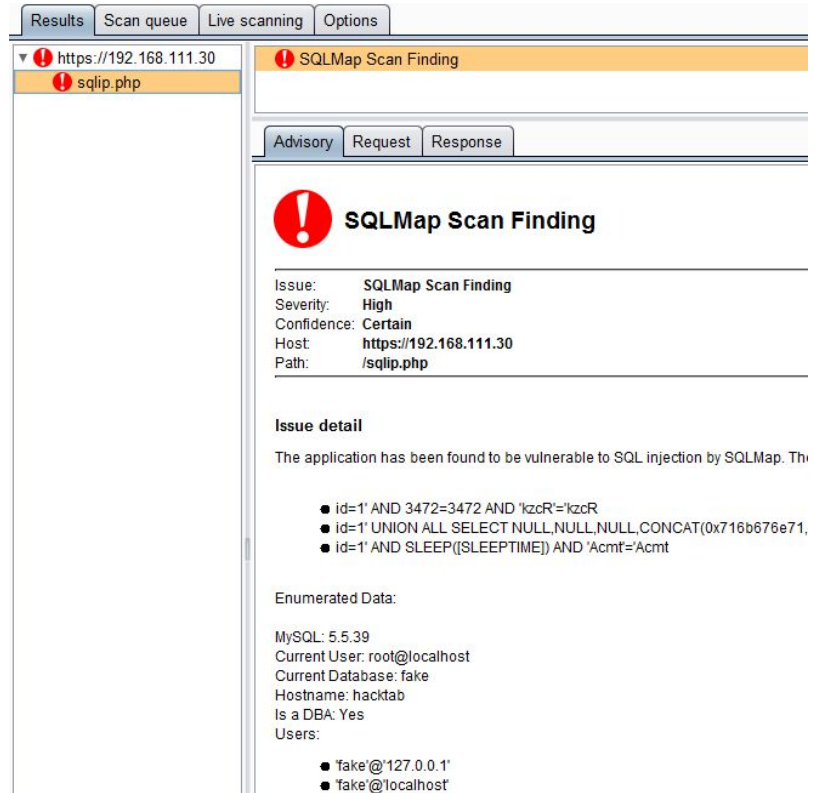
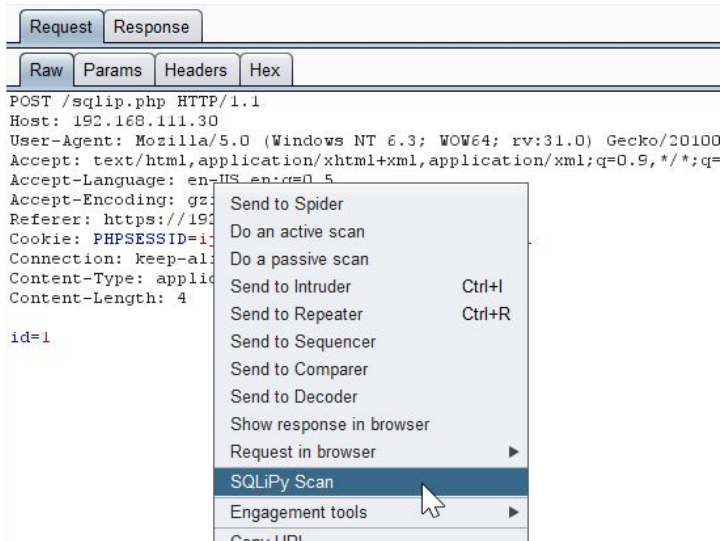


SQL Injection

- Does this page look like it might need to call on stored data?
 - There are some polyglots for SQLi - e.g. SLEEP(1)/*' or SLEEP(1) or ''' or SLEEP(1) or"*/
 - Mattias on polyglots - check out his pres:
(<https://www.slideshare.net/MathiasKarlsson2/polyglot-payloads-in-practice-by-avlidienbrunn-at-hackpra>)
 - Seclists has a lot of fuzzlists for SQLi
 - Some observations/thoughts:
 - Not so much error-based these days; mostly blind
 - SQLMap is your friend...
 - Can use SQLMap with -L to parse Burp log files
 - Common params:
 - ID values; currency values; sorting params; JSON and XML values; cookie values; custom headers

SQLiPy

- SQLiPy
 - Burp plugin
 - right click on any request, to send to SQLMap



SQLi DBMS specific resources...

mySQL

[PentestMonkey's mySQL injection cheat sheet](#)

[Reiners mySQL injection Filter Evasion Cheatsheet](#)

MSSQL

[Evil SQL Error/Union/Blind MSSQL Cheatsheet](#)

[PentestMonkey's MSSQL SQLi injection Cheat Sheet](#)

ORACLE

[PentestMonkey's Oracle SQLi Cheatsheet](#)

POSTGRESQL

[PentestMonkey's Postgres SQLi Cheatsheet](#)

Others

[Access SQLi Cheatsheet](#)

[PentestMonkey's Ingres SQL Injection Cheat Sheet](#)

[pentestmonkey's DB2 SQL Injection Cheat Sheet](#)

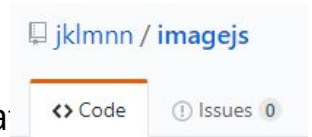
[pentestmonkey's Informix SQL Injection Cheat Sheet](#)

[SQLite3 Injection Cheat sheet](#)

[Ruby on Rails \(Active Record\) SQL Injection Guide](#)

File uploads

- Malicious file upload
 - Can we upload and run that content?
- Possible attacks
 - Upload an unexpected file format to achieve code exec (php, jsp, aspx, ++)
 - Not so likely these days to get full on code execution :(
 - More likely to be able to upload an html file - leading to xss, etc
 - Execute XSS ^, also inject on the filename -- images as well → imagejs
 - Attack the parser to DoS the site or XSS via storing payloads in metadata
 - XXE (often on pdf uploads, but other places as well) → oxml_xxe tool
 - Ex: Imagetrack (not XXE, but using file upload to get code exec)
 - Bypass security zones and store malware on target site via file polyglots
 - Files that execute in different contexts!



Local File Inclusion #LFI

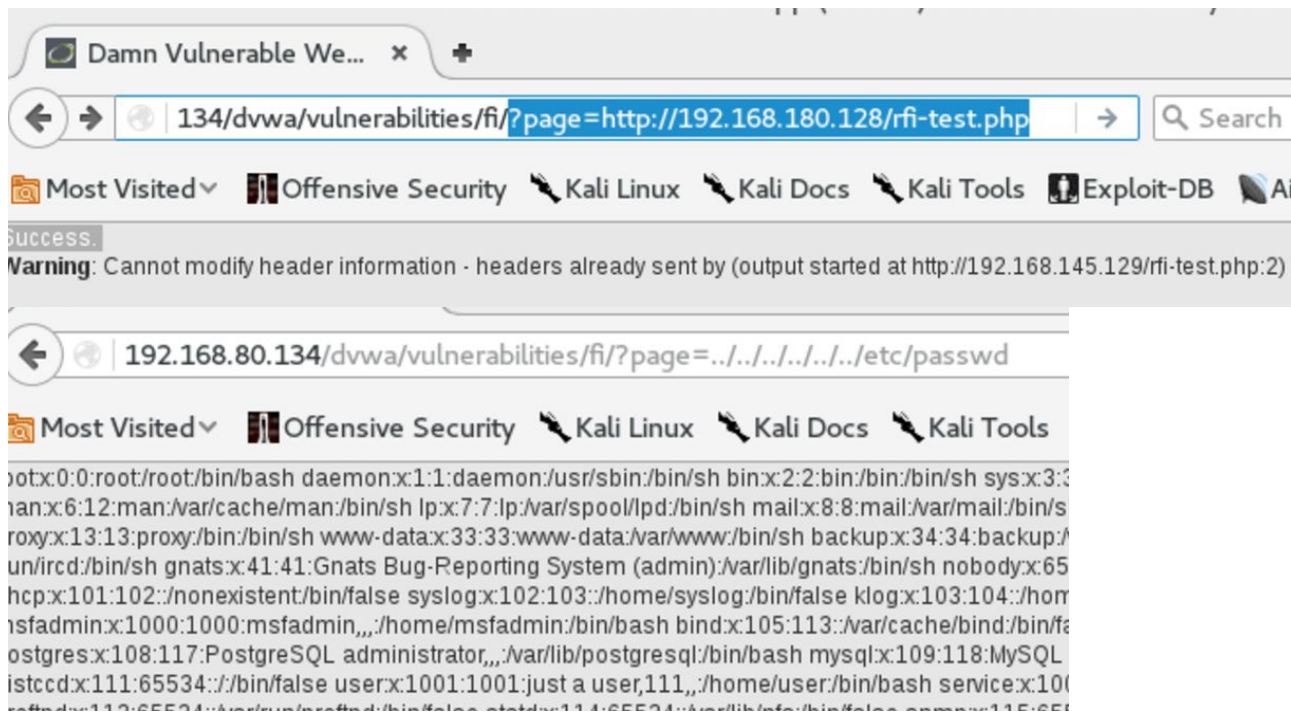
- Does it or can it interact with the server file system?
- Tools:
 - Liffy → github.com/hvqzao/liffy
 - Seclists fuzzing list →
- Common Parameters or Injection points
 - file=
 - location=
 - locale=
 - path=
 - display=
 - load=
 - read=
 - retrieve=

[illegible]

Arbitrary File Inclusion {AFI/RFI}

- Common Parameters/Injection points

- File=
- document=
- Folder=
- root=
- Path=
- pg=
- style=
- pdf=
- template=
- php_path=
- doc=



File includes and redirects

- Look for any param with another web address in it.
 - A lot of the same params from LFI can often work here
- Common bypasses include:
 - escape "/" with "/" or "/" with "/"
 - try single "/" instead of "/"
 - remove http i.e. "continue=//google.com"
 - "/\", "|", "/%09"
 - encode, slashes
 - "../" CHANGE TO "..//"
 - "../" CHANGE TO "...//"
 - "/" CHANGE TO "/"

Redirects

- Common params/injection points
 - Anytime it's pulling from or pointing to a resource - try make it grab something else, or even point it to your own; ex: "redirect_to" giving back internal files /etc/passwd - etc.
 - dest=
 - continue=
 - redirect=
 - url= (or anything with "url" in it)
 - uri= (same as above)
 - window=
 - next=

CSRF

- Burp PoC makes things pretty easy these days
- Focus on bypasses
 - Removing the param
 - Using old values
 - Values from other sessions
 - Modifying the values, etc
- Burpy (github.com/debasishm89/burpy)
 - Helps automate finding CSRF bypasses
 - Enable site logging in Burp; crawl the site, doing all the actions
 - Create a template, then run it against your log
 - Burpy then tries removing tokens, etc, and we can diff responses
- Or focus on pages with no token... (github.com/arvinddoraiswamy/mywebappsripts/blob/master/BurpExtensions/csrf_token_detect.py)
 - Keep in mind that it's not always in the POST body...

Crafted Request [Token Removed from Request]

```
POST /messages/action/ HTTP/1.1
Content-Length: 61
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:20.0) Gecko/20100101 Firefox/20.0
Host: www.facebook.com
Referer: http://www.facebook.com/messagingconfirmation?action_url=/messages/action/?
mm_action=delete&tids=mid.1375723992343%3A9fb37a810424df201&tid=mid.1375723992343:9fb37a810
Fun: Fun
Cookie: Deleted
Content-Type: application/x-www-form-urlencoded
```

mm_action=delete&tids=mid.1375723992343:9fb37a810424df2016&

Live Response

```
HTTP/1.1 408 Client timeout
date: Thu, 17 Oct 2013 07:54:30 GMT
connection: keep-alive
content-type: text/html; charset=utf-8
content-length: 2131
```

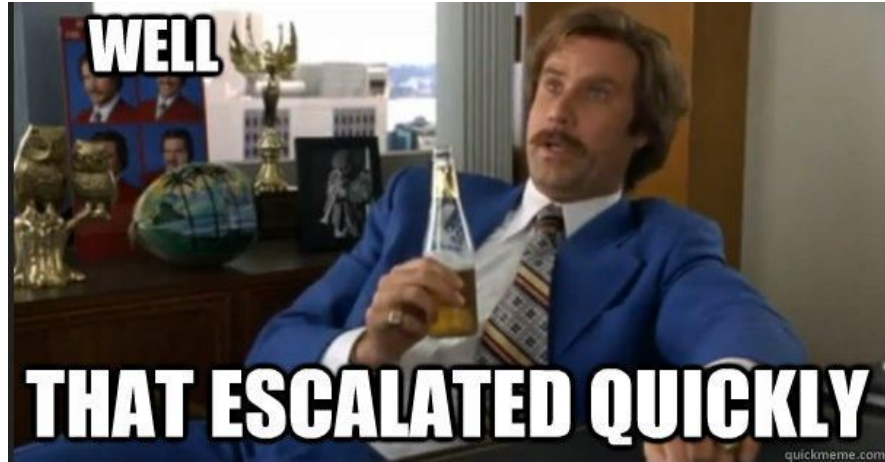
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1
```

CSRF

- Common critical functions to check for CSRF
 - Need CSRF to be on something of value
 - Add/upload file
 - Password/email change
 - Delete file
 - Profile edit
 - And so on...

Privilege escalation

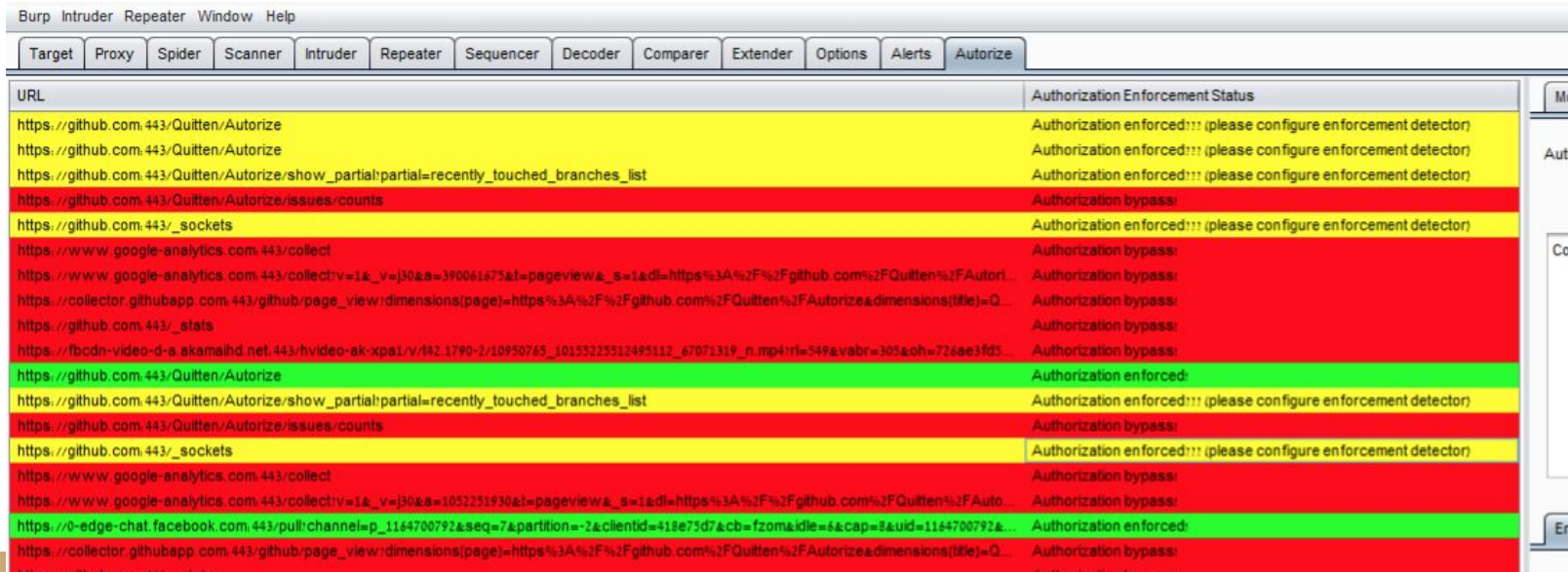
- Can a given user level do what they otherwise shouldn't be able to do?
 - Usually will need accounts of varying privileges
- Common places to check
 - Add/delete/modify user
 - Change account info
 - Customer analytics view



- Things on the backend, where they never would have thought the lower level user would have the knowledge to make that request
- A lot of apps just have the 'vibe' that they relied on security through obscurity

Authorize

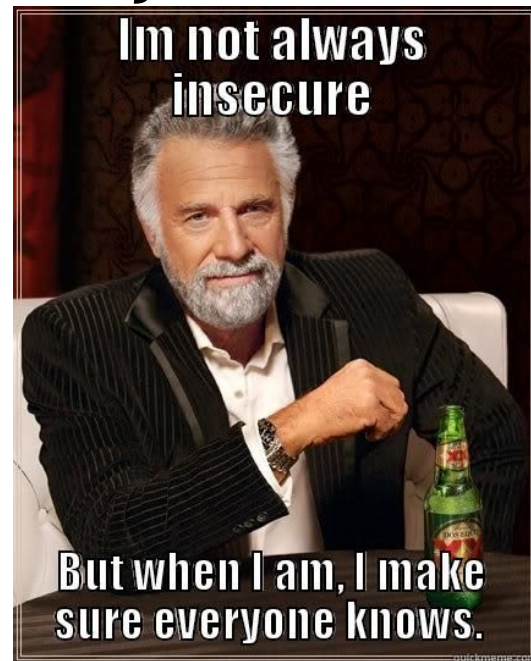
- Burp plugin (github.com/Quitten/Autorize)
 - Helps with privilege testing
 - Browse app with high priv user; then have it resend all those same requests authenticated as a lower priv user, and see what actions succeed



URL	Authorization Enforcement Status
https://github.com.443/Quitten/Autorize	Authorization enforced!!! (please configure enforcement detector)
https://github.com.443/Quitten/Autorize	Authorization enforced!!! (please configure enforcement detector)
https://github.com.443/Quitten/Autorize/show_partial?partial=recently_touched_branches_list	Authorization enforced!!! (please configure enforcement detector)
https://github.com.443/Quitten/Autorize/issues/counts	Authorization bypass:
https://github.com.443/_sockets	Authorization enforced!!! (please configure enforcement detector)
https://www.google-analytics.com.443/collect	Authorization bypass:
https://www.google-analytics.com.443/collect?v=1&_v=j30&a=390061675&_t=pageview&_s=1&dl=https%3A%2F%2Fgithub.com%2FQuitten%2FAutorize&dimensions(title)=Q...	Authorization bypass:
https://collector.githubapp.com.443/github/page_view?dimensions(page)=https%3A%2F%2Fgithub.com%2FQuitten%2FAutorize&dimensions(title)=Q...	Authorization bypass:
https://github.com.443/_state	Authorization bypass:
https://fbcdn-video-d-a.akamaihd.net.443/hvideo-ak-xpa1/v/142.1790-2/10950745_10155225312495112_67071319_n.mp4?ri=54&vabr=305&oh=726&e3fd5...	Authorization bypass:
https://github.com.443/Quitten/Autorize	Authorization enforced:
https://github.com.443/Quitten/Autorize/show_partial?partial=recently_touched_branches_list	Authorization enforced!!! (please configure enforcement detector)
https://github.com.443/Quitten/Autorize/issues/counts	Authorization bypass:
https://github.com.443/_sockets	Authorization enforced!!! (please configure enforcement detector)
https://www.google-analytics.com.443/collect	Authorization bypass:
https://www.google-analytics.com.443/collect?v=1&_v=j30&a=1052251930&_t=pageview&_s=1&dl=https%3A%2F%2Fgithub.com%2FQuitten%2FAutorize&dimensions(title)=Q...	Authorization bypass:
https://q-edge-chat.facebook.com.443/pull/channel=p_1164700792&seq=7&partition=-2&clientid=418e75d7&cb=fzom&idle=&cap=8&uid=1164700792&...	Authorization enforced:
https://collector.githubapp.com.443/github/page_view?dimensions(page)=https%3A%2F%2Fgithub.com%2FQuitten%2FAutorize&dimensions(title)=Q...	Authorization bypass:

IDORs (insecure direct object references)

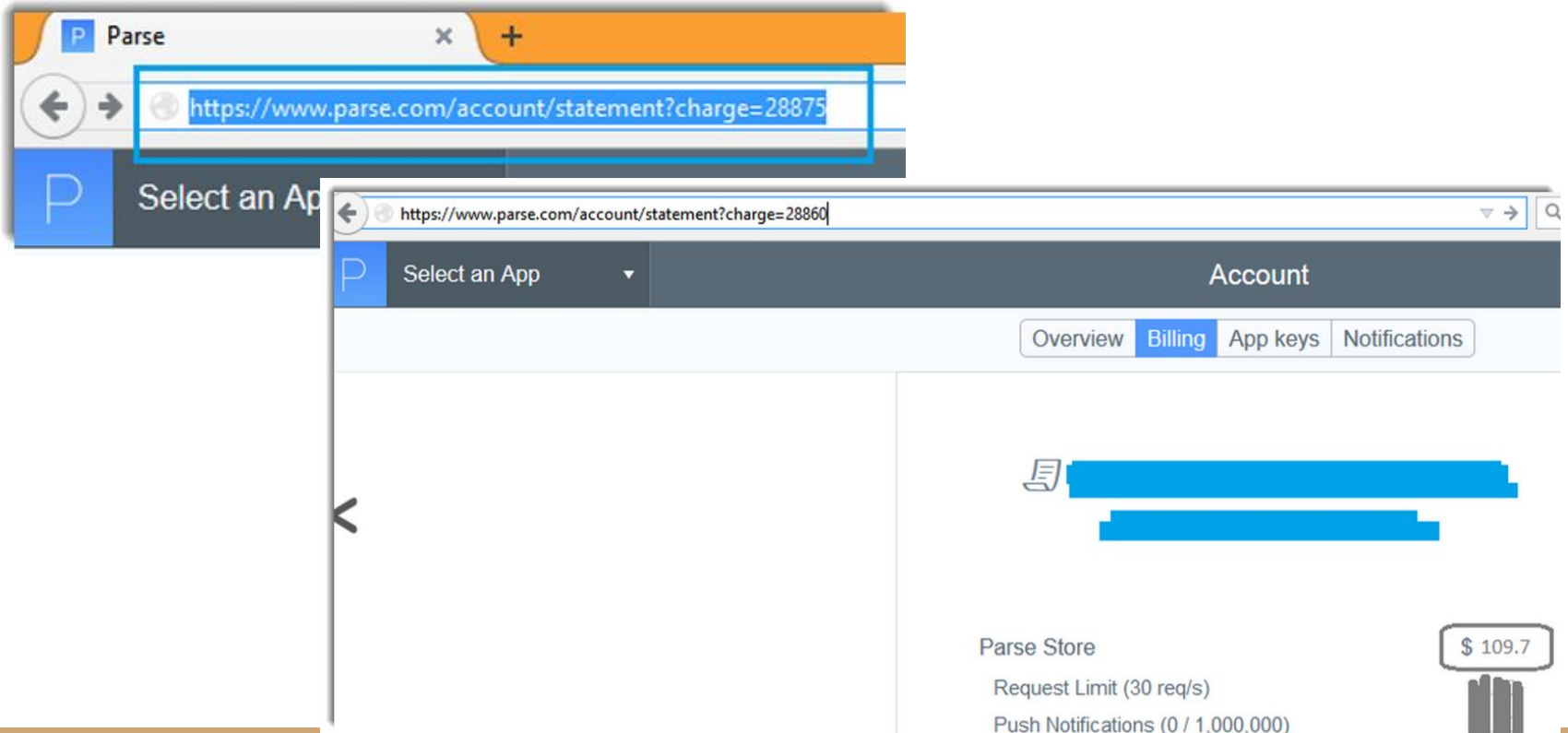
- Near impossible to find with scanners
 - As a result, pretty common on bug bounties!
 - Can you access or modify something that's not yours?
- When testing...
 - Find any and all UUIDs
 - Increment AND decrement
 - Try negative values
 - Attempt to perform sensitive actions using another user's ID
 - Change password
 - Forgot password
 - Admin only functions
 - Try not to modify other people's accounts; test only against what you own



IDORs

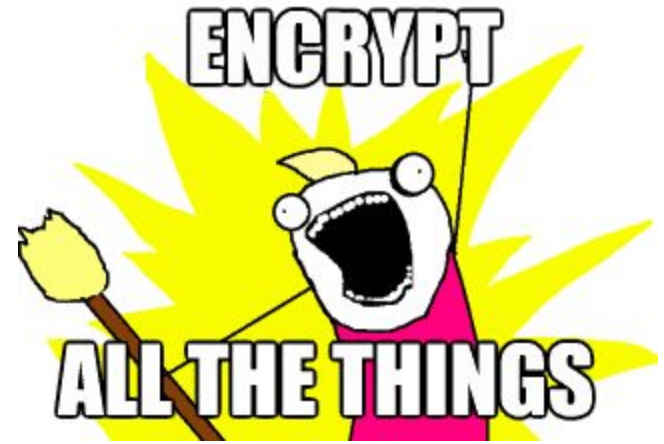
- Other IDOR thoughts...
 - Try pretty much any sensitive action across accounts
 - Again, scanners won't really find these things...
 - We see them a LOT
 - Can you access without authentication, etc?
 - Note other UUIDs or ways users are identified - hashes, emails, etc
 - Receipts (what happens if I put one less on my receipt?)
 - Same concept for files
 - Shipping/purchase order ids, etc
 - Messages sent/delete

IDOR example



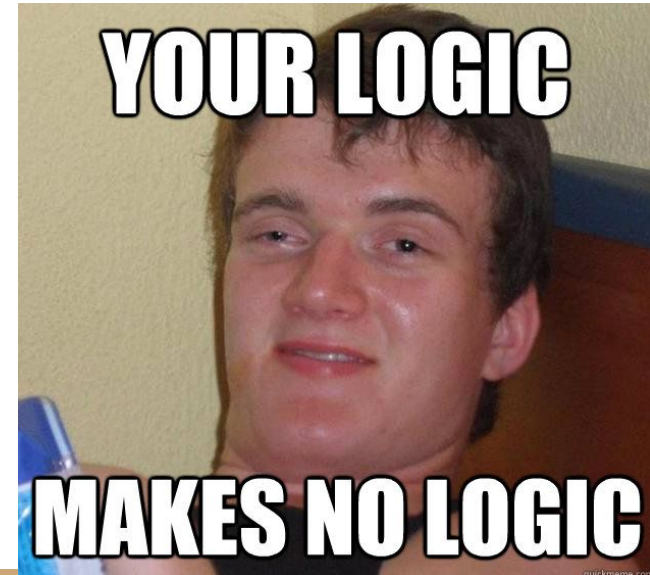
Transport

- Make sure everything is over HTTPS
- Examples:
 - Sensitive images transported over HTTP
 - Login forms over HTTP
 - Analytics with session data / PII leaked over HTTP
- ForceSSL
 - Tool that takes https links and tries them over http
 - github.com/arvinddoraismamy/mywebappscripts/tree/master/ForceSSL



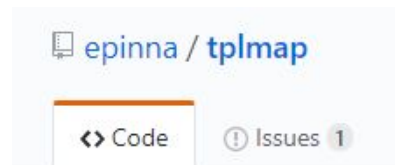
Logic

- Logic flaws that are tricky, mostly manual:
 - substituting hashed parameters
 - step manipulation
 - use negatives in quantities (can you get money TO your account?)
 - authentication bypass
 - application level DoS
 - (massive parameter values, 999999 pages, etc)
 - Timing attacks



Server side template injection (SSTI)

- Does the site use a templating engine?
 - A lot of times one can get code exec or file read
- TPLmap
 - Like SQLmap for template injection
 - Covers a lot of the major templating engines (flask, etc)
 - Can even give shells!
 - Reviewing source code, you can see how to do this manually
 - Lot more manual content online, but worth looking into #notAnExpert



```
$ ./tplmap.py -u 'http://www.target.com/page?name=John'
[+] Tplmap 0.3
    Automatic Server-Side Template Injection Detection and Exploitation Tool

[+] Testing if GET parameter 'name' is injectable
[+] Smarty plugin is testing rendering with tag '{*}'
[+] Smarty plugin is testing blind injection
[+] Mako plugin is testing rendering with tag '${*}'
...
[+] Jinja2 plugin is testing rendering with tag '{{{*}}}'
```

SSRF (Server Side Request Forgery)

- Like LFI (same params, etc)
 - But some other things/ideas for you can do
 - For instance, possibly hitting internal machines or services that aren't accessible externally e.g. 127.0.0.1:8080/admin, etc
- SSRF bible cheat sheet (google doc)
 - docs.google.com/document/d/1v1TkWZtrhzRLy0bYXBcdLUedXGb9njTNIJXa3u9akHM/
- Once again, #notAnExpert, but a PoC is usually easy enough to put together, and a clear demonstration of the issue is enough to get paid.

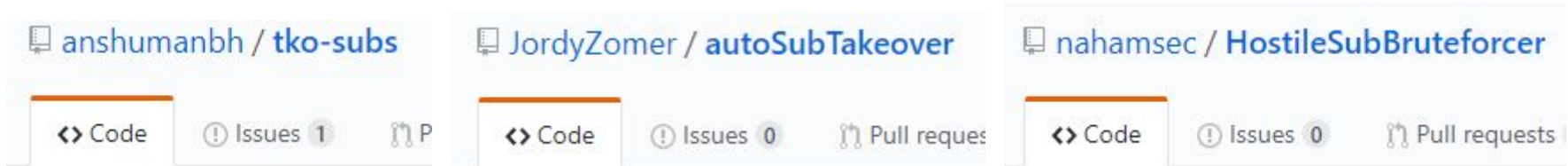
Subdomain Takeover

- Going back to discovery...
 - Sometimes orgs forget about dangling cnames that were once setup for services.
 - All we have to do is find these pages waiting to be claimed...
- Common forgotten services...
 - heroku
 - github
 - tumblr
 - shopify
 - squarespace
 - salesforce
 - desk
 - Aws (s3 buckets)
 - fastly
 - hubspot
 - and on and on



Subdomain Takeover

- Don't break ToS; often claiming the domain isn't necessary
- Some tools:
 - autoSubTakeover
 - HostileSubBruteforcer
 - Tko-subs
 - All do roughly the same thing; matter of preference.



Ok. Some notes...

- Always read the scope. It's important.
 - VRT
- Where are people NOT testing?
 - APIs
 - Boring, but often untested
 - IDORs, etc (but loses XSS, etc)
 - Mobile apps
 - Which often boils down to an API
 - Binary apps
 - Web/ui testing is again, easier



Dealing with people...

- People are temperamental creatures
- Remember:
 - The person on the other side is a person
 - They have ups/downs
 - People to report to, etc
 - Bottom line: follow the golden rule
 - Treat others as you'd like to be treated
- This also applies to reports
 - Demonstrating impact on a report is huge
 - They usually have to explain this to non security people

Bill is on the internet.

**Bill sees something
that offends him.**

Bill moves on.

Bill is smart.

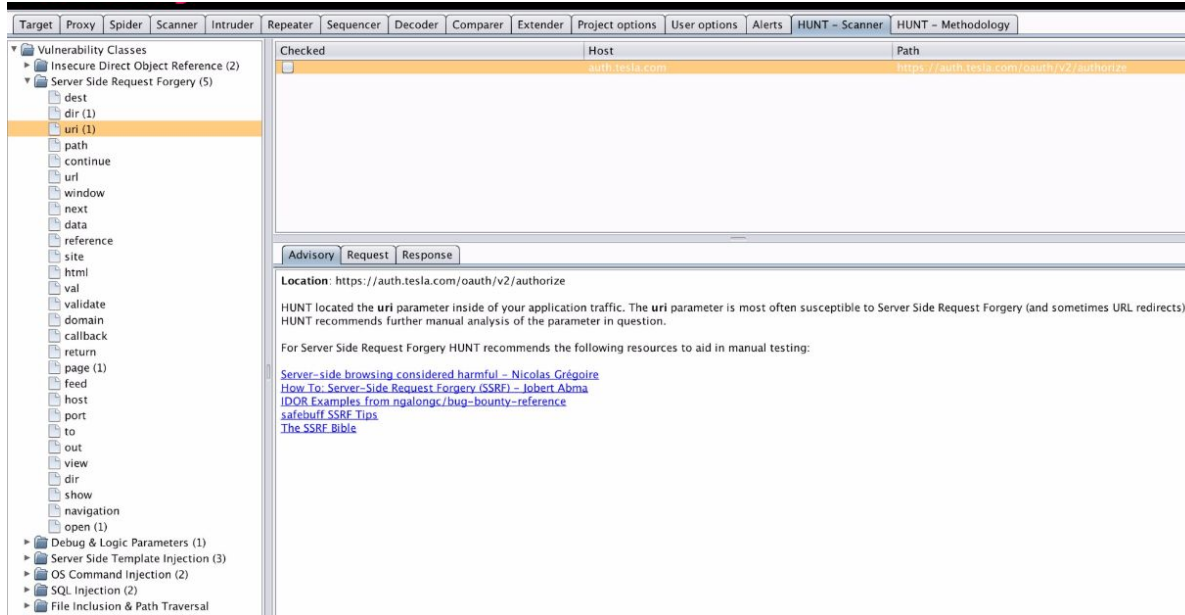
Be like Bill.



One more tool...

HUNT

- Hunt!
 - Burp extension that looks for a lot of the params that we've talked about today!
 - github.com/bugcrowd/HUNT



Data Driven Assessment

- How to test an app in n minutes...
 - Visit the search, registration, contact, password reset, and comment forms and hit them with your polyglot strings
 - Scan those specific functions with Burp's built-in scanner
 - Check your cookies, log out, check cookies, log in, check cookies. Submit old cookies, see if there's access.
 - Do a reset and see if; the password comes plaintext, uses a URL based token, is predictable, can be used multiple times, or logs you in automatically
 - Find numeric account identifiers anywhere in URLs and rotate them for context change
 - Find the security-sensitive function(s) or files and see if vulnerable to non-auth browsing (idors), lower-auth browsing, CSRF, CSRF protection bypass, and see if they can be done over HTTP.
 - Directory brute for top short list on SecLists
 - Check upload functions for alternate file types that can execute code (xss or php/etc/etc)

And that's that.

- There was a lot.
- Hopefully you learned at least ONE thing
 - I learned a ton while putting this together!
- Would love to talk, get feedback, etc.
 - @grantmcmusic
 - [linkedin.com/in/grantmccracken](https://www.linkedin.com/in/grantmccracken)

