

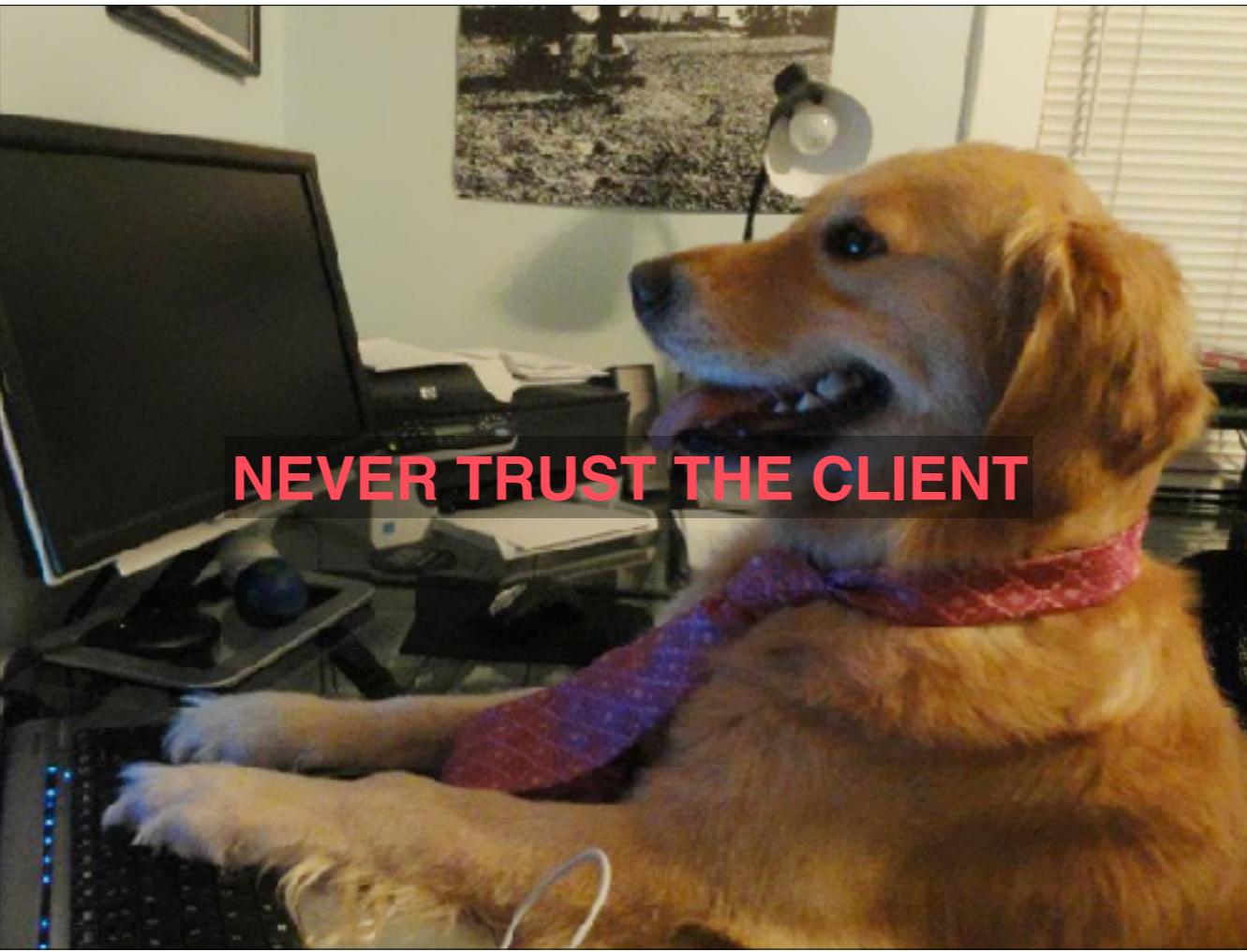
SECURING YOUR WEBSERVICE

@garrybodsworth

<https://www.yagro.com>

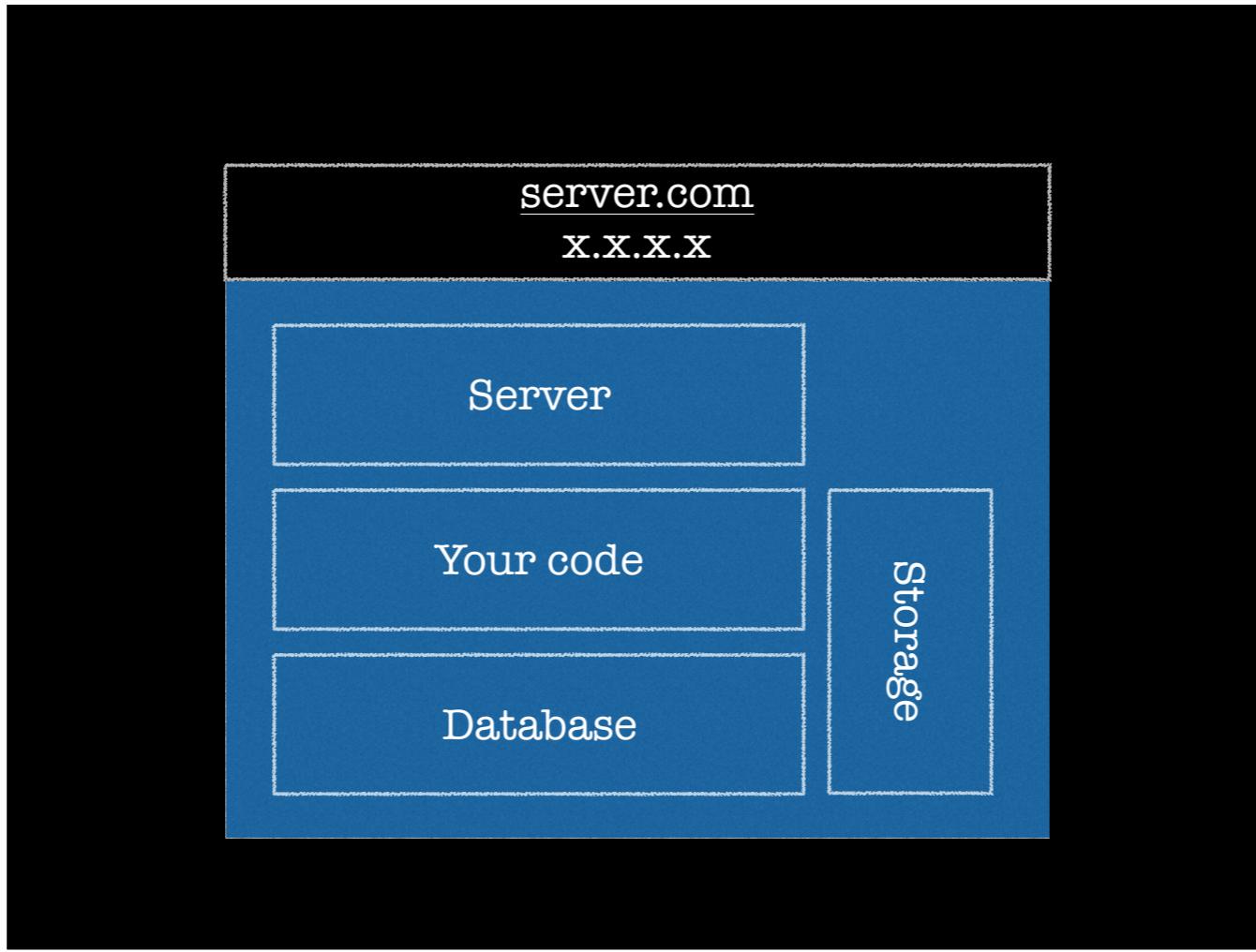
About me:

- devices
- web services
- Microsoft
- Bromium + browsers



I'll come back to this.

Critical thing to learn when building all webservices



Standard server layout

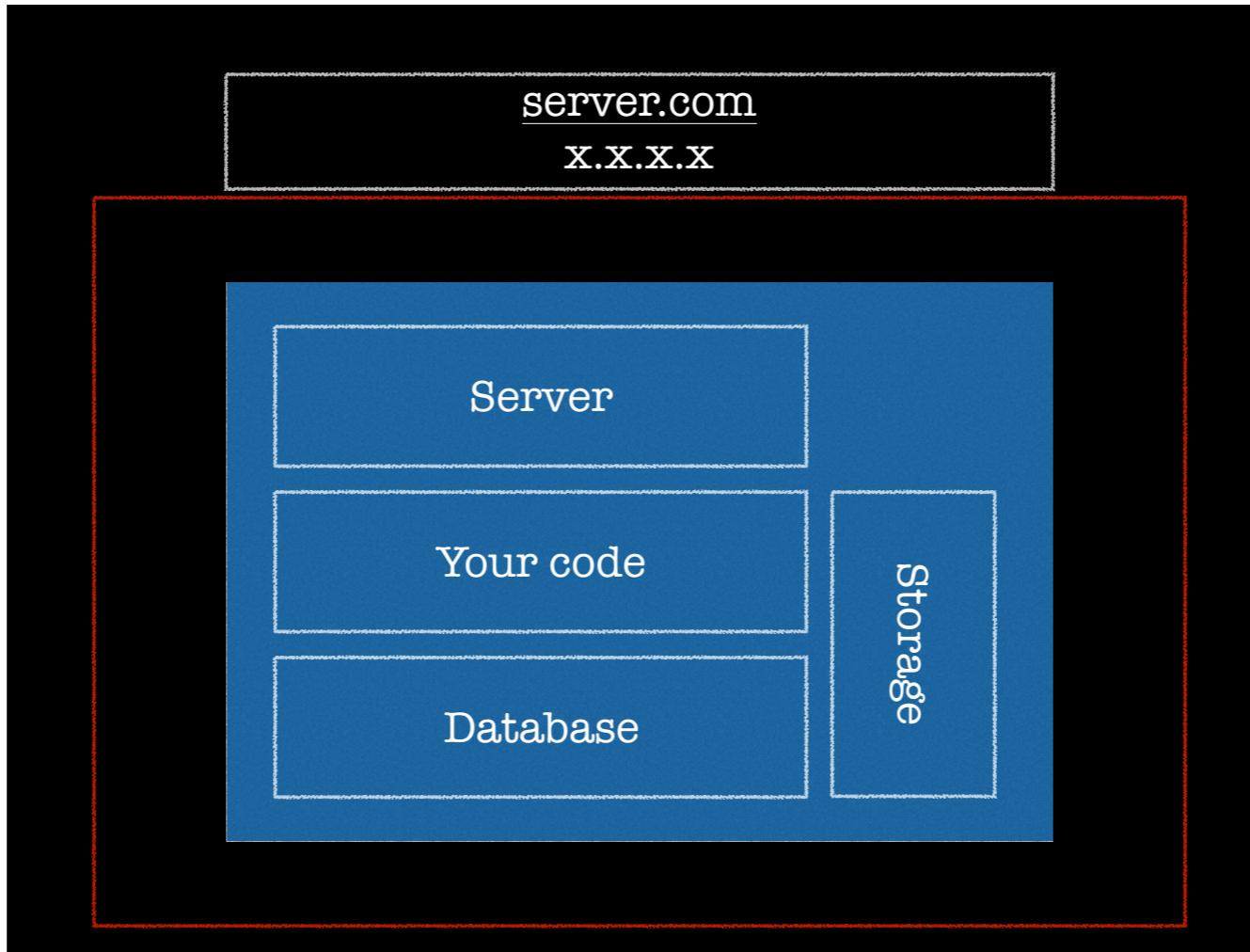
Exposed single IP address

DNS maps to only one

Server fully exposed on Internet

Must always be kept patched and fully up to date

Compromise this machine and get everything



Architecture can solve some of the basic issues surrounding access to servers

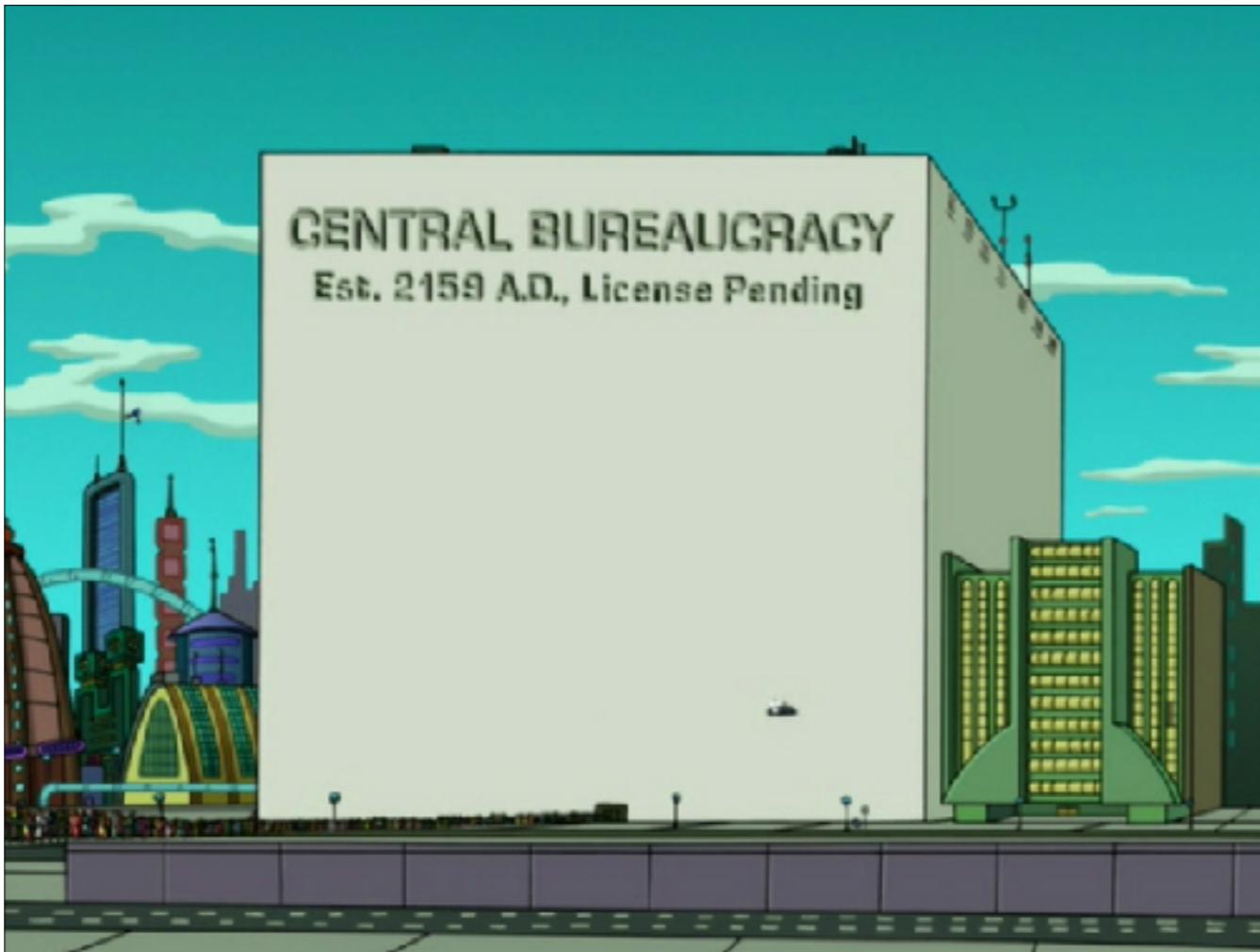
First of all the most basic thing we can do is protect the server from the general internet

IP tables/firewalls are depressing and confusing.

Put in its own private network which is NATed away from the world.



Load balancing + reverse proxy
Decouple DNS name from target
HTTPS termination
Only way for public internet to get into your network
Can target multiple machines



Separate the database

Completely remove any public access from the internet

Only allow clients that should access it (through firewall rules)

Also - make sure you only need the root user for things like backups.

Create a user per database and also have the ability to have readonly users

Advantage of single machine model is unix domain sockets and configuring so no external access.



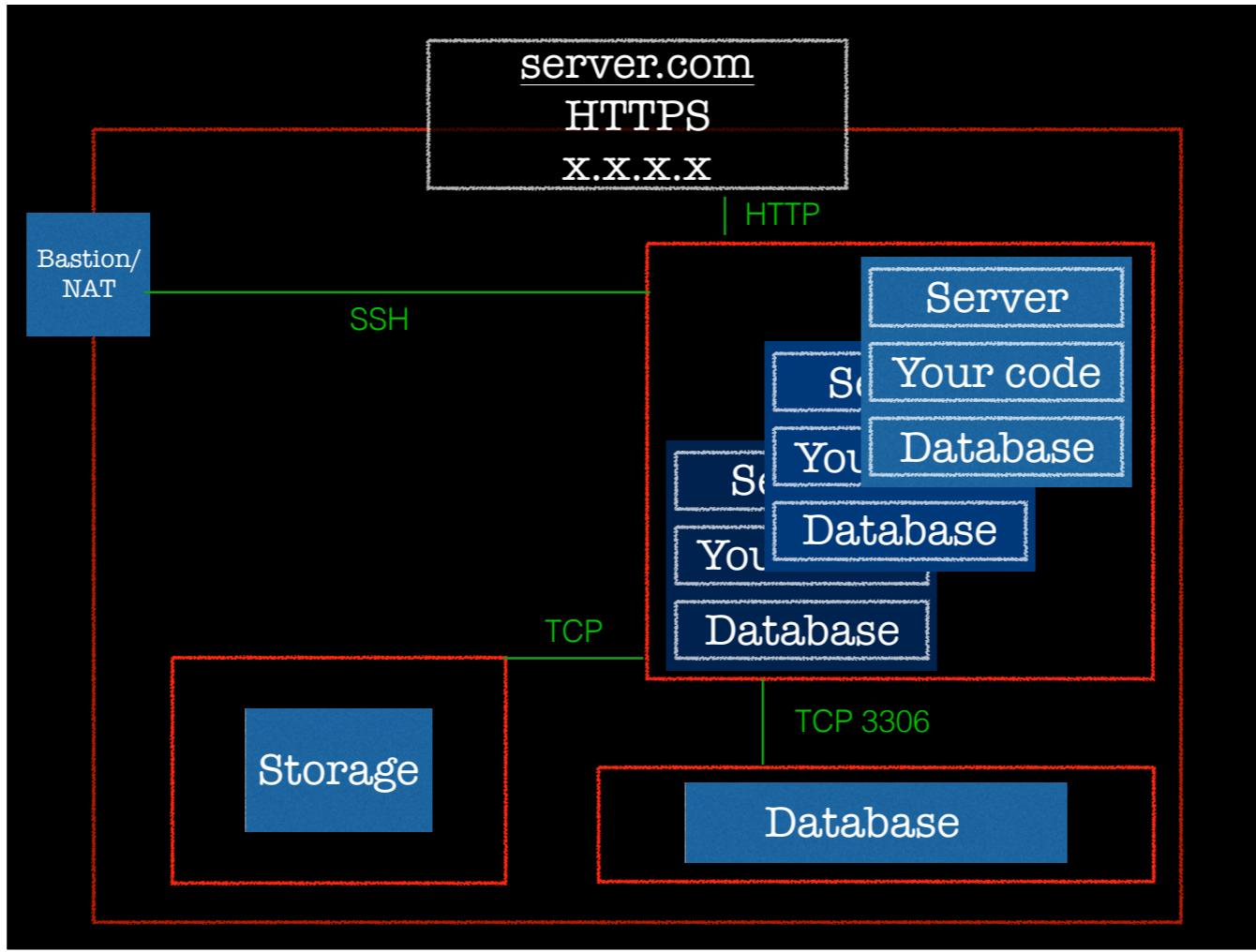
How do you get into the servers?

Bastions or NATs

Depending on the deployment model of push vs pull (ie, Ansible vs Puppet)

Only should allow ssh for bastion and also only from known IPs

For NAT requires setting up the server instances iptables and networking so they can access the internet



We start to look a little like this.



Out of the box your choice of framework is insecure
What can you do?
Learn stuff
Some have best security practices - e.g. django cookie cutter

Gear Up!



MAKE GIFS AT GIFSOUP.COM

Right, we've got problems, now we need to fix them.



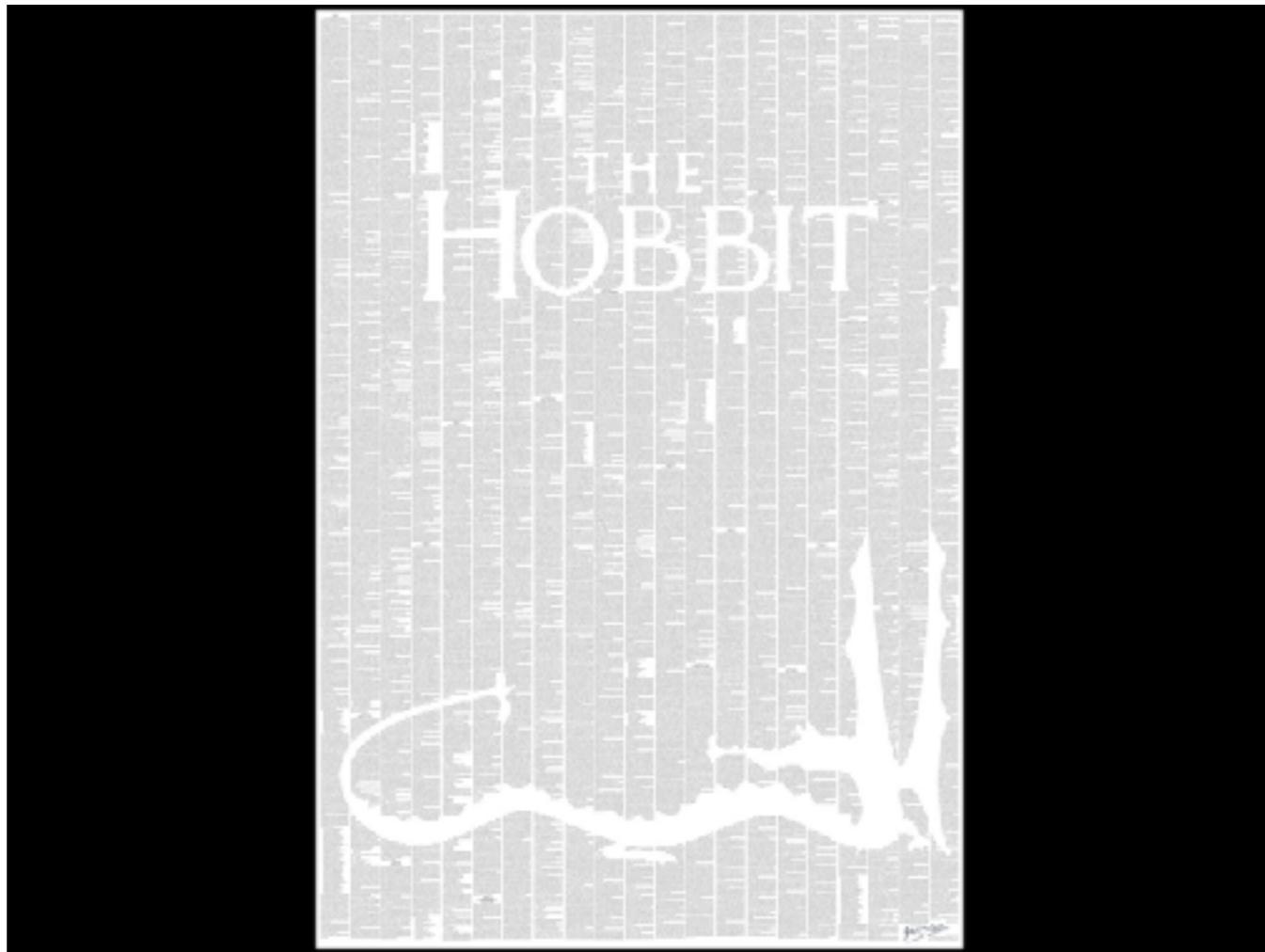
Lets talk a little about cookies

What are they?

How do they work?

What properties do they have?

domain, path, secure, httponly



Tokens

SPA + JWT

JWT supposed to be stateless

Bad for sessions

Can be reused in other contexts

Javascript can access it

Combine with sessions?

Then you have CSRF problems again. There is no clear example.

My current solution.

SSL CERTIFICATES



- Recent revocations
- Recent improvements - Amazon, LetsEncrypt
- Configuring servers, protocols and ciphers

ENCRYPTION

- Transport
- Data at rest

A B C D E F G H I J K L M

N O P Q R S T U V W X Y Z

- Cryptographic hash (one-way)
- Reversible (two-way)

H E L L O

U R Y Y B

- Symmetric
- Asymmetric

- Transport - SSL, TLS, HTTPS
- Data at rest - your storage - database, filesystem, nosql
- Cryptographic hash - for storing passwords
- Reversible - you eventually need the original data like files on your hard drive
- Symmetric - Both know the same secret - less computationally intensive typically
- Assymetric - Public key encryption - big primes

MANAGING SECRETS

- Updating certificates
- Rolling keys
- Service passwords
- Who knows all your secrets?
- Are your secrets safe?
- “Secret Server”, Vault, KMS

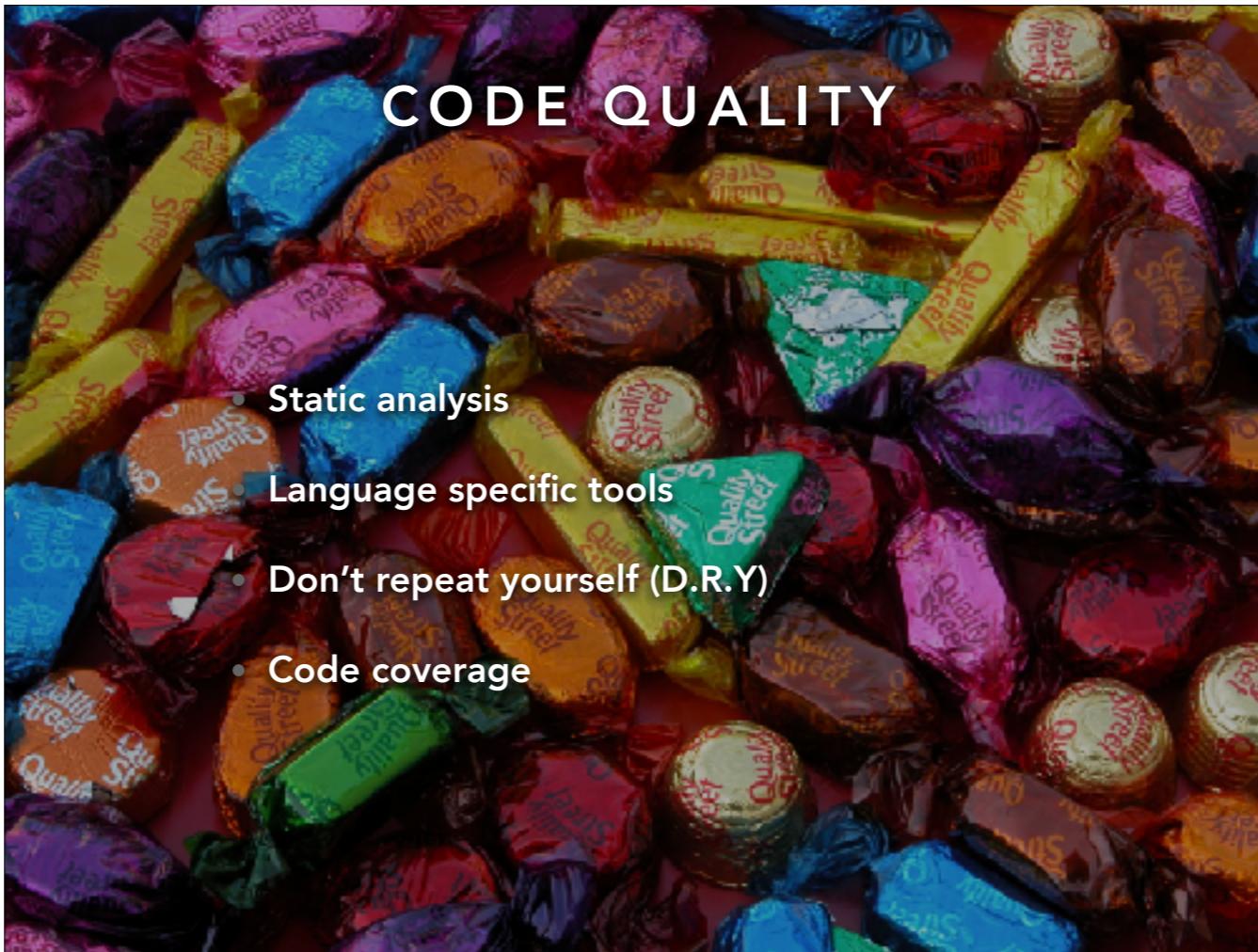


- We know all your secrets
- Where do you store
 - certificates
 - passwords
 - critical information
- Source control? VERY BAD
- Secret server/service? Allows you to decouple secrets from the service.
- Encrypted file? Not bad.
- Rolling keys - have a plan for keys being changed or rotated. Reasons: compromise, age, management....
- USE PASSWORDS ON YOUR DATABASE

SQL INJECTION



- SQL Injection
- Always use the ORM!
- Escaping!
- Django has no known SQL injection attacks open



CODE QUALITY

- Static analysis
- Language specific tools
- Don't repeat yourself (D.R.Y)
- Code coverage

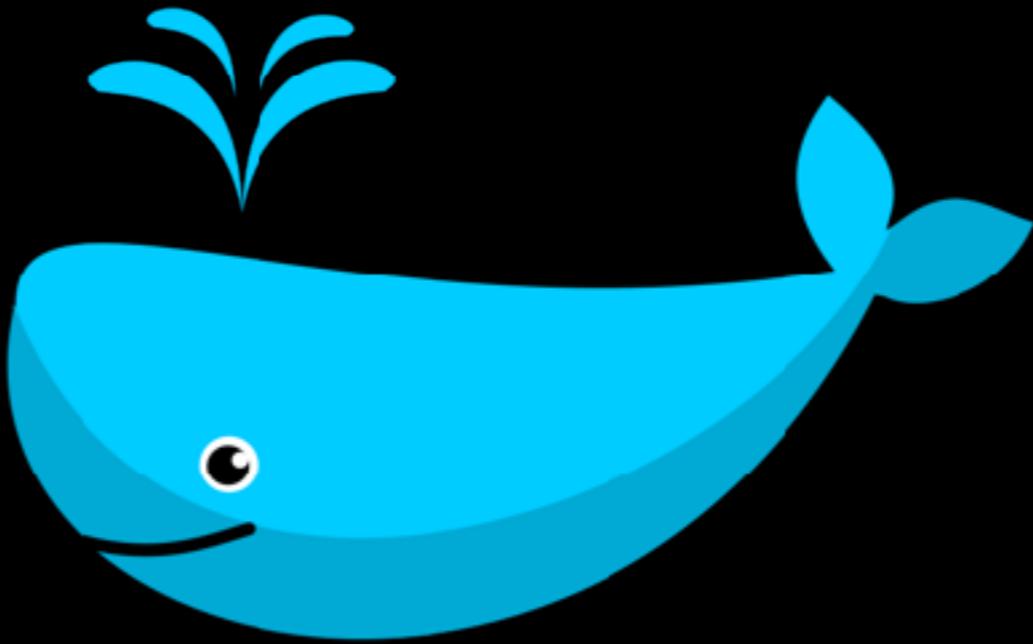
- Start with the most secure settings.
- All warnings on, all warnings as errors
- Static analysis - stops the dumbest of errors
- Language specific tools - Python: e.g. PEP8, PyLint, PyFlakes
- Don't Repeat Yourself - fix a bug once - easier to test
- Code coverage - if the code is never executed except in production how do you know it is good?
- Testing
- Continuous Integration

CONTINUOUS DEPLOYMENT



- Continuous deployment
- 3rd party code drops and updates
- Updating certificates and third party service dependencies
- Use short(ish) expiries to get used to updating
- Be ready to switch providers, the decision may be out of your hands for money, buddies, or plain security disasters (I have an example there...)

Bundling



- * Package the app
- * Docker
- * Install environment
- * Decided on Alpine once I'd convinced myself it worked.
- * Do not use Alpine Python images - you will eventually use a package that fails to install due to some library misnaming. Normal Alpine works great.
- * Use standard Alpine - remember python3 not python on command line
- * Supervisord and uwsgi
- * Fat Docker - convenient packaging rather than micro services.
- * Also developing on Windows gives access to Linux

AWS



- Cloudformation
- Troposphere
- EC2 Container Service
- Credentials/Secrets

* <https://github.com/cloudtools/troposphere>

- * ECS
- * Private registry, updates as part of infrastructure
- * Cloudformation - infrastructure as code. Update infrastructure and packages.
- * EBS, S3, Cloudfront, RDS, auto-scaling groups, security groups, ECS
- * Troposphere <https://github.com/cloudtools/troposphere>
- * SSL
- * Creds/secrets
- * Terraform

CI/CD



GitLab

- * Gitlab
- * All steps are fabric scripts that can be run locally
- * Gitlab CI hooks up to merge requests with gating etc.
- * Causes update via update-stack for cloudformation
- * Master -> staging -> production

Updates



- * Needs AWS creds
- * Generate cloudformation template YAML
- * Call update stack
- * Servers are destroyed and replaced with the new software

- X-CSRF
- Forms
- Referer
- Origin



Exploits your trust of another site to craft an attack - forms with side-effects

Referer can be spoofed even if you check

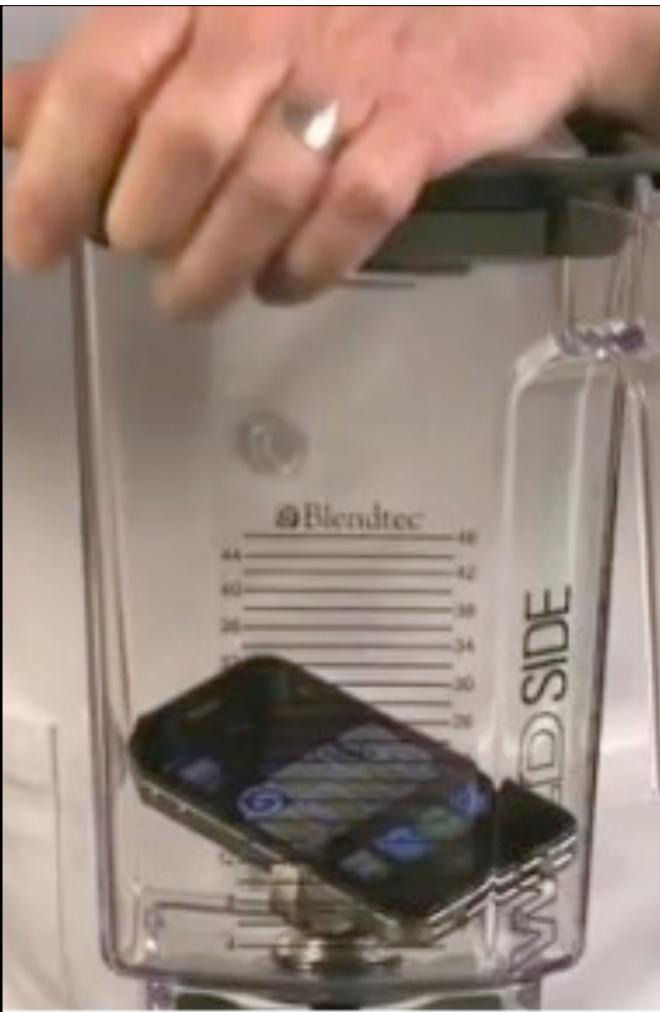
CSRF “synchroniser” tokens in forms

Cookie to header token - sets in JS - relies on same origin security

Uniqueness of CSRF tokens

Related - single use tokens

MIXED CONTENT



- Mixed content
- HTTP can extract HTTPS content on the page - they share the DOM
- Understand your cookies, secure, httponly
- Serve on single domain if you can

IFRAMES



Use XFrame headers - you have to be sure the browsers implement it right

Indicates whether a page should be allowed to be framed and a set of rules for that DENY, SAMEORIGIN, ALLOW-FROM

Rely on the browser for implementation

- XSS
- Reflected
- Persistent
- UXSS



Reflected from form submission

Stored and persistent

Escape output - very difficult

Tie session cookies to the IP address - forgets we are a mobile world

httponly cookies to stop the being stolen

Newest mitigation - Content Security Policy (CSP)



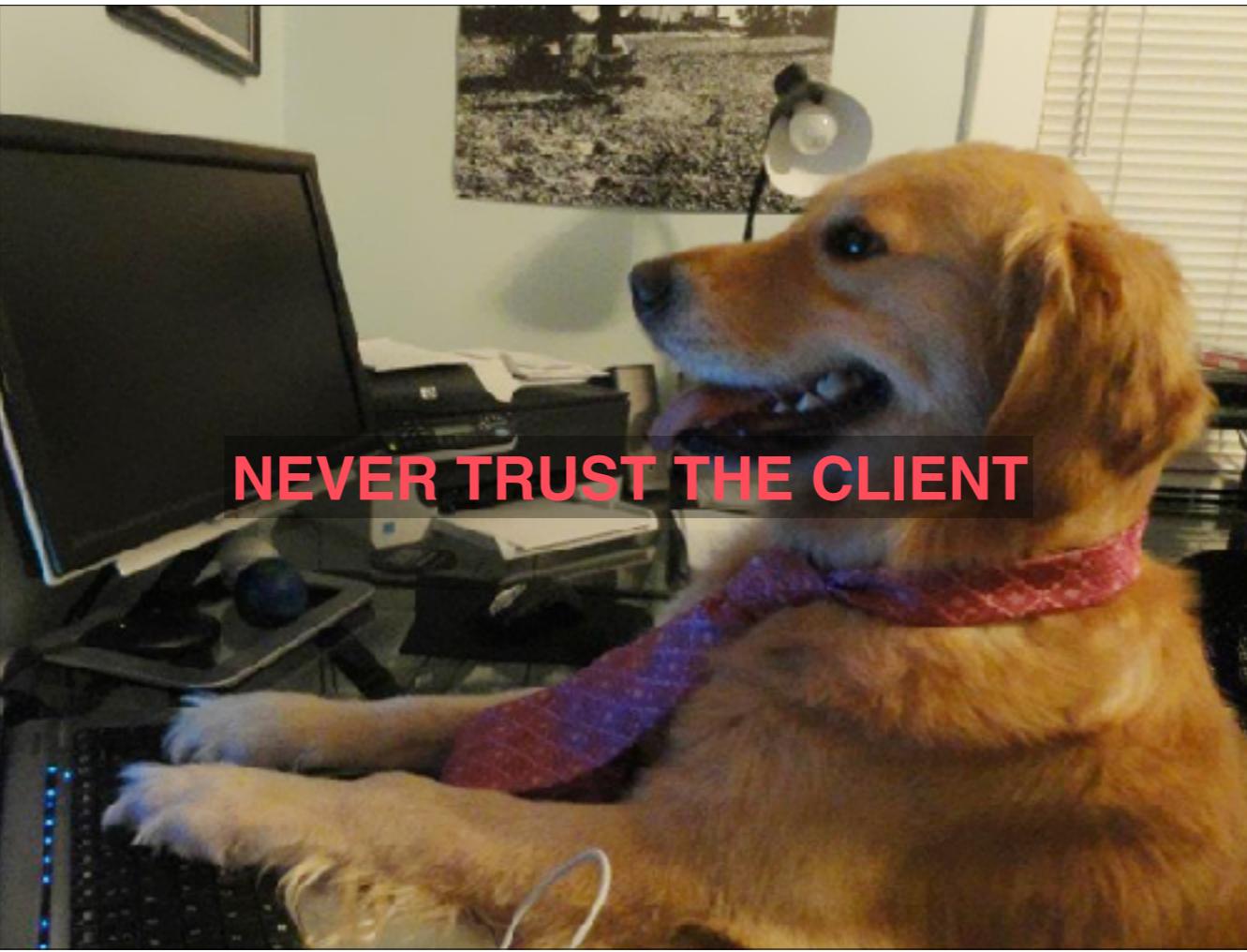
Pretty complex.

Very easy to break your site.

Do you know how your site is set out?

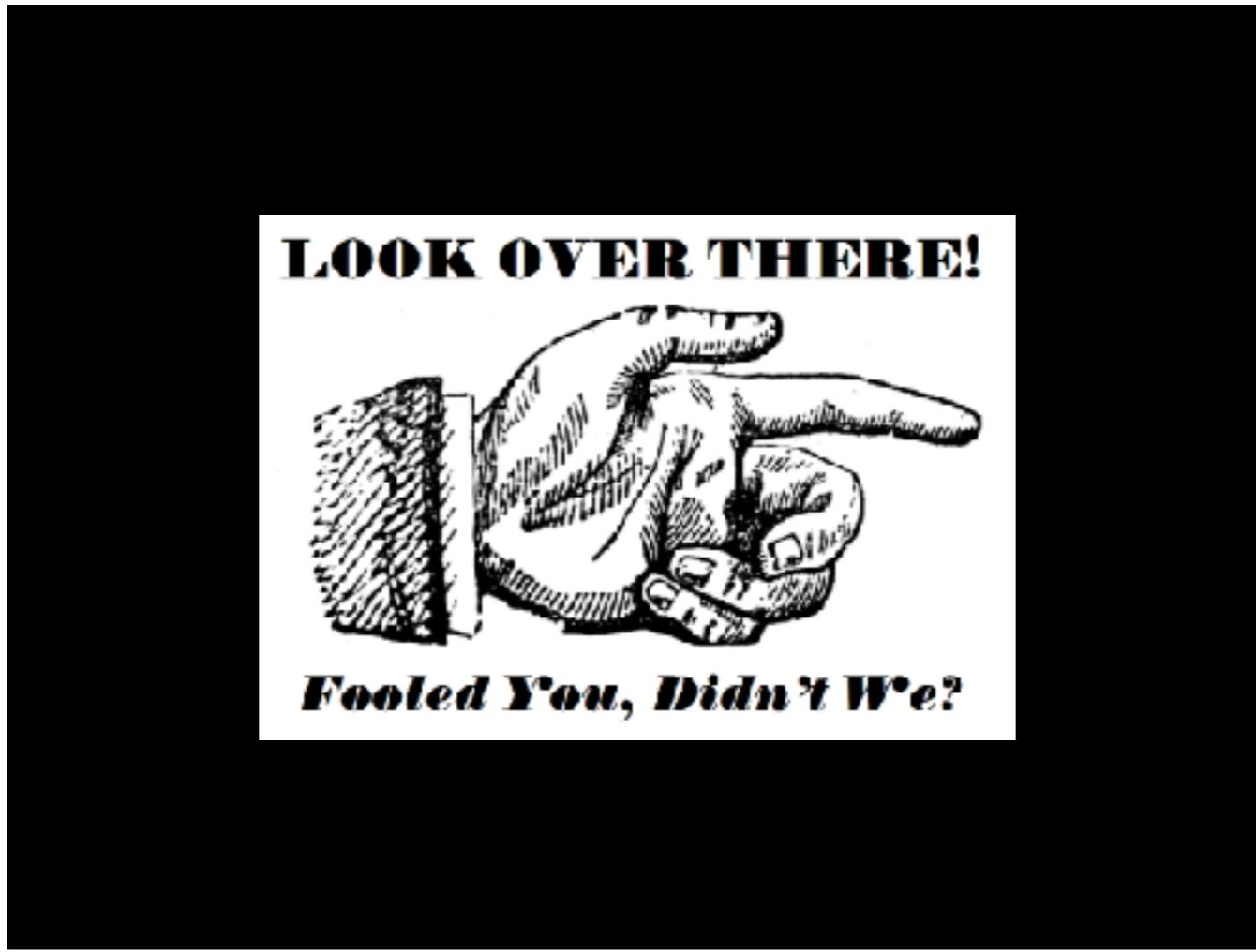
Third-party? Google analytics, advertising, tracking, and so on.

- complicated - relates to serving from multiple domains



Referers

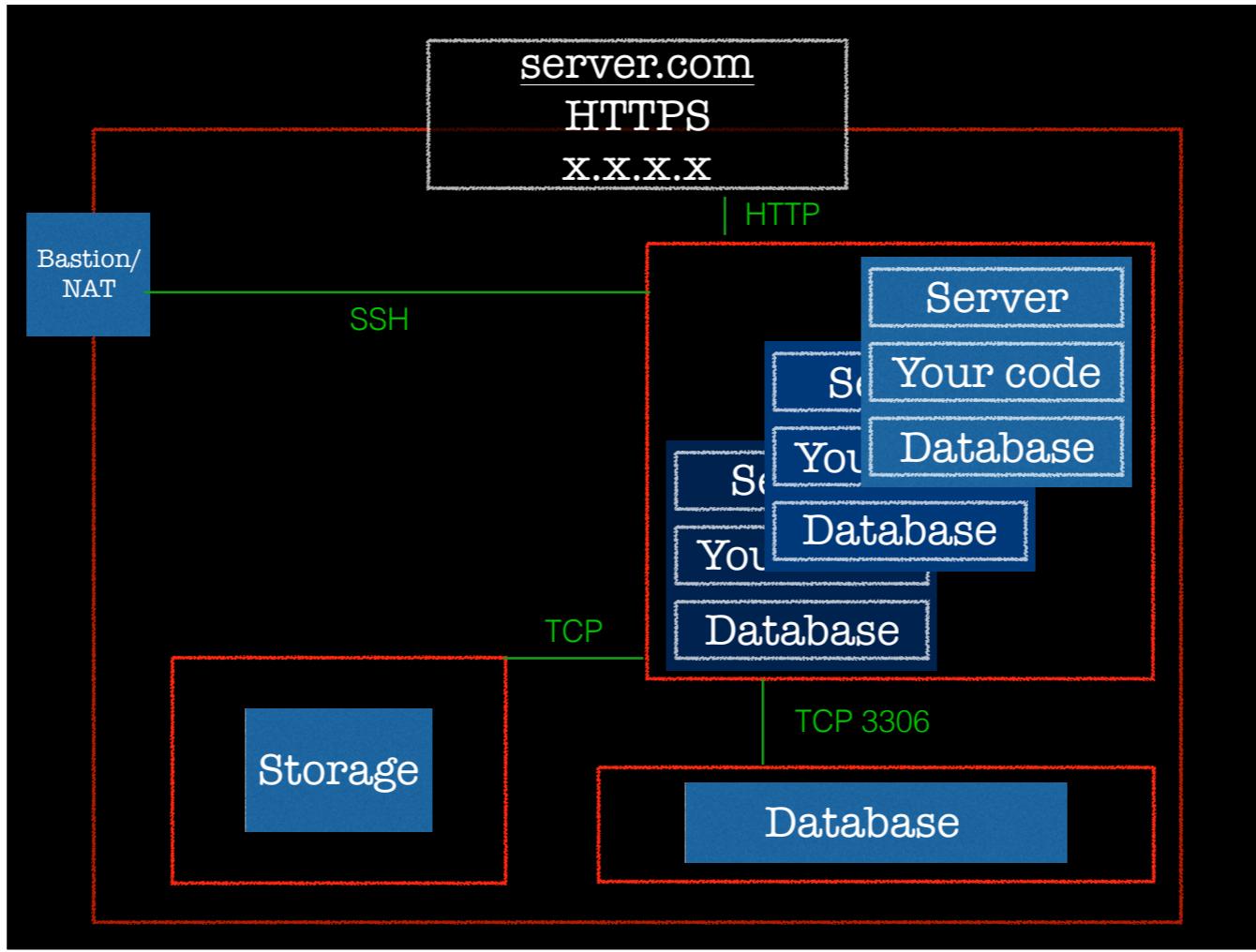
Trusting headers for constructing things



Open redirects

Do not use user input for constructing forwards and redirects

Allows phishing scams by looking like the link is legitimate



Where to put headers and the advantages/disadvantages

HSTS would make sense at load balancer - but no configuration access

Web frameworks add the headers there. No good for static content.

Add to the server nginx/Apache

List of things:

- Secure cookie + httponly
- XSS filter
- HSTS
- SSL check
- Same origin policy
- Content Security Policy
- Sessions

<https://observatory.mozilla.org>



What have I missed?

- Multiple headers
- DoS - head of line blocking
- Brute force
- Hashing is not encryption!
- SSL ciphers
- Certificate pinning
- PCI compliance
- PII
- CVEs and vulnerability disclosures
- File format parsing
- Pen tests
- Physical pen tests
- Reproducible builds
- Guessable URLs
- Examples of exploits!
- SChannel
- Enterprise networks
- CORS
- MitM
- RSA4
- goto fail;
- Malicious actors
- Physical intrusion
- More passwords stolen
- Incident response
- Physical security solutions
- Fuzz testing
- Session hijacking
- Misconfiguration
- And sooooooooooooo much more



So much, so much, my head, this could take HOURS!

And finally..... get a password manager!

**BUT DON'T USE THE BROWSER
PLUGIN!!!**



- **OWASP** <https://www.owasp.org/>
(see Reference section on left)
- **Mozilla Security Blog** <https://blog.mozilla.org/security/>
- **Content Security Policy** <https://scotthelme.co.uk/content-security-policy-an-introduction/>
- **CVEs** <https://cve.mitre.org>
- **SPA Security** <http://www.slideshare.net/carlo.bonamico/angularjs-security-defend-your-single-page-application>
- **HTML5 Security Cheatsheet** https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet

Compromises I had to make on security