SECURITY IN THE AGE OF FRAMEWORKS

LUKA KLADARIC // L@K.HR // @KLL

PARENTAL

ADVISORY

EXPLICIT CONTENT

ONCE UPON A TIME...

WE WROTE OUR OWN CODE

ALL OF IT.

SO WE KNEW **WHAT WAS IN IT.**

WE KNEW **EVERY LITTLE BIT.**

WHO IS THIS GUY?

Luka Kladaric (1985)
started doing web stuff around 1997

javascript before jQuery
server-side with ASP/VBScript (~1999)

ran away to PHP (~2002)

ran away to Python (~2013)

ran away to devops

TODAY WE **"KNOW BETTER"** THAN TO CODE EVERYTHING **FROM SCRATCH**

SO WE RELY ON **FRAMEWORKS**

AND **LIBRARIES**

& A BUNCH OF **3RD PARTY CODE**

WE OUTSOURCE **AUTHENTICATION...**

LOGGING...

DATABASE INTERACTION...

API CONNECTIVITY...

EVERYTHING THAT ISN'T **STRICTLY UNIQUE**
TO THE PROBLEM WE'RE SOLVING

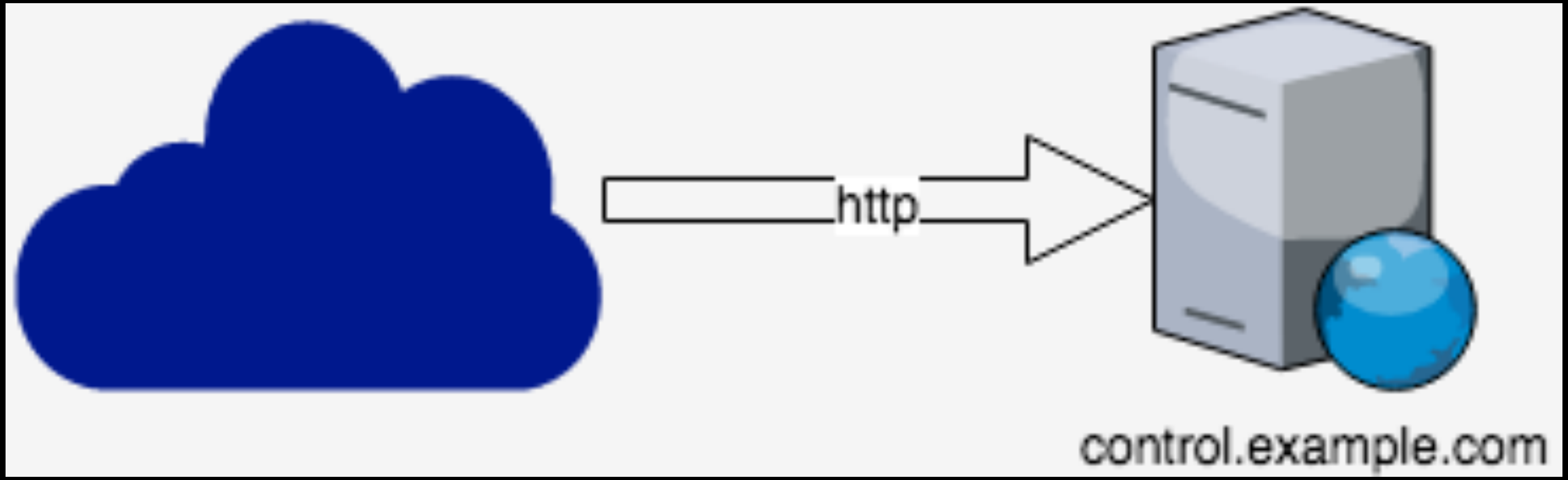
THIS PRESENTS A **RTFM** PROBLEM

HOW DOES THIS PROBLEM MANIFEST ITSELF?

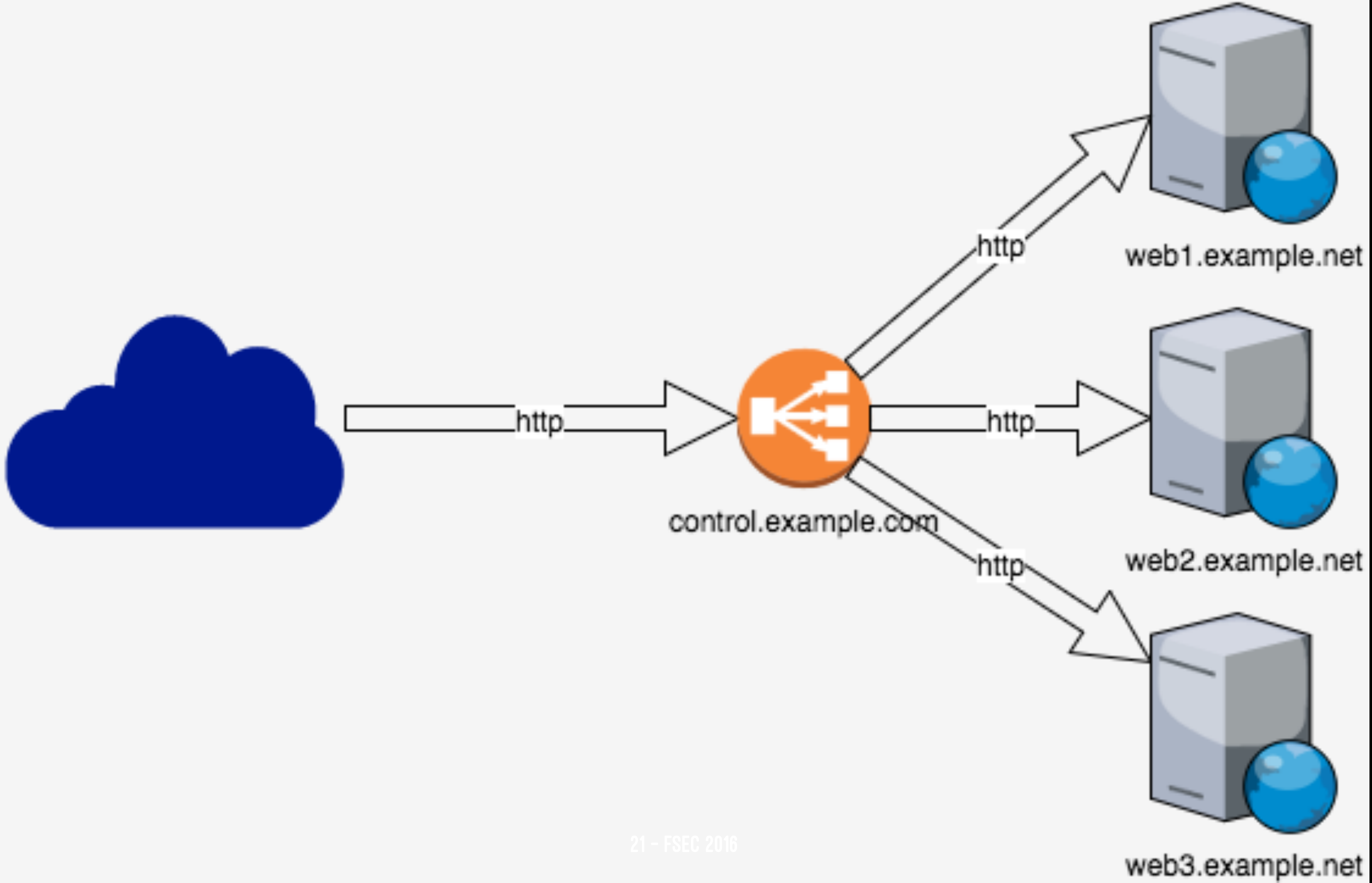
STORY TIME! (DEMO 1)

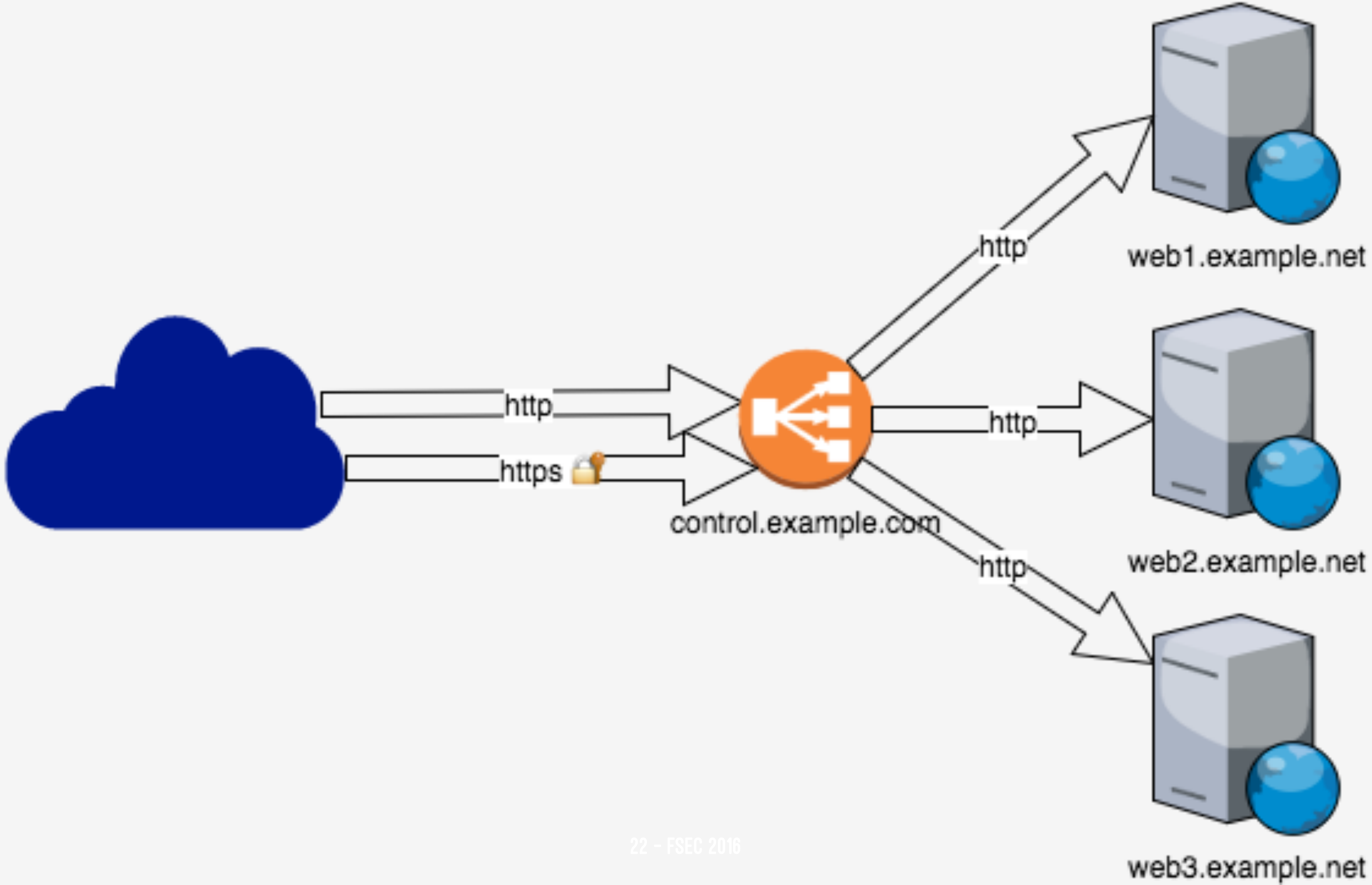
EXAMPLE CO BUILDS THEIR FIRST WEBAPP

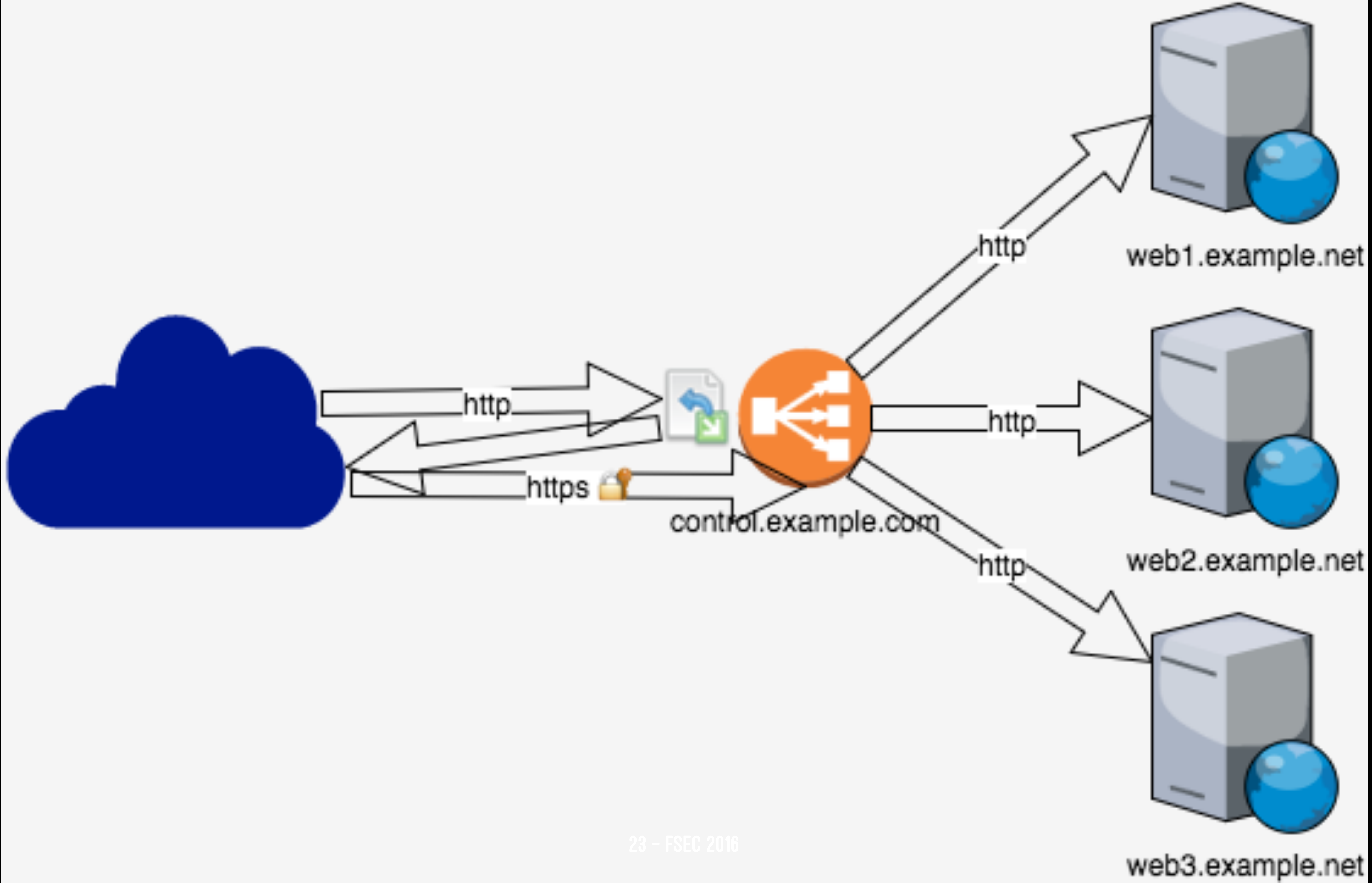
- AN INTERNAL DASHBOARD AT **CONTROL.EXAMPLE.COM**



- AUTHENTICATION?


http
http
http
http
control.example.com
web1.example.net
web2.example.net
web3.example.net


http
https
control.example.com
http
http
http
web1.example.net
web2.example.net
web3.example.net


http
https
control.example.com
http
http
http
web1.example.net
web2.example.net
web3.example.net

BUT IF THE SERVERS ONLY EVER SEE **HTTP**,
HOW WILL THEY KNOW ABOUT **HTTPS?**

SPOILER: THEY WON'T

(NOT BY DEFAULT ANYWAY)

WHAT WE'D LIKE TO SEE

```
$ curl -sIL http://control.example.com
HTTP/1.0 302 Found
Location: https://control.example.com/
Server: BigIP

HTTP/1.1 302 FOUND
Server: nginx/1.4.6 (Ubuntu)
Location: https://control.example.com/login/?next=%2F

HTTP/1.1 302 FOUND
Server: nginx/1.4.6 (Ubuntu)
Location: https://accounts.google.com/o/oauth2/auth
            ?redirect_uri=https%3A%2F%2Fcontrol.example.com%2Flogin%2Fauthorized

[... user logs in & grants example.com access to google account...]

HTTP/1.1 302 FOUND
Server: GSE
Location: https://control.example.com/login/authorized?code=ACCESS_TOKEN
```

```
$ curl -sIL http://control.example.com
HTTP/1.0 302 Found
Location: https://control.example.com/
Server: BigIP

HTTP/1.1 302 FOUND
Server: nginx/1.4.6 (Ubuntu)
Location: http://control.example.com/login/?next=%2F

HTTP/1.0 302 Found
Location: https://control.example.com/login/?next=%2F
Server: BigIP

HTTP/1.1 302 FOUND
Server: nginx/1.4.6 (Ubuntu)
Location: https://accounts.google.com/o/oauth2/auth
            ?redirect_uri=http%3A%2F%2Fcontrol.example.com%2Flogin%2Fauthorized

[... user logs in & grants example.com access to google account...]

HTTP/1.1 302 FOUND
Server: GSE
Location: http://control.example.com/login/authorized?code=ACCESS_TOKEN
```

FIXES:

- RESPECT X-FORWARDED-PROTO HEADER

WATCH OUT FOR VARIOUS COMPONENTS TREATING SECURITY HEADERS DIFFERENTLY

- SSL CONFIG CHANGE TRIGGERS AUDIT

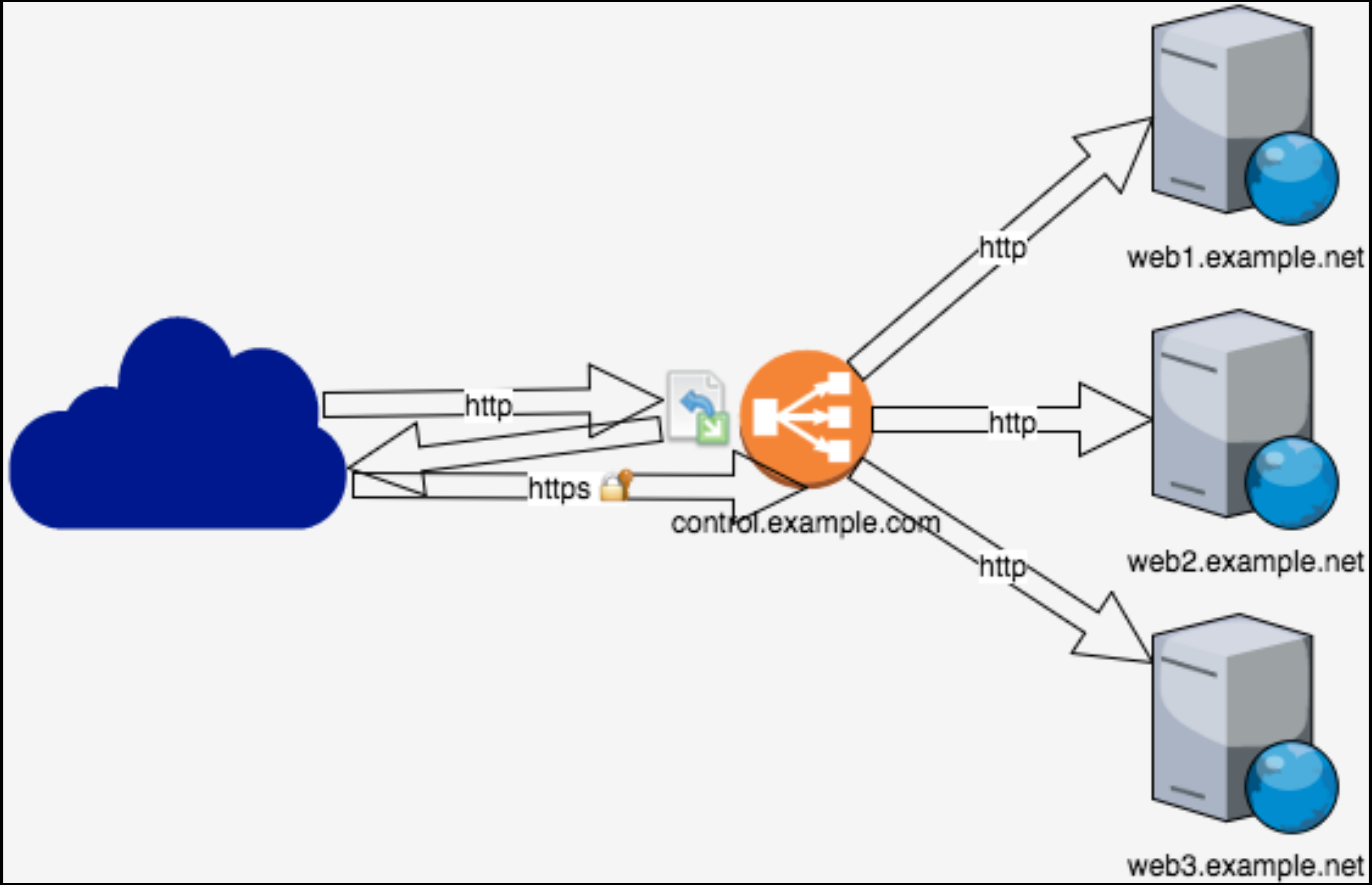
- REMOVE HTTP FROM OAUTH WHITELIST

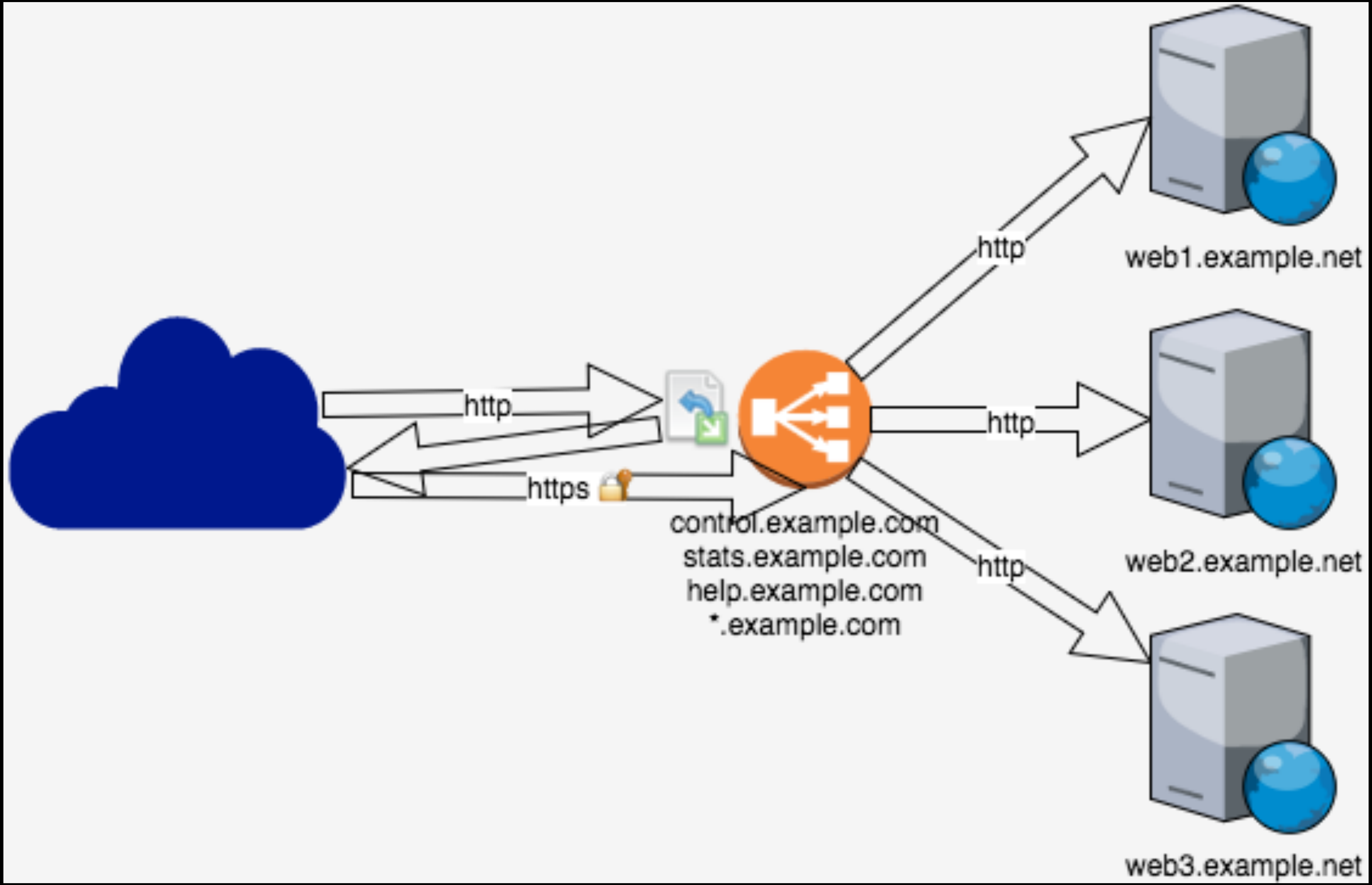
- HSTS HEADERS[1]

[1] HTTP STRICT TRANSPORT SECURITY

DIFFICULT TO CATCH, BROWSERS DON'T REPORT THIS AS BAD BEHAVIOR

STORY TIME! (DEMO 2)


http
https
control.example.com
http
http
http
web1.example.net
web2.example.net
web3.example.net


http
https
control.example.com
stats.example.com
help.example.com
*.example.com
http
http
http
web1.example.net
web2.example.net
web3.example.net

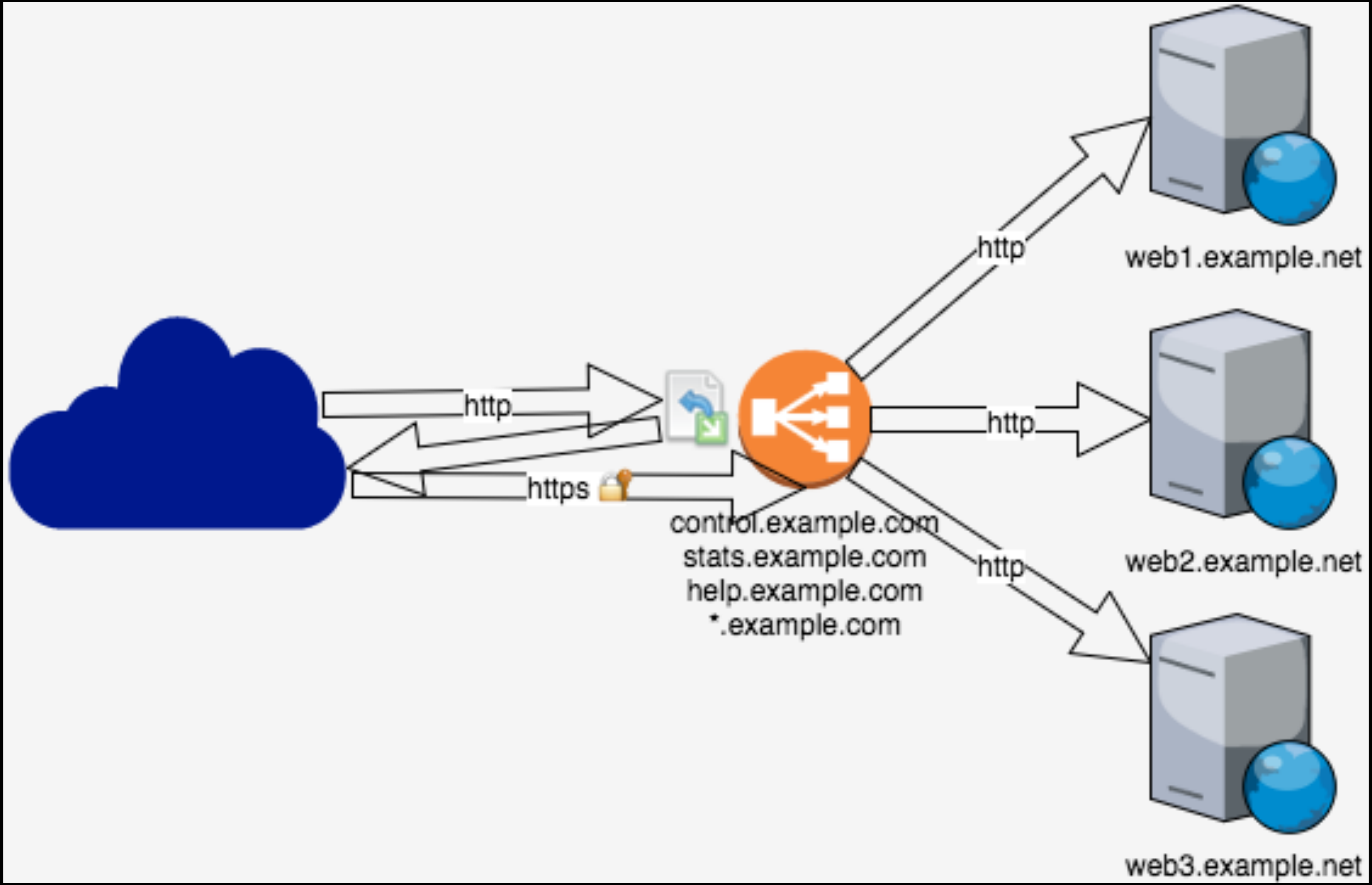
IF YOU'VE EVER SET UP OAUTH

FOR MULTIPLE APPS

AGAINST THE SAME PROVIDER

… DID YOU BOTHER SETTING UP SEPARATE OAUTH CLIENTS?

YEAH.

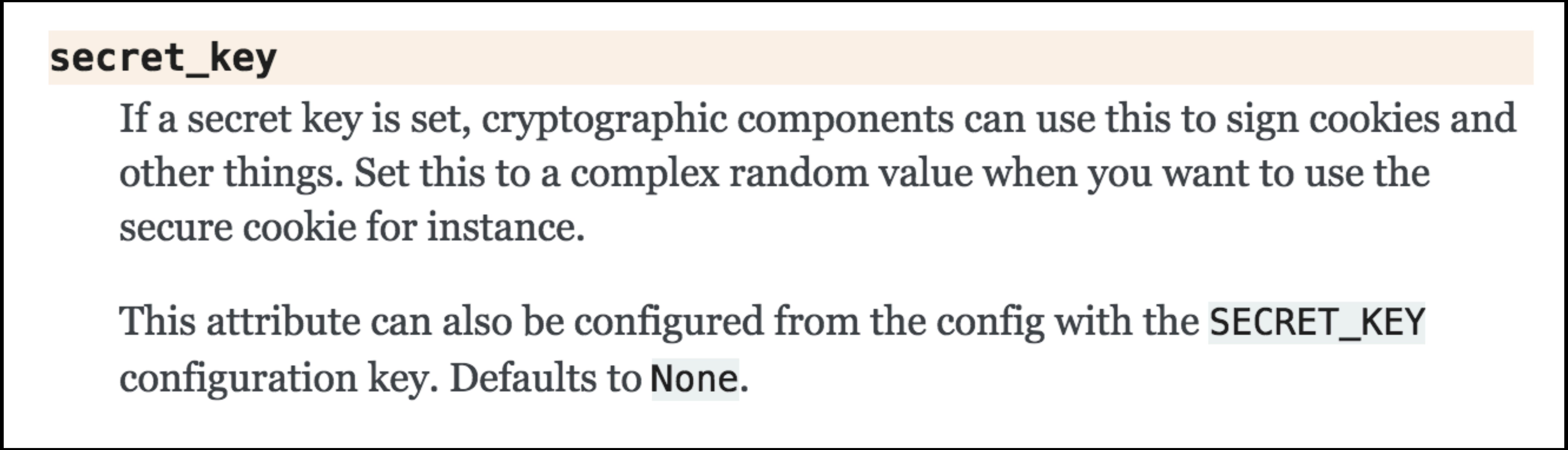

http
https
control.example.com
stats.example.com
help.example.com
*.example.com
http
http
http
web1.example.net
web2.example.net
web3.example.net

STORY TIME! (DEMO 3)

```
Cookie:
session=.eJxljssKgkAYRl8l_rWJl1ScnW
kLC8msaBEik046po7YGF7w3Su3bg7f4vBxR
M1Zl-C6g3BpcVqS6M1xWQOSN4ZqKrJqatMX
y0NLzg.CrvTRg.5bTUddcAEVMFMth_I
uPteZTOOjA; HttpOnly; Path=/
```

HOW IS COOKIE SIGNED

FLASK

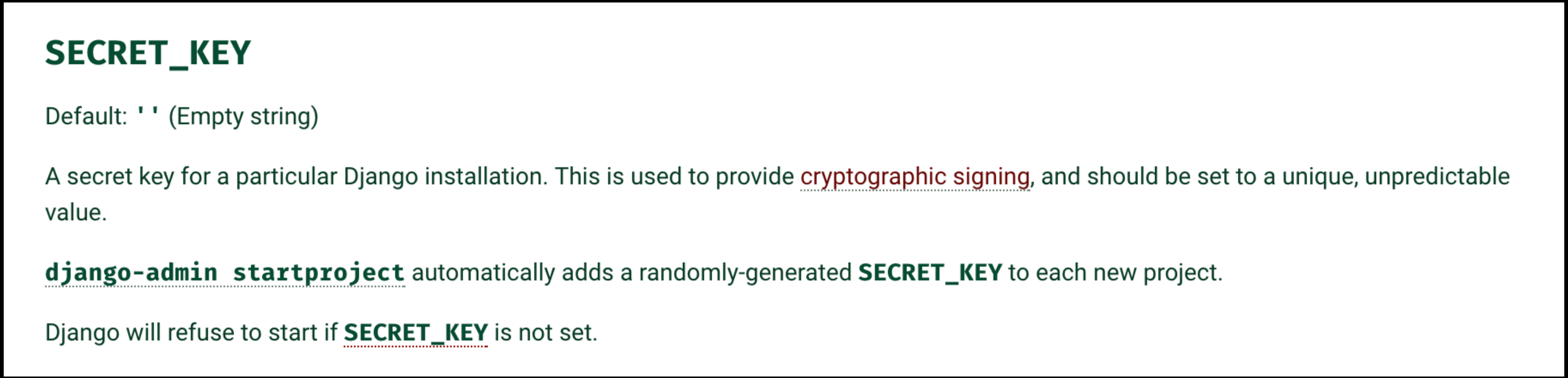


`secret_key`

If a secret key is set, cryptographic components can use this to sign cookies and other things. Set this to a complex random value when you want to use the secure cookie for instance.

This attribute can also be configured from the config with the `SECRET_KEY` configuration key. Defaults to `None`.

DJANGO



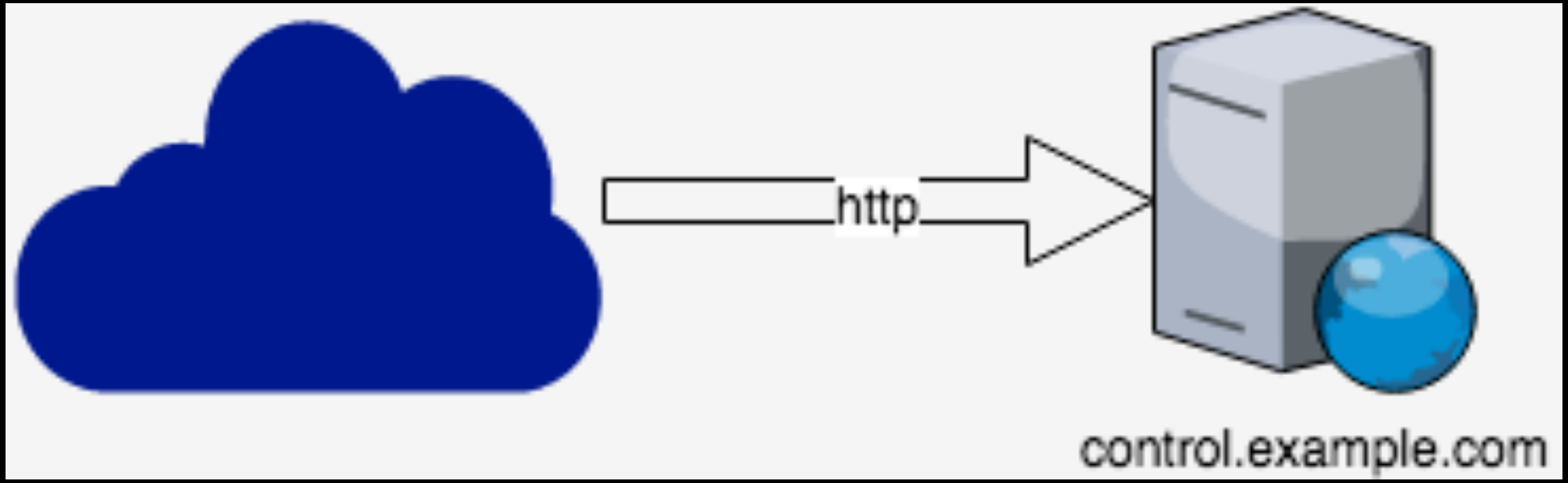
SECRET_KEY

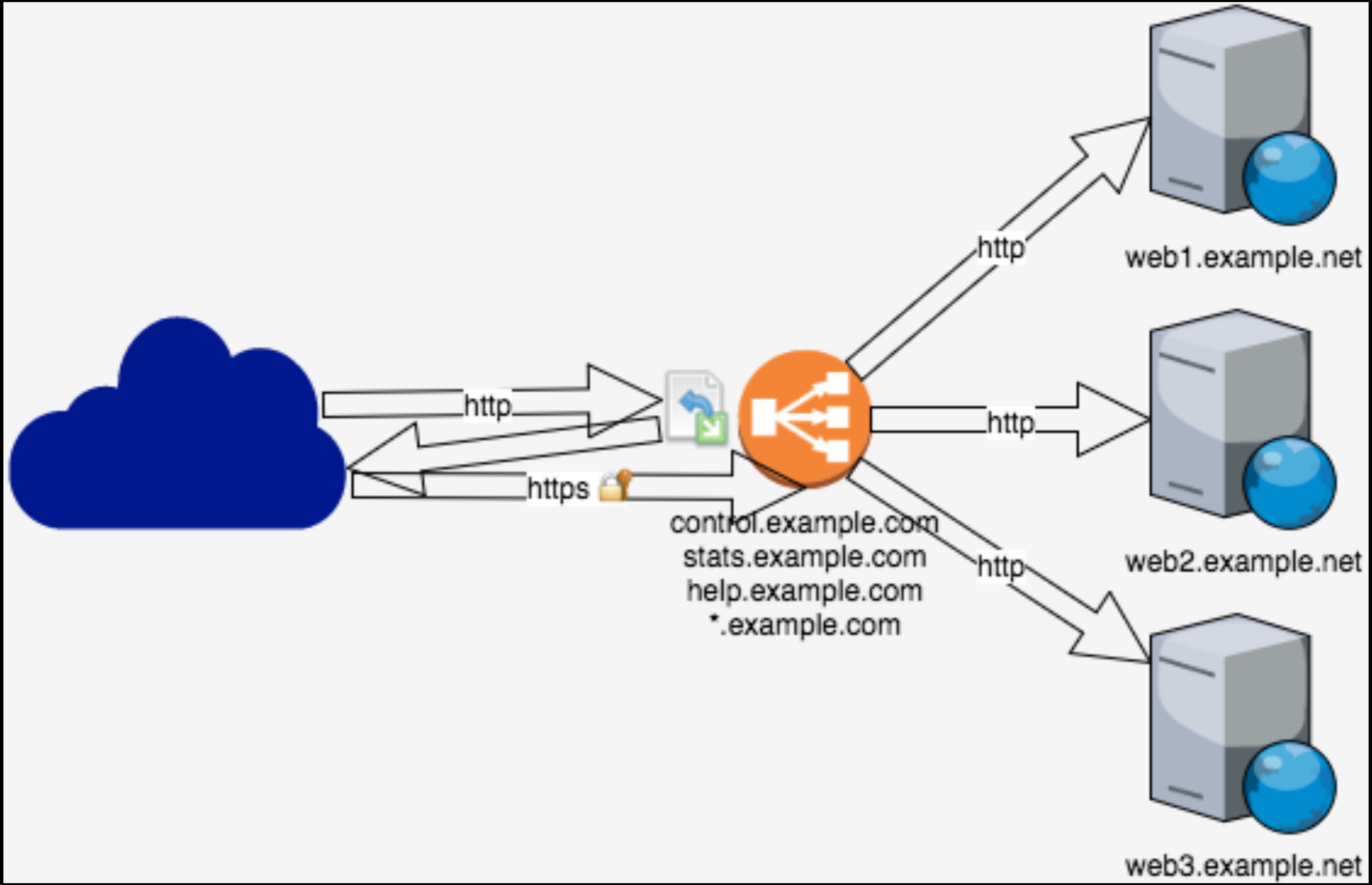
Default: `''` (Empty string)

A secret key for a particular Django installation. This is used to provide cryptographic signing, and should be set to a unique, unpredictable value.

`django-admin startproject` automatically adds a randomly-generated **SECRET_KEY** to each new project.

Django will refuse to start if **SECRET_KEY** is not set.


http
control.example.com


http
https
control.example.com
stats.example.com
help.example.com
*.example.com
http
http
http
web1.example.net
web2.example.net
web3.example.net

SECRET_KEY BEING CONFIGURED BADLY IS NEXT TO IMPOSSIBLE TO CATCH

ALL OF THESE COMBINED MEAN:

SPEAR PHISHING VECTOR

PRIVILEGE ESCALATION

FORCED INSECURE COMMS

ATTACKER TRAFFIC INDISTINGUISHABLE FROM REGULAR TRAFFIC

THE LB HERE ISN'T TO BLAME

PEOPLE HAVE ANCIENT BOOKMARKS POINTING TO HTTP

ALL THINGS BEING EQUAL BUT RUNNING ON A SINGLE MACHINE THIS WOULD STILL BE A VULNERABLE SETUP

HSTS IS A MUST

STORY TIME! (DEMO 4)

CSRF PROTECTION DISABLED

YUP. FOUND THIS ONE TOO.

IT COMES WITH THE THING LEAVE CSRF PROTECTION ALONE

TAKEAWAYS

UNDERSTAND THE STUFF YOU USE BETTER.

BE MORE VIGILANT WITH CODE REVIEWS ON SECURITY-IMPACTING STUFF

BRING IN A FRESH PAIR OF EYES EVERY ONCE IN A WHILE

THIS TALK IS NOT AN ENDORSEMENT
TO ROLL EVERYTHING YOURSELF.

THANK YOU

LUKA KLADARIC // L@K.HR // @KLL