

Production Networking

Chris Timperley · Wednesday, October 21, 2026

Administrivia

- **Homework:**
  - baseline PR will be submitted later tonight
  - we will send new Admin API and Mailpit credentials
  - for now, build telemetry around the ticket purchasing journey
- **Midsemester Grades:**
  - A+ (97%), A (93%), A- (90%), 87% (B+), 84% (B), 80% (B-), 75% (C)
  - boundaries for final grades may shift slightly

🎯 Learning Goals

- Trace a web request through DNS, firewalls, ports, and listeners to a service
- Explain how TLS and certificates protect a connection against interception and server impersonation
- Choose an ACME challenge for issuing and renewing certificates for public or private services
- Explain how a reverse proxy routes requests and centralizes TLS and login checks
- Find unintended access paths and propose fixes using network isolation, authentication, and authorization

Today: How do people reach our services?

https://scottybites.example/menu

FRESH COFFEE. LUNCH BETWEEN CLASSES.

ScottyBites

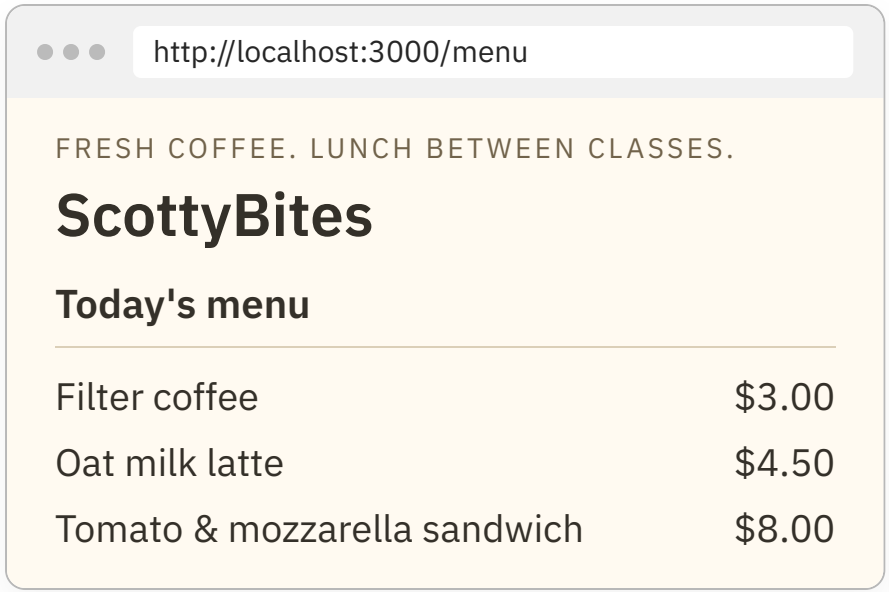
Today's menu

Filter coffee	$3.00
Oat milk latte	$4.50
Tomato & mozzarella sandwich	$8.00



Photo: Clifford / Unsplash

ScottyBites works on our laptop



Frontend: displays the menu in our browser.

Backend · app:3000
Serves the frontend and menu API

SQL queries

Database · db:5432
Stores menu items and prices

```
our laptop
$ docker compose up -d app db
Container scottybites-db-1   Started
Container scottybites-app-1  Started
```

We want to deploy to a server on campus

- A campus-hosted **virtual machine (VM)**

  will run everything

- We register `scottybites.example` through a **registrar** and host its records with a **DNS provider** that has an API

- Asking the university for `scottybites.cmu.edu` would make every record change a request to its administrators. That will matter later

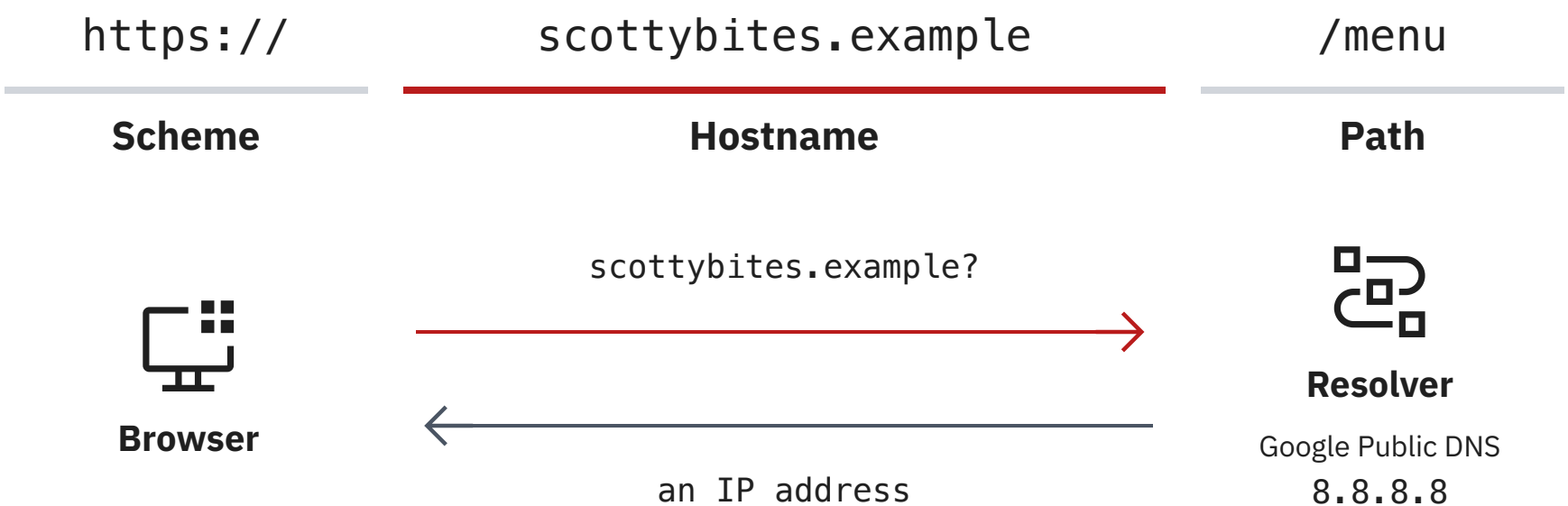


Photo: panumas nikhomkhai / Pexels

Names, Addresses, and Ports

How does a request find the right machine and process?

DNS finds an address for the hostname



DNS (Domain Name System) turns names into addresses. A **resolver** is the DNS service the client asks. The web page itself never passes through DNS.

DNS records store information about names

A **DNS record** is a published entry with a **name**, a **type**, and a **value**.

```
; Name                        TTL  Class Type   Value
scottybites.example.          300  IN    A      192.168.1.23
scottybites.example.          300  IN    AAAA   2001:db8::10
www.scottybites.example.      300  IN    CNAME  scottybites.example.
scottybites.example.          300  IN    TXT    "site-verification=abc123"
```

A: an IPv4 address

AAAA: an IPv6 address

CNAME: another name to look up

TXT: text, often used to verify domain control

TTL 300: cache for up to 300 seconds. **IN:** the Internet class of records.

Where do DNS records come from?

We create DNS records through our DNS provider

Name	Type	Content
*.mycoolwebpage.xyz	A	192.0.2.3
mycoolwebpage.xyz	A	203.0.113.1
sub1.mycoolwebpage.xyz	A	192.0.2.1
www.mycoolwebpage.xyz	A	192.0.2.100

Cloudflare DNS dashboard · Source: Cloudflare

- The provider's **authoritative DNS servers** serve the records we publish.

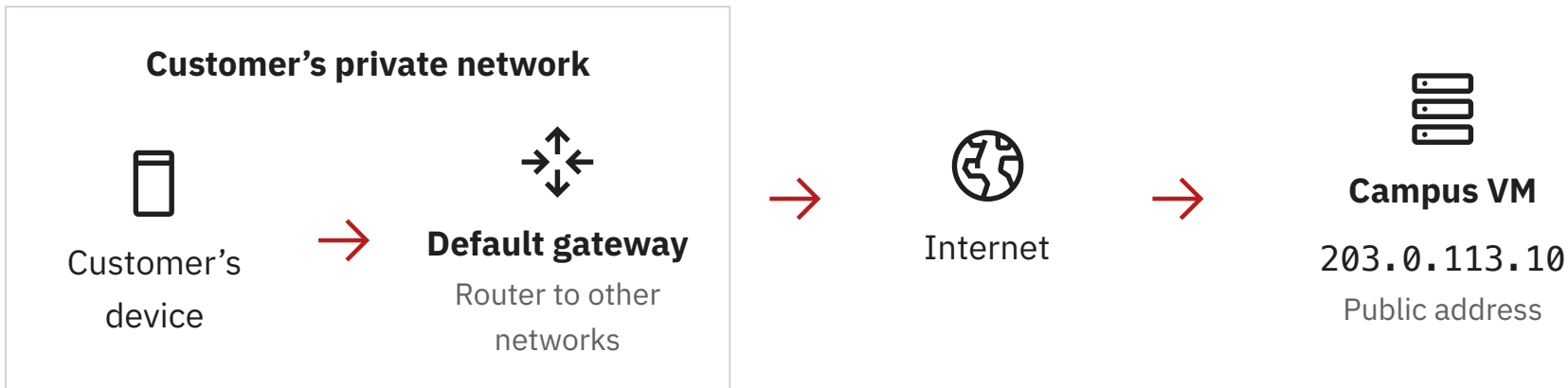
Which IP address should we publish?

One candidate

192.168.1.23

- DNS can return this address. **Can a customer on another network reach it?**

Publish the VM's public address



Private addresses, such as `192.168.1.23`, are reused across networks. Internet routers do not route traffic to them.

```
scottybites.example.    A    203.0.113.10
```

Campus assigns the address; DNS publishes it.

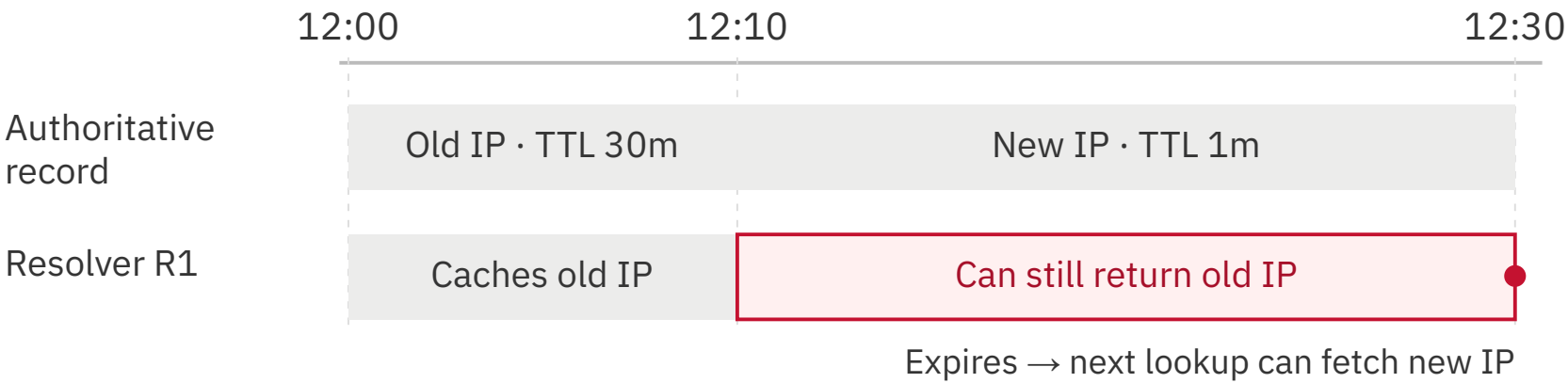
What if our public IP address changes?

- Suppose we host a service at home. Our ISP assigns a **dynamic** public address, which can change; a **static** address stays assigned.
- A **dynamic DNS updater** detects the change and updates the record through the DNS provider's API.

```
Before: home.example.    A    203.0.113.10
After:  home.example.    A    203.0.113.20
```

We updated the record. Can customers still get the old address?

⚠ DNS caches can keep an old answer

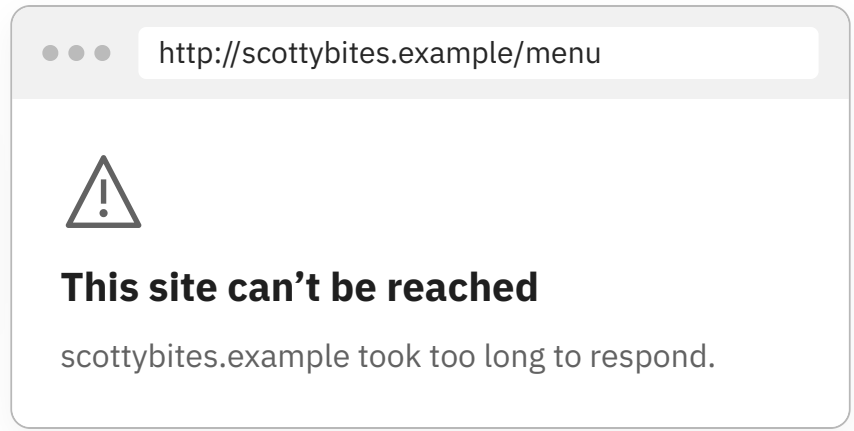


- After our home server's public IP changes, resolvers may return the old IP address until the cached answer's **TTL (time to live)** expires

The name resolves, but the page never loads

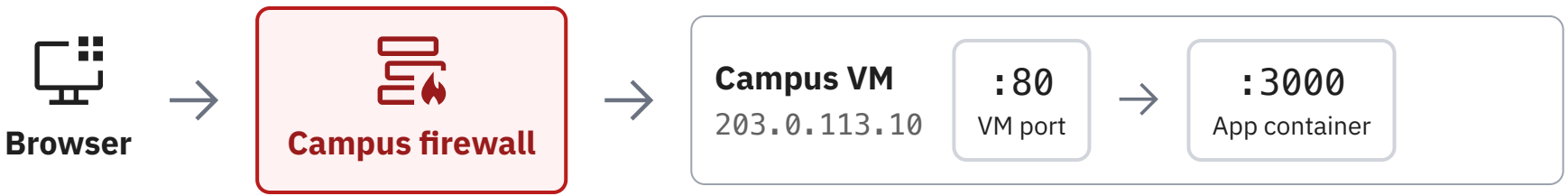
customer's laptop

```
$ dig +short scottybites.example
203.0.113.10
```



- DNS found the right address
- **Something between the browser and the app is still missing**

Firewalls drop traffic we have not allowed

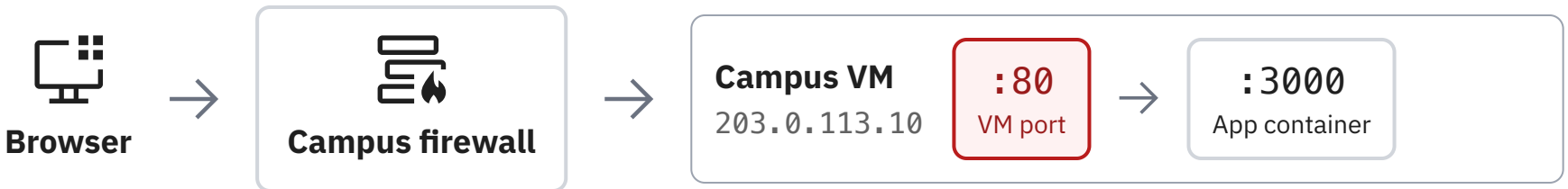


- The campus firewall drops connections that no rule allows (i.e., block by default).
- We ask it to allow **TCP port 80**.

```
# Same policy on a host (UFW)
ufw default deny incoming
ufw allow 22/tcp
ufw allow 80/tcp
```

Why deny everything by default instead of blocking the dangerous ports?

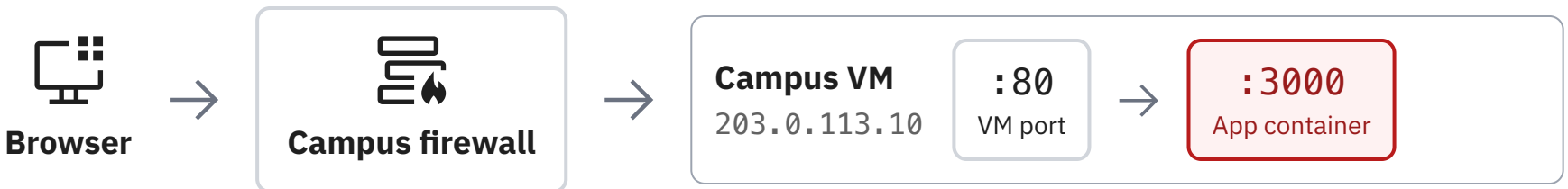
Publish the app's port on the VM



An address picks the machine; a **TCP port** picks the process. Nothing on the VM accepts connections on port 80, so we forward it to the app.

```yaml
services:
  app:
    ports:
      - "80:3000"   # temporary
```

The app must listen beyond its own container



A **listener** is a process waiting for connections at an address and port. The app worked when tested inside its container, but not from outside it:

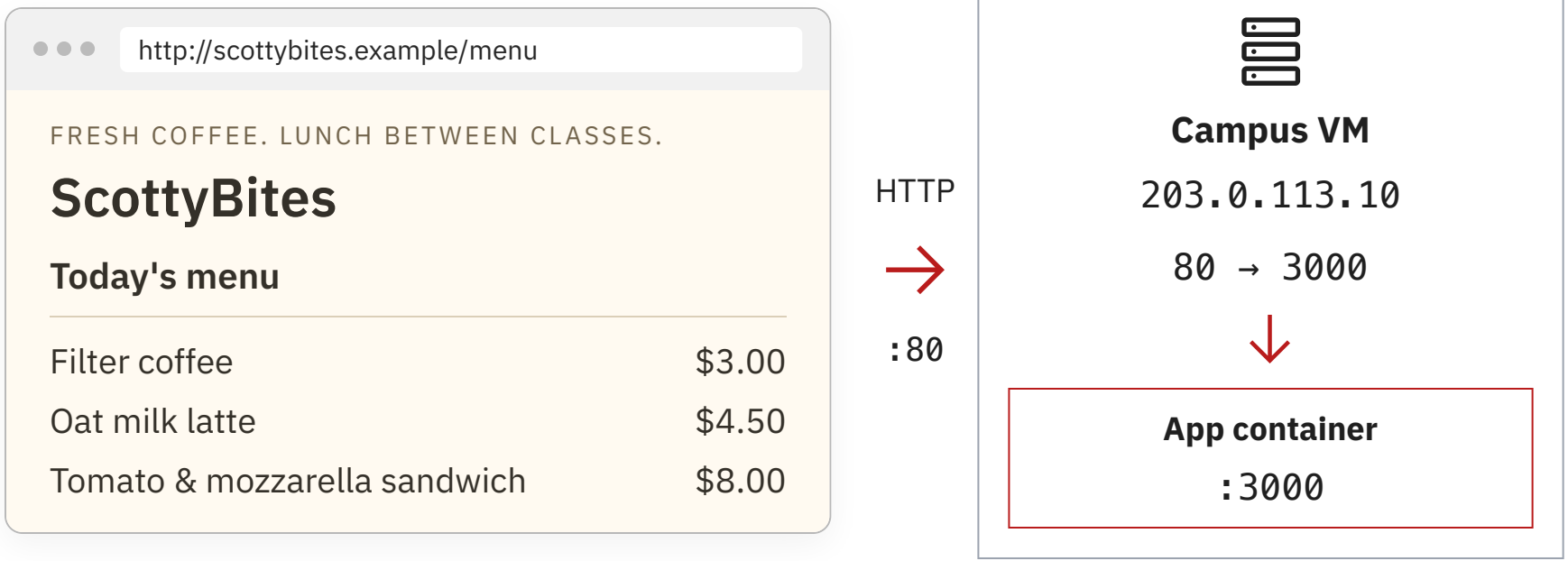
`127.0.0.1:3000`

Only the container itself (**loopback**)

`0.0.0.0:3000`

Connections on any of its interfaces

The menu now loads over HTTP



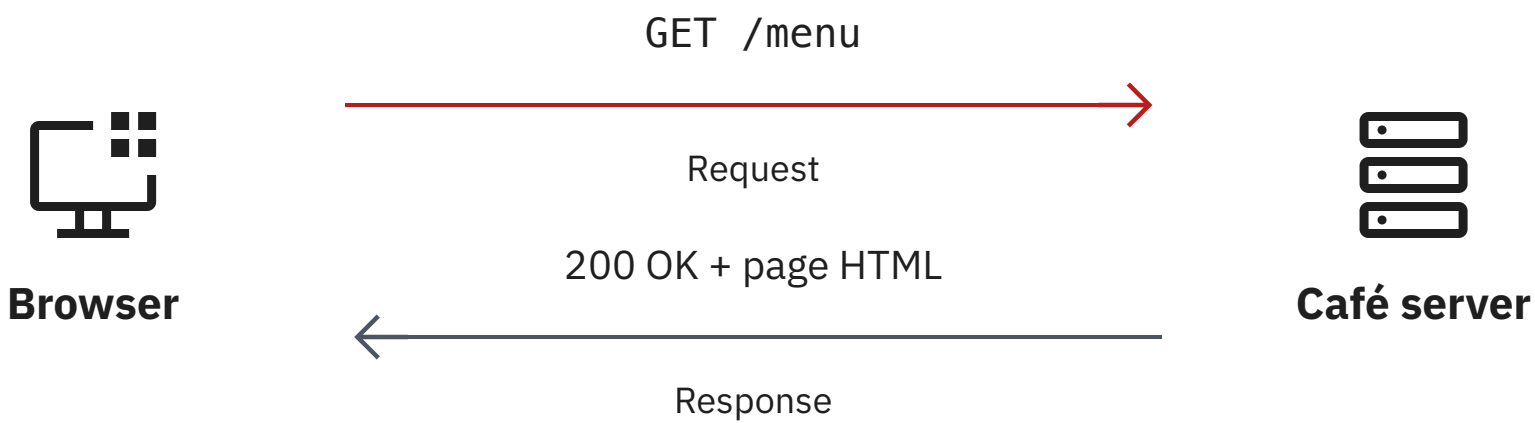
The menu loads. Are we ready for customers?

TLS and Certificates

How does HTTPS verify the server and protect the conversation?

HTTP carries requests and responses

HTTP (Hypertext Transfer Protocol) carries messages between a browser and a web server.



Plain HTTP sends these messages **without encryption**.

An attacker can control the café hotspot



Photo: Clifford / Unsplash

A customer joins a hotspot run by an attacker. Requests to our website pass through it.



Browser
Customer's device

Hotspot
Attacker controls it

Café server
Our website

This is an **on-path attack**, also called a **man-in-the-middle (MITM)** attack.

⚠ Attackers can read and change HTTP

```
POST /login HTTP/1.1
Host: scottybites.example

user=sam&password=hunter2
```

```
<p>Coffee: $3</p>
```

becomes:

```
<p>Coffee: $30</p>
```

- Plain HTTP provides neither **confidentiality** (i.e., hiding messages) nor **integrity** (i.e., detecting changes).
- With traffic captured at that hotspot, an attacker can use tools like **Wireshark** to intercept and read HTTP requests as text.

⚠ Attackers have several ways onto the path



Run the network
A rogue hotspot, often named after the real one, carries every customer's traffic



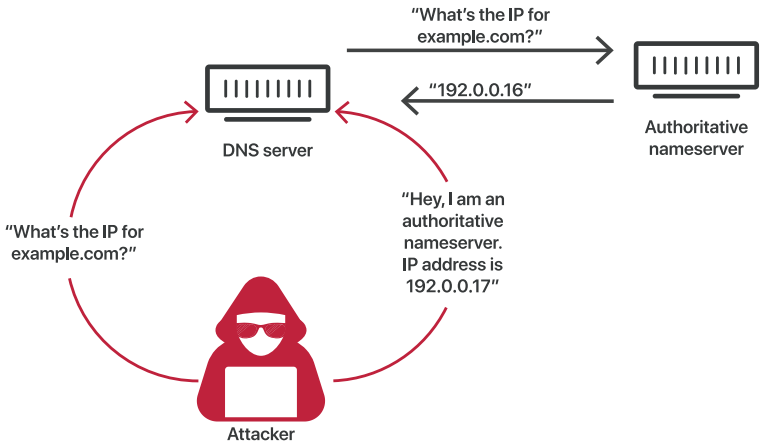
Impersonate the gateway
On a shared network, ARP spoofing claims the router's address, so neighbors send their traffic to the attacker

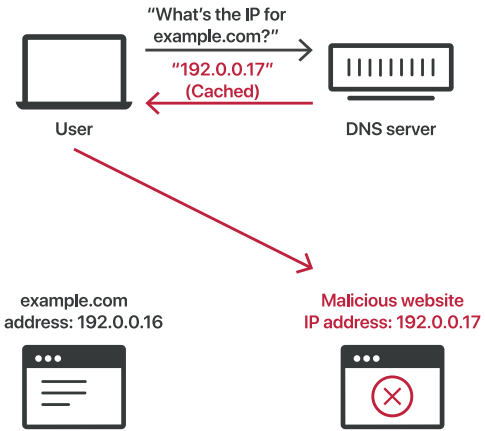


Forge the DNS answer
The browser asks for scottybites.example and receives the attacker's address

The last one needs no position on the path! **It moves the path to the attacker.**

⚠ A poisoned resolver misleads every user

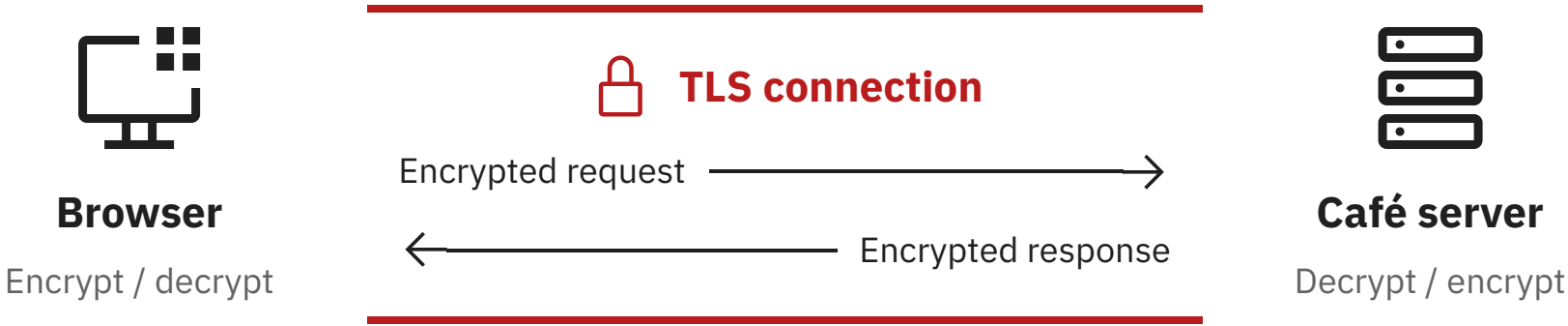




- Plain DNS answers carry no proof of who sent them.
- **The forged answer stays cached until its TTL expires.**

Diagrams: Cloudflare Learning Center

HTTPS protects HTTP traffic with TLS



- **TLS (Transport Layer Security)** encrypts HTTP messages and detects changes in transit. **HTTPS is HTTP over TLS.**
- The hotspot forwards encrypted traffic. It cannot read the HTTP messages or change them undetected.

TLS uses public and private keys

The server has a matching **public/private key pair**.



Public key

Can be shared with anyone.
The browser uses it to check the server's proof.



Private key

Stays secret on the server.
The server uses it to prove it holds this key.

During connection setup, the server proves it holds the private key. Separate shared **session keys** encrypt and decrypt the HTTP messages.

Encryption needs an identity check

An attacker has keys too. Without an identity check, it could set up **two separate encrypted connections**:



Browser
Encrypted connection to attacker

Attacker
Reads, changes, and forwards messages

Café server
Separate encrypted connection

The attacker can read messages because **each connection ends at the attacker**.

Does this public key belong to `scottybites.example`? A **certificate** helps the browser check.

Certificates link hostnames to public keys



scottybites.example

Public key

Issuer’s signature

Validity dates

- The certificate names the **hostname and public key** that belong together.
- A **certificate authority (CA)** signs that claim. Its **digital signature** lets the browser verify the signer and detect changes.
- During the **TLS handshake** (connection setup), the server proves it holds the matching **private key**.

How do I get a certificate?

A certificate authority checks and signs

- A **certificate authority (CA)** is an organization that issues digitally signed certificates
- For our website, it checks that we **control the hostname**, then signs a certificate connecting that name to our public key







ACME automates certificate requests



Generate keys
Keep the private key secret.

Request
Send the hostname and public key.

Prove control
Answer a CA challenge.

Receive
CA returns a signed certificate.

- Manual renewals are repetitive; a missed renewal can take a service offline.
- **ACME** is the protocol software uses to request and renew certificates with a CA.
- A **challenge** is a task the CA checks to confirm we control the name.

ACME Challenges



HTTP-01 · public port 80

Serve a challenge response at a specific HTTP URL. The CA fetches it.



DNS-01 · public DNS

Publish a challenge value in a TXT record. The CA looks it up; the website can stay private.



TLS-ALPN-01 · public port 443

Answer a special TLS challenge. The CA checks it during connection setup.

Which challenge would you use?

1 **The public menu site; ports 80 and 443 are open**

HTTP-01 OR TLS-ALPN-01 — The CA can reach the site either way.

2 **The campus firewall allows only port 443**

TLS-ALPN-01 — Answered on the HTTPS port itself.

3 **Grafana, on a private network with no public address**

DNS-01 — The CA cannot connect, but it can read public DNS.

4 **One certificate for *.scottybites.example**

DNS-01 — The only one of these that issues wildcards.

5 **A private site on scottybites.cmu.edu; record changes need a ticket to campus IT**

NONE AUTOMATES — DNS-01 needs API access to the records.

How does a certificate establish trust?

Trust follows signatures to a known root

The browser checks each issuer's signature until it reaches a **root CA** it already trusts.



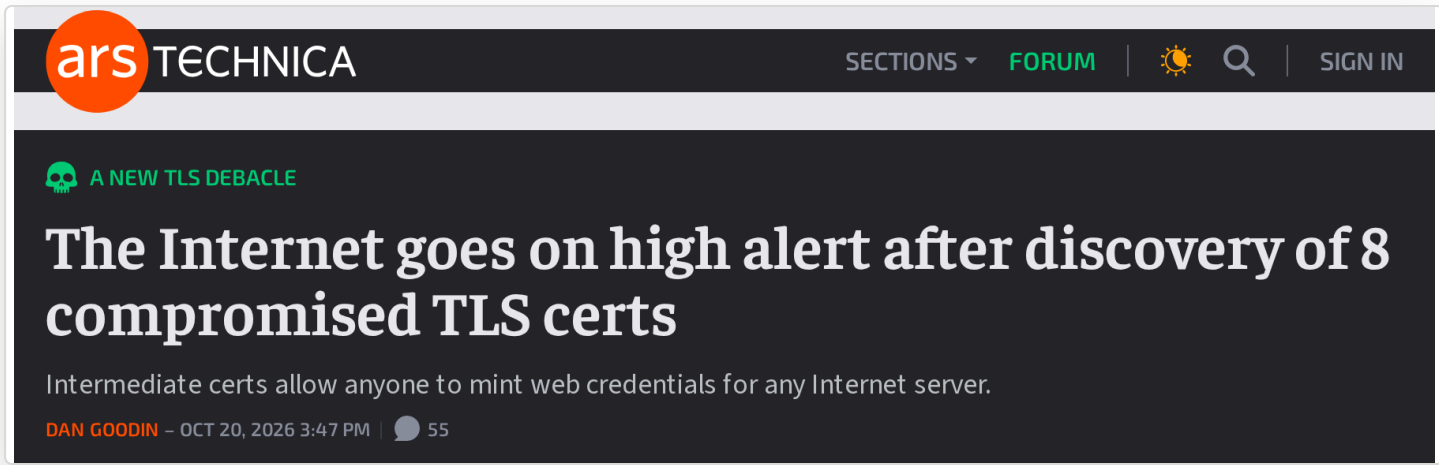
Site certificate
Names scottybites.example

Intermediate CA
Signs the site certificate

Trusted root CA
Signs the intermediate

- The browser or operating system's **trust store** contains the trusted roots.
- The server sends its certificate and the intermediate certificates.

A compromised CA key puts many sites at risk



- Eight compromised eMudhra **intermediate CA keys** could let attackers issue certificates for **websites they did not control**
- Clients stay vulnerable until they **block the affected chains**

Incident confirmation: Firefox Security Team · Mozilla distributed blocks through OneCRL.

The browser checks trust, hostname, and dates

Trust

Signatures lead to a trusted root CA.

Hostname

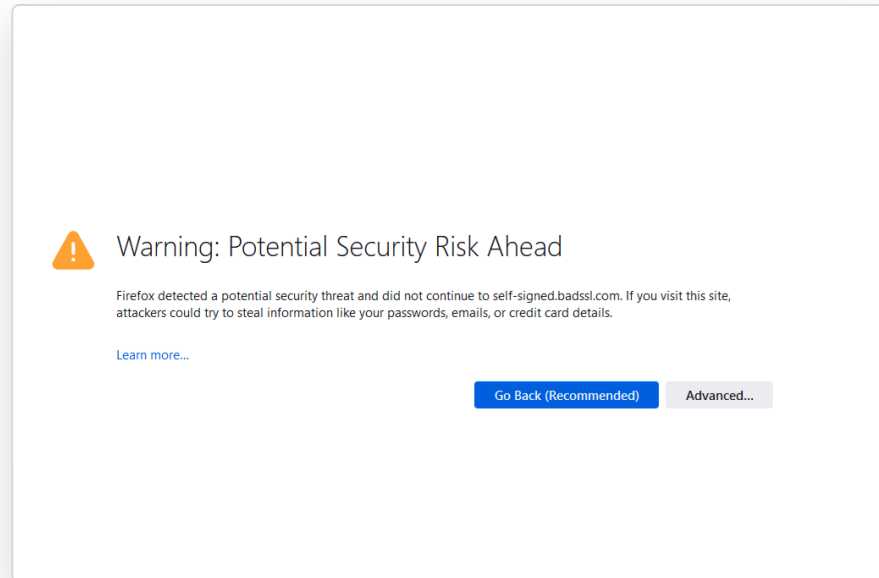
The certificate names scottybites.example.

Validity

Today is within the certificate's dates.

Private-key proof

The server holds the matching private key.



Screenshot: Mozilla Support

If a required check fails, the browser stops the connection.

Can a copied certificate fool the browser?

The hotspot attacker copied our valid certificate for `scottybites.example` and redirects the browser to its own server.

Can the attacker impersonate our website?

No. The certificate is public. The attacker still needs our **private key** to prove it is our server.

What changes if the attacker steals our private key too?

It can present our certificate and prove it holds the key. **These checks can pass.**

DNSSEC lets resolvers reject forged answers



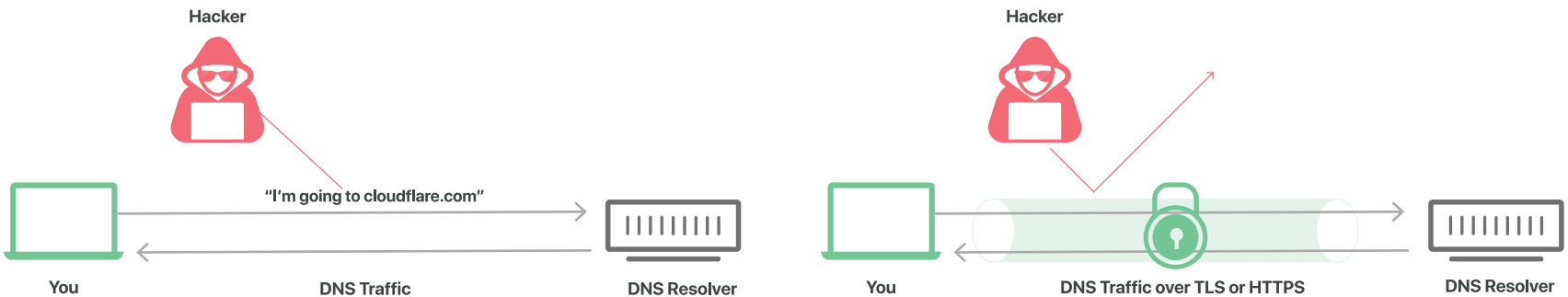
Root
Vouches for the .example key

.example
Authenticates our domain's key

scottybites.example
Signs the A record

- **DNSSEC** adds **digital signatures** to DNS records
- **Validating resolvers** reject forged answers before caching them
- **Answers stay readable:** DNSSEC does not encrypt them

Encrypted DNS hides the lookup



- **DoT** (DNS over TLS) and **DoH** (DNS over HTTPS) encrypt the lookup: the hotspot cannot read or forge it, but **the resolver still sees every name.**

Diagrams: Cloudflare Learning Center

Which protection stops it?

1 **The hotspot reads which names a customer looks up**

DOT / DOH — The lookup travels encrypted to the resolver.

2 **A forged answer tries to poison the resolver's cache**

DNSSEC — A validating resolver rejects it; HTTPS still warns.

3 **The hotspot reads a customer's login password**

HTTPS — Only the browser and server can decrypt it.

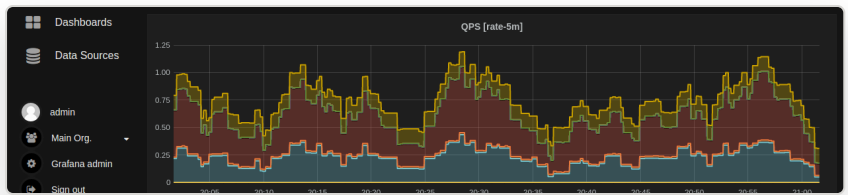
4 **We published the wrong address ourselves**

NONE — Each one faithfully delivers our mistake.

Reverse Proxies

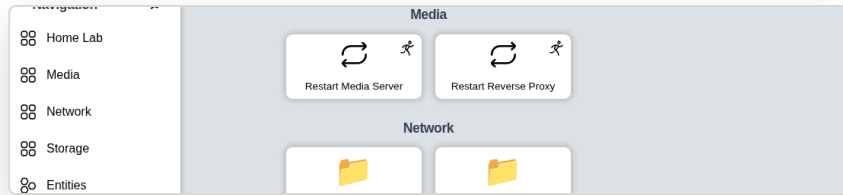
How do several services share one address?

The team also wants to host third-party tools



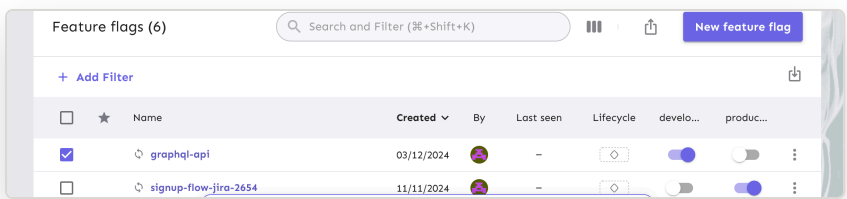
Grafana

Dashboards · staff



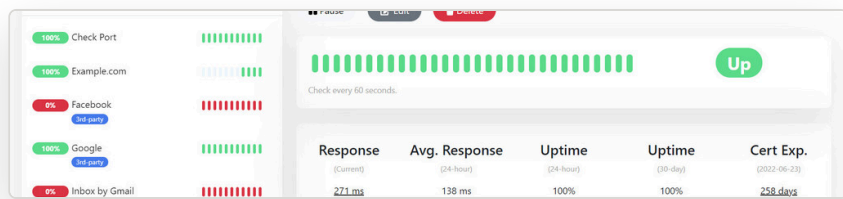
OliveTin

Restart and rollback buttons · on-call staff



unleash

Feature flags · staff UI, app API



Uptime Kuma

Uptime and certificate checks · staff

We could publish each service on its own port

```yaml
services:
  app:
    ports: ["80:3000"]
  grafana:
    ports: ["3001:3000"]
  olivetin:
    ports: ["1337:1337"]
  unleash:
    ports: ["4242:4242"]
  uptime-kuma:
    ports: ["3002:3001"]
```

```
http://scottybites.example
http://scottybites.example:3001
http://scottybites.example:1337
http://scottybites.example:4242
http://scottybites.example:3002
```

What are some possible problems with this setup?

⚠ Each tool must solve these problems alone



Each tool must do HTTPS itself
Many serve only plain HTTP; the rest each need a certificate and key

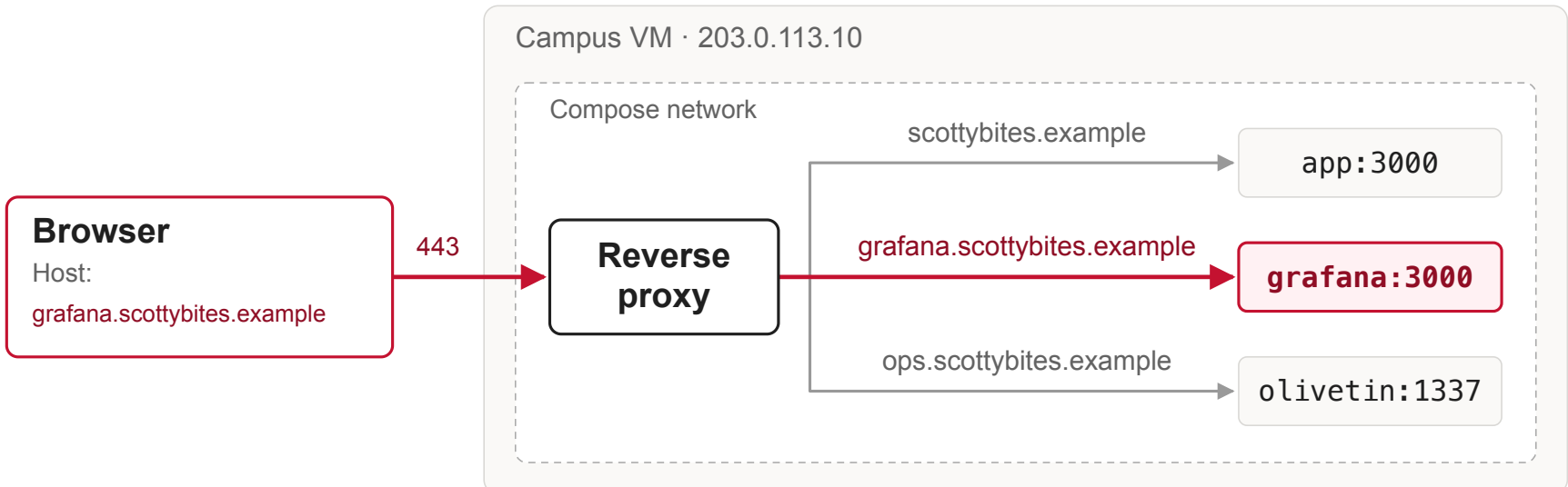


Each port is another door
Each faces the internet behind its own login. No one place to lock down



Each URL is tied to this machine
Moving a tool, or running two copies of the app, changes the URL

A reverse proxy on 443 routes by hostname



- A **reverse proxy** accepts requests for other services. It chooses an **upstream service** by hostname or path and opens its own connection to it.

A proxy is one place to add shared features



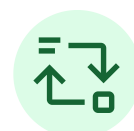
TLS termination
One place holds the certificates and keys.



Load balancing
Spread requests across copies, or move a service, without changing URLs.



Access control
One door to guard, in front of tools we did not write.



Caching and compression
Answer repeated requests without asking the upstream.

Caddy can route by subdomain or by path

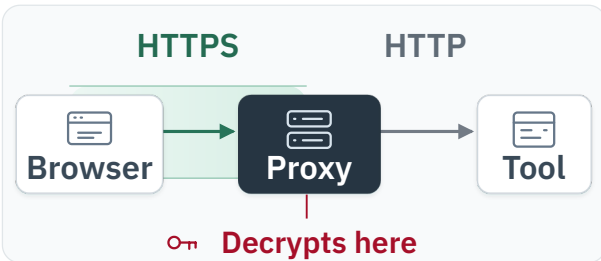
```
scottybites.example {
  reverse_proxy app:3000
}

grafana.scottybites.example {
  reverse_proxy grafana:3000
}

ops.scottybites.example {
  reverse_proxy olivetin:1337
}
```

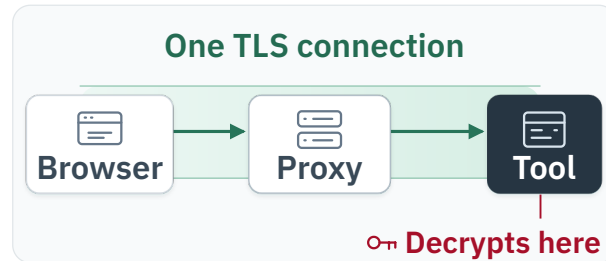
```
scottybites.example {
  handle_path /grafana/* {
    reverse_proxy grafana:3000
  }
  handle_path /ops/* {
    reverse_proxy olivetin:1337
  }
  handle {
    reverse_proxy app:3000
  }
}
```

The proxy must decrypt to act on the request



TLS termination (our setup)

- Proxy holds the keys and decrypts
- Can route by **path** and check **logins**
- Upstream traffic is plain HTTP



TLS passthrough

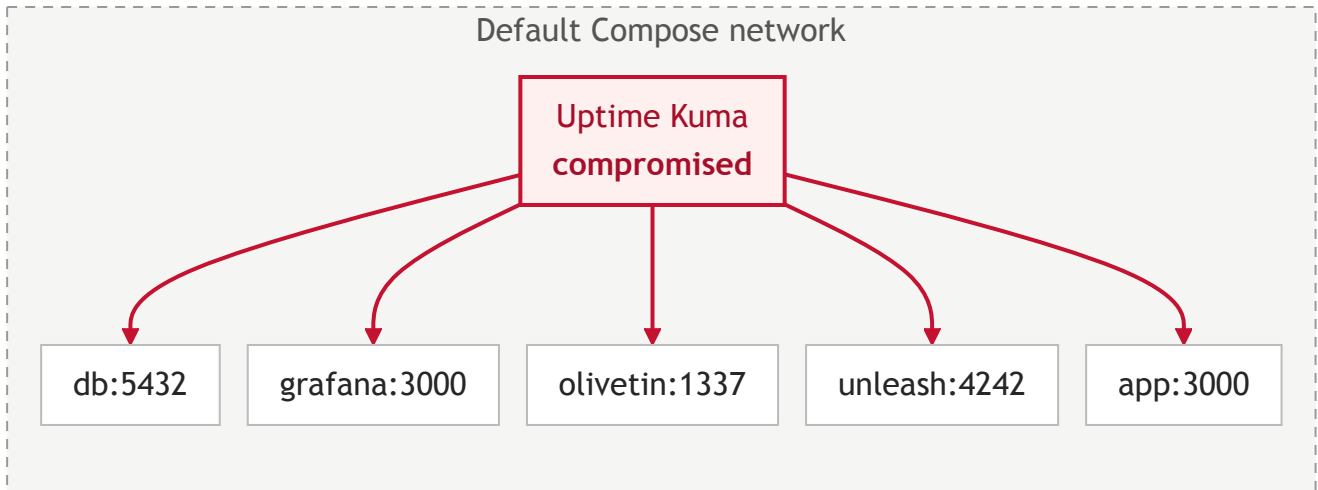
- Routes by TLS server name only
- Each tool keeps its own key
- No path routing, no proxy login

Funnel all traffic through the proxy (80/443)

```yaml
services:
  proxy:
    image: caddy:2
    ports:
      - "443:443"                          # only expose proxy
      - "80:80"                            # redirect HTTP->HTTPS
    volumes:
      - ./Caddyfile:/etc/caddy/Caddyfile:ro
      - caddy_data:/data                   # persist certs
  app:                                     # no ports
  db:                                      # no ports
  grafana:                                 # no ports
  # ...
volumes:
  caddy_data:
```

Should every tool be able to reach every other?

⚠ A compromised tool reaches every service



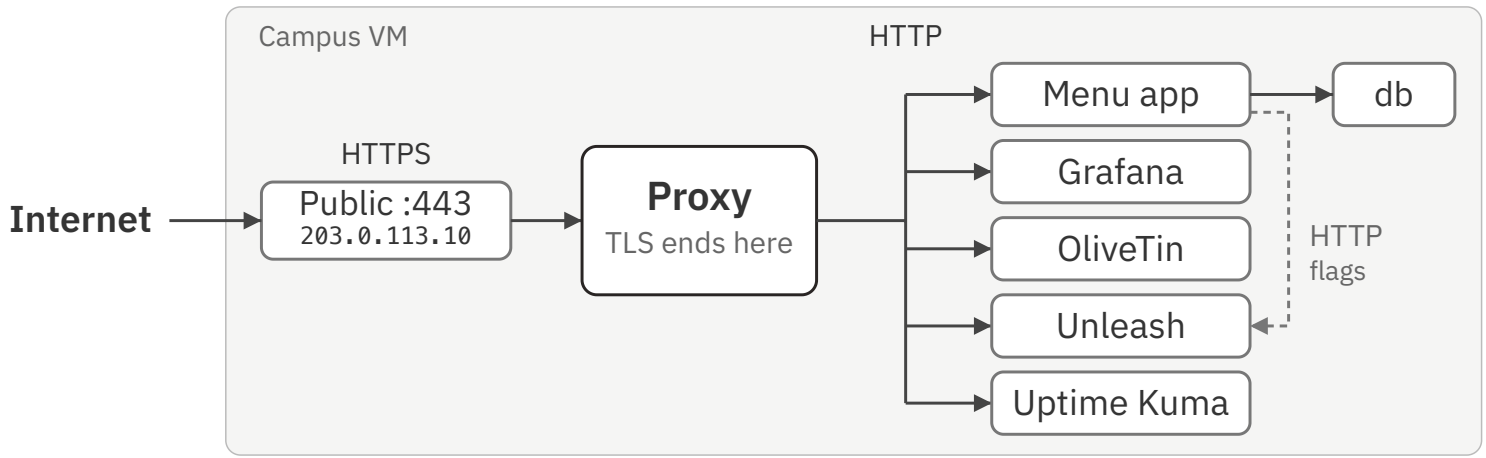
- Compose puts every service on **one default network**
- Any container can connect to any service, without passing through the proxy

Separate networks shrink the damage

```yaml
services:
  proxy:
    networks: [web, grafana, kuma]
  app:
    networks: [web, backend]
  db:
    networks: [backend]
  grafana:
    networks: [grafana]
  uptime-kuma:
    networks: [kuma]
  # ...
```

- Each tool shares a network **only with the proxy**
- Only the app reaches `db`
- A compromised Uptime Kuma reaches only the proxy

Example: TLS protection ends at the proxy



- In this setup, the proxy forwards requests to services over plain HTTP
- Calls between services also use HTTP (e.g., app reads flags from `unleash:4242`)
- The proxy holds the TLS private keys for all public hostnames

Private Access

How do we keep internal tools for staff only?

Discuss: Is each tool's own login enough?

Service	Who should reach it	Who can reach it now
Grafana	Staff	Anyone on the internet, then its login
OliveTin	On-call staff	Anyone on the internet, then its login
Unleash	Staff (admin UI), the app (API)	Anyone on the internet, then its login
Uptime Kuma	Staff	Anyone on the internet, then its login

Is that acceptable? What would you change?

⚠ Assume public services will be found

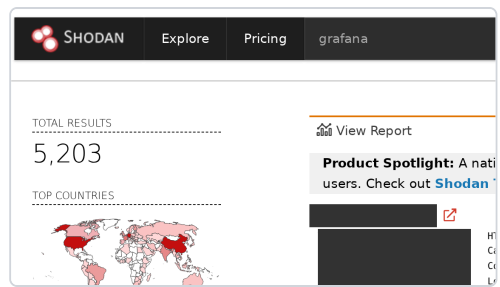
Scanning

Scanners such as Nmap, ZMap, and masscan probe public IP addresses and ports.

```
PORT      STATE SERVICE    VERSION
22/tcp    open  ssh        OpenSSH 6.
| ssh-hostkey:
|   1024 ac:00:a0:1a:82:ff:cc:55:99:d
|   2048 20:3d:2d:44:62:2a:b0:5a:9d:b
|_  256 96:02:bb:5e:57:54:1c:4e:45:2f
80/tcp    open  http       Apache htt
|_http-server-header: Apache/2.4.7 (U
|_http-title: Go ahead and ScanMe!
31337/tcp open  tcpwrapped
```

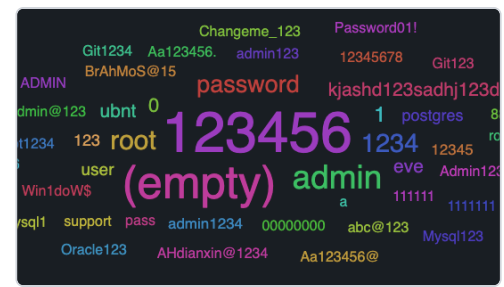
Indexing

Shodan and Censys make responding services searchable.



Automated attacks

Bots try default passwords and known exploits against the services they find.



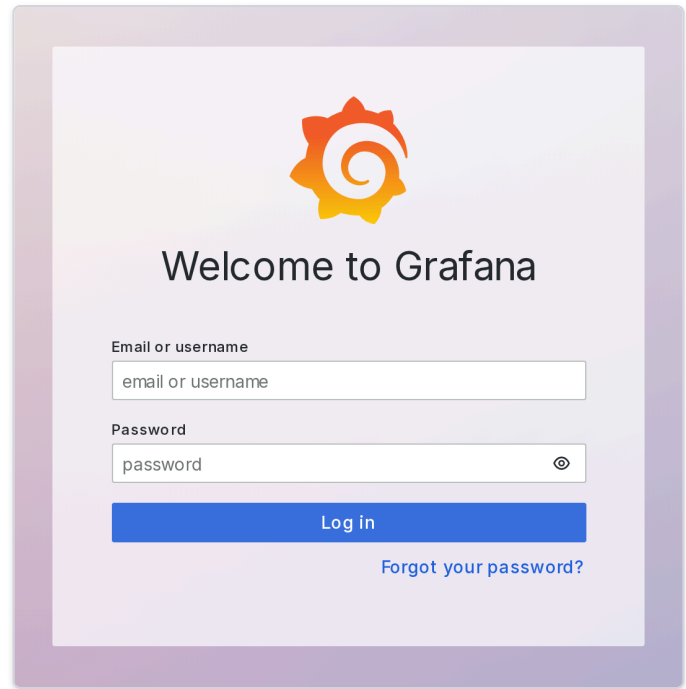
A login in every tool has many weak points

We could rely on each tool's own login: four tools, four login systems.

Uneven protection
Some lack MFA, or any login at all.

Too many accounts
A leaked, reused password opens several.

Missed revocations
Leavers must be removed from every tool.



`grafana.scottybites.example`

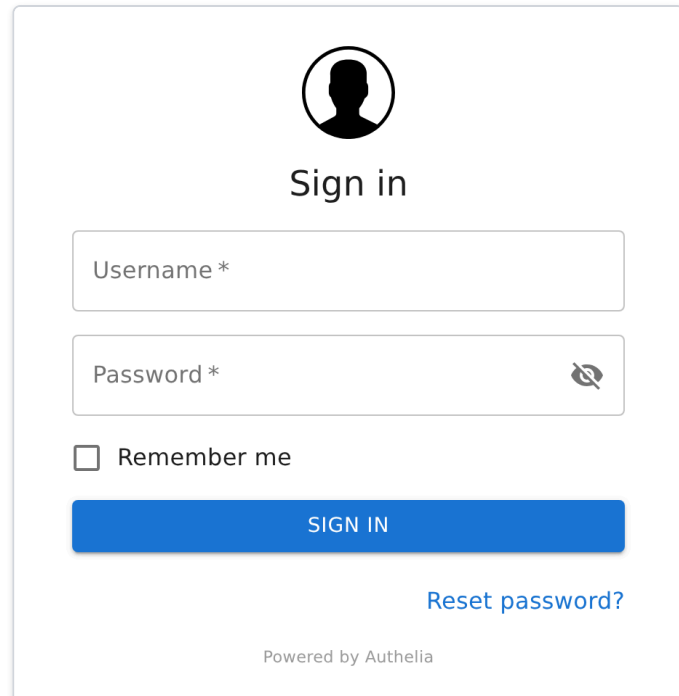
Single sign-on gives every tool one login

With **single sign-on (SSO)**, one **identity provider** checks who you are, and every tool trusts its answer.

One strong login
MFA and password rules cover every tool.

One account per person
Staff remember one password, not four.

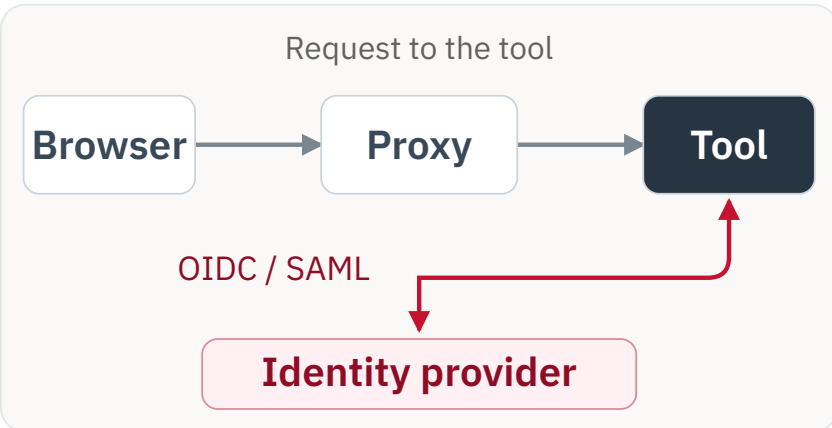
One revocation
Disable one account to lock every tool.



`auth.scottybites.example`

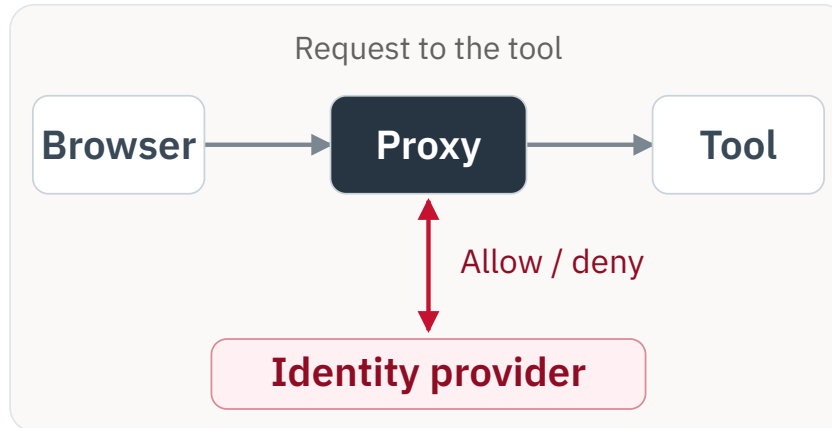
Tools hand login to SSO in two ways

The tool asks



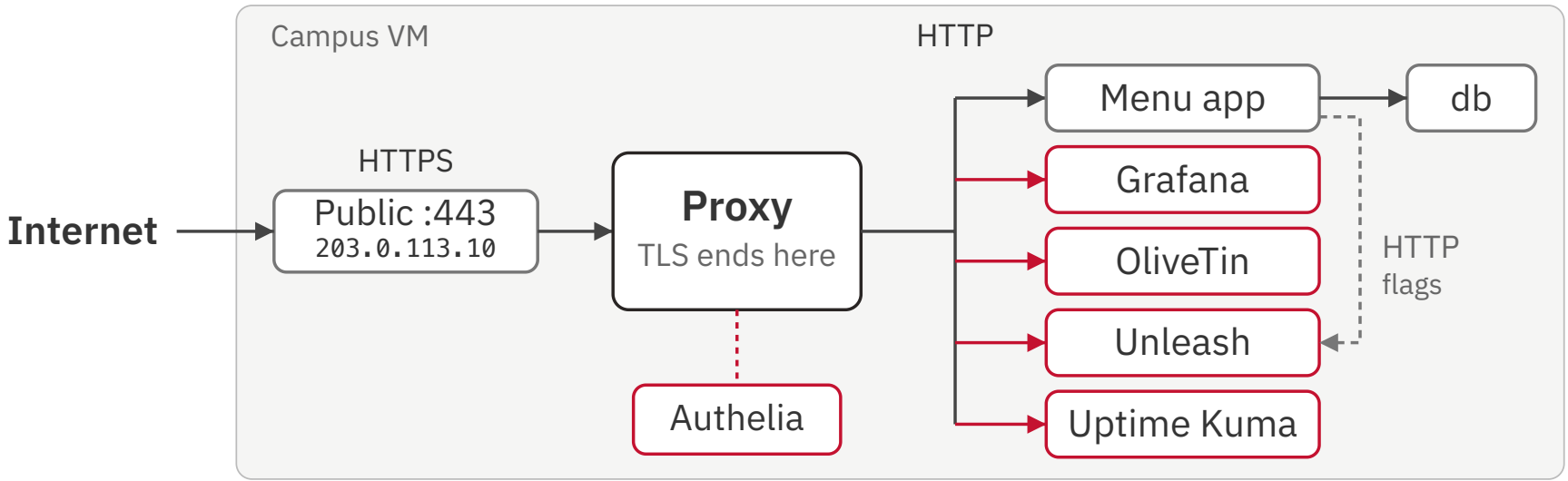
The tool sends you to the identity provider to sign in. It checks the result and logs you in.

The proxy asks



The proxy checks access with the identity provider before forwarding each request. The tool needs no built-in SSO support.

Authelia checks each request for the proxy



Authelia checks whether the request is allowed. The proxy redirects users to sign in, or forwards the request with the username in a header such as `Remote-User`.

A tool such as Grafana can trust that header and skip its own login.

Authelia rate-limits and logs every login

```
regulation:
  max_retries: 3
  find_time: 2m
  ban_time: 5m
```



3 failed attempts
Within 2 minutes.

5-minute ban
Sam cannot sign in.

Retry allowed
The ban ends automatically.

Authelia · login log (excerpt)

```
01:05:47 level=error msg="Unsuccessful 1FA authentication attempt by user 'sam'"
```

Authelia controls who can open each tool

For our ScottyBites server, **OliveTin** provides restart and rollback buttons.

Authelia: may you open the tool?

Checks whether you are in the on-call group before letting you reach OliveTin.

OliveTin: may you run this action?

Must check whether you may roll back a deployment before running the command.

Authelia does not check permissions for individual operations inside the tool.

Anyone on the internet can still reach the login page.

A VPN lets only members connect

A **virtual private network (VPN)** joins devices into a private network over encrypted tunnels across the internet. A service that listens only on that network cannot be reached by strangers.



Gateway VPN

Every device connects to one VPN server, which admits it to the network behind it.

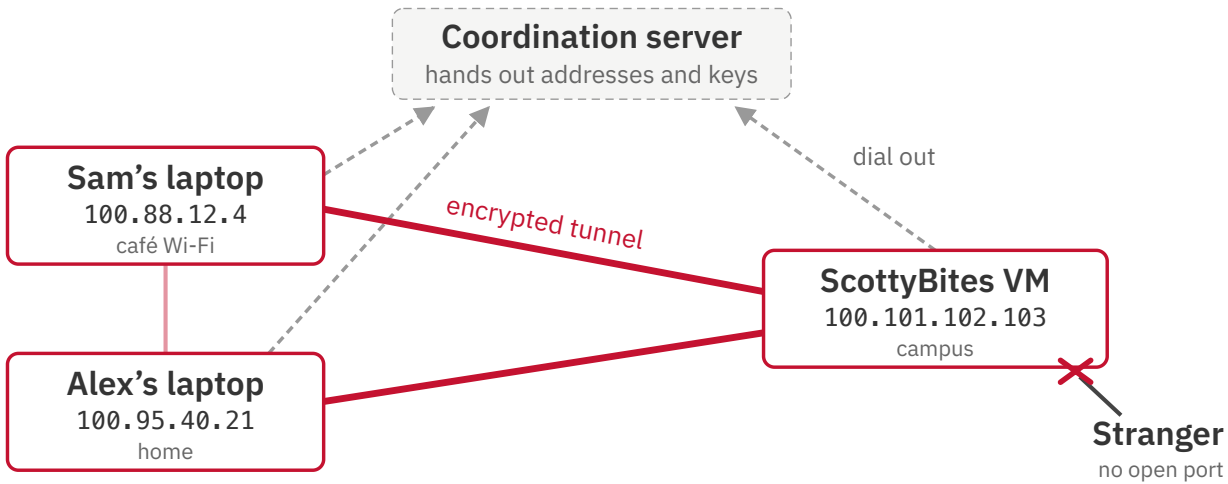


Mesh VPN

Devices connect directly to each other. Tailscale, NetBird, and ZeroTier work this way.

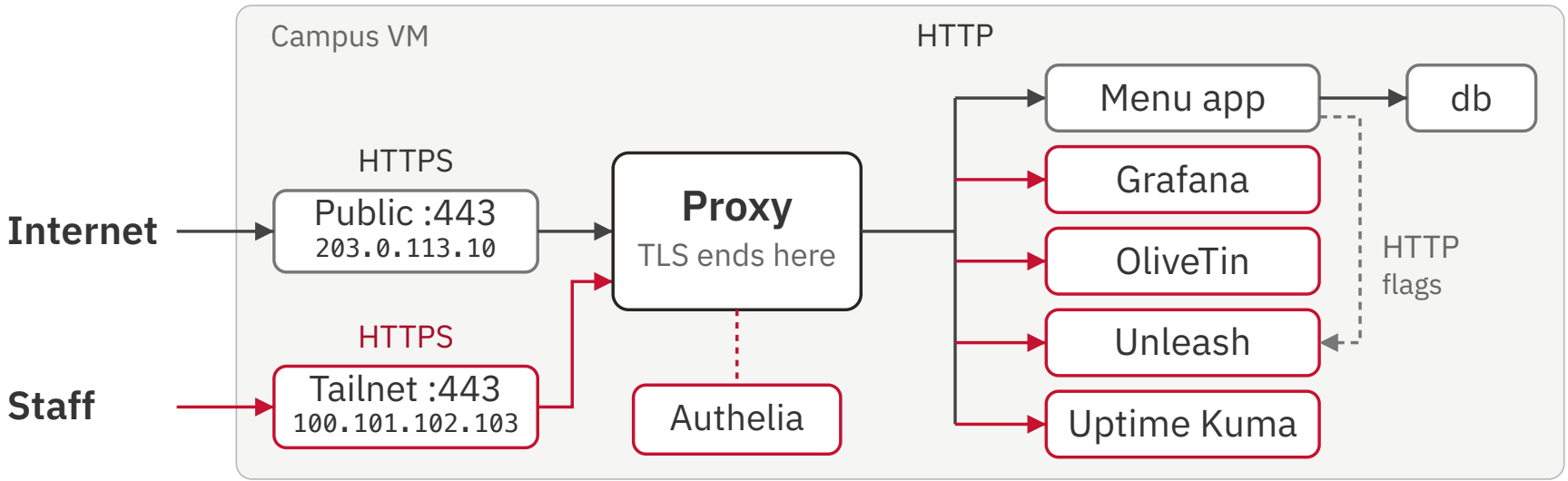
A tailnet keeps internal tools off the internet

A **tailnet** is one team's private network in Tailscale, a mesh VPN.



Addresses come from `100.64.0.0/10`. Tailnet DNS resolves `grafana.` and friends; their public records are removed.

Internal names answer only on the tailnet



The proxy gains a **second listener** on its tailnet address for the internal names. Staff still log in through Authelia; strangers cannot even reach the login page.
DNS-01 lets these private sites renew their certificates.

Allow only the connections each group needs

An **access control list (ACL)** specifies who may connect to which devices and ports. This example uses Tailscale's newer **grants** syntax.

```
{
  "groups": {
    "group:staff": ["sam@scottybites.example", "alex@scottybites.example"],
    "group:oncall": ["sam@scottybites.example"]
  },
  "tagOwners": { "tag:vm": ["group:oncall"] },
  "grants": [
    { "src": ["group:staff"], "dst": ["tag:vm"], "ip": ["tcp:443"] },
    { "src": ["group:oncall"], "dst": ["tag:vm"], "ip": ["tcp:22"] }
  ]
}
```

- Staff → HTTPS; on-call staff → SSH
- Other connections are denied

Break into the ScottyBites deployment



Outsider

On the internet. Not on the tailnet; no account.



Alex’s laptop

Malware on a staff laptop. It can use Alex’s signed-in session.



Compromised tool

You run code inside Uptime Kuma.



1 · Find one path

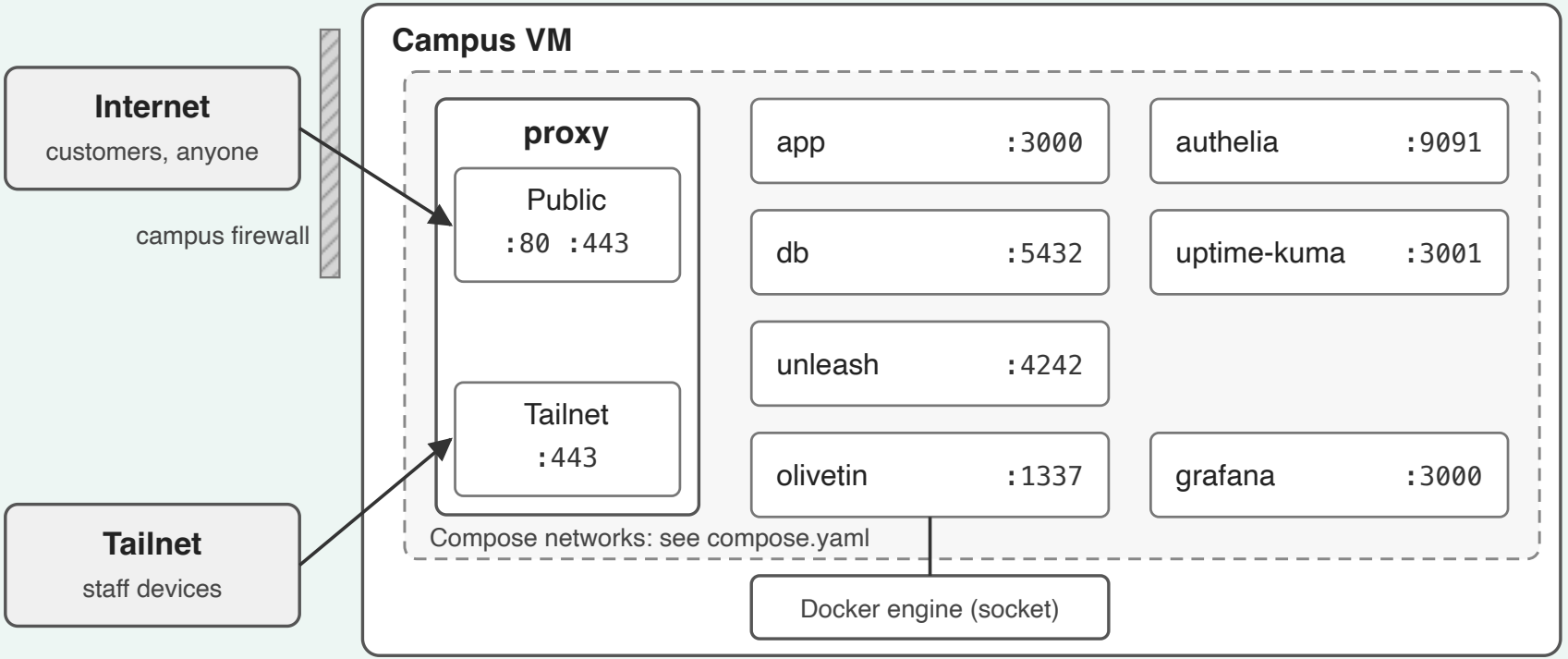
Get access to Grafana, OliveTin, or the database that your role should not have. Show where you start, how you reach it, and what you can do.



2 · Stop that path

Propose one change. Keep access for the people and services that need it.

The “finished” deployment



The deployment's configuration (excerpts)

compose.yaml

```yaml
services:
  proxy:
    ports:
      - "203.0.113.10:80:80"
      - "203.0.113.10:443:443"
      - "100.101.102.103:443:8443"
    networks: [web, auth, grafana, ops, flags, kuma]
  authelia:
    networks: [auth]
  app:
    networks: [web, backend, flags]
  db:
    networks: [backend]
  grafana:
    networks: [grafana]
  olivetin:
    networks: [ops]
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
  unleash:
    networks: [flags]
  uptime-kuma:
    networks: [kuma, web, backend, grafana, ops, flags]
```

Proxy routes

```
Port 80 (both): redirect to HTTPS

Public · 203.0.113.10:443
  scottybites.example
    → app:3000
  grafana.scottybites.example
    → grafana:3000

Tailnet · 100.101.102.103:443
Every request is checked by Authelia
  grafana.… → grafana:3000
  ops.…     → olivetin:1337
  flags.…   → unleash:4242
  status.…  → uptime-kuma:3001
```

Login rules (Authelia)

```
Top to bottom; unmatched → denied
  *.scottybites.example → group staff
```

Each check answers a different question

Question	Path we found	Fix
Can this device connect?	The outsider reaches Grafana through the public listener	The public listener serves only the menu
Who is this?	The outsider sends `Remote-User: admin`, and Grafana believes it	The proxy replaces the header on every listener
May they do this?	Alex's session presses "roll back"	`ops.`: allow on-call, deny everyone else
What else can it reach?	Kuma connects straight to the database, Grafana, and OliveTin	Kuma shares a network only with the proxy

No check substitutes for another. Each fix answers one question.

Test each fix, then ask what a takeover gives

Grafana now accepts `Remote-User` only from the proxy. Does the outsider's forged header still work?

- Yes: that request arrives **from** the proxy
- Also replace the header in the proxy, or remove the public route

OliveTin mounts the Docker socket. What does taking over OliveTin give an attacker?

- Control of the Docker engine, which is root on the VM
- Keep the socket only where an action truly needs it

🚀 Today, we learned how to ...

- Trace a web request through DNS, firewalls, ports, and listeners to a service
- Explain how TLS and certificates protect a connection against interception and server impersonation
- Choose an ACME challenge for issuing and renewing certificates for public or private services
- Explain how a reverse proxy routes requests and centralizes TLS and login checks
- Find unintended access paths and propose fixes using network isolation, authentication, and authorization