ORIGIN SERVER VS CDN

Prepared by Edie Sibanda

Prepared for Basic security shielding

Project Overview

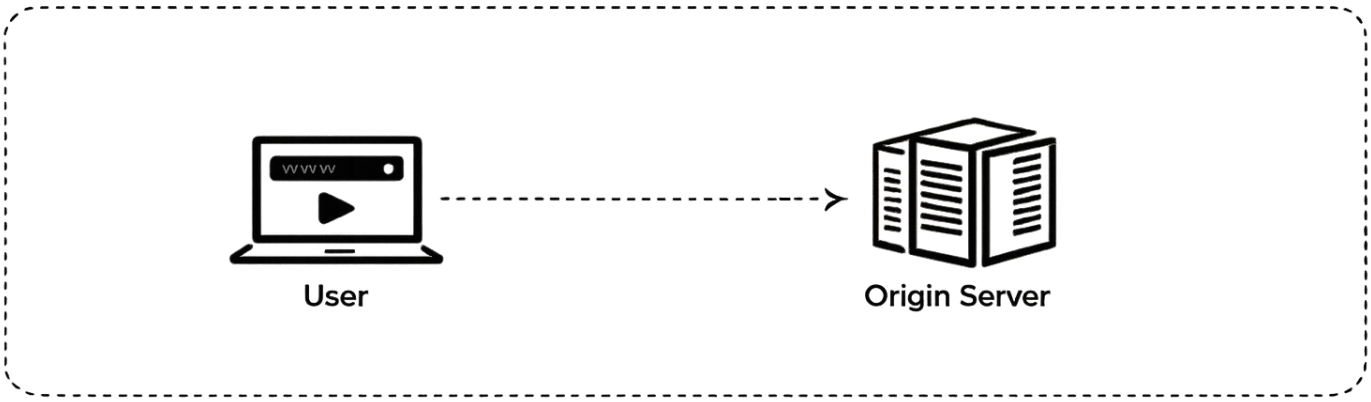
In modern web infrastructures, **CDNs** are widely used to protect origin servers. This project compares a **direct server setup** with a **CDN-protected setup** to analyze differences in exposure , security, and attack surface.

Introduction

In my recent experience analyzing web application security, I've noticed that modern infrastructures heavily rely on CDNs. They prevent direct access to the origin server,reducing exposure to the pulic internet. In this project,I'll explore ehy using a CDN is more beneficial than a server exposed directly to the public.

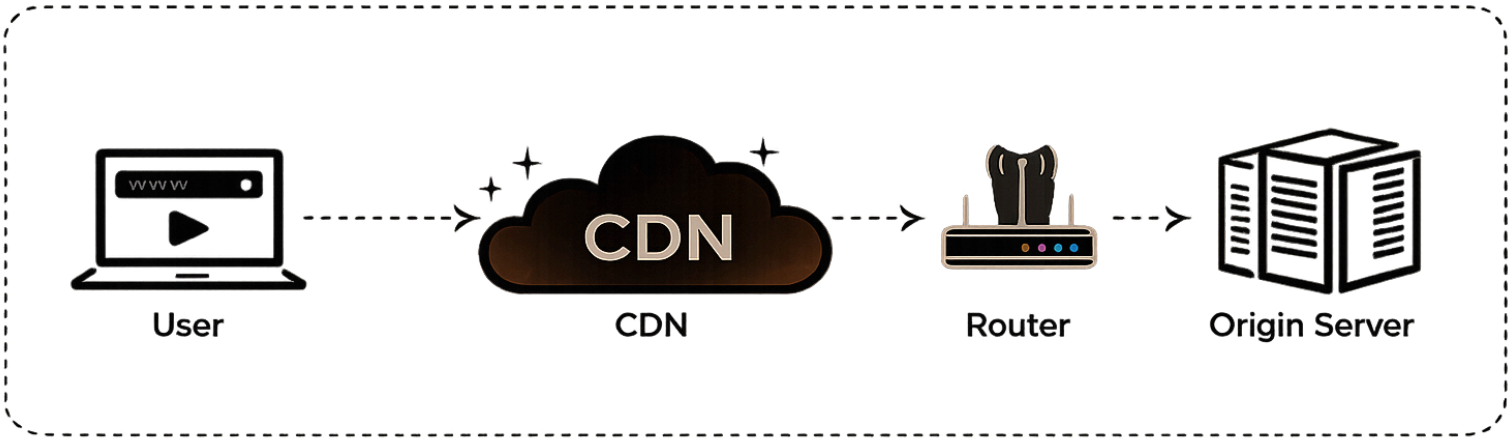
Architecture Diagrams 1

Users directly interact with the origin server. The IP is exposed and services visible to the public and all traffic not filtered :



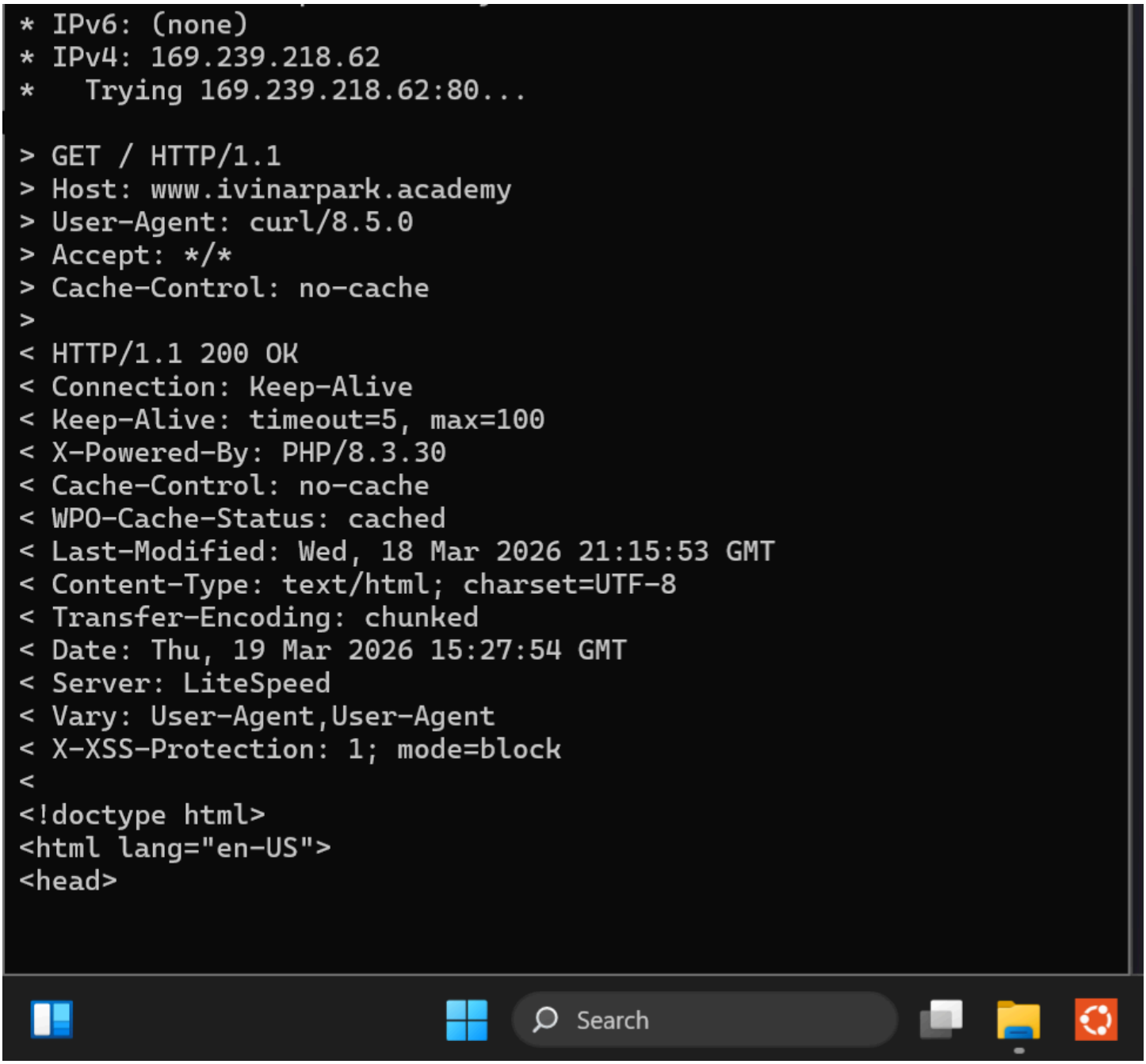
Architecture Diagrams 2

Users do not directly interact with the server, they interact with the CDN layer first - making the origin server IP hidden and the CDN filters requests acting a protection barrier for the server. Reducing the attack surface.



Observation on Direct server

Looking at the response headers from the request, a few things immediately stand out that point toward this being a **direct origin server** rather than traffic going through a CDN.



First, the connection goes straight to an IP:

Trying 169.239.218.62:80...

That already tells me something important - I'm not being routed through a CDN edge node (like Cloudflare or Akamai). Instead, I'm hitting what is very likely the actual hosting server itself.

Now when I look deeper into the response headers:

Server: LiteSpeed
X-Powered-By: PHP/8.3.30

This is where it gets even clearer.

A CDN usually **masks or strips backend details** to reduce fingerprinting. But here, the server is openly telling me:

- The web server software (**LiteSpeed**)
- The backend language and version (**PHP 8.3.30**)

That level of transparency is a giveaway — this isn't being filtered or protected upstream. This is raw exposure from the origin.

There's also:

WPO-Cache-Status: cached

Which suggests caching is happening at the application/server level (likely a WordPress optimization plugin), not at a CDN edge. Another small signal that there's no external caching layer sitting in front.

IP Address Exposure

The biggest issue here though is the exposed IP:

169.239.218.62

This is critical.

When a site is properly behind a CDN, the real origin IP is hidden — you only see the CDN's IPs. That creates a layer of protection:

- Prevents direct attacks on the server
- Forces all traffic through filtering (WAF, rate limiting, etc.)

But here, the origin IP is fully visible and directly reachable.

That means:

- The server can be targeted directly (bypassing any potential frontend protections)
- Attack surface increases significantly
- Recon becomes easier because I'm interacting with the real system, not a shield

So from this alone, I can confidently say this is a **direct server exposure scenario**.

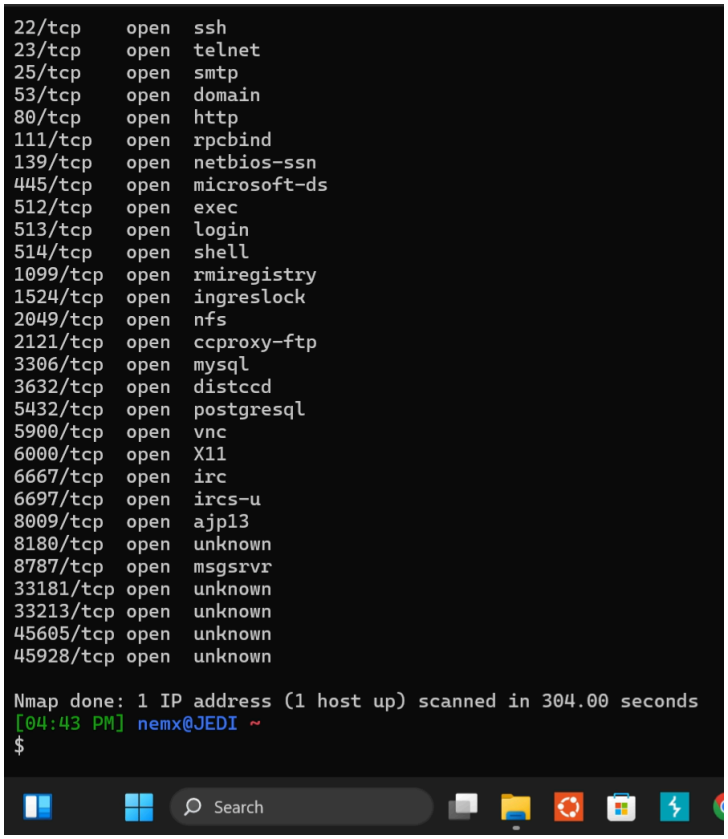
No CDN masking, no abstraction layer - just straight communication with the backend.
It's raw, it's informative... but from a security perspective, it's also risky.

This is exactly the kind of setup where small misconfigurations can turn into real entry points, because there's nothing sitting in front to absorb or filter the interaction.

At first glance, seeing an IP doesn't immediately mean it's a direct server - CDNs also resolve to IPs.
But in this case, the behavior of the connection and the absence of any CDN-related headers, combined with full backend exposure (LiteSpeed, PHP version), makes it clear that this IP is not an edge node but the actual origin server.

After confirming that I'm likely interacting with the origin server, I took it a step further and scanned for open ports.

What came back was... not normal for something that's supposed to be publicly exposed on the internet.



This isn't just a web server anymore.
This is a **fully exposed system**.

Normally, for a public-facing website, you'd expect:

- Port 80 (HTTP)
- Port 443 (HTTPS)

Maybe SSH (22) if needed — but even that is usually restricted.

But here?

– Multiple internal and sensitive services are opened

This confirms something deeper than just “no CDN”

This server is not only directly exposed — it’s **broadly exposed**

There’s no segmentation.
No filtering layer.
No controlled surface.

Every open port is:

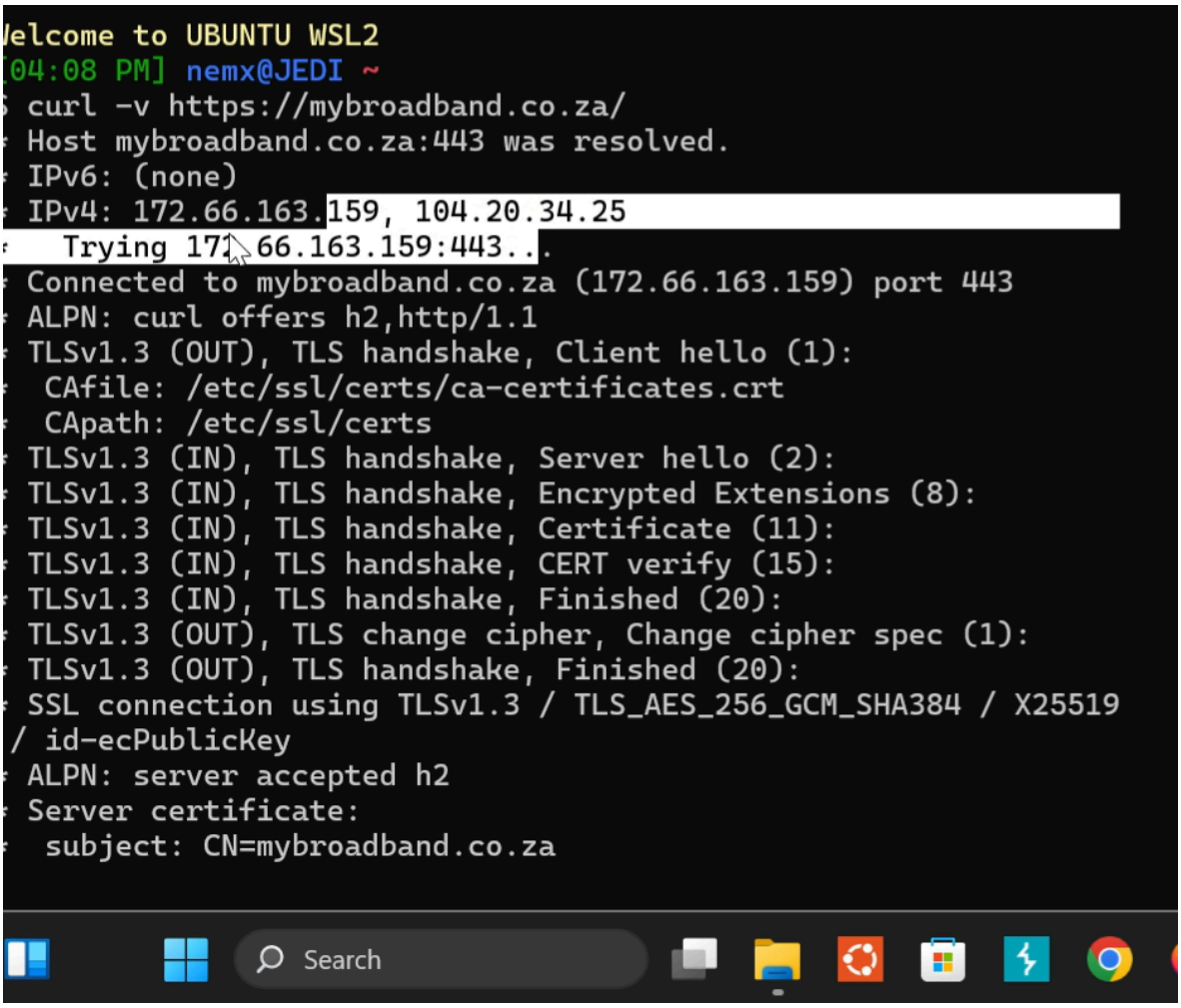
- Another entry point
- Another fingerprint
- Another potential weakness

Direct Server section

At this point, it becomes clear that this is not just a direct server setup, but an overexposed one. The absence of a CDN or protective layer means that not only is the web application reachable, but the underlying services and infrastructure are also directly accessible. This significantly increases the attack surface, turning what should be a limited interface into a wide and potentially vulnerable system.

Observation on Content Delivery Network (CDN)

Right away, the behavior here feels different.



That's the first signal.

Instead of resolving to a **single IP**, the domain resolves to **multiple IP addresses**.

IPv4: 172.66.163.159, 104.20.34.25

This isn't a direct server anymore.

-> **This is distribution.**

CDNs operate using **multiple edge nodes**, so depending on:

- Location
- Load balancing
- Network conditions

You'll get different IPs serving the same domain. Even though I connect to one IP, I know:

- It's just one of many
- It represents an **edge node**, not the backend

TLS & Protocol Handling

SSL connection using TLSv1.3
ALPN: server accepted h2

This is another subtle signal.

CDNs often:

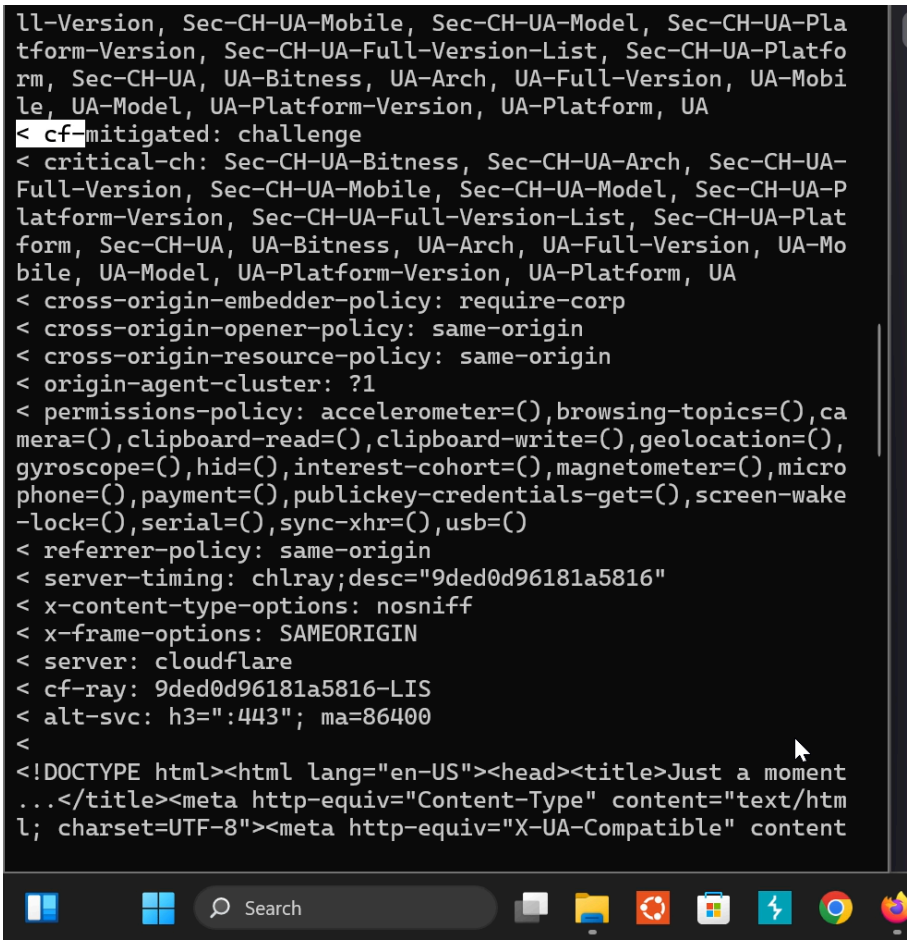
- Enforce modern TLS versions
- Support HTTP/2 (h2)
- Handle encryption at the edge

-> Meaning encryption is terminated *before* reaching the origin server

The Key Difference in Thinking

Compare this to the direct server:

- Before → I was reaching the system
- Now → I'm reaching a **layer in front of the system**



The first thing that proves a CDN is being used here is:

server: cloudflare

- This is the most direct giveaway.
- It explicitly tells you the HTTP response is being served by Cloudflare's edge infrastructure, not the origin server.

cf-ray: 9ded0d96181a5816-LIS

- cf-ray is a **Cloudflare-specific header**.
- It's a unique request ID assigned by Cloudflare.
- The suffix LIS indicates the **edge data center location (Lisbon)**.
- This proves your request hit a Cloudflare edge node, not the actual backend server.

cf-mitigated: challenge

- This shows Cloudflare is actively applying **bot protection / challenge mechanisms**.
- That's part of Cloudflare's CDN + security layer (WAF, anti-DDoS).

alt-svc: h3=":443"

- Indicates HTTP/3 support, commonly enabled by CDNs like Cloudflare for performance.

Conclusion

This setup shifts the entire interaction model - from direct exposure to controlled access. The presence of multiple IPs, modern protocol handling, and identifiable CDN headers shows that the system is no longer directly reachable, but instead protected behind an intermediary layer designed to filter, distribute, and mask the underlying infrastructure.

Final Section (Security Implication)

Comparing both setups, the difference is not just in structure but in exposure.

The direct server allowed full interaction with the backend system - revealing its IP address, technologies, and even internal services through open ports. This creates a wide attack

surface, where an attacker can move beyond the web application and begin targeting the system itself.

In contrast, the CDN-backed setup introduces a controlled layer that abstracts the origin server. The real IP is hidden, backend technologies are masked, and all traffic is filtered through intermediary nodes. This significantly reduces direct exposure and limits how much information can be gathered during reconnaissance.

What stands out is not just what is visible in each case, but what is *hidden*. The CDN doesn't remove the system - it simply places it behind a boundary that changes how it can be reached and analyzed.