The Login Page That Never Appears

Captive portal problems are usually detection problems. Understand what triggers the popup, and most failures explain themselves.

Every guest network gets this ticket.

A visitor connects to the guest Wi-Fi and waits for the login page.

Nothing appears.

They open a browser and try a website. They get an error, a blank page, or an endless spinner.

The ticket says, “Guest Wi-Fi doesn’t work.”

Someone reboots an access point.

Nothing changes, because the access point was never the problem.

Here is the part about captive portals that is rarely explained properly:

The portal page itself is usually not broken.

What breaks is detection.

It is the small exchange between the device and the network that tells the device a captive portal exists.

Understand that exchange, and most portal tickets start to make sense.

How the popup actually appears

When a device joins a network, it does not simply wait for the user to open a browser.

It quietly sends a connectivity probe: a plain HTTP request to a known address operated by the device’s operating system vendor.

Apple devices check an Apple-owned address. Android checks a Google connectivity endpoint. Windows uses Microsoft’s network connectivity test.

Each operating system knows exactly what the expected response should look like.

Then one of three things happens.

If the probe receives the expected response, the device concludes that internet access is available. No portal is detected, so no popup appears.

If the probe receives something different, usually a redirect injected by the network, the device concludes that it is behind a captive portal and opens the login window.

If the probe receives no response at all, the device concludes that connectivity is unavailable.

Usually, there is still no popup.

That is the key point:

The popup appears only when the network successfully intercepts the probe and returns a redirect.

The portal is not a page the device actively searches for.

It is a trap the network has to spring correctly.

The failure modes, in the order I usually meet them

The probe leaks through

The walled garden, the small set of destinations an unauthenticated client is allowed to reach, is too permissive.

If the probe reaches the real vendor server and receives the expected response, the device believes the internet is working normally.

No popup appears.

Real traffic may still be blocked, leaving the user connected but unable to do anything. It looks like an internet problem, but it is really a portal-detection problem.

The ironic version happens when someone whitelists operating system domains to solve a different issue and accidentally includes the probe destinations.

Whitelist the probe and you kill the popup.

Every time.

DNS is blocked before authentication

Detection usually starts with a DNS lookup.

The device must resolve the probe hostname before it can send the request.

If unauthenticated clients cannot reach DNS, the detection process never starts.

The walled garden must allow DNS before authentication.

That is not optional. It is part of the mechanism.

The HTTPS trap

You cannot cleanly intercept HTTPS.

Redirecting an encrypted request without the correct certificate produces a security warning, not a login page. That is exactly how HTTPS is designed to behave.

Captive portal detection works because the operating system probes use plain HTTP.

This is also why the classic user test often fails.

The guest opens a browser, which loads an HTTPS homepage, and sees a certificate warning, an error, or a dead page instead of the portal.

The engineer’s test is different:

Use a plain HTTP website.

A site such as `neverssl.com` exists specifically for this purpose.

If the portal appears when visiting a plain HTTP site but not when loading the user’s normal homepage, you have just seen the HTTPS trap in action.

The garden has gaps

Sometimes the popup appears, but the page is broken.

It may be blank, partially rendered, or have a login button that does nothing.

The portal hostname may be allowed while its stylesheets, scripts, fonts, or images are hosted somewhere else.

Social login may require access to an identity provider.

Payment may depend on a third-party processor.

The portal is a small web application, and every service it relies on must be reachable before authentication.

That can include:

- The portal hostname
- CDN-hosted assets

- Identity providers
- Payment processors
- API endpoints
- Certificate validation services

The portal itself should also be served over HTTPS using a certificate that matches its hostname. Otherwise, the device’s captive portal browser may reject it.

MAC randomisation strikes again

Modern devices use private MAC addresses.

They often generate a different MAC address for each network, and some may rotate it over time.

Captive portal sessions are commonly tied to the client MAC address.

That means the device that “keeps asking the user to log in” may not be malfunctioning.

It may simply be presenting a new identity.

From the portal’s point of view, it is a new device.

Per-network private addresses also mean that a returning guest may appear as a completely new guest.

Session duration, reauthentication, and registration flows need to be designed around that reality.

The old assumption of a permanent client MAC address no longer holds.

It works when we test it

This is my favourite version of the ticket.

Staff insist the portal is working.

They test it from their desk and the login page appears immediately.

The complaints continue.

The problem is that the staff devices may already have valid sessions.

Their MAC addresses are known, authorised, or remembered from a previous visit.

Guests are joining with fresh private MAC addresses and no session history.

The staff are testing the authenticated path.

The guests are experiencing the unauthenticated path.

Test like a guest.

Forget the network. Use a private MAC address. Join from a cold state with no previous session.

That is the experience your visitors are actually having.

The method

Start by scoping the problem, as always.

Is every guest affected, or only one platform?

If all Android devices fail but iPhones work, investigate how the network handles Android’s connectivity probe.

If only one device is affected, look at that device’s portal history, cached session, private MAC behaviour, and DNS state.

Then work through the mechanism.

Confirm that unauthenticated clients can reach DNS.

Confirm that the operating system probes are being intercepted rather than allowed through.

The probe destinations should not be included in the walled garden.

Test detection using a plain HTTP site, not an HTTPS homepage.

If the portal appears but does not render properly, inspect every dependency in the garden:

- Portal hostname
- CDN assets
- Identity provider
- Payment services
- API calls
- Certificate chain

Check that the portal certificate is valid and matches the hostname.

Finally, test from a clean state using a randomised MAC address.

Test like a guest, not like a member of staff.

Final Thoughts

The captive portal is the visible part of the process, but it is rarely the part that failed.

Detection is the mechanism:

A plain HTTP probe.

An interception.

A redirect.

A popup.

Four steps.

Nearly every captive portal problem is one of those steps failing.

Know the probe.

Control the garden.

Test like a guest.

The login page that never appeared was not lost.

It was never requested.

This article is part of an ongoing wireless troubleshooting series. The method remains the same: scope the problem first, then investigate the layer the symptom actually points towards.

Jarryd De Oliveira, CWNE #594

Revision #1
Created 30 July 2026 15:16:22 by Jarryd
Updated 30 July 2026 15:49:55 by Jarryd