Autonomous Connection to Guest Wi-Fi with an Embedded Device: Opportunities, Challenges and Limitations

FATIH CEMIL DEMIR, University of Twente, The Netherlands

As Internet of Things (IoT) devices become increasingly common in public environments, ensuring their seamless and secure access to network connectivity is essential. Public and guest Wi-Fi networks, while widely available, often use captive portals, browser-based login pages designed for human users, which present a major obstacle for embedded devices lacking graphical interfaces. This research addresses the unexplored challenge of enabling autonomous connection to such networks for resource-constrained embedded systems, specifically using the ESP32 platform. By investigating how these devices can independently detect, negotiate and authenticate through various captive portal workflows, this study explores the technical feasibility, resource implications, performance trade-offs, and security risks of automated guest Wi-Fi onboarding. The results aim to inform the development of robust and self-sufficient IoT systems capable of operating securely in dynamic public network environments.

Additional Key Words and Phrases: Autonomous Connection,Guest Wi-Fi,Embedded Systems

1 INTRODUCTION

The Internet of Things (IoT) is rapidly expanding into our public spaces, with embedded devices powering everything from environmental monitoring systems to smart city infrastructure. For these devices to be effective, they require constant and reliable internet connectivity to transmit sensor data, receive updates, or interact with cloud services. Public and guest Wi-Fi networks present an ideal solution, offering ubiquitous and often free internet access without relying on cellular data plans [24]. This availability opens the door for truly "plug-and-play" IoT devices that can be deployed anywhere with minimal setup.

However, this seemingly simple solution was not designed for autonomous machines. Access to most public Wi-Fi is governed by captive portals—web pages that intercept a user's connection and require manual interaction, such as accepting terms of service or entering credentials, before granting full internet access [29]. This human-centric mechanism creates a fundamental barrier for headless embedded devices, which typically lack the graphical interfaces, web browsers, and direct input methods needed to navigate these portals [24].

The challenge of automating this connection is compounded by two other critical factors: security and resource constraints. Public Wi-Fi hotspots are notoriously insecure, exposing connected devices to data interception and cyberattacks [24]. While human users may not always grasp the severity of these risks, an autonomous device cannot afford to ignore them [18]. Furthermore, the IoT devices themselves are inherently resource-constrained, possessing limited processing power, memory (RAM), and storage. Therefore, any potential solution must not only be intelligent enough to handle diverse portal workflows but also be lightweight and secure, creating a difficult trade-off between functionality, safety, and efficiency.

This paper provides a systematic survey and feasibility analysis to navigate this complex landscape. To ground the analysis, we use the widely adopted ESP32 microcontroller as a representative model for a resource-constrained IoT device, theoretically assessing the feasibility of various onboarding schemes against its known capabilities [17]. By deconstructing and evaluating the different methods for autonomous connection, this work aims to build a clear framework of the opportunities, challenges, and fundamental limitations involved.

1.1 Related Work

To effectively address these combined challenges of autonomous interaction, resource efficiency, and security, it is essential to first survey the existing landscape of research. Significant work has explored various facets of this problem domain, including the security of public Wi-Fi, the mechanics of captive portals, and the capabilities of embedded devices. However, as the following analysis will show, these areas have often been investigated in isolation. Studies have extensively documented the security vulnerabilities inherent in public Wi-Fi networks [24] and analyzed user awareness and behavior concerning these risks [18]. The mechanics and security weaknesses of captive portals, including their detection and potential bypass techniques, have also been investigated [23, 29]. Researchers have proposed alternative or enhanced authentication methods, such as integrating FIDO2/WebAuthn (Fast IDentity Online/Web Authentication) or employing EAP (Extensible Authentication Protocol) protocols, sometimes in conjunction with captive portals, to improve security. Furthermore, embedded devices like the ESP32 have been utilized in related security research, for instance, to simulate network threats or study user responses [18].

1.2 Research Question and Sub-Research Questions

Despite this body of work, a critical gap exists regarding the convergence of these issues specifically for resource-constrained embedded devices. While individual aspects like captive portal interaction or security protocols are understood, there is a lack of investigation into the practical feasibility of enabling an embedded device, such as an ESP32, to autonomously and securely handle the diversity of real-world guest Wi-Fi onboarding processes, particularly those involving captive portals, without any human interaction. Specifically, current scientific works do not adequately address the combined challenges of:

- Developing robust autonomous detection and interaction logic for varied captive portal implementations.

TScIT 43, July 4, 2025, Enschede, The Netherlands

- Quantifying the resource implications (flash memory, RAM usage) of implementing such logic on constrained embedded hardware.
- Evaluating the performance characteristics, particularly connection latency, introduced by the autonomous process.
- Understanding the potential security compromises or trade-offs necessary to achieve autonomous connection within device limitations.
- Assessing the overall reliability and success rate across different types of guest networks.

Research Question (RQ).

*To what extent can an **embedded device** autonomously **detect** and **complete** the onboarding flows used by contemporary guest-Wi-Fi networks, and how do **resource usage** (flash/RAM), **connection latency**, and **security compromises** jointly determine the feasibility of each flow?*

The study decomposes this overarching question into three tightly coupled subresearch questions (sRQs):

sRQ 1: **Architectural landscape**
Which guest-Wi-Fi onboarding schemes are most prevalent, and what sequence of network and application-layer messages does an embedded client have to perform for each scheme?

sRQ 2: **Resource and performance cost**
*For each identified onboarding scheme, what are the **anticipated implementation complexity, resource requirements (in terms of processing and memory), and performance characteristics (in terms of connection latency)** for a resource-constrained embedded device?*

sRQ 3: **Security trade-offs**
*What **security compromises** are necessary to automate each onboarding flow on an embedded devices, and how do these compromises affect the residual risk compared with manual, user-mediated onboarding?*

Therefore, the proposed solution investigated in this research is to explore and determine the feasibility of achieving fully autonomous guest Wi-Fi connection, including captive portal negotiation, for resource-constrained embedded devices. This involves designing, implementing, and evaluating mechanisms that allow a device like the ESP32 to independently connect to various guest networks it encounters.

The primary contribution of this research will be to provide a comprehensive understanding of the practicalities involved in this autonomous connection process. The findings will illuminate the technical hurdles, quantify the resource costs (memory, storage), measure the performance impact (connection time), and analyze the security considerations and potential trade-offs. This knowledge is essential for developers and engineers aiming to design and deploy robust, reliable, and secure embedded IoT devices capable of operating autonomously in environments reliant on public or guest Wi-Fi infrastructure. Ultimately, this work will inform the feasibility and design choices for future autonomous embedded systems requiring network access in diverse public settings.

Together, SRQ 1 determines the *recognition* challenge, SRQ 2 quantifies the *implementation cost*, and SRQ 3 contextualizes these costs in terms of *security impact*. Addressing all three enables a grounded answer to the stated Research Question.

1.3 Paper Structure

The remainder of this paper is structured to systematically address the research questions. Section 2 presents the core of the literature survey that was made, establishing a taxonomy of the different guest Wi-Fi onboarding architectures an autonomous device might encounter; this directly addresses sRQ1. Section 4 presents and discusses findings, synthesizing the analysis of the theoretical survey. It is organized to provide direct answers to sRQ1, sRQ2, and sRQ3 by interpreting the data summarized in the Appendices, followed by a broader discussion of the resulting challenges and opportunities. Finally, Section 5 concludes the paper by summarizing this paper's key contributions and offering a final perspective on the feasibility of autonomous guest Wi-Fi connection for embedded devices.

2 TYPE OF GUEST NETWORKS

To understand the challenges of autonomous connection, it is first necessary to establish a clear taxonomy of the different network architectures an embedded device might encounter. This section addresses sRQ1 by categorizing and analyzing the most common guest Wi-Fi onboarding schemes, from modern standardized protocols to ad-hoc captive portal systems. The analysis is derived from a review of academic literature, industry standards (RFCs), and vendor documentation. Each of the following subsections details a specific architecture, its typical protocol flow, and the inherent complexities it presents for a headless embedded device like the ESP32.

2.1 Open Access & Wi-Fi Enhanced Open (OWE)

Most public hotspots still broadcast an un-encrypted open SSID (Service Set Identifier) because it works on every client and needs no user interaction; Cisco counted around 628 million such hotspots by 2023, dwarfing all other Wi-Fi models [6]. RFC 8110 introduces Opportunistic Wireless Encryption (OWE), adopted by WPA3 "Enhanced Open," so venues can run a second BSSID (Basic Service Set Identifier) that silently encrypts traffic for modern devices while legacy stations remain on the plain-open BSSID [11].

For a small IoT station such as an ESP32 the connection flow stays almost the same as a classic open SSID. The only additions, when the chip supports OWE, are: the station attaches its own Diffie-Hellman public key to the Association Request, parses the AP's public key (or recognizes a cached Pairwise Master Key (PMK)), verifies that key, runs the Elliptic-curve Diffie-Hellman (ECDH) exchange, feeds the shared secret into HKDF (HMAC-based Key Derivation Function) to derive a new Pairwise Master Key, and then completes the normal Wi-Fi Protected Access (WPA) 4-way handshake to install the session keys. RFC 8110 specifies that these cryptographic steps require parsing an extra element and running ECDH/HKDF. While these steps require extra processing, these procedures are standards and does not require software implementation on client side. Also enabling this feature on ESP32 will allow the embedded device to connect to the OWE supported guest network[11, 28].

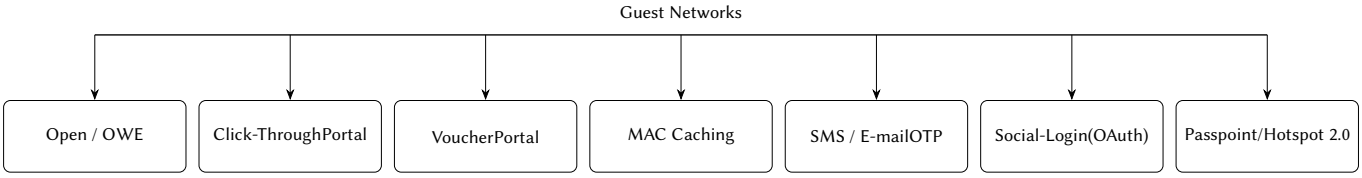

Fig. 1. Guest-network types.

From a feasibility standpoint, this scheme offers a significant security improvement over legacy open networks (sRQ3) at the cost of a measurable, albeit small, increase in firmware size and connection latency due to the cryptographic overhead (sRQ2).

2.2 Click-Through Captive Portals

The majority of cafés and airports expose guests to a one-tap "Accept & Continue" splash page. [1] The technical template for these portals is laid out in RFC 8952 Captive Portal Architecture, which says a hotspot should signal captivity via provisioning protocols (Dynamic Host Configuration Protocol (DHCP) or IPv6 Router Advertisements (RA)) and present a web user-portal instead of forging DNS/HTTP traffic. The URI of that portal is conveyed with DHCP/RA Option 114, defined in RFC 8910, so modern operating systems can pop up the consent page automatically [14, 15, 22]. Connection procedure and its complexity can be generalized by the following sequence:

(1) Associate to open SSID.
(2) DHCP/RA provisioning: the ACK/RA includes Option 114 carrying the Captive-Portal-API URI.
(3) Portal discovery: the client performs an HTTP GET to that URI (or any site); the enforcement device blocks general traffic and returns an HTTP 302 redirect to the splash page, exactly as the workflow in Section 4.1 of RFC 8952 describes. [15]
(4) User action: through the portal page user confirms the conditions and send a post form.
(5) Enforcement update: the portal i.e. API server instructs the enforcement device to whitelist the client's MAC/IP, after which the browser reloads the original URL and full internet access is granted (step 5 of RFC 8952's initial-connection workflow). [15]

Captive portal systems may appear to be straightforward "click-through" web pages, but their implementations vary widely and often involve complex web interactions. Supporting a broad range of captive portals in automated or embedded devices (such as IoT modules or travel routers) is non-trivial due to diverse scripting requirements and protocol tricks. Remaining part of this subsection, summarizes key factors contributing to this complexity, and how they impact automation, resource use, and security.

Diverse Implementations

Modern captive portals lack a single standardized implementation. Vendors historically implemented portals as ad hoc network hacks – intercepting traffic and redirecting users to login pages. In fact, researchers describe today's captive portals as "an ad hoc, Web-based man-in-the-middle (MitM) hack" that often confuses client devices and users [19]. The IETF, in RFC 8952, similarly notes that existing captive portal solutions take various non-standard approaches, like forging DNS or HTTP responses and even attempting HTTPS interception [15]. This means each portal can behave differently: some inject custom HTTP redirects, others rewrite DNS, etc., making it hard for a generic client to handle all cases. The IETF explicitly acknowledges "the existence of a huge variety of pre-existing portals" and the need for a new standardized architecture rather than trying to support every proprietary method [15].

This variability extends to how users are notified or redirected. Traditional methods basically trick devices into opening a browser. Many operating systems now perform "canary" connectivity checks (e.g. a known HTTP URL) and if they get an unexpected result, they launch a captive portal mini-browser [15]. These mini-browsers themselves are simplified and sometimes outdated web clients, which creates another layer of inconsistency. A recent security analysis of captive portal mini-browsers found that different devices' mini-browsers lack modern security measures and behave inconsistently. In some cases, devices forego a mini-browser and just use the default browser. All of this underscores that captive portals are a patchwork of different techniques, not a uniform mechanism [29].

JavaScript, Tokens, and Dynamic Web Flows

Although a captive portal login page might look like a static form or a simple "Accept terms" button, under the hood it often relies on active web content (JavaScript, meta refreshes, dynamic tokens, etc.). Hidden form fields and CSRF tokens are common – many captive portals generate a one-time token that must be included in the acceptance POST request for security. For example, Cisco's Identity Services Engine (ISE) guest portals include a CSRF token in the login requests, and will reject submissions without it (a standard protection against cross-site request forgery) [7]. This means an automated client must fetch the portal page and parse out such tokens before posting.

In addition, JavaScript logic is frequently required to complete the login process. One example is MikroTik's RouterOS Hotspot: its captive portal login can use an HTTP-CHAP mechanism where the password must be hashed in the browser via a provided script. The standard pages include a md5.js file – "JavaScript for MD5 password hashing" – that the login form calls to compute a response [20]. An automated client would need to replicate this hashing (or fully execute the JavaScript) to log in successfully when CHAP is enabled.

Meta-refresh HTML tags are yet another technique: some portals deliver a page that automatically refreshes to a given URL. Aruba's documentation notes that a captive portal "landing page contains the meta-refresh tag to reload the page using real browser applications" [3]. This is actually a clever workaround to move the session from a mini-browser to the user's full browser – after initial sign-on,

the meta-refresh triggers the device to open the real browser to a specified page. Automated systems would need to detect and follow meta-refresh tags, or else the login flow might never complete. In general, a client might encounter JavaScript-driven forms (e.g. a form that auto-submits via script), timers or polling loops that check if a user is still online, or AJAX/XHR calls that report login status to the portal. For instance, some captive portals use periodic XHR polling to update usage time or to detect when the user has accepted terms on a secondary page before granting full access. An automation solution must either implement a mini web browser or specifically code for all these possibilities – greatly increasing complexity.

Impact on Automation and Embedded Systems

The diversity of scripts and web content means headless or embedded devices struggle with captive portals. In many IoT scenarios, there is no user interface to click through a portal. The IETF's architecture acknowledges this, noting that while its new API can let devices detect captivity, it "does not describe a mechanism for [UI-less] devices to negotiate for unrestricted network access" [15]. In practice, most IoT devices simply cannot connect to networks behind captive portals without user help. Solutions like Wi-Fi travel routers or enterprise IoT gateways try to bridge this gap, but they effectively have to include a web browser component or predefined scripts. This is resource-intensive and error-prone.

For embedded systems, supporting a wide variety of captive portals means increased resource usage. Memory and CPU constraints can make it infeasible to run a full browser engine. Yet, without one, the device might not handle modern login pages that load multiple scripts, images, or third-party content. Even parsing HTML to find form fields or meta-refresh tags can add code complexity and processing overhead on a microcontroller. Each new portal quirk (a hidden input here, a dynamic parameter there, a non-standard form submission) potentially needs a custom handler. Maintaining a library of workarounds for every vendor's portal is a heavy burden for firmware (sRQ2). This complexity in code can also introduce security vulnerabilities (e.g., if the parser mishandles an unexpected input) (sRQ3). Essentially, to automate portal login, an embedded device almost needs to impersonate a web browser – a task far beyond a simple Wi-Fi client.

2.3 Voucher / Username–Password Portals

Voucher- or credential-based portals build on the click-through model described above but add a credentials step.[15, 18]. This approach is widely deployed in open Wi-Fi "hotspots" to onboard new users and enforce acceptable-use policies. Unlike secure 802.1X enterprise Wi-Fi, these networks start unencrypted and rely on an application-layer (HTTP) login, which is an ad-hoc solution but has become very popular for public access hotspots [19]

The captive portal authentication workflow in a voucher/login scenario generally proceeds as follows. Steps 1–2 match Section 2.1; here we begin with Step 3.

(3) User Login (Voucher/Credentials): The captive portal web page is displayed, usually asking the user to either enter a voucher code or username/password, or to accept terms of service [15, 18]. The user fills in the required credentials on this portal form and submits it. (Voucher-based systems use one-time codes given to guests, while login-based systems validate against an account database – but the portal mechanism for input is similar in both cases.)
(4) Back-End Authentication: Upon submission, the portal backend (sometimes called a UAM – Universal Access Method handler) forwards the credentials to an authentication server on the network (commonly via Remote Authentication Dial-In User Service (RADIUS) in the AAA (Authentication, Authorization, and Accounting) backend) for verification. For example, the UAM component contacts a RADIUS server with the username/password or voucher code to authenticate the user, often using a centralized subscriber database or RADIUS AAA service [19].
(5) Access Granted: After successful authentication, the network marks the user as authenticated and lifts the captive restrictions [19]. The user's device is then allowed to send and receive traffic to the Internet freely. In practice, the portal may redirect the user to their originally requested page or show a welcome/confirmation page at this stage [18]. The client can now use the network normally until the session ends or times out (at which point the portal might require re-authentication) [15].

The same headless-device limitation described in Section 2.1 applies here, with the added burden of securely provisioning credentials [15]. This added burden stems from having to pre-load vouchers or passwords into the device (or deliver them over a protected side channel) and keep them in tamper-resistant storage so that attackers on the same Wi-Fi cannot steal or replay them.

2.4 MAC-Authentication Bypass / Caching

Public guest hotspots often "remember" devices that have already passed their captive portal by storing the device's MAC address and automatically authorizing the same address on subsequent connections. A large-scale measurement study of 67 North-American cafés, libraries and transit hubs found that nearly two-thirds of venues used exactly this MAC-caching technique to waive the splash page for repeat visitors during a 24-hour grace period [1]. When a new client appears, the gateway forces the captive-portal login; once the user accepts the terms or enters credentials, the gateway adds the client's MAC to its allow-list. Any later association from that MAC is granted service immediately until the cache entry expires. For devices with no browser—ticket kiosks, streaming sticks, embedded sensors—this mechanism can be a practical way to regain Internet access without human help.

Some public deployments extend the idea with MAC-Authentication Bypass (MAB), a RADIUS workflow originally formalized for 802.1X networks [25]. If a client fails EAP authentication, the access point submits a RADIUS Access-Request that uses the client's MAC address as the username (via the Calling-Station-Id attribute). The RADIUS server checks the MAC against its guest database; an Access-Accept can include tunnel attributes that place the device directly into a "guest" Virtual LAN (VLAN), while an Access-Reject keeps the client quarantined. This approach requires no changes on the

device side and lets operators apply role-based VLANs or ACLs (Access Control List), exactly as they would for 802.1X users.[4].

The protocol sequence for an embedded client such as an ESP32 is therefore simple:

(1) Associate to the open SSID and run DHCP.
(2) Gateway queries its MAC cache (or RADIUS).
(3) Known MAC → immediate access; unknown MAC → captive-portal redirect.
(4) After a successful web login, the gateway inserts the MAC into the cache and returns the client to normal traffic flow.

Because the ESP32 only executes the standard Wi-Fi handshake and DHCP exchange, no browser or TLS (Transport Layer Security) code is required. The trade-off is security: MAC addresses are visible over the air and trivially spoofed. Attack experiments have shown that a rogue device can impersonate an authorized MAC and hijack the session in seconds [27]. In addition, modern phones rotate their MAC for privacy; the IETF MADINAS working-group draft warns that such randomization "breaks stateful services that rely on a stable MAC and forces re-authentication" [12]. Large-venue studies likewise report operational headaches when thousands of rotating MACs overflow controller caches [30]. Consequently, MAC caching and MAB are best viewed as convenience features: they ease guest onboarding and enable headless IoT clients, but they must be combined with short cache lifetimes, network segmentation or additional monitoring to offset spoofing and privacy risks. This scheme is therefore ideal from a resource and performance cost perspective (sRQ2) as it requires no additional logic on the embedded client, but it represents the most significant security compromise (sRQ3), making the device's identity trivially vulnerable to impersonation.

2.5 SMS / E-mail One-Time-Passcode Portals

Many modern public Wi-Fi hotspots now require users to enter a phone number or email address and then input a one-time code sent to that contact, instead of simply clicking "Accept & Continue" on a splash page. Enterprise Wi-Fi systems like Cisco ISE and Aruba ClearPass include built-in support for this workflow – for example, Cisco's guest portal can be configured to deliver login credentials via SMS or email [8].

Despite the added SMS/Email step, the underlying network exchange begins like any captive portal. The access point signals captivity using DHCP or Router Advertisement Option 114 to provide the captive portal's URL (as defined in RFC 8910) [14], enabling client devices to automatically open the login page [15]. The user's first HTTP(S) POST to that portal transmits their phone number or email to register. According to industry leader Cisco's documentation, the actual sending of the one-time passcode (via an SMS gateway or email server) happens out-of-band – off the local Wi-Fi link – and the client's only in-band interactions are: (1) the initial registration request, (2) a second HTTP(S) POST to submit the received OTP code, and (3) the network controller's authorization update. Specifically, after the correct OTP is entered, the portal's backend issues a RADIUS Change-of-Authorization (CoA) or similar API call to the Wi-Fi gateway, instructing it to lift the captive restrictions for that client [8]. This means the client does not need any new 802.1X/EAP handshake; all the complexity stays in the web/API layer. Once the CoA is processed and the client's MAC or IP is whitelisted, the user gains normal internet access [8] (in accordance with the standard captive portal workflow in RFC 8952) [15].

For headless IoT devices (like sensors or microcontrollers without UIs), an OTP-based captive portal is essentially a brick wall. Such a device has no phone number or email to receive a code, nor a user interface to read and input the OTP. Current standards explicitly acknowledge this limitation: devices without any user interface cannot independently complete web captive portal logins for full network access [15]. In theory, a workaround would require an external trusted service to receive the SMS/email on behalf of the device and feed the code into the portal – which would entail adding a TLS-enabled client, maintaining a backend session, and storing credentials/tokens on the device. Implementing all that is far beyond the typical firmware footprint of constrained IoT hardware like ESP32. Moreover, from a security standpoint, integrating an SMS/email OTP mechanism into an IoT device would expand its attack surface. Research has shown that one-time passwords sent via SMS are vulnerable to social engineering and malware interception: users can be phished into divulging OTP codes in look-alike prompts [16], and malicious apps or attackers on a phone can illicitly read OTP messages to hijack the authentication channel. In short, embedding an OTP portal workflow into a tiny device not only poses a resource and implementation challenge but also introduces well-documented OTP security weaknesses [16] that could put the device and network at risk.

2.6 Social-Login (OAuth) Portals

OAuth-based captive portals leverage third-party identity providers for this login step. In public Wi-Fi hotspots and guest networks, the splash page often offers "social login" options (e.g. Facebook). In this model, the Wi-Fi service delegates authentication to an external OAuth 2.0 provider. The user selects a social network or SSO provider on the portal, which initiates an OAuth flow; this allows the user to sign in with an existing account and share basic profile information, while the captive portal itself never sees the password (it only receives an OAuth token or profile data upon completion) [7]. Such OAuth-based captive portals are commonly used in venues like cafés, airports, and enterprise guest WLANs to simplify login and tie network access to a real identity or social account for analytics.

When the user opts for social login, the captive portal initiates an OAuth 2.0 Authorization Code grant sequence with the chosen provider [10]. The portal's web server directs the user's browser to the OAuth authorization endpoint of the provider (via an HTTP 302 Redirect to a HTTPS URL carrying the portal's client ID, requested scopes, redirect URI, etc.) [7, 10].

The user is then presented with the provider's login page and prompted to authenticate and grant consent for the Wi-Fi service to obtain their basic profile (identity) information [7]. Upon successful authentication and approval, the authorization server (e.g. Google or Facebook) redirects the user-agent back to the captive portal's callback URI with an authorization code in the URL query string [10]. The captive portal (now acting as the OAuth client) exchanges this code for an access token by making a back-channel HTTPS POST

to the provider's token endpoint, including its client credentials as required [10]. Once the token is issued (and optionally an ID token or refresh token), the portal can query the provider's API for the user's identity details or decode the token if it's a self-contained ID token. Finally, the portal logs the user in and notifies the network gateway to lift the access restrictions for that device [7]. This entire process involves multiple HTTP redirects and interactions between the browser and various domains. For example, a Facebook login sequence will contact facebook.com and several related domains for OAuth dialogs and static content (e.g. fbcdn.net, akamaihd.net for scripts/images), all of which must be accessible through the captive portal. These dependencies on external web resources and secure token exchanges mean the protocol flow is entirely browser-driven and encrypted, appearing as a series of TLS web fetches and redirects in a packet trace (with OAuth tokens and credentials conveyed over HTTPS as per OAuth 2.0 security requirements) [10].

A fundamental limitation of OAuth-based captive portals is that they assume a human user with a full web browser. Devices without a user interface or browser (e.g. IoT sensors, printers, or other headless gadgets) cannot perform the interactive OAuth handshake and thus cannot get online via these captive portals [15]. The multi-step flow – involving HTML forms, JavaScript-driven redirects, and user consent dialogs – is fundamentally unsolvable for an unattended device that cannot display pages or input credentials. In fact, even on phones or laptops, the mini browser used for Wi-Fi sign-on (such as the iOS or Android captive network assistant) may be treated as an insecure or unsupported user-agent by OAuth providers. For example, Google does not allow OAuth login prompts to run inside embedded web-views for security reasons, and it blocks them with an error. This means a Google-based Wi-Fi login will fail in the captive portal's built-in browser, forcing users to launch a normal browser to complete the process [9]. In summary, the OAuth social-login approach for captive portals is interactive and browser-dependent by design. It cannot be navigated by headless or automated clients, which lack the capability to execute the required web content and user interactions, a shortcoming acknowledged by both industry documentation and standards bodies

2.7 Passpoint / Hotspot 2.0

Wi-Fi Passpoint (also known as Hotspot 2.0) is a certification by the Wi-Fi Alliance that enables automatic, secure Wi-Fi access in public networks [2]. Introduced in the early 2010s (built on the IEEE 802.11u standard), Passpoint allows mobile devices to seamlessly discover and join Wi-Fi hotspots (e.g. in airports, hotels, city networks) without the user manually selecting SSIDs or handling web captive portals [21]. The network advertises its identity and services to devices, and if a device has a matching credential (such as a mobile carrier or roaming agreement), it can auto-connect with enterprise-grade security. This creates a cellular-like experience for Wi-Fi: once provisioned, users roam onto partner Wi-Fi networks automatically and securely [2].

Typical authentication flow could be described as follows:

Discovery & ANQP: Access points broadcast Passpoint/Hotspot 2.0 capabilities in beacon frames (per 802.11u), indicating network type, internet availability, and supported service providers [5]. A Passpoint-capable device detects these signals and uses the Access Network Query Protocol (ANQP) to query the hotspot for detailed information (e.g. venue info, provider list, authentication methods) before associating [5].

Credential Matching: The device's Passpoint profile (installed beforehand) is examined to find if it has a subscription or credential for any of the advertised providers [2]. For example, a device might recognize a roaming consortium OI or realm belonging to its mobile carrier or another identity provider. If a match is found, the device selects that network for connection.

Secure 802.1X Authentication: The device then automatically initiates an 802.1X authentication (WPA2-Enterprise or WPA3-Enterprise) using an appropriate EAP method (such as EAP-TLS, EAP-SIM/AKA for SIM-based IDs, or TTLS/PEAP) to authenticate with the network's backend [13]. The hotspot's AAA server (or a proxy) validates the credentials (often via RADIUS to the user's home operator or identity provider). Upon successful EAP authentication, a secure encryption key is established (4-way handshake), and the device gains internet access with no captive portal needed. The entire process is transparent to the user and can persist across roaming events, so users stay connected securely as they move between Passpoint-enabled hotspots.

While Passpoint greatly simplifies user access, its implementation on embedded or headless IoT devices presents significant challenges, primarily concerning automation and standards compliance. These devices typically have limited UI or input, making it hard to install the necessary Passpoint profiles or interact with onboarding portals. They also must implement the full 802.11u/Passpoint protocol (ANQP queries, etc.) and support 802.1X EAP authentication, which may exceed the capabilities or memory of lightweight Wi-Fi stacks. As a result, many IoT devices currently stick to traditional Wi-Fi connection methods due to these implementation complexities and resource requirements. Automation and standards compliance are key concerns – credentials would need to be provisioned in the device at manufacturing or via a secure out-of-band mechanism to enable truly hands-free access. The industry is beginning to address this gap: for instance, the Wireless Broadband Alliance has been working to extend Passpoint to IoT use cases to "support dynamic IoT roaming and streamline authentication and interoperability" [26]. This includes exploring cloud-based credential management and new provisioning frameworks to allow IoT devices to securely leverage Passpoint with minimal user intervention.

3 RESULTS & DISCUSSIONS

3.1 Answer to sRQ1

The first sub-research question (sRQ1) sought to identify prevalent guest-Wi-Fi onboarding schemes and their required protocol flows. The detailed investigation in Section 2, summarized in **Table 1**, reveals a highly fragmented and non-standardized landscape. The architectures can be broadly classified into two categories: standardized, machine-friendly protocols (OWE, Passpoint) and human-centric, ad-hoc portal systems. While the former provide a clear and automatable connection path, the latter, which constitute a significant portion of public networks, rely on diverse and unpredictable mechanisms involving HTTP redirection and

Table 1. Analysis of onboarding flows and key protocols for an autonomous device.

Guest Network Type	Autonomous Onboarding Flow	Key Mechanisms / Protocols
Wi-Fi Enhanced Open (OWE)	1. Associate with DH key. 2. Perform ECDH exchange. 3. Complete WPA handshake.	OWE (RFC 8110), ECDH, HKDF.
Click-Through Captive Portal	1. Connect; get HTTP redirect. 2. Fetch & parse splash page. 3. Find & submit "accept" form.	DHCP, HTTP Redirect, HTML/JS Parsing.
Voucher / Username-Password Portal	1. Connect; get HTTP redirect. 2. Fetch & parse login page. 3. Submit pre-provisioned creds.	HTTP Redirect, HTML Parsing, Credential Management.
MAC-Auth Bypass / Caching	1. Associate to SSID. 2. Get IP via DHCP. (No further client steps).	Client MAC address as identifier; network-side logic.
SMS / E-mail OTP Portal	Infeasible for headless device. Requires external service to receive & forward OTP.	Out-of-band communication, multiple HTTP POSTs.
Social-Login (OAuth) Portal	Infeasible for headless device. Requires interactive, multi-domain browser flow.	OAuth 2.0, JavaScript, secure user interaction.
Passpoint / Hotspot 2.0	1. Discover via 802.11u beacon. 2. Query network info (ANQP). 3. Match pre-installed profile. 4. Authenticate via 802.1X/EAP.	802.11u, ANQP, WPA2/3-Enterprise, EAP, RADIUS.

HTML/JavaScript interaction. Furthermore, the analysis identifies schemes like SMS/OTP and OAuth-based portals as fundamentally infeasible for a headless device to navigate autonomously due to their reliance on out-of-band channels or complex, interactive web flows. This diversity represents the primary recognition challenge for any autonomous client.

3.2 Answer to sRQ2

The second sub-research question (sRQ2) focused on the resource and performance costs associated with automating each scheme on an ESP32. The qualitative assessment presented in **Table 2** quantifies these costs. A direct correlation exists between the complexity of the onboarding flow and its impact on the embedded device. Schemes like MAC-Auth Bypass are ideal from a resource perspective, imposing minimal overhead on flash, RAM, or CPU. In contrast, any method requiring captive portal negotiation introduces large overhead due to the need for HTTP client libraries, TLS stacks, and HTML parsers, which consume significant flash memory and runtime RAM. This complexity also translates to higher connection latency, as multiple network round-trips are required to fetch and submit portal forms. The most resource-intensive but automatable scheme is Passpoint, which requires a full 802.1X/EAP supplicant.

3.3 Answer to sRQ3

The investigation into sRQ3 addressed the security compromises necessary for automation. The findings, summarized in **Table 3**, highlight a critical inverse relationship between the ease of automation and the level of security provided. The simplest method to implement, MAC-Auth Bypass, is also the most insecure, as it is

Table 2. Qualitative cost assessment for implementing autonomous flows on an ESP32.

Guest Network Type	Implementation Complexity [1]	Resource Impact (RAM/CPU)[2]	Performance Impact (Latency)[3]
Wi-Fi Enhanced Open (OWE)	Low	Low	Low
Click-Through Captive Portal	High	Moderate	High
Voucher / Username-Password	High	Moderate-High	High
MAC-Auth Bypass / Caching	Minimal	Minimal	Low
SMS / E-mail OTP Portal	Infeasible	Infeasible	Infeasible
Social-Login (OAuth) Portal	Infeasible	Infeasible	Infeasible
Passpoint / Hotspot 2.0	High	High	Moderate

[1] Code / firmware effort required to support the flow.
[2] Additional runtime RAM / CPU consumed by onboarding logic.
[3] Extra time added to establish usable internet connectivity.

trivially vulnerable to MAC address spoofing and session hijacking. Automating voucher or password-based portals introduces the severe risk of insecure on-device credential storage, making the device a high-value target. Blindly automating click-through portals may also lead to the acceptance of malicious terms of service. Conversely, the most secure methods, OWE and Passpoint, offer robust, encrypted communication from the outset but demand higher implementation complexity and a secure method for initial credential provisioning. This demonstrates that the choice of an automation strategy is fundamentally a risk management decision, where convenience is often achieved at the expense of security.

3.4 Challenges

The findings reveal several fundamental challenges that impede the deployment of network-agile autonomous IoT devices. The findings reveal a primary obstacle: the architectural fragmentation of guest networks. The lack of standardization, particularly in captive portals, makes creating a universal 'solver' for an autonomous device nearly impossible. This is compounded by the headless device impasse, where modern authentication methods like OAuth and OTP are explicitly designed for human interaction and are thus inaccessible to embedded systems. This leads to a critical security-versus-automation dilemma, where the most easily automated connection methods (e.g., MAC-based authentication) are the most insecure. Finally, all these issues are worsened by resource constraints of embedded hardware, which makes implementing complex web-parsing and cryptographic stacks a significant technical burden.

3.5 Opportunities

Despite these challenges, the analysis also points to clear opportunities for progress. The most impactful long-term opportunity lies in the wider industry adoption of IoT-friendly standards like Passpoint and OWE, which solve the problem at the network level. Meanwhile, an opportunity still exists to develop lightweight, heuristic-based portal solvers for embedded systems that can handle the most common click-through scenarios without the overhead of a full browser engine. Furthermore, developers can implement a tiered connection strategy in device firmware, prioritizing secure and simple networks (OWE, Passpoint) before falling back to less secure or more complex methods. A final, powerful opportunity is the use of hybrid solutions, where an auxiliary device (e.g., a smartphone) performs the initial complex authentication and securely passes session tokens or status to the embedded device via a side-channel like Bluetooth.

4 CONCLUSIONS

This research investigated the feasibility of enabling a resource-constrained embedded device, the ESP32, to autonomously connect to contemporary guest Wi-Fi networks. The study established a taxonomy of onboarding architectures, revealing a fragmented landscape dominated by human-centric captive portals that are inherently difficult for headless devices to navigate. A critical trade-off was identified: methods that are simple to automate, such as MAC-Auth Bypass, are highly insecure, while secure, standardized methods like Passpoint and OWE present a higher implementation burden. The conclusion is that while autonomous connection is technically feasible for certain network types, the current guest Wi-Fi ecosystem presents substantial practical barriers to robust and secure operation. True network autonomy for IoT devices in public spaces will depend less on developing complex client-side workarounds and more on the wider industry adoption of machine-friendly standards. The primary contribution of this work is a clear framework for understanding these technical and security trade-offs, providing developers with a grounded assessment of the costs, risks, and practical limitations of deploying autonomous IoT devices in diverse network environments.

5 AI DISCLOSURE

Portions of the paper's language were refined with the assistance of OpenAI ChatGPT (model 4o). The tool was used solely for improving readability and correcting grammar; all ideas, analysis, and conclusions are the author's own.

REFERENCES

[1] Suzan Ali Ahmad Ali. 2020. *A Large-Scale Evaluation of Privacy Practices of Public WiFi Captive Portals.* Master's thesis. Concordia University. https://spectrum.library.concordia.ca/id/eprint/987023/

[2] APAC Task Group. 2024. WBA OPENROAMING™ INTRODUCTION GUIDE. https://wballiance.com/wp-content/uploads/2024/10/WBA-OpenRoaming-Guide-APAC-TG_v1.0.0-FINALPublic.pdf#:~:text=For%20over%20a%20decade%2C%20Passpoint%C2%AE,cumbersome%20process%20of%20onboarding%20the

Table 3. Analysis of security risks introduced by automation and overall IoT feasibility.

Guest Network Type	Primary Automation Security Risk	Data-in-Transit Security	Overall IoT Feasibility [1]
Wi-Fi Enhanced Open (OWE)	Minimal; a security improvement.	OWE Encrypted	High
Click-Through Captive Portal	Blindly accepting ToS; parser vulnerabilities.	Open/Unencrypted	Medium
Voucher / Username-Password	Insecure on-device credential storage.	Open/Unencrypted	Low
MAC-Auth Bypass / Caching	MAC address spoofing and session hijacking.	Open/Unencrypted	Medium-Low
SMS / E-mail OTP Portal	Dependency on vulnerable out-of-band channel (SMS).	Open/Unencrypted	Not Feasible
Social-Login (OAuth) Portal	N/A (automation is infeasible).	Open/Unencrypted	Not Feasible
Passpoint / Hotspot 2.0	Secure provisioning of initial device profile/credential.	High (WPA2/3-Ent.)	Medium

[1] *Overall feasibility* balances implementation cost against security benefits for autonomous IoT.

[3] Aruba. 2021. ArubaOS 6.4.4.x Guide. https://support.hpe.com/hpesc/public/docDisplay?docId=a00107917en_us

[4] Cisco. 2011. MAC Authentication Bypass Deployment Guide. https://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Security/TrustSec_1-99/MAB/MAB_Dep_Guide.html

[5] Cisco. 2016. Cisco Catalyst 9800 Series Wireless Controller Software Configuration Guide, Cisco IOS XE Gibraltar 16.12.x - Hotspot 2.0 [Cisco Catalyst 9800 Series Wireless Controllers]. https://www.cisco.com/c/en/us/td/docs/wireless/controller/9800/16-12/config-guide/b_wl_16_12_cg/hotspot2.html

[6] Cisco. 2020. Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[7] Cisco. 2025. Cisco Identity Services Engine Administrator Guide, Release 3.4 - Guest and Secure WiFi [Cisco Identity Services Engine 3.4]. https://www.cisco.com/c/en/us/td/docs/security/ise/3-4/admin_guide/b_ise_admin_3_4/b_ISE_admin_guest.html

[8] Michal Garcarz and Nicolas Darchis. 2015. ISE Version 1.3 Self Registered Guest Portal Configuration Example. https://www.cisco.com/c/en/us/support/docs/security/identity-services-engine/118742-configure-ise-00.html

[9] Google Inc. 2016. Modernizing OAuth interactions in Native Apps for Better Usability and Security- Google Developers Blog. https://developers.googleblog.com/en/modernizing-oauth-interactions-in-native-apps-for-better-usability-and-security/

[10] Dick Hardt. 2012. *The OAuth 2.0 Authorization Framework*. Request for Comments RFC 6749. Internet Engineering Task Force. https://doi.org/10.17487/RFC6749 Num Pages: 76.

[11] Dan Harkins and Warren Kumari. 2017. *Opportunistic Wireless Encryption*. Request for Comments RFC 8110. Internet Engineering Task Force. https://doi.org/10.17487/RFC8110 Num Pages: 12.

[12] Jerome Henry and Yiu Lee. 2024. *Randomized and Changing MAC Address: Context, Network Impacts, and Use Cases*. Internet Draft draft-ietf-madinas-use-cases-19. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-madinas-use-cases Num Pages: 21.

[13] Naureen Hoque, Hanif Rahbari, and Cullen Rezendes. 2022. Systematically Analyzing Vulnerabilities in the Connection Establishment Phase of Wi-Fi Systems. In *2022 IEEE Conference on Communications and Network Security (CNS)*. IEEE, Austin, TX, USA, 64–72. https://doi.org/10.1109/CNS56114.2022.9947252

[14] Warren Kumari and Erik Kline. 2020. *Captive-Portal Identification in DHCP and Router Advertisements (RAs)*. Request for Comments RFC 8910. Internet Engineering Task Force. https://doi.org/10.17487/RFC8910 Num Pages: 11.

[15] Kyle Larose, David Dolson, and Heng Liu. 2020. *Captive Portal Architecture*. Request for Comments RFC 8952. Internet Engineering Task Force. https://doi.org/10.17487/RFC8952 Num Pages: 19.

[16] Zeyu Lei, Yuhong Nan, Yanick Fratantonio, and Antonio Bianchi. 2021. On the Insecurity of SMS One-Time Password Messages against Local Attackers in Modern Mobile Devices. In *Proceedings 2021 Network and Distributed System Security Symposium*. Internet Society, Virtual, 19. https://doi.org/10.14722/ndss.2021.24212

[17] Alexander Maier, Andrew Sharp, and Yuriy Vagapov. 2017. Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things. In *2017 Internet Technologies and Applications (ITA)*. IEEE, 143–148. https://doi.org/10.1109/ITECHA.2017.8101926

[18] Hyago Santana Mariano and Daniel Chaves Café. 2024. Measuring Public Wi-Fi Security Awareness via Captive Portal Connections Using a Microcontroller. In *2024 Workshop on Communication Networks and Power Systems (WCNPS)*. 1–6. https://doi.org/10.1109/WCNPS65035.2024.10814259 ISSN: 2768-0045.

[19] Nuno Marques, André Zúquete, and João Paulo Barraca. 2020. Integration of the Captive Portal paradigm with the 802.1X architecture. *Wireless Personal Communications* 113, 4 (Aug. 2020), 1891–1915. https://doi.org/10.1007/s11277-020-07298-y arXiv:1908.09927 [cs].

[20] MikroTik. 2021. Manual:Customizing Hotspot - MikroTik Wiki. https://wiki.mikrotik.com/Manual:Customizing_Hotspot

[21] Aruba Networks. 2012. Wi-Fi Certified Passpoint Architecture for Public ... - Aruba Networks. https://www.yumpu.com/en/document/view/6883907/wi-fi-certified-passpoint-architecture-for-public-aruba-networks

[22] Tommy Pauly and Darshak Thakore. 2020. *Captive Portal API*. Request for Comments RFC 8908. Internet Engineering Task Force. https://doi.org/10.17487/RFC8908 Num Pages: 11.

[23] Martiño Rivera-Dourado. 2022. *Captive Portal Network Authentication Based on WebAuthn Security Keys*. Master's thesis. Universidade da Coruña, A Coruña. https://ruc.udc.es/dspace/handle/2183/31921 Accepted: 2022-10-31T14:36:03Z.

[24] Muhammad Sangeen, Naveed Anwar Bhatti, Kashif Kifayat, Abeer Abdullah Alsadhan, and Haoda Wang. 2023. Blind-trust: Raising awareness of the dangers of using unsecured public Wi-Fi networks. *Computer Communications* 209 (Sept. 2023), 359–367. https://doi.org/10.1016/j.comcom.2023.07.011

[25] Anew H. Smith, Glen Zorn, John Roese, Bernard D. Aboba, and Paul Congdon. 2003. *IEEE 802.1X Remote Authentication Dial In User Service (RADIUS) Usage Guidelines*. Request for Comments RFC 3580. Internet Engineering Task Force. https://doi.org/10.17487/RFC3580 Num Pages: 30.

[26] Bryan Smith. 2018. Wi-Fi 6 (802.11ax) and Next Generation Hotspot boost industry confidence in Wi-Fi. https://wballiance.com/wi-fi-6-and-next-generation-hotspot-boost-industry-confidence-in-wi-fi/

[27] B A Sujathakumari and Husna Sabhat. 2018. A Theoretical Survey on MAC Address Blacklisting. *International Journal of Engineering Research* 6, 15 (2018), 4.

[28] Espressif Systems. 2025. Wi-Fi Security - ESP32 - — ESP-IDF Programming Guide v5.4.1 documentation. https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/wifi-security.html

[29] Ping-Lun Wang, Kai-Hsiang Chou, Shou-Ching Hsiao, Ann Tene Low, Tiffany Hyun-Jin Kim, and Hsu-Chun Hsiao. 2023. Capturing Antique Browsers in Modern Devices: A Security Analysis of Captive Portal Mini-Browsers. In *Applied Cryptography and Network Security: 21st International Conference, ACNS 2023, Kyoto, Japan, June 19–22, 2023, Proceedings, Part I*. Springer-Verlag, Berlin, Heidelberg, 260–283. https://doi.org/10.1007/978-3-031-33488-7_10

[30] Yoshiaki Watanabe, Makoto Otani, Hirofumi Eto, Kenzi Watanabe, and Shin-ichi Tadaki. 2013. A MAC address based authentication system applicable to campus-scale network. In *2013 15th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. 1–3. https://ieeexplore.ieee.org/document/6665242