

Captive Portal Analyzer Project

Amr Elganainy

Contents

1	Introduction	2
2	Methodology	3
2.1	Custom Captive Portal Development	3
2.2	Network Traffic Capture Implementation	4
3	Setting up a Captive Portal on Raspberry Pi	5
3.1	Prerequisites	5
3.2	Structure Overview	6
4	Setting up the Remote Server	8
4.1	Authentication website	9
4.2	Authentication API	10
5	Captive Portal Analyzer Mobile App [3]	11
5.1	Overview	11
5.2	App Architecture	12
5.3	Screens Overview and Implementation	18
5.3.1	Welcome Screen	18
5.3.2	PCAPDroid Setup Screen (<code>SetupPCAPDroidScreen</code>)	19
5.3.3	Manual Connect Screen (<code>ManualConnectScreen</code>)	20
5.3.4	Analysis Screen (<code>AnalysisScreen</code>)	21
5.3.5	Screenshot Flagging Screen (<code>ScreenshotFlaggingScreen</code>)	25
5.3.6	Session List Screen (<code>SessionListScreen</code>)	26
5.3.7	Session Details (<code>SessionScreen</code>)	27
5.3.8	Automatic Analysis Screen (Now a Multi-Screen Flow: <code>AutomaticAnalysisGraphRoute</code>)	30
5.3.9	Settings Screen (<code>SettingsScreen</code>)	35
5.4	Data Upload and Server View	36

6	Results	38
6.1	Networks with Privacy Violations	38
6.1.1	Test CaptivePortal (Test Environment)	38
6.1.2	ASK4 WiFi (Hotel Network)	39
6.1.3	Vodafone Hotspot (Telecom)	40
6.2	Networks with No Security Issues	40
6.2.1	Networks Using eu.network-auth.com Service Provider	40
6.2.2	Networks with Different Service Providers	42
6.3	Summary of Findings	46
7	Limitations and Workarounds	47
7.1	Android Request Inspector WebView	47
7.1.1	Workaround	47
7.2	Screenshot Functionality	47
7.2.1	Workaround	48
8	Conclusion	49

Foreword

This report is part of the master’s project titled *WiFi Hotspot Analyzer Project*, conducted under the supervision of Dr. Tobias Urban, Research Associate Henry Hosseini, and Research Associate Christian Böttger during the winter semester of 2024/25 at the *Westfälische Hochschule Gelsenkirchen*, within the Department of Computer Science and Communication. The project was carried out in collaboration with the *Institute for Internet Security—if(is)*.

1 Introduction

This technical report outlines the implementation and analysis of a modern captive portal system integrated with a remote authentication server, specifically designed for deployment on a Raspberry Pi. The captive portal seamlessly redirects users to the authentication server, where they can log in to gain internet access for a limited duration of one hour.

Complementing this infrastructure, a dedicated mobile application has been developed to analyze captive portals in a comprehensive way. The app facilitates direct interaction with captive portals through an integrated WebView and, for deeper inspection, supports packet capture by integrating with the PCAPdroid application. This dual approach enables users to collect and analyze critical data such as POST request bodies, headers, session details, raw network traffic

(PCAP files), and associated HTML/JavaScript content. With advanced features like AI-powered data analysis via the Gemini AI SDK, the application evaluates privacy practices and data collection policies through a guided, multi-step process allowing fine-grained data selection. Users can also capture screenshots, mark them for relevance (e.g., Terms of Service), and upload all analyzed data for further examination by our team, ensuring a robust and thorough evaluation process.

Together, through the integration of the Raspberry Pi-based captive portal and the enhanced analytical capabilities of the mobile app (including WebView analysis, PCAPdroid integration, and multi-step AI analysis), a cohesive framework is formed for implementing, managing, and comprehensively analyzing captive portal solutions.

Analyzing captive portals is crucial as it sheds light on network access practices, user data handling, and potential vulnerabilities. These aspects are increasingly relevant in securing modern networks and ensuring user privacy, particularly with the varied data collection methods employed by different captive portal implementations.

2 Methodology

Our research methodology addresses two primary technical challenges in network traffic analysis: captive portal access and comprehensive traffic capture. We detail our approaches to overcome these challenges while maintaining practical usability and research validity.

2.1 Custom Captive Portal Development

To ensure consistent and controlled testing conditions, we implemented a custom captive portal system using Raspberry Pi hardware which will be discussed in sections 3 and 4. This approach was necessitated by several limitations in accessing public captive portals:

1. The availability of public portals during development is dependent on the developer's location.
2. Restricted configuration options in existing portals.
3. Inability to simulate security violations for testing.

Our custom implementation provides:

- Complete configuration control over portal settings.

- Consistent availability for development and testing.
- Capability to simulate various security scenarios.
- Reproducible testing environment.

The portal was configured to mirror typical public WiFi authentication systems while allowing necessary modifications for research purposes.

2.2 Network Traffic Capture Implementation

Initial attempts to capture network traffic using traditional MITM proxy methods revealed significant technical limitations, particularly with modern Android devices. We identified four major constraints:

1. Android 10+ security restrictions: The system prevents users from installing system-level CA certificates required for man-in-the-middle (MitM) attacks for general applications.
2. Limited device compatibility: Less than 5% of devices are running compatible versions of Android 10 or earlier[2] that might allow easier system-level certificate installation for older MITM techniques.
3. External hardware dependency: Traditional MITM proxy setups often require an attached laptop or a separate device to act as the proxy, which can be inconvenient for mobile users.
4. Certificate Pinning: Some captive portal sites may implement certificate pinning, hindering traditional MitM attacks by rejecting unrecognized CA certificates.

To address these challenges and provide a robust traffic capture solution, our mobile application, detailed in Section 5, employs a multi-faceted approach:

- **WebView-based Capture:** Utilizes a WebView with JavaScript injection for capturing HTTP/HTTPS requests, including POST bodies and headers, directly within the app. This method is effective for analyzing application-layer interactions with the captive portal.
- **PCAPdroid Integration:** For comprehensive network-level analysis, the application integrates with PCAPdroid, a third-party Android application. This allows for the capture of raw network packets (PCAP files) with appropriate user setup for TLS decryption. This approach overcomes the limitations of system-level CA certificate installation for general apps by leveraging PCAPdroid's capabilities, which often involve using a VPN service or root access (though the app aims for non-root usage via PCAPdroid's VPN mode).

- **In-app Data Processing:** Collected data, including WebView interactions and processed information from PCAP files, is managed and analyzed within the app.
- **Integrated AI-powered Privacy Analysis:** A guided, multi-step process allows users to select specific data points for analysis by the Gemini AI model, offering insights into privacy practices.
- **User-Friendly Interface:** Provides step-by-step guidance for both WebView-based analysis and PCAPdroid setup and usage.
- **Remote Storage Capability:** Facilitates the upload of collected session data, including screenshots and PCAP file references, for further research.

This combined methodology enables comprehensive traffic analysis, from application-layer details to raw packet data, while maintaining practical usability for non-technical users and ensuring research validity for academic purposes.

The implementation details of both the custom captive portal and the mobile application's traffic capture and analysis system are further elaborated in subsequent sections of this paper.

3 Setting up a Captive Portal on Raspberry Pi

This section provides a walkthrough of how the Raspberry Pi emulates a captive portal using a combination of iptables firewall rules, a database, and Node.js for traffic redirection. An additional JavaScript service manages device sessions, granting one-hour access after successful authentication with the remote server. Although tailored for the Raspberry Pi, this approach is compatible with any Linux system with WLAN capabilities. For detailed implementation steps, refer to the project's GitHub repository.[4].

3.1 Prerequisites

- **Operating System:** Any Ubuntu/Debian RPi distribution
- **Network Setup:** Ethernet cable connection for the RPi
- **Domain Name:** (optional) Used for the remote Authentication website
- **Dynamic Hosting Provider:** AWS or other preferred hosting services

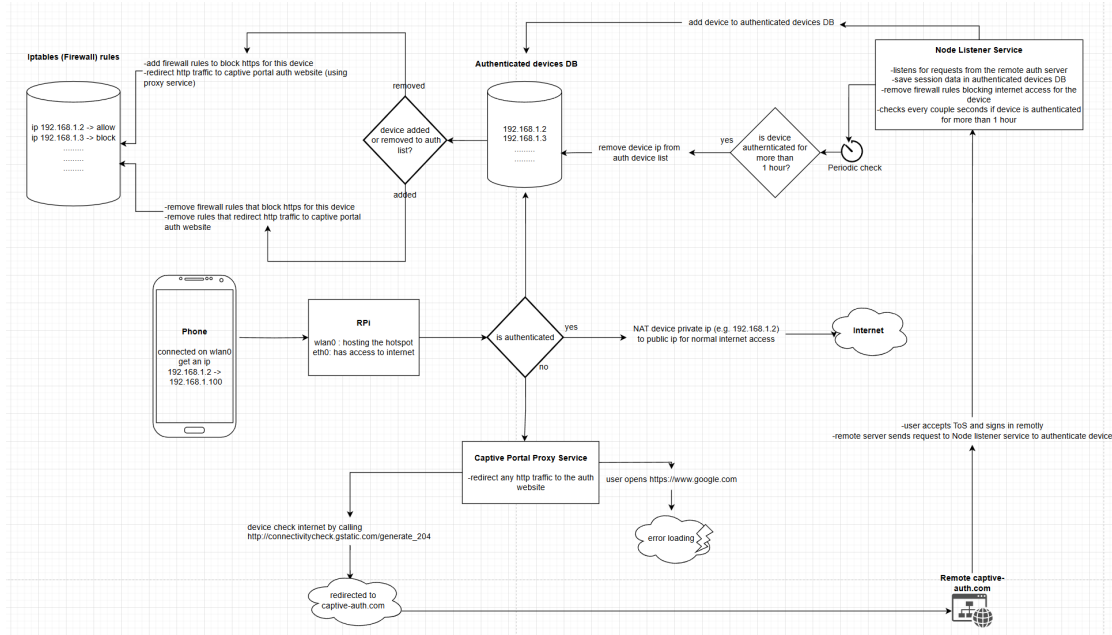


Figure 1: A sketch of the Raspberry Pi setup. This shows the components involved in mimicking the captive portal functionality.

3.2 Structure Overview

Hotspot Setup

- NetworkManager is used for hotspot creation.
- At this point, this is just a normal Wi-Fi network where users can either:
 - Enter the password (if one is set), or
 - Connect directly (if no password is set),

to gain full internet access.

DNS and DHCP Configuration

- dnsmasq is used to:
 - Control the IP range assigned to devices connecting to the Raspberry Pi's Wi-Fi network, which in this case is from 192.168.1.2 to 192.168.1.100.

- Manage DNS resolution, manually pointing to the IP of the authentication server.

Network Routing

- The Raspberry Pi has two interfaces:
 - `wlan0`: For devices connecting to the hotspot.
 - `eth0`: For the Raspberry Pi's internet access.
- IP forwarding must be enabled to allow devices to move packets from `wlan0` to `eth0` and back.

Firewall Configuration

- Initially, all devices are blocked from accessing the internet by setting up `iptables` rules for traffic control.
- Example commands:

```
sudo iptables -t nat -A PREROUTING -i wlan0 -p tcp -s
↳ $UNAUTHENTICATED_DEVICE_IP --dport 80 -j DNAT
↳ --to-destination 192.168.1.1:8080
sudo iptables -A FORWARD -i wlan0 -p tcp -s
↳ $UNAUTHENTICATED_DEVICE_IP --dport 443 -j REJECT
↳ --reject-with icmp-port-unreachable
```

- The above commands:
 - Redirect HTTP traffic to port 8080, where the Captive Portal Proxy Service listens.
 - Block HTTPS traffic for the unauthenticated device.

Traffic Management

- Direct redirection to a specific website using `iptables` isn't possible, plus we need to include some extra header information to the HTTP request.
- Instead, all HTTP traffic is redirected to the Captive Portal Proxy Service.
- This service:
 - Adds headers containing device information (e.g., MAC address, IP, user-agent).

- Forwards the request to the remote authentication website, allowing it to identify valid requests.

Remote Authentication

- Remote authentication is performed by the remote server (details covered in section 4).
- Once authentication is complete, the remote server sends a request to port 4001 of the Raspberry Pi.
- Another service (Node Listener Service) listens for these requests.
- A persistent connection service is configured.

Node Listener Service

- This service performs several tasks:
 - Listens for requests from the remote authentication server.
 - Saves session data in the authenticated devices database.
 - Removes firewall rules blocking internet access for authenticated devices.
 - Periodically checks if a device has been authenticated for more than one hour and, if so, re-applies firewall rules to block internet access.

For detailed implementation steps, configuration files, and scripts, please refer to the project's GitHub repository [4].

4 Setting up the Remote Server

This section outlines the process of setting up the remote server infrastructure for the captive portal system, which the Raspberry Pi will use to remotely authenticate users, simulating the behavior of real-world captive portals. For detailed implementation steps, refer to the project's GitHub repository [4], particularly the "remote-website" and "remote-auth-server" folders.

The website we created for testing is currently accessible at <https://captive.ganainy.online/>. However, please note that the login process cannot be completed unless the website is accessed through redirection from the captive portal, which sends the necessary session data required for user authentication.

Why not host the website for login and the API for authentication directly on the Raspberry Pi and instead choose a remote server?

The decision to use a remote server is made to mimic the real-world behavior of captive portals commonly used by most companies (except for some small businesses that may use local servers to reduce costs by not relying on hosting services). These businesses typically use a remote server with a central database for users, allowing users to access their credentials at different locations within the business. The central database also makes it easier to gather data for marketing purposes.

4.1 Authentication website

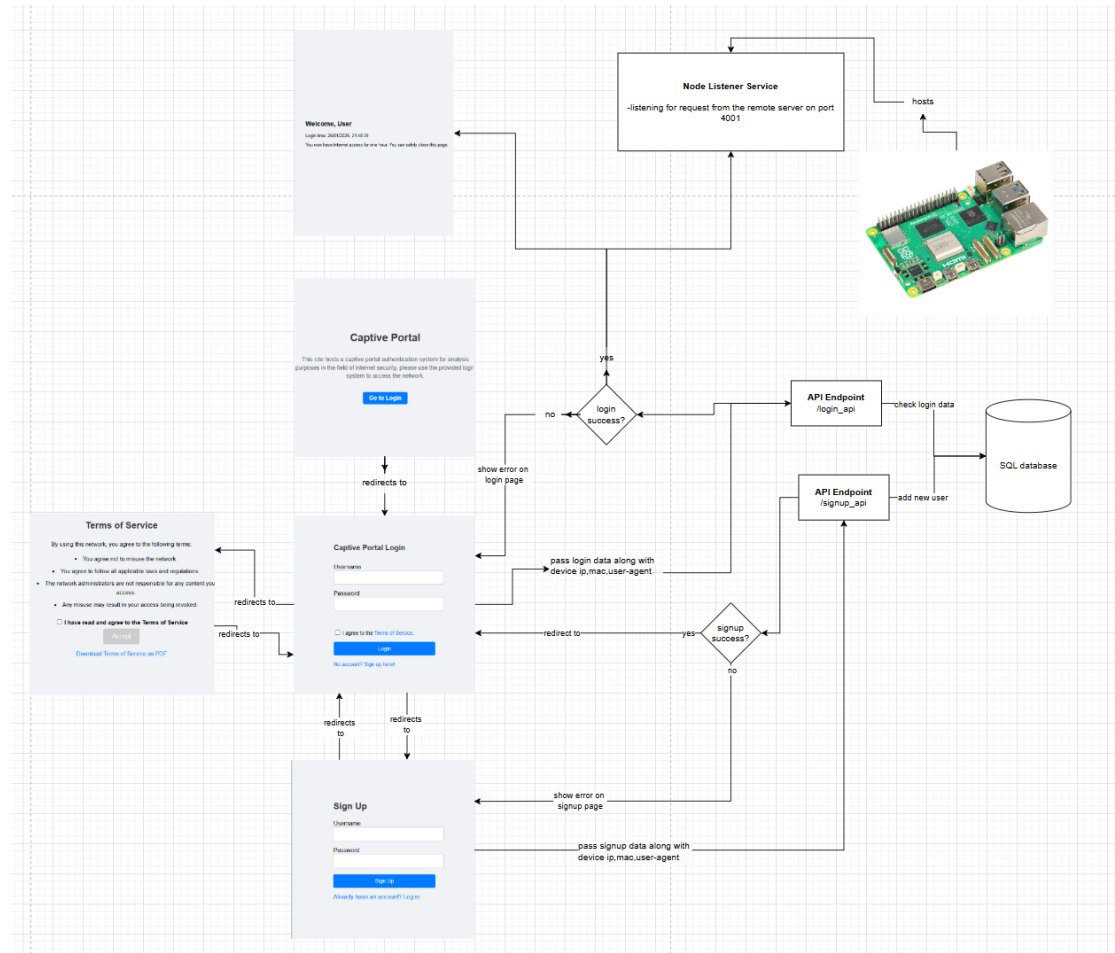


Figure 2: A sketch of the remote server setup. This shows the components involved in the authentication process.

1. The website consists of four main pages:

- **Welcome Page**

This page greets the user after a successful login and informs them that they have internet access for one hour.

- **Sign-In Page**

This page prompts the user to log in with a username and password or navigate to the sign-up page. Users must approve the Terms of Service (ToS) before logging in. When a login request is made, the website sends a request to the authentication API endpoint `auth.ganainy.online/login-api` and includes details such as the IP and MAC address of the device attempting to log in. If the login fails, an appropriate error message is displayed to notify the user.

- **Sign-Up Page**

This page allows users to create a new account by sending a request to the authentication API endpoint `auth.ganainy.online/signup-api` with information such as the IP and MAC address of the device. Upon successful sign-up, the user is redirected to the sign-in page to log in. If the sign-up fails, a relevant error message is displayed to inform the user.

- **Terms of Service Page**

A simple page where users can read a mock version of the Terms of Service.

4.2 Authentication API

2. Authentication API has two main endpoints:

- `/login_api`: Authenticates users and validates credentials.

Uses MySQL to check if the user login request is valid. Implements password hashing with `bcrypt`. Integrates with the Node Listener Service discussed earlier. Upon a successful login, a request is sent to the Node Listener Service, which is hosted on the Raspberry Pi marking the login as successful and granting the user internet access for one hour. It also returns a success or error code to the authentication website to display the appropriate HTML (e.g., welcoming the user or prompting them to check their username or password).

- `/signup_api`: Registers new users.

Checks if the given signup data is valid and whether the information is

already used by other users. If valid, it stores the data in the database and returns a success code, allowing the website to navigate the user to the login page. Otherwise, it returns an error message (e.g., username already taken, password too short) to be displayed to the user upon a failed signup.

For detailed implementation steps, configuration files, and scripts, please refer to the project's GitHub repository [4].

5 Captive Portal Analyzer Mobile App [3]

5.1 Overview

The Captive Portal Analyzer app is designed to streamline the process of analyzing captive portals by offering an intuitive interface for users to interact with them directly. The app supports two primary analysis modes:

- **WebView-based Analysis:** This mode leverages a custom implementation of WebView [1] with JavaScript injection capabilities, enabling it to capture a wide range of data. This includes POST request bodies, headers, URLs, URL parameters, screenshots, and the HTML and JavaScript files associated with the captive portal.
- **PCAPdroid-based Packet Capture:** For more in-depth network analysis, the app integrates with PCAPdroid (a third-party application). This allows for the capture of raw network packets during the captive portal interaction. The app guides users through setting up PCAPdroid for TLS decryption and targeted app capture. Captured PCAP files can then be processed and included in the analysis.

The collected data is systematically organized into "sessions," providing a comprehensive record of user interactions. These sessions can be analyzed within the app using an integrated AI model (Gemini AI SDK [7]) through a new multi-step flow, offering quick insights into the portal's behavior and privacy practices. This flow allows users to select specific data points (requests, screenshots, web content, and processed PCAP data) and customize prompts for the AI. Additionally, the app allows users to review the collected data before optionally uploading it to a remote server for further manual analysis by researchers or other stakeholders. After the primary analysis (WebView or PCAP-based interaction), users are guided to a new **Screenshot Flagging Screen** to mark relevant screenshots (e.g., ToS, privacy policy) before viewing the session list.

5.2 App Architecture

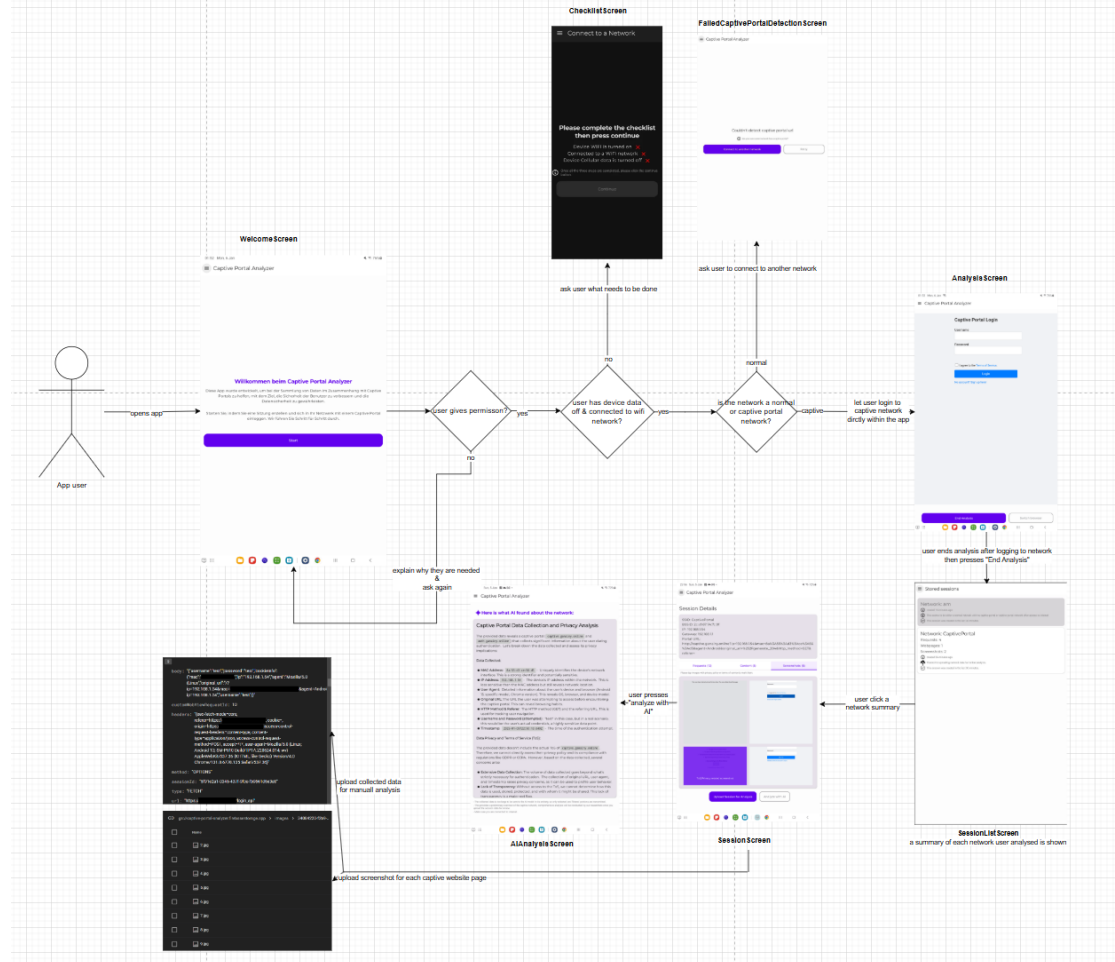


Figure 3: A sketch of the app flow from the start to uploading and analyzing collected data.

General App Software Architecture

The app follows the MVVM (Model-View-ViewModel) architecture, where each screen composable (e.g., `ManualConnectScreen`, `AnalysisScreen`) typically has its own `ViewModel` (e.g., `ManualConnectViewModel`, `AnalysisViewModel`) responsible for business logic. `ViewModels` interact with the `NetworkSessionRepository`, which serves as the single source of truth for both local (Room database) and remote (Firebase) data. Screens observe `UiState` from their `ViewModels` and

pass user actions back to them. This structure separates concerns and facilitates decoupling between UI and business logic.¹

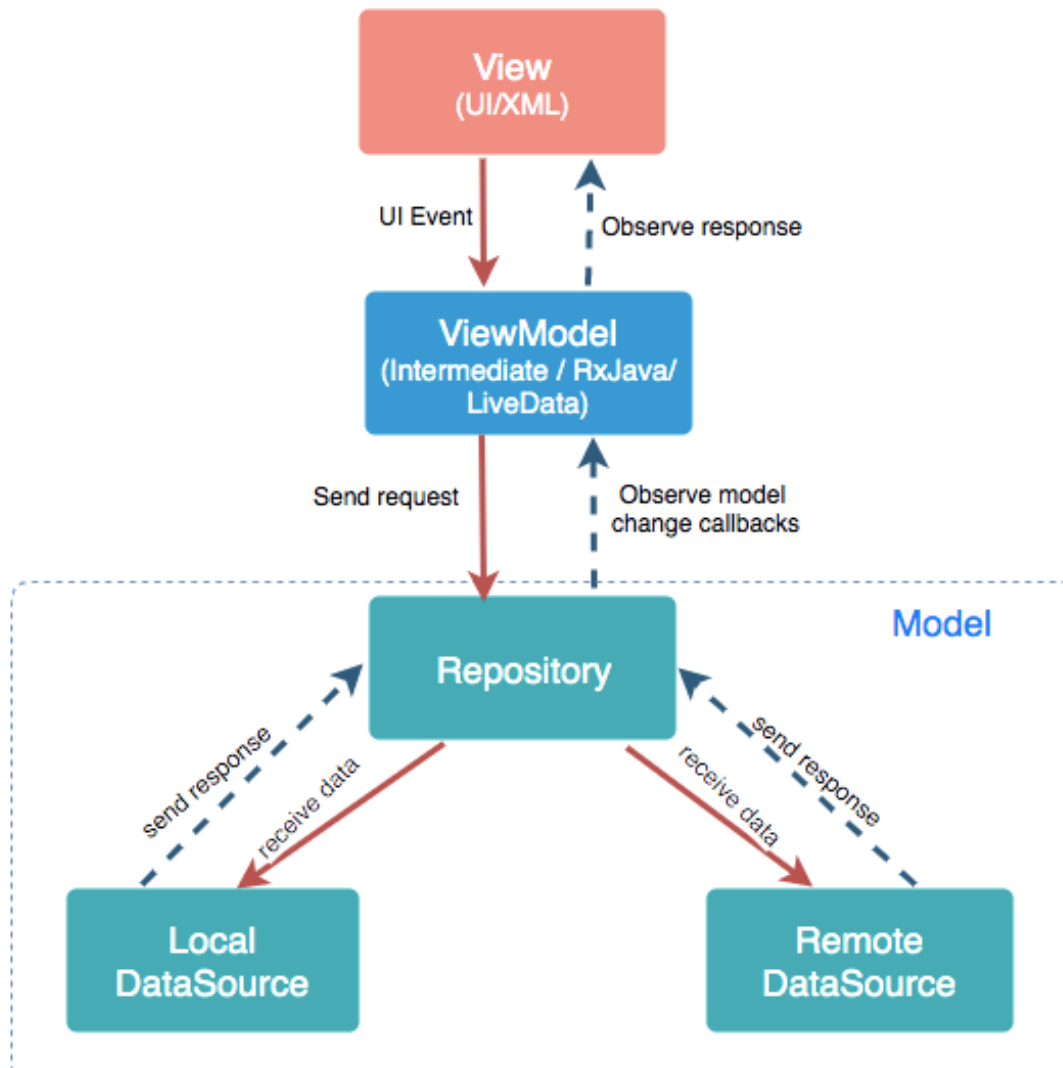


Figure 4: MVVM Architecture.

The App uses a single activity, `MainActivity.kt`, which initializes critical components like `AppDatabase`, `NetworkSessionManager`, and the `MainViewModel` (for global state, dialogs, toast messages, PCAPdroid interaction, and theme/locale

¹MVVM (Model-View-ViewModel) is an architectural pattern that separates the development of the graphical user interface from the business logic or back-end logic (Model).

settings). **MainActivity** also handles the setup and results for PCAPdroid control intents (start, stop, status) and the Storage Access Framework (SAF) for selecting PCAP files. A **BroadcastReceiver** is used to monitor PCAPdroid's capture status externally. Navigation is managed by a **NavController** within **AppNavGraph.kt**. A significant architectural update is the introduction of **nested navigation graphs**:

- **PrimaryAnalysisGraphRoute**: Encapsulates the main analysis flow, starting with **AnalysisScreen** and proceeding to the new **ScreenshotFlaggingScreen**. ViewModels like **AnalysisViewModel** and **ScreenshotFlaggingViewModel** are scoped to this graph, allowing them to share data and state during this flow.
- **AutomaticAnalysisGraphRoute**: Manages the multi-step AI analysis, including **AutomaticAnalysisInputScreen**, **PcapInclusionScreen**, **AutomaticAnalysisPromptPreviewRoute**, and **AutomaticAnalysisOutputRoute**. The **AutomaticAnalysisViewModel** is scoped to this graph.

AppNavGraph.kt defines **Screen** objects with routes, navigation actions (e.g., **actions.navigateToPrimaryAnalysisGraph**, **actions.navigateToAutomaticAnalysisGraph**), and handles dependency injection for screens and their graph-scoped ViewModels.

MainActivity.kt As the app's entry point, it initializes critical classes used throughout the application:

AppDatabase Responsible for local data persistence using the Room library. Key methods include:

- **insertSession(session: NetworkSessionEntity)**
 - Purpose: Stores a new network session in the local database
 - Use case: When a user connects to a new network, this method creates a persistent record
 - Key parameters:
 - * **session**: Contains all metadata about the network session (SSID, connection details, PCAP file path, etc.)
- **getSessionByNetworkId(networkId: String): NetworkSessionEntity?**
- **updateSession(session: NetworkSessionEntity)**
- **updatePortalUrl(sessionId: String, portalUrl: String)**

- `updateIsCaptiveLocal(sessionId: String, isLocal: Boolean)`
- `getSessionRequests(sessionId: String): List<CustomWebViewRequestEntity>`
- `insertWebpageContent(content: WebpageContentEntity)`
- `getSessionWebpageContent(sessionId: String): List<WebpageContentEntity>`
- `getCompleteSessionData(sessionId: String): Flow<SessionData>`
- Screenshot-related methods: (`insertScreenshot`, `getSessionScreenshots` - now also includes `isPrivacyOrTosRelated` flag)
- Remote upload method (`uploadSessionData`): The `SessionUploader` (part of repository) handles this. Uploaded data may now include PCAP file references or summaries.

NetworkSessionManager Responsible for creating and managing network sessions:

- `getCurrentSession(): NetworkSessionEntity?`
 1. Checks app permissions
 2. Retrieves network identifiers (SSID, BSSID, gateway address, DHCP server address)
 3. Creates unique network identifier (using last step network data)
 4. Checks if network is already stored
 5. Loads existing session or creates new one (based on last step result), now potentially including initializing PCAP-related fields.
- Uses `ConnectivityManager.NetworkCallback()` to:
 - Detect WiFi connection changes
 - Call `getCurrentSession()` to ensure session information is updated on network changes

NetworkConnectivityObserver Monitors device network connectivity to:

- Observe network connection status
- Provide listeners with connection loss updates
- Listeners can decide to take actions based on this information (e.g. refuse operations requiring internet connection such as uploading the session to the remote server or performing AI analysis).

MainViewModel Manages cross-screen and app-wide state:

- Share data between compose screens and MainActivity (e.g., `clickedSessionId`, `PcapDroidCaptureState`, `copiedPcapFileUri`).
- Manage global UI interactions (dialogs, toasts).
- Handle app-wide settings (language, dark mode) via `DataStore`.
- Manages state and intents for PCAPdroid interaction.
- Manages the `skipSetup` preference for PCAPdroid.

AppNavGraph.kt Defines application navigation:

- Create `Screen` objects with routes (now includes `PCAPDroidSetup`, `ScreenshotFlagging`, and nested graph routes).
- Define navigation actions within `NavigationActions` class.
- Manages dependency injection for screens, including graph-scoped `ViewModels`.
- Implements nested navigation for `PrimaryAnalysisGraphRoute` and `AutomaticAnalysisGraphRoute`.

Example screen definition (now within a nested graph for Analysis and ScreenshotFlagging):

```
// --- Nested Navigation Graph for Primary Analysis Flow ---
navigation(
    startDestination = Screen.Analysis.route,
    route = PrimaryAnalysisGraphRoute
) {
    composable(route = Screen.Analysis.route) { backStackEntry ->
        val graphEntry = remember(backStackEntry) {
            navController.getBackStackEntry(PrimaryAnalysisGraphRoute)
        }
        val analysisViewModel: AnalysisViewModel = viewModel(
            viewModelStoreOwner = graphEntry, /* ... factory ... */
        )
        AnalysisScreen(
            /* ...dependencies... */
            onNavigateToScreenshotFlagging =
```



```
        actions.navigateToScreenshotFlaggingScreen
    )
}
composable(route = Screen.ScreenshotFlagging.route) { backStackEntry ->
    // Get the SAME ViewModel instance or a new one scoped to the graph
    /* ... ScreenshotFlaggingScreen setup ... */
}
}
```

This structure ensures that `AnalysisViewModel` and `ScreenshotFlaggingViewModel` can share data if needed, as they are scoped to the `PrimaryAnalysisGraphRoute`. Navigation between them happens within this graph.

- **repository, sessionManager:** As described before.
- **mainViewModel:** Provides access to global states like PCAPdroid status, dialogs, etc.
- Navigation actions like `navigateToScreenshotFlaggingScreen` are defined to move between screens within the same nested graph, or actions like `navigateToSessionListScreen` navigate out of the graph.

5.3 Screens Overview and Implementation

Now that we have covered the shared app components, we will explore each screen from both the frontend and backend perspectives.

Frontend: A screenshot of the screen, along with a text description that provides an overview of the specific screen.

Backend: This section outlines the functions and flow of each screen, explaining how the app operates behind the scenes. Typically, each screen is divided into a `ViewModel` which handles the logic and a `ScreenCompose` that manages the UI. However, for simple screens like `AboutScreen`, `SettingsScreen`, or `WelcomeScreen` (which may now have more logic due to `skipSetup`), a `ViewModel` might not be strictly necessary if logic is minimal.

Notes:

- Users are advised to use placeholder data during the signup and login processes, as this data, with user consent, will be uploaded to the server for further analysis.
- For a more thorough understanding of the app's functionality, a video preview is available in the app's GitHub repository: [3].

5.3.1 Welcome Screen

Frontend

On app launch, the user is shown a welcome page (See Figure 5) with options to either proceed to PCAPdroid setup or directly to the manual connection checklist if PCAPdroid setup has been previously skipped (controlled by a `skipSetup` preference).

Backend

- Checks the `skipSetup` preference (managed by `MainViewModel` and stored in `DataStore`).
- If `skipSetup` is true or the user chooses to skip, navigates to `Screen.ManualConnect.route`.
- Otherwise, navigates to `Screen.PCAPDroidSetup.route`.

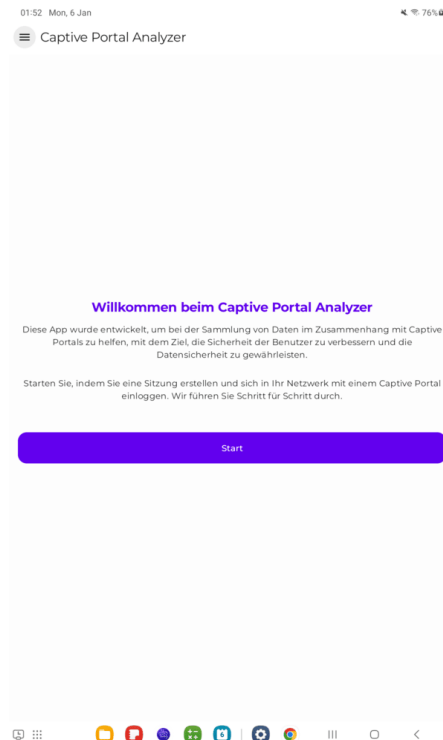


Figure 5: Welcome and permission granting screen.

5.3.2 PCAPDroid Setup Screen (SetupPCAPDroidScreen)

Frontend

A multi-step guided tutorial (using a `HorizontalPager`) that instructs the user on how to:

1. Install PCAPdroid from the Play Store (if not already installed, detected by `SetupPCAPDroidViewModel`).
2. Enable TLS decryption in PCAPdroid settings.
3. Add decryption rules in PCAPdroid for the "Captive Portal Analyzer" and "Captive Portal Login" apps.
4. Select "Captive Portal Analyzer" and "Captive Portal Login" as target apps in PCAPdroid.

Videos (`LoopingVideoPlayer`) are embedded to demonstrate steps 3 and 4. A checkbox allows users to "Don't show this setup again," which updates the `skipSetup` preference.

Backend (SetupPCAPDroidViewModel)

- Checks if PCAPdroid is installed.
- Manages the current step of the tutorial.
- Handles the `skipSetup` checkbox state.
- `updateSkipSetup` callback updates the preference in `MainViewModel`.
- Navigates to `Screen.ManualConnect.route` upon completion or if the user skips introductory steps via a dedicated link.

5.3.3 Manual Connect Screen (ManualConnectScreen)**Frontend**

Prompts the user to grant necessary permissions (Location, Wi-Fi state). Displays a checklist (As shown in Fig 6) for:

- Device Wi-Fi turned on.
- Connected to a Wi-Fi network.
- Device Cellular data turned off.

The "Continue" button is enabled only when all permissions are granted and checklist items are met.

Backend (ManualConnectViewModel)

- Manages permission requests (`checkPermissions`, `onPermissionResult` using `ActivityResultContracts.RequestMultiplePermissions`).
- Monitors Wi-Fi state, cellular data state, and network connectivity.
- Updates the UI checklist.
- When requirements are fulfilled and "Continue" is pressed, it now navigates to the `PrimaryAnalysisGraphRoute` (which starts with `AnalysisScreen`) via `actions.navigateToPrimaryAnalysisGraph`.

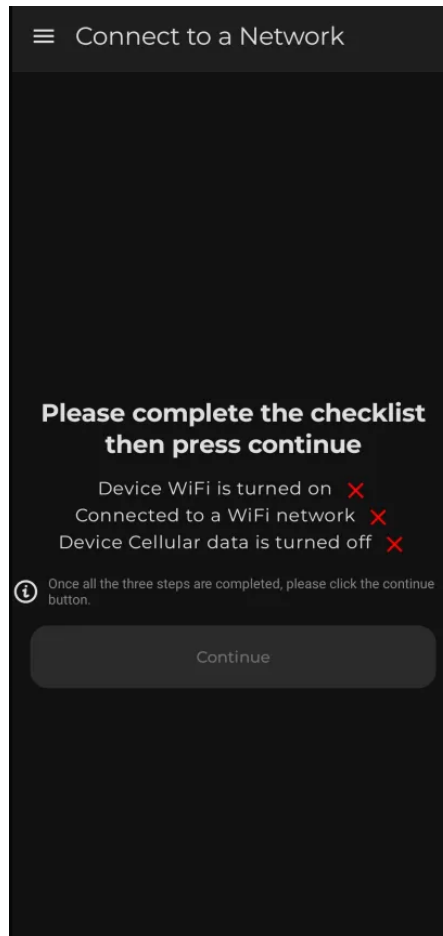


Figure 6: Checklist of steps needed for starting analysis.

5.3.4 Analysis Screen (AnalysisScreen)

The Analysis screen is the core of data collection and is the first screen in the `PrimaryAnalysisGraph`.

Frontend

- Features a `TabRow` with two tabs: "WebView" and "Packet Capture".
- **WebView Tab:**
 - Attempts to load the network's captive portal page using an HTTP request in the `WebView`. Handles redirects to detect captive portals or confirms direct internet access (prompting to connect to a different network if so, similar to Figure 7).

- Displays the captive portal URL.
- Allows interaction with the portal, logging pages, capturing screenshots, and monitoring HTTP requests/responses using either the "Android Request Inspector WebView" [1] (with JS injection) or a vanilla WebView (switchable via a "Switch Browser" button, similar to Figure 8).
- An "End Analysis" button.
- A "Packet Capture Mode Preference" section (`PreferenceSetupContent`) that appears if a captive portal URL is not yet detected. This section allows the user to choose between:
 - * "Continue with Packet Capture (Advanced)": Requires PCAPdroid setup. Navigates to `SetupPCAPDroidScreen` if PCAPdroid is not installed or not configured. If PCAPdroid is ready, it starts packet capture via `MainActivity` intents and sets `PcapDroidPacketCaptureStatus.PCAPDROID_CAPTURE_ENABLED`.
 - * "Continue without Packet Capture (Default WebView)": Sets `PcapDroidPacketCaptureStatus.PCAPDROID_CAPTURE_DISABLED` and proceeds with WebView-only analysis.

- **Packet Capture Tab:**

- Its content depends on `PcapDroidPacketCaptureStatus` (managed by `MainViewModel` and observed by `AnalysisScreen`):
 - * `INITIAL`: Shows `PacketCaptureInitialContent` (prompting to select a mode via the WebView tab's preference section).
 - * `PCAPDROID_CAPTURE_DISABLED`: Shows `PacketCaptureDisabledContent` (informing the user analysis will proceed without packet capture and to use "End Analysis" from WebView tab).
 - * `PCAPDROID_CAPTURE_ENABLED`: Shows `PacketCaptureEnabledContent`. This view has a multi-step UI:
 1. **Start/Stop Capture:** Buttons to "Start" (if `captureState` is `IDLE/STOPPED`) or "Stop" (if `RUNNING`) PCAPdroid capture via `MainActivity` intents. Displays current `captureState` (`IDLE`, `STARTING`, `RUNNING`, `STOPPING`, `FILE_READY`, `WRONG_FILE_PICKED`) and `statusMessage` from `MainViewModel`. Includes a refresh icon to re-check status.
 2. **Select and Copy PCAP File:** Once capture is stopped and `captureState` is `FILE_READY`, prompts the user to "Open File" (triggering SAF via `MainActivity`) to select the expected PCAP file (name provided by `targetPcapName`). If the wrong

file is picked, `WRONG_FILE_PICKED` state is shown with an error. On correct selection, `copiedPcapFileUri` is updated in `MainViewModel`.

3. **Save PCAP File:** A "Save PCAP File" button (enabled when `copiedPcapFileUri` is not null and state is `FILE_READY`) which calls `analysisViewModel::storePcapFilePathInTheSession` and then `analysisViewModel::markAnalysisAsComplete`.

Backend (AnalysisViewModel - Scoped to PrimaryAnalysisGraphRoute)

- Handles captive portal detection (`getCaptivePortalAddress`) and UI state management (`AnalysisUiState`, `AnalysisUiData`) for the WebView tab.
- Manages WebView type (custom/normal) and JS injection.
- Saves web requests, webpage content (HTML/JS), and screenshots to the repository.
- Initiates PCAPdroid actions (start, stop, status) by calling methods on `MainViewModel` which then launch the intents via `MainActivity`.
- Stores the path of the selected PCAP file into the current session via `storePcapFilePathInTheSession` when the user successfully selects a file in the Packet Capture tab.
- When "End Analysis" (from WebView tab if packet capture disabled) or "Save PCAP File" (from Packet Capture tab if enabled) is pressed, it calls `markAnalysisAsComplete`.
- If `markAnalysisAsComplete` is successful and all data is saved, it updates `AnalysisUiState` to `AnalysisCompleteNavigateToNextScreen`.
- The `AnalysisScreen` composable observes this state and navigates to the `ScreenshotFlaggingScreen` using `navigationConfig.onNavigateToScreenshotFlagging`.
- Checks if PCAPdroid is installed via `isPcapDroidAppInstalled` (delegated from `SetupPCAPDroidViewModel`).

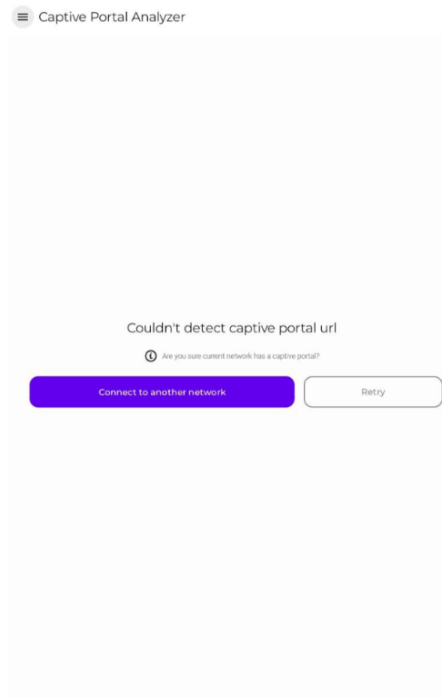


Figure 7: Automatic detection of captive/normal networks.

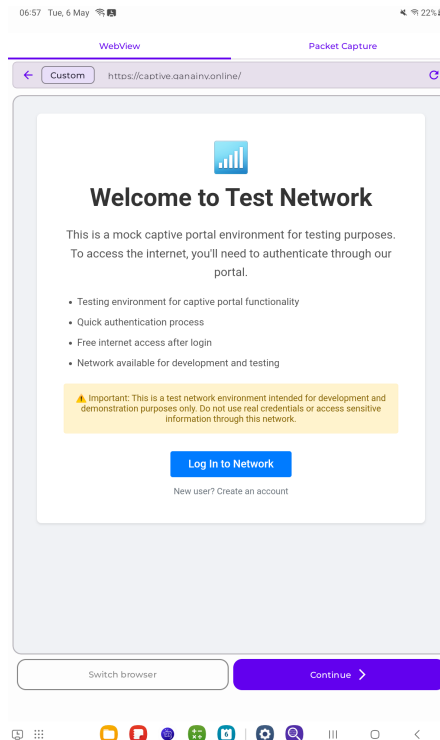


Figure 8: Interacting with the captive portal directly from the app, gathering requests, request bodies, and headers. (WebView Tab view)

5.3.5 Screenshot Flagging Screen (ScreenshotFlaggingScreen)

This screen is the second step in the `PrimaryAnalysisGraphRoute`.

Frontend

- Displays screenshots captured during the `AnalysisScreen` interaction.
- Allows the user to tap on images to mark/unmark them as "Privacy/ToS related".
- A "Finish Flagging" button.

Backend (ScreenshotFlaggingViewModel - Scoped to PrimaryAnalysisGraphRoute)

- Receives the current session ID (implicitly, as it shares the graph with `AnalysisViewModel` which created/updated the session).
- Loads screenshots for the current session from the repository.

- Handles the toggling of the `isPrivacyOrTosRelated` flag for screenshots and updates them in the repository.
- When "Finish Flagging" is pressed, navigates to the `SessionListScreen` via `onFinishFlagging` (which calls `actions.navigateToSessionListScreen`).

5.3.6 Session List Screen (`SessionListScreen`)

Frontend

- Displays a `TabRow` with two `FilterTab` composables: "Captive" and "Normal".
- Shows a list of network sessions (`NetworkSessionItem`) based on the selected filter.
- Each `NetworkSessionItem` displays SSID, IP, gateway, and (for captive portals) counts of requests, webpages, and screenshots. It also shows when the session was created (time ago).
- Captive portal items are clickable to view details. Normal network items are not interactive for navigation.
- An icon button allows deletion of a session (with a confirmation dialog shown via `MainViewModel`).
- Displays appropriate empty state messages if no sessions exist overall, or if no sessions match the current filter (e.g., "No captive sessions found").

Backend (`SessionListViewModel`)

- Fetches all `SessionData` (session entity + counts) from `NetworkSessionRepository`.
- Manages the `selectedFilter` state (`SessionFilterType.CAPTIVE` or `SessionFilterType.NORMAL`).
- Provides a `filteredSessionDataList` based on the selected filter and whether a session `isCaptivePortal()` (determined by having requests, web content, or screenshots).
- Handles session deletion (`deleteSession`), which also includes deleting associated files from device storage (PCAP file from `NetworkSessionEntity.pcapFilePath`, screenshots from `ScreenshotEntity.path`, and HTML/JS files from `WebpageContentEntity.htmlContentPath/jsContentPath`).

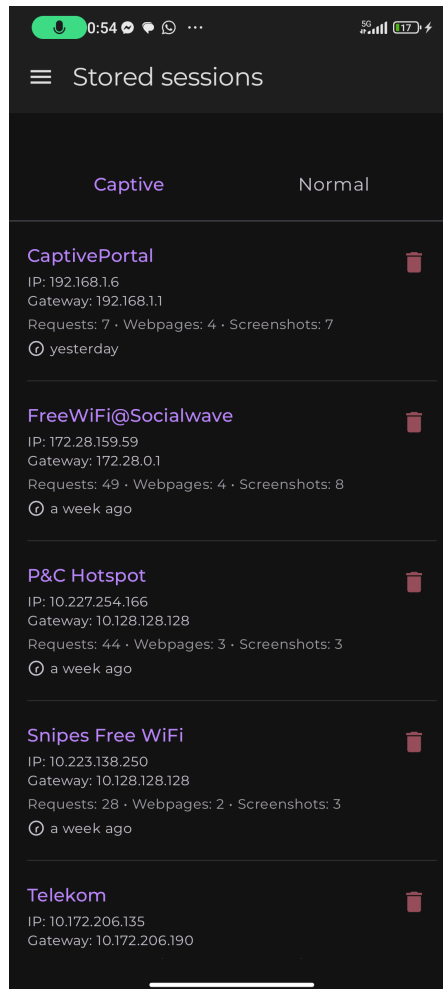


Figure 9: List of collected data for different networks.

5.3.7 Session Details (SessionScreen)

Frontend

- Displays detailed data for a selected captive portal session.
- Uses a `LazyColumn` with a `stickyHeader` for the tabs.
- The general session details (SSID, BSSID, IP, Gateway, Portal URL, and new **PCAP file path** if available) are displayed at the top (`SessionGeneralDetails`) and can scroll off. This section is expandable/collapsible.
- The `stickyHeader` contains a `TabRow` with three tabs: "Requests (*count*)", "Content (*count*)", and "Images (*count*)".

- **Requests Tab:** Shows a list of HTTP requests (`RequestListItem`), separated by `ListSectionHeader` into "Before Authentication" and "After Authentication" sections (based on `CustomWebViewRequestEntity.hasFullInternetAccess`). A "Filters" button (`FiltersHeader`) in the sticky header (when Requests tab is active) allows toggling a `FilterBottomSheet`.
- **Content Tab:** Shows a list of saved HTML/JS content (`ContentItem`).
- **Images Tab:** Displays screenshots (`ScreenshotDisplayCard`) in a `FlowRow`, separated by headers into "Privacy-Related Screenshots" and "Other Screenshots" based on the `isPrivacyOrTosRelated` flag.
- Action buttons (`SessionActionButtons`) at the bottom, horizontally scrollable if needed: "Upload Session for Analysis" (if never uploaded/re-uploading enabled) or status text (e.g., "Already Uploaded", "Thanks for Uploading"), and "Automatic Analysis".
- The `FilterBottomSheet` (for the Requests tab) allows filtering requests by method (GET, POST, etc.) and by whether the request body is empty.

Backend (`SessionViewModel`)

- Loads complete `SessionData` for the `clickedSessionId` (passed from `MainViewModel`).
- Manages `SessionState` (Loading, NeverUploaded, Uploading, AlreadyUploaded, Success, ErrorLoading, ErrorUploading, ReUploading).
- Handles session upload (`uploadSession`) to Firebase, now potentially including the processed PCAP data. The `uploadHistory` `StateFlow` provides step-by-step feedback messages (`SessionUploadHistoryLog`) during upload, showing a progress indicator and log.
- Manages filter states for requests (`showFilteringBottomSheet`, `isBodyEmptyChecked`, `selectedMethods`).
- Navigation:
 - "Automatic Analysis" button navigates to `AutomaticAnalysisGraphRoute` (start of the AI analysis flow).
 - Clicking a request item navigates to `Screen.RequestDetails.route`.
 - Clicking a content item navigates to `Screen.WebPageContent.route`.

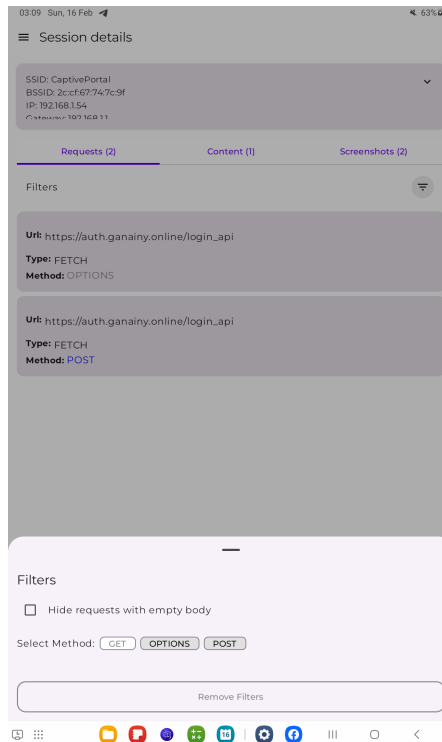


Figure 10: Session Requests can be filtered by type or body content.

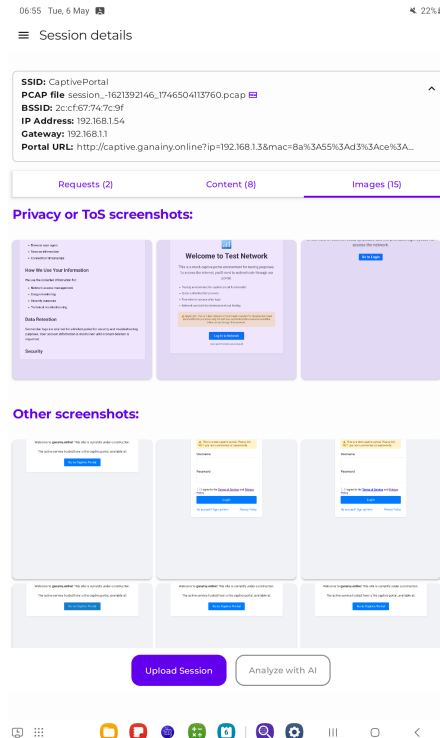


Figure 11: Detailed view of collected session data.

5.3.8 Automatic Analysis Screen (Now a Multi-Screen Flow: AutomaticAnalysisGraphRoute)

The automatic analysis feature is now a sequence of screens within the AutomaticAnalysisGraphRoute, all managed by the graph-scoped AutomaticAnalysisViewModel.

Frontend

- 1. AutomaticAnalysisInputScreen:
 - Allows the user to enter a custom text prompt for the AI in an `OutlinedTextField`.
 - Provides a collapsible section (`DataSelectionCategory`), toggled by a header, for users to select which data components from the current session to include in the AI analysis:
 - * Network Requests (with count)
 - * Screenshots (filtered for `isPrivacyOrTosRelated`, with count)
 - * Webpage Content (with count)

- Each category allows selecting/deselecting all items or individual items via checkboxes.
- A "Proceed to PCAP Step" button (`RoundCornerButton`), enabled when data is loaded and at least one item is selected. Loading/error states for initial session data load are handled.
- **2. PcapInclusionScreen:**
 - Checks if a PCAP file is associated with the session (`uiState.isPcapIncludable`).
 - If available, shows a `PcapSelectedItem` allowing the user to toggle inclusion of the PCAP file using a checkbox.
 - Displays the PCAP processing state (Idle, Processing, Success, Error). If an error occurred during a previous conversion attempt, a retry button is shown.
 - If PCAP is not available for the session, a message "PCAP not available" is displayed.
 - "Analyze with Pro" (`RoundCornerButton`) and "Analyze with Fast" (`GhostButton`) buttons.
 - A clickable text "You can check the prompt that will be sent to the model by clicking here" navigates to `AutomaticAnalysisPromptPreviewRoute`.
 - An animated visibility section shows a loading indicator if AI analysis or PCAP conversion is in progress.
- **3. AutomaticAnalysisPromptPreviewScreen (AutomaticAnalysisPromptPreviewRoute):**
 - Displays status cards (`DataItemStatusCard` for requests, screenshots, web content; `PcapStatusCard` for PCAP) summarizing what data is currently selected/included for analysis and its status (e.g., PCAP converting, truncated).
 - Shows the fully constructed prompt (`uiState.promptPreview`) that will be sent to the AI model, including the base instructions, user's custom prompt, and the selected/processed data, rendered using `MarkdownText`.
 - A scroll-to-top `FloatingActionButton` appears if content is long.
 - A loading indicator is shown if the prompt preview itself is being generated.
- **4. AutomaticAnalysisOutputScreen:**
 - Displays a collapsible header (`AnalysisInfoHeader`) showing the prompt used and a summary of data included in the analysis.

- Shows the AI-generated analysis report (`uiState.outputText`) using `MarkdownText`, prefixed by a Gemini icon and "AI Analysis" title.
- Displays loading indicators (`Analyzing data...` or a subtle streaming indicator) or error messages if AI interaction fails. Supports streaming display of results.
- Includes a collapsible "Hints" section (`CollapsibleAnalysisHints`) at the bottom, which can be dismissed.

Backend (`AutomaticAnalysisViewModel` - Scoped to `AutomaticAnalysisGraphRoute`)

- **Initialization & Data Loading:** Loads `SessionData` for the `clickedSessionId`. Manages `AutomaticAnalysisUiState`.
- **Prompt and Data Selection Management:**
 - Manages the user's `inputText` (prompt).
 - Manages selected IDs for requests, screenshots, and webpage content, and the `isPcapSelected` state.
- **PCAP Processing:** Handles PCAP file conversion to a summarized text/JSON format suitable for the AI prompt. Manages `pcapProcessingState` (Idle, Processing, Polling, Success, Error) and `isPcapConverting` flags.
- **Prompt Generation for Preview:** The `generatePromptForPreview()` method constructs the full prompt based on current selections and data, updating `uiState.promptPreview`.
- **AI Analysis:**
 - The `analyzeWithAi(modelName: String)` method:
 - * Constructs the final prompt including selected data and processed PCAP summary.
 - * Interacts with the Gemini AI SDK using the specified model (`LATEST_GEMINI_PRO_MODEL` or `LATEST_GEMINI_FLASH_MODEL`).
 - * Updates `outputText` with the AI's response (handles streaming if supported).
 - * Manages loading (`isLoading`) and error (`error`) states for the AI interaction.
- **Navigation:** Handles navigation between the screens of the automatic analysis flow (`AutomaticAnalysisInputRoute` → `PcapInclusionRoute` → `AutomaticAnalysisPromptPreviewRoute` (optional) → `AutomaticAnalysisOutputRoute`).

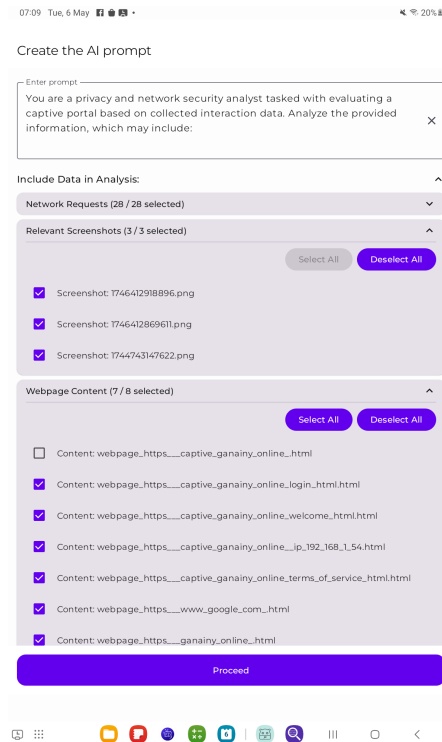


Figure 12: Fine grained control of the prompt and files sent to the AI Model (Input Screen example).

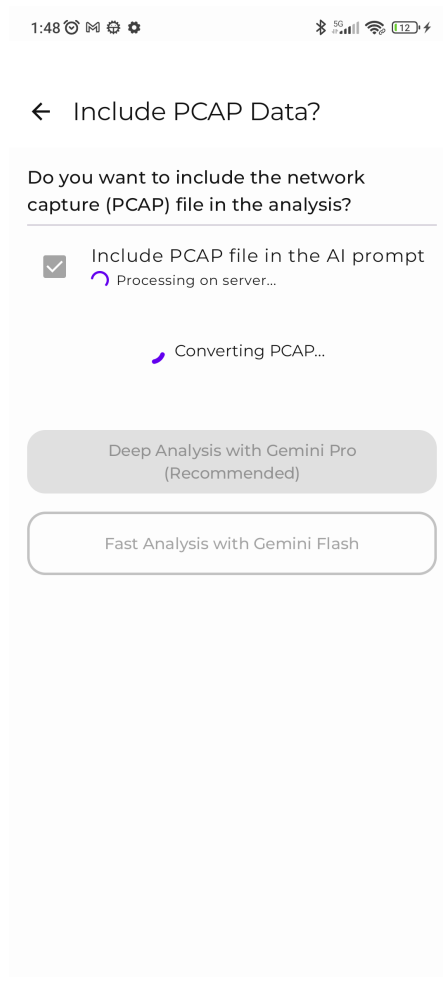


Figure 13: A Json version of the .pcap file can be included in the AI analysis for better results

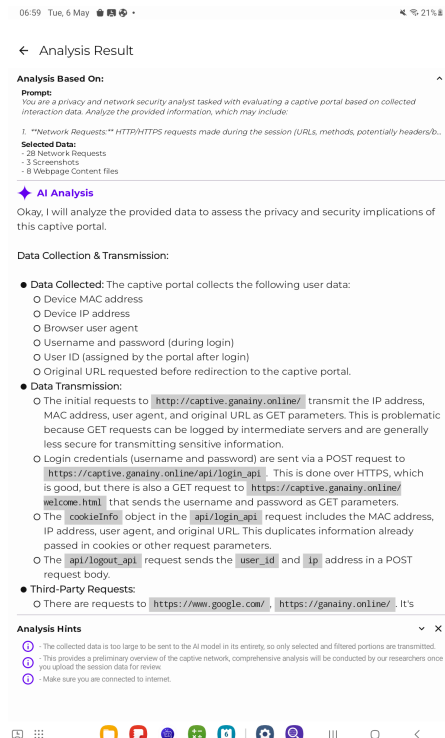


Figure 14: AI insights on the captive portal (Output Screen example).

5.3.9 Settings Screen (SettingsScreen)

Frontend

The app currently supports two languages: English and German. Additionally, the app offers both dark and light modes, or it can follow the system's theme for user convenience (see Figure 15).

Backend

The `SettingsScreen` logic is relatively simple and does not require its own `View-Model`. Adding support for more languages is simple—just supply an XML file containing the desired translations (e.g., `strings.es.xml` for Spanish). The screen receives two callback functions and two arguments from the `MainActivity` (passed via `MainViewModel`):

- `onLocalChanged: (Locale) -> Unit`: Notifies the `MainViewModel` when the user changes the app language, which then saves it to `DataStore` and triggers activity recreation to apply the locale.

- **onThemeChange:** (ThemeMode) -> Unit: Similar to `onLocalChanged`, it notifies the `MainViewModel` when the user changes the app theme mode, which is saved to `DataStore`.
- **themeMode:** ThemeMode: Represents the currently selected theme mode set by the user.
- **currentLanguage:** String: Stores the currently selected language set by the user (as a language tag).

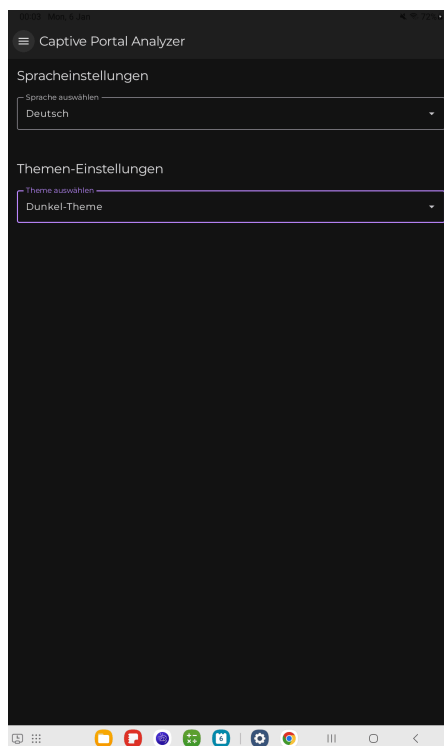


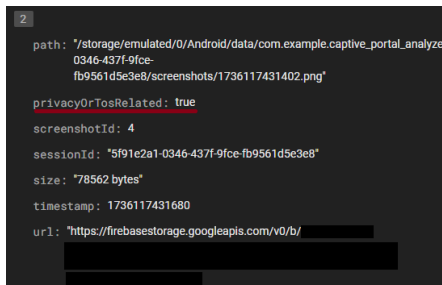
Figure 15: Support for languages and dark mode.

5.4 Data Upload and Server View

Users can upload session data to a remote server (e.g., Firebase [5, 6]). The server stores HTTP request bodies and identifies ToS/privacy-related screenshots for easier analysis (See Figure 16).

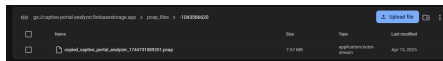
- HTTP request details (including bodies).
- HTML and JavaScript files from visited pages.

- Screenshots, with an indicator for those marked as ToS/privacy-related (from the `ScreenshotFlaggingScreen`).
- The path to the locally stored PCAP file (`NetworkSessionEntity.pcapFilePath`) if one was captured and associated with the session during the `AnalysisScreen` flow. The `SessionUploader` is responsible for deciding whether to upload the PCAP file itself, its processed summary, or just its metadata reference.



```
2
path: "/storage/emulated/0/Android/data/com.example.captive_portal_analyzer_0346-437f-9fce-fb9561d5e3e8/screenshots/1736117431402.png"
privacyOrTosRelated: true
screenshotId: 4
sessionId: "5f91e2a1-0346-437f-9fce-fb9561d5e3e8"
size: "78562 bytes"
timestamp: 1736117431680
url: "https://firebasestorage.googleapis.com/v0/b/..."
```

Figure 16: Screenshots marked as ToS/Privacy-related for separate analysis.



Name	Size	Type	Last modified
captive_portal_analyzer_1736117431402.pcap	7.21 MB	application/octet-stream	Apr 19, 2025

Figure 17: The .pcap file is also stored remotely for further analysis.

Captive Portal Reconstruction

The app saves HTML and JavaScript files from each captive portal page, along with screenshots, allowing researchers to recreate the captive portal's structure later if needed (See Figure 18, 19).

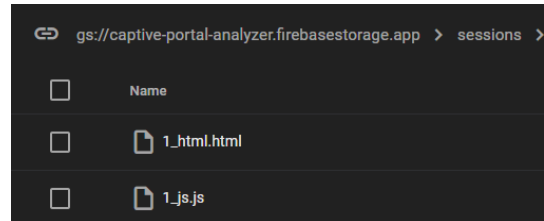


Figure 19: HTML and JavaScript files of captive portal pages uploaded to the backend.

Our analysis included a total of 14 captive portal networks, comprising 13 randomly selected networks encountered in various locations within the city center of Dortmund, along with a controlled test environment. The controlled network, named "CaptivePortal," was specifically designed with intentional privacy vulnerabilities to validate the effectiveness and reliability of our testing methodology, as detailed in Subsection 2.1. This combination of real-world networks and a controlled environment allowed us to evaluate the application's performance and its ability to detect potential privacy issues across different scenarios. The findings are categorized into networks with identified privacy violations and those with no significant security issues.

6.1 Networks with Privacy Violations

- **Issue Identified:** Plaintext credentials (username & password) in request body (See Listing 6.1.1)

```
{
  "username": "test",
  "password": "test",
  "cookieInfo": {
    "mac": "8a:55:d3:ce:56:d5",
    "ip": "192.168.1.54",
    "agent": "...",
    "original_url":
      ↪  "/?ip=192.168.1.54&mac=8a%3A55%3Ad3%3Ace%3A56%3Ad5&agent=...",
  }
}
```

Listing 1: Request body - Network CaptivePortal (for Testing only)

```
http://captive.ganainy.online/?ip=192.168.1.54&mac=8a%3A55%3Ad3%3Ace%
↪  3A56%3Ad5&agent...
```

Listing 2: URL parameters - Network CaptivePortal (for Testing only)

6.1.2 ASK4 WiFi (Hotel Network)

- **Issue Identified:** PII (Personally Identifiable Information) such as full name, email, phone number, and room number is exposed in plain text in the request body of a request made to the remote server. This data could potentially be intercepted by malicious actors. (See Listing 6.1.2)

```
{
  "personal_details[firstname]": "Test",
  "personal_details[surname]": "Test",
  "personal_details[phone]": "",
  "personal_details[email][first]": "test@test.com",
  "personal_details[email][second]": "test@test.com",
  "personal_details[residence]": "room|335722",
}
```

Listing 3: Request body - Network ASK4 WiFi

6.1.3 Vodafone Hotspot (Telecom)

- **Issues Identified:** Plaintext transfer of private data
 - Request body containing the user's password is in plaintext and other personal data such as phone number and email

```
{
  "method": "PUT",
  "body": [ {
    "parameters": {
      "email1": "amr@gmail.com",
      "email2": "amr@gmail.com",
      "phoneNumber": "017000000000"    }  },  {
    "username": "amr@gmail.com",
    "password1": "123456",
    "password2": "123456",
    "captcha": "cw43g"
  }  ]}
```

Listing 4: Request body - Network Vodafone Hotspot

6.2 Networks with No Security Issues

6.2.1 Networks Using eu.network-auth.com Service Provider

The upcoming networks are grouped together since they use the same captive portal service provider `eu.network-auth.com` and thus have almost identical results of the backend authentication process and interacting with the captive portal with different frontend customized for each different network.

While the networks shared the IP and MAC as URL parameters, this is a standard approach for authentication in captive portal networks. The IP is private and is only used for local network routing. On the other hand, the MAC address is used for device tracking within the local network and doesn't compromise the real device's MAC address, thanks to the device's MAC randomization².

- **P&C Hotspot (Retail)** (See Listing 5)

²MAC randomization is a privacy feature that periodically changes the MAC address to prevent consistent tracking of a device across different networks.

Additionally, the `cfuvid`³ cookie might refer to a cookie used by Cloudflare for rate limiting which is a legitimate usage case in the context of captive portals. (See Listing 6.2.1)

- **Free Wifi s.Oliver (Retail)** (See Listing 6.2.1)
- **Free Wifi Comma (Retail)** (See Listing 6.2.1)
- **Welcome to Lush (Retail)** (See Listing 6.2.1)

Wormland (Retail) (See Listing 7)

Snipes Free WiFi (Retail) (See Listing 6.2.1)

```
https://eu.network-auth.com/.../.../?mac=E4%3A55%3AA8%3A01%3A0C%3ADB&
↪ real_ip=10.242.168.39&client_ip=10.242.168.39&client_mac=EA:D8:9E
↪ :70:21:C2&vap=...
```

Listing 5: URL parameters - Network P&C Hotspot

```
{cookie=_cfuvid=xSthDft0B20L3bLXrPXaYDnR3ba6n3Z181JNoH147aE-173722384
↪ 6975-0.0.1.1-604800000}
```

Listing 6: Request header - Network P&C Hotspot

```
https://eu.network-auth.com/.../.../?mac=E0%3A55%3A3D%3AF2%3AE3%3A4A&r
↪ eal_ip=0.0.0.0&client_ip=10.113.164.40&client_mac=8E:6B:3F:7F:83:
↪ A6&vap=...
```

Listing 7: URL parameters - Network Wormland

```
https://eu.network-auth.com/.../.../?mac=E0%3ACB%3ABC%3A32%3A6B%3A2C&
↪ real_ip=10.16.41.5&client_ip=10.16.41.5&client_mac=8E:DE:22:C5:FF
↪ :DA&vap=..
```

Listing 8: URL parameters - Network Snipes Free WiFi

³Source: <https://captaincompliance.com/education/cfuvid-cookie/>

```
https://eu.network-auth.com/splash/N8-_Kawc.14.542/?mac=E0%3ACB%3ABC%  
↪ 3A49%3A4A%3AA2&real_ip=172.17.5.6&client_ip=10.71.147.41&client_m  
↪ ac=E2:41:48:DA:32:1D
```

Listing 9: URL parameters - Network Free Wifi s.Oliver

```
mac=F8:9E:28:7A:E6:DB  
client_mac=AA:58:62:8C:CE:4C  
real_ip=172.17.15.246  
client_ip=10.221.188.246  
clickthrough-sessions=MTczNzIyNDAxNXxEWDhFQVFMX2dB.....  
a=761f22b2c1593d0bb87e0b606f990ba4974706de  
b=5162152  
key=de3415913da977b5e1e4f314f251456a8643c24d  
referer=https://eu.network-auth.com/splash/K5BM5bwc.13.542/?mac=F8%3A  
↪ 9E%3A28%3A7A%3AE6%3ADB...  
continue_url=http%3A%2F%2Fclien.....
```

Listing 10: Request body - Network Free Wifi Comma

```
https://eu.network-auth.com/splash/e9Lgfa8.2.60/?mac=E4%3A55%3AA8%3A2  
↪ 6%3AC0%3A78&real_ip=10.94.113.247&client_ip=10.94.113.247&client_  
↪ mac=2A:8D:AA:FF:48:2C&vap=2&a=.....&auth_version=5&key=.....&a  
↪ cl_ver=P15167294V2&continue_url=http%3A%2F%2Fclients3.google.com%  
↪ 2Fgenerate_204
```

Listing 11: URL parameters - Network Welcome to Lush

6.2.2 Networks with Different Service Providers

The upcoming four networks use different captive portal service providers but showed very similar behaviors as the networks in 6.2.1:

- **Volksbank (Public Network)** uses the provider `portal.myspotmarketing.com` which also shares `mac` and `ip` in the URL parameters (See Listing 6.2.2) (See Listing 6.2.2)

- **H&M Free WiFi (Retail)** uses the provider `euw1.cloudguest.central.arubanetworks.com` which also shares `mac` and `ip` in the URL parameters (See Listing 16 & 17)
 - **device_brand & device_model & device_id:** The app seems to try to collect information about the user's device (although it failed to get the brand and model) more than needed in the context of captive portal authentication (See Listing 16).
 - **nas-id⁴ identifier:** This seems to be an identifier related to policy control in RADIUS that helps to identify authentication requests, which seems like a legitimate identifier for the context of captive portals (See Listing 17).
- **Müller-FreeWifi Network (Retail)** uses the provider `portal.myspotmarketing.com` which also shares `user-mac` and `user-ip` in the URL parameters (See Listing 6.2.2)
 - While `FX_username` (See Listing 6.2.2) might refer to the base64-encoded username used for authentication, which is insufficient encryption, the exposure of the username itself is not a significant security issue. This is assuming that the same method is not applied to the password, which was not found in the request body.
- **Zara-Emp (Retail)** uses the provider `wifi.inditex.com`
 - Website seems to collect a lot of session IDs (See 6.2.2), but on further research, it turns out the cookies are mostly related to bot detection and anti-fraud, which is legitimate for the captive portal context:
 - * `_abck`: The ABCK Akamai security cookie was created by Akamai to prevent fraud and abuse and bot detection ⁵.
 - * `ak_bmsc`: Used for distinguishing between humans and bots ⁶.
 - * `bm_sv`: Helps differentiate between web requests generated by humans and those generated by bots ⁷.
 - * `bm_mi`: An Akamai cookie for performance tracking ⁸.

⁴Source: <https://www.securew2.com/blog/what-is-nas-id>

⁵Source: https://captaincompliance.com/education/_abck/

⁶Source: https://www.cookie.is/ak_bmsc

⁷Source: <https://techdocs.akamai.com/identity-cloud/docs/hosted-login-cookies-and-local-storage-1>

⁸Source: <https://www.nokia.com/cookies/cookie-list/>

* **bm_sz**: Seems to serve the same purpose ⁹.

- **Telekom (Public Network)** The Telekom captive portal network did not share any data in the URL parameters. Additionally, the captured request body only contained a random session token for the user and some non-personally identifiable device details, such as the user-agent, with no other private data included. (See Listing 6.2.2)

```
{  
  "mac": "2E:91:3B:C9:3B:48",  
  "ip": "10.220.11.237",  
}
```

Listing 12: Request body - Network Volksbank

```
http://portal.myspotmarketing.com/api/v3/.../loginbranding?myspot_mac  
↪ _address=B8%3A69%3AF4%3A99%3ACC%3A4F&mac_address=2E%3A91%3A3B%3AC  
↪ 9%3A3B%3A48&...&....&...
```

Listing 13: URL parameters - Network Volksbank

```
{  
  "AP-MAC": "a4817a535d00",  
  "redirect-url": "http://clients3.google.com/generate_204",  
  "ssid": "-Müller-FreeWifi-",  
  "user-ip": "100.64.66.79",  
  "user-mac": "a6428adab4ee",  
  "FX_username": "jyty5LUfQe6K0sdEu%2BEqHD%2FU2hs%3D",  
  "FX_password": "easy",  
  "FX_lang": "en",  
  "username": "jyty5LUfQe6K0sdEu%2BEqHD%2FU2hs%3D",  
  "password": "easy"  
}
```

Listing 14: Request body - Network Müller-FreeWifi

⁹Source: <https://stackoverflow.com/questions/57121107/what-is-the-purpose-of-abck-and-bm-sz>

```
https://start.cloudwifi.de/?AP%2DMAC=a4817a535d00&redirect%2Durl=...&
↳ ssid=..&user%2Dip=100%2E64%2E66%2E79&user%2Dmac=a6428adab4ee&logi
↳ n%2Durl=..
```

Listing 15: URL parameters - Network Müller-FreeWifi

```
{"device_brand":"unknown","device_model":"unknown","os":"android","os
↳ _version":"13","device_id":"d2899574343f8e850d9ef59ed5f5e768","de
↳ vice_type":"smartphone"}}}
```

Listing 16: Request body - Network H&M Free WiFi

```
https://euw1.cloudguest.central.arubanetworks.com/portal/scope.cust-0
↳ 99f0b04f86d486ab23619343699852b/DE_HM_V1/capture?cmd=login&mac=2a
↳ %3A84%3Ad5%3A35%3A40%3Aaa&ssid=H%26M%20Free%20WiFi&ip=192.168.20
↳ 2.190&apname=de0709iap002&apmac=34%3A8a%3A12%3Ace%3A08%3A40&vcnam
↳ e=de0709wvc001&switchip=securelogin.hpe.com&url=http%3A%2F%2Fclie
↳ nts3.google.com%2Fgenerate_204&nas-id=1677c1ba-fbfd-45cb-ac76-fe9
↳ bf20c0874
```

Listing 17: URL parameters - Network H&M Free WiFi

```
{
↳ _abck=C405AE60434B.....&ak_bmsc=DF033FCA48533D3C.....&GSI
↳ D=gahsocvdt72r7q.....&bm_mi=3F01D3D73A5.....&bm_sv=97
↳ ADD6742FA82C59EFDCC97532.....&bm_sz=81C1B2E2A1564F04867361B
↳ 5E3BD0DE1.....
}
```

Listing 18: URL parameters - Network Zara-Emp

```
{
  "cookie": "accessToken=1737225187,K016467,9IpnWVajvmT9DYVLAygPzQ",
  "user-agent": "Mozilla...",
}
```

Listing 19: Request body - Network Telekom

6.3 Summary of Findings

Table 1 presents an overview of the analyzed networks, highlighting their security properties and potential privacy concerns. A significant portion of the networks did not exhibit major security vulnerabilities, which could be attributed to the widespread use of a common captive portal service provider `eu.network-auth.com`. This provider appears to be particularly popular among retail stores, ensuring a consistent level of security across multiple networks. However, a few exceptions were identified where privacy violations were observed, indicating that some networks may not fully adhere to best security practices.

Table 1: Summary of Analyzed Networks and Their Security Properties

Network Name	Type	Identified Issues	Service provider
ASK4 WiFi	Hotel	Plaintext PII in Request body	gw.customer.ask4.lan
Vodafone Hotspot	Telecom	Plaintext PII in Request body including passwords	hotspot.vodafone.de
P&C Hotspot	Retail	None	eu.network-auth.com
Wormland	Retail	None	eu.network-auth.com
Snipes Free WiFi	Retail	None	eu.network-auth.com
Welcome to Lush	Retail	None	eu.network-auth.com
Free Wifi s.O	Public	None	eu.network-auth.com
Free Wifi comma	Retail	None	eu.network-auth.com
Müller-FreeWifi	Retail	None	myspotmarketing.com
Volksbank	Banking	None	myspotmarketing.com
Telekom	Telecom	None	hotspot.t-mobile.net
H&M Free WiFi	Retail	None	arubanetworks.com
Zara-Emp	Retail	None	wifi.inditex.com

7 Limitations and Workarounds

7.1 Android Request Inspector WebView

The Android Request Inspector WebView is an essential tool for intercepting PUT and POST request bodies containing sensitive user information before transmission to remote backends. However, it faces certain limitation:

- The tool operates by modifying the JavaScript code of captive portal files to enable additional logging capabilities
- This source code modification may potentially cause rendering issues on some websites

7.1.1 Workaround

During extensive testing across 14 different captive portals, no rendering issues were encountered. However, as a precautionary measure, an alternative version of the Inspector WebView has been implemented with the following characteristics:

- Operates without JavaScript injection
- Successfully captures URLs, URL parameters, and headers
- Cannot capture POST request body content
- Serves as a reliable fallback when the primary method encounters issues

7.2 Screenshot Functionality

During the testing phase across 14 different captive portals, a screenshot-related limitation was identified which is a screenshots may be captured before the page completes loading so that the screenshot show loading indicator instead of actual page content (See Figure 20).

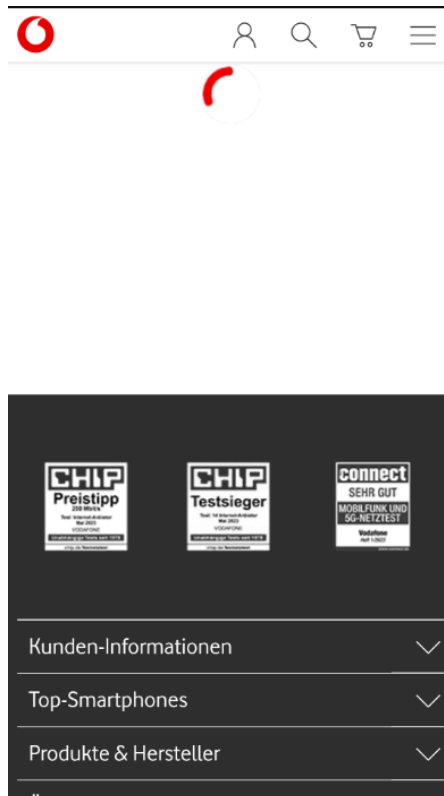


Figure 20: Example of a screenshot captured during page loading state

7.2.1 Workaround

To address the screenshot limitations, the following solution has been implemented:

- HTML content for each captive portal page is saved separately
- Content files are uploaded to the remote server
- These files can be executed to reproduce the intended page content
- This provides a reliable alternative when screenshots are incomplete

8 Conclusion

The comprehensive analysis of 14 captive portal networks provides valuable insights into the security and privacy practices of public WiFi implementations. Contrary to initial concerns, the majority of analyzed networks did not exhibit significant security vulnerabilities. However, a few exceptions were identified, particularly in cases where sensitive user data was transmitted in plaintext, raising privacy concerns.

While most captive portals followed standard security practices, notable risks were observed in specific networks. Two major issues stood out: first, the presence of plaintext personally identifiable information (PII) in request bodies, which increases the risk of data interception by malicious actors; and second, inconsistencies in data handling across different service providers, where some networks collected more user information than necessary for authentication.

Furthermore, the widespread use of a small number of captive portal service providers—especially in retail environments—suggests that privacy policies and security implementations may be dictated by these providers rather than individual businesses. This centralization presents both opportunities for standardized security improvements and risks if vulnerabilities exist in widely deployed solutions.

The app successfully detects and interacts with various types of captive portals directly within the app, enabling complete data capture both locally and remotely for further analysis by professionals. This functionality ensures that security assessments can be conducted efficiently while remaining accessible to regular users.

The controlled test environment played a crucial role in validating our methodology, as it provided a consistent and modifiable captive network regardless of the development location. This allowed us to introduce specific security and privacy scenarios for rigorous testing, ensuring that the mobile app could effectively detect real-world vulnerabilities.

References

- [1] Android. *Android Request Inspector WebView*. <https://github.com/acsbendi/Android-Request-Inspector-WebView>.
- [2] Composables. *Android Distribution Chart*. URL: <https://composables.com/android-distribution-chart>.
- [3] Amr Elganainy. *captive-portal-analyzer-kotlin*. URL: <https://github.com/ganainy/captive-portal-analyzer-kotlin>.
- [4] Amr Elganainy. *Raspberrypi-captive-portal*. URL: <https://github.com/ganainy/raspberrypi-captive-portal/tree/remote-captive>.

- [5] Google. *Firebase Firestore*. <https://firebase.google.com/products/firestore>.
- [6] Google. *Firebase Storage*. <https://firebase.google.com/products/storage>.
- [7] Google. *Gemini AI SDK*. <https://developer.android.com/ai/google-ai-client-sdk>.