目次

Airoha RACE: TWS の脆弱性検証 1
1 おことわり 1
2 はじめに 1
3 Bluetooth 仕様 超速習版 2
4 脆弱性解説 3
5 実験 7
6 最後に 20
参考文献 21
そうはならんやろ - eMudhra 社インシデントの一考察 23
1 はじめに 23
2 何が起こったか 23
3 eMudhra 社の説明 27
4 何故起こったか 29
5 どうすれば良かったか 30
6 おわりに 31
参考文献 32

Airoha RACE: TWS の脆弱性検証

1　おことわり

本稿では無線通信デバイスにおける脆弱性を取り上げ、その脆弱性を用いてデバイスへのアクセスを試みていますが、すべて自身が所有・管理するデバイスに対して行っています。本稿の内容を実際に試す際には、必ず自己の管理下において実施するようお願いします。

2　はじめに

TWS[*1]は当初バッテリーのもちが悪い、落としそう、音質が微妙、耳からうどんが生えてるなどと言われ、ごく一部のガジェット好きくらいにしか刺さっていませんでしたが、それも遙か昔の話で、今となっては街や電車の中で TWS を使っている人を見かけない日はまずありません。TWS は通常スマートフォンと接続して、電話をしたり音楽を聴いたりします。その際の通信には Bluetooth という規格が使われます。

今回取り上げるのは、発見者が Airoha RACE と呼称している脆弱性です。これは TWS の Bluetooth 通信を担う SoC[*2]の SDK[*3]において、その仕様上の瑕疵によって、認証なしに TWS に接続して、そのデバイス情報を取得したり、RAM の内容を読み書きしたりできる脆弱性です。

本稿では、まずこの脆弱性に関わる Bluetooth の仕様をかなり端折って説明します。続いてこの脆弱性について説明します。最後に、実際に TWS に対してこの脆弱性を使って実際に RACE プロトコルで通信してみます。

---

*1　True Wireless Stereo の略。日本語では「完全ワイヤレスイヤホン」などと呼称します。本稿では TWS という呼称で統一します。

*2　System on Chip の略。ここでは Bluetooth 通信に関する機能を一手に担うチップのこと。

*3　Software Development Kit の略。ここでは SoC を扱うためのライブラリや文書一式のこと。

5.1 実験環境

脆弱性のあるデバイスとしてリストに挙がっていたもののうち、Sony の WF-C500 と、JBL の Endurance Race 2 を購入しました。WF-C500 は 2025 年 7 月 11 日に Amazon.co.jp で新品を購入しました。JBL Endurance Race 2 は秋葉原の e☆イヤホンにて中古品を購入しました。いずれも購入後にファームウェアアップデートは行っていません。

攻撃は PC から行いました。Ubuntu 26.04 LTS をインストールした Linuxマシンに、サンワサプライの Bluetooth アダプタ MM-BTUD46 を接続しています。

余談ですがここでちょっと紆余曲折があって、このマシンのマザーボードはオンボードで Bluetooth に対応しているのですが、これはうまく動作しませんでした。そこでバッファローの BSBT54D200BK というアダプタを購入して使ったところ、今度は搭載チップが新しすぎて正常に動作するドライバがありませんでした。最終的に前述の MM-BTUD46 を「信頼性の高い Qualcomm 社製チップ」という宣伝文句を信じて購入したところようやく動作しました。そもそも今時 Bluetooth アダプタというもの自体需要があまりなくてラインナップが限られる中で相性問題を踏み抜くとは思っておらず、割とこの時点で疲れていた気がします。

実装は Go で行いました。Bluetooth ライブラリとして `tinygo.org/x/bluetooth`[*17] を採用しました。実際には後述の問題があったため、フォーク版の `github.com/tinkerizer/go-bluetooth`[*18]を使用しています。

5.2 事前実験

[4]には、iOS アプリの nRF Connect for Mobile[*19]を使用してデバイスが脆弱かどうかを確認する手順が記載されています。この手順に従って WF-C500 に対して RACE プロトコルの `GET_BUILD_VERSION` を実行したところ、以下のような結果になりました。

1. デバイスへの接続は成功
2. 受信用の characteristic を subscribe する時点でペアリング要求が発生

*17 `https://github.com/tinygo-org/bluetooth`

*18 `https://github.com/tinkerizer/go-bluetooth`

*19 `https://apps.apple.com/jp/app/nrf-connect-for-mobile/id1054362403`

そうはならんやろ - eMudhra 社インシデントの一考察

1　はじめに

2026 年 10 月 20 日、インドの認証局 eMudhra 社が発行する電子証明書 8 個が強度の弱い秘密鍵を使っており、第三者が秘密鍵を特定できる状態だったことが研究者による報告で明らかになった[1]。これにより、世界中の攻撃者がその認証局のものとして有効な偽の証明書を発行できたり、それを利用して TLS 通信に対する中間者攻撃を行えたりする状態だったことが判明した。問題の証明書は主要な OS・ブラウザーによって信頼されていたため、各ベンダーは対応に追われた。

問題となった強度の弱い秘密鍵では、本来は非常に大きい範囲を取り特定困難な乱数であるべき鍵[*1]の値が、非常に狭い範囲$[1, 256)$の中にあり、特定が容易な状態となっていた。 これを研究者が発見し報告したことで事態が明らかになった。

その頃、世の中の話題は PHP の重大な脆弱性（PHP2Shell、CVE-2026-109618 および CVE-2026-109619）[2]で持ち切りだった。 しかし、PHP の脆弱性はあくまでインシデント（脆弱性）として「ありがち」なものだったのに対し、このインシデントは認証局として通常の運用を行っていれば起こり難いものであり、筆者の主観で言えば「そうはならんやろ」としか言いようがない異常なインシデントのように思われた。そこで、この記事では入手可能な情報の範囲でこのインシデントについてまとめて考察する。

2　何が起こったか

今回のインシデント対応は Bugzilla の Bug 2083449[1]で行われた。

---

*1　概ね$[1, 2^{384})$の範囲にある整数値。

3 については Full Incident Report での説明が不足しているので一旦論じない。

4 については、これが良くなかったように思われる。 Full Incident Report を読む限り、これは「キーセレモニーの後に、先頭 1 バイトに切り詰められる前の鍵対で署名された CRL[*8]を切り詰められた後の鍵対で検証したら署名が一致せず、異常事象として報告されていた」ということらしい。 このミスマッチは、最終的に切り詰められた後の鍵対で再署名を行うことで「解決」されている。 定められた手順で運用されている認証局のマテリアルで署名のミスマッチが発生したのだから、根本原因の調査が行われることが望ましかったと筆者は考える。

これらに加えて、そもそも鍵の export/import や wrap/unwrap といった**暗号関係の操作を独自実装のプログラムで行うのは本質的に危険**なので、それぞれの直前・直後で厳密な鍵の同一性チェックが行われるべきだったろうと考える[*9]。

6 おわりに

ここまで、eMudhra 社の認証局インシデントについて、現在入手可能な情報で振り返り、筆者なりの考察を加えた。 このようなインシデントは今後起こらないことが望ましいが、残念ながらこの世界は「そうはならんやろ」なコードと運用で溢れており、きっとそうはならない。 また、AI の隆盛によって理解と実践の距離が離れつつあるように思われ、今までとは別ベクトルでのびっくりインシデントが起こってくるような気がしてならない。これからも気を強く持ってセキュリティをやっていきたい。

著者紹介

op

セキュリティをやっている人。リバエンが主だったが何だかんだいろいろやる。最近のマイブームはまぜそば。

*8 Certificate Revocation List のこと。

*9 鍵の比較を行っていなかったことは要因 2 でも触れられている。

urandom vol.16

発行日	2026 年 11 月 23 日（技術書典 21）
著者	mayth, op
連絡先	https://urandom.team/
印刷所	株式会社栄光