Moje hodinky, moje data

Chytré hodinky pro mobilní Linux

Petr Hodina

OpenAlt 2026

O mně



Petr Hodina

Kontraktor

Embedded firmware

Přispěvatel do Mobile NixOS

Agenda

1. Ekosystém
2. Úrovně otevřenosti
3. Cíl
4. Architektura
5. Bluetooth LE
6. Pairing a handshake
7. Data
8. Aplikace
9. Závěr

Mainline Linux na telefonech je dnes použitelný

- Mainline kernel a bootloader na telefonech už dnes funguje na každodenní použití
- Hardware je z velké části vyřešený: displej, dotyk, Wi-Fi, modem, kamery ...
- Ale telefon samotný nestačí

Lidé nekupují telefon. Kupují ekosystém.

Ekosystémy: Google, Apple, Samsung

- Telefon, hodinky, sluchátka, domácnost, platby, fotky: vše je navzájem **propojené**
- Uvnitř jedné platformy to funguje opravdu dobře: spárujete, a jede to
- Každý má **walled garden**: Apple, Google, Samsung, Xiaomi, Huawei ...
- Vaše data tečou přes servery výrobce

Používáte chytré hodinky? Jaké a s čím je párujete?

Otevřenost má několik úrovní

Úroveň	Příklad	
Otevřený hardware i software	ZSWatch	`github.com/ZSWatch/ZSWatch`
Otevřený software	Bangle.js	`banglejs.com`
Uzavřený systém	většina hodinek na trhu	

Většina hodinek používá **uzavřený systém**.

Co když vám hodinky vyhovují?

- Někomu může **UI na hodinkách** naprosto vyhovovat, i když je uzavřené
- Nepotřebujeme nový systém v hodinkách, ale démona, který se s nimi bude bavit
- Otevřená implementace místo uzavřené aplikace výrobce
- Přesně to dělá **Gadgetbridge**

Co byste od takového démona čekali?

Android má Gadgetbridge

- Open-source aplikace, která mluví s **desítkami** různých hodinek a náramků
- Bez cloudu výrobce, data zůstávají u vás
- Unifikuje přístup: jeden uživatelský zážitek nad mnoha různými protokoly
- Každé zařízení má vlastní ovladač (*device coordinator*), společné jsou jen datový model a UI

Na Linuxu nic takového není

- Žádná společná vrstva pro chytré hodinky a náramky
- **BlueZ** není ideální místo pro integraci
- Proč: BlueZ je **obecný systémový démon** (spojení, párování, GATT přes D-Bus). Nezná protokoly výrobců ani datový model zdraví, takže vlastní handshake, framing a ovladače hodinek musí řešit vrstva nad ním

Co s tím?

Cíl: hodinky přímo s Linuxem

1. Sparovat hodinky přímo s Linuxem přes **Bluetooth LE**
2. Projít nezdokumentovaným **šifrovaným autentizačním handshakem**
3. Číst reálná data: **tep, kroky, spánek, notifikace**

Bez Android telefonu, bez black box aplikace, bez cloudu mezi tím.

Jak zjistit, co aplikace výrobce dělá

Android

- Vzít telefon s aplikací výrobce
- Povolit **dump HCI** Bluetooth komunikace (`btsnoop_hci.log`) a otevřít ve Wiresharku
- **Root** zařízení a použít **Frida** pro dynamickou analýzu aplikace

iOS

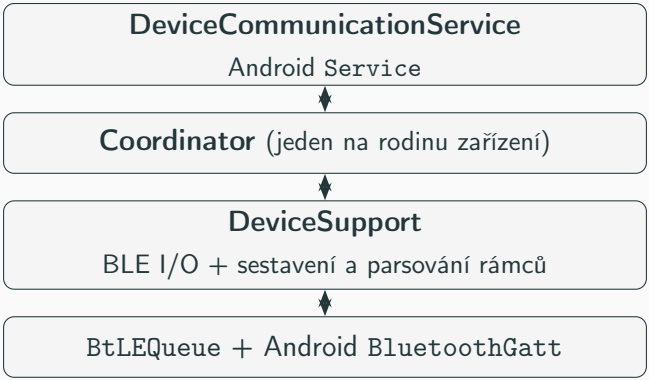
- Podobný postup, ale **složitější**: nutný jailbreak
- Existují i nástroje jako **Corellium Falcon**

Moje hodinky na hraní

Hodinky	Protokol
Garmin Vivoactive	Multi-Link v2 + GFDI (COBS)
WearFitPro (QD9)	framing `AB/EA/C4`, Nordic UART
Suunto (Movesense)	NSP + SBEM logbook
Samsung Galaxy Watch	SCSP, JSON přes GATT
TicWatch (Wear OS)	Mobvoi, magic hlavička

Které hodinky používáte vy?

Gadgetbridge: jak je postavený



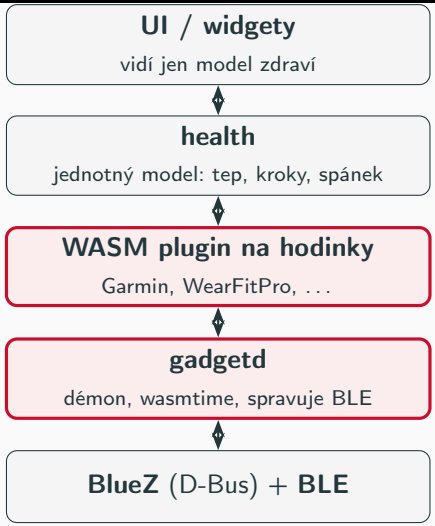
Aktuální master (listopad 2026): cca **3 300 souborů Java**, přes **580 coordinatorů**, 159 adresářů zařízení

Proč Gadgetbridge není vhodný pro Linux

- Je to **Android aplikace**: více než 1 700 souborů importuje `android.*`
- BLE jde přes `BluetoothGatt`, běh přes `Service`, notifikace přes `NotificationListenerService`
- Protokol a I/O jsou svázané jen částečně: novější ovladače (Garmin, Xiaomi) mají zprávy oddělené, ale starší (Huami, WearFit) míchají vše v jedné třídě na tisíce řádků; orchestrace a spojení zůstávají v Androidu
- Vše běží v jednom procesu bez izolace: chyba v parseru nebo škodlivé hodinky mají přístup ke všem datům aplikace; kontroly vstupů jsou jen u jednotlivých ovladačů
- Databáze a UI jsou svázané s Androidem
- Na Linuxu by to znamenalo Android runtime nebo celé přepsání

Použitelné je to, co v něm je: reverzně zjištěné protokoly jako reference.

Odpověď: vrstva nad BlueZem



- Každý plugin běží izolovaně: chyba nebo škodlivé hodinky se k ostatním datům nedostanou
- **WASM** je nenáročný, jednoduchý VM
- **Prohlížeč** umí spustit WASM a přes **Web Bluetooth** dá po povolení uživatelem přístup k Bluetooth API: plugin tak poběží i bez démona

Data ven: Grafana a MQTT

Gadgetbridge

- Grafanu ani MQTT **standardně nepodporuje**
- Data jsou v SQLite na telefonu
- Okliky: export databáze → skript → InfluxDB → Grafana (gb2influxdb)
- Nebo přes Health Connect a další aplikaci (Life Dashboard Companion)

Náš přístup

- **MQTT** a **Grafana** jako first class citizen
- Model zdraví v gadgetd je dostupný přímo, bez exportů
- Hodinky se tak stanou jedním ze zdrojů dat pro domácí automatizaci a dashboardy

Proč takhle

- **gadgetd** řeší BLE přes BlueZ jednou; plugin jen deklaruje GATT služby a dostává události
- Každý výrobce je **plugin** ve WASM: izolovaný, přenositelný, nezávisle sestavitelný
- Protokolová knihovna je čistá (encode/decode bez I/O), proto se dobře testuje na zachycené komunikaci
- Společné rozhraní pro čtení zdravotních dat: UI se nemění, mění se jen zdroj dat
- Protipól Gadgetbridge: ne jedna monolitická aplikace pro Android, ale démon a pluginy pro Linux

`plugin.toml`: GATT služby, povolené schopnosti (notifikace na desktop, přístup k síti jen na uvedené domény)

Bluetooth LE v kostce

- **GAP**: reklamy (advertising), připojení, pairing a bonding
- **GATT**: služba → charakteristika → hodnota, identifikované UUID
- Charakteristika umí `read`, `write`, `write-no-response` a `notify`
- Výchozí MTU 23 B, tedy 20 B dat: delší zprávy si výrobci fragmentují sami
- Bonding šifruje spojení (SMP, klíč LTK), obsah dat nad tím ale řeší každý po svém

Příklady UUID

WearFitPro, Nordic UART:

`6e400001-b5a3-f393-e0a9-e50e24dcca9e`

Garmin Multi-Link RX / TX:

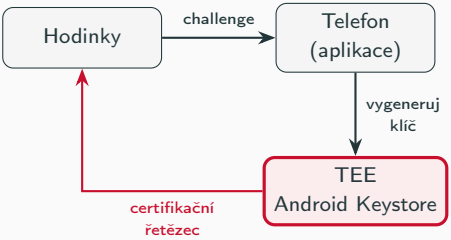
`6a4e2810-667b-11e3-949a-0800200c9a66`

`6a4e2820-667b-11e3-949a-0800200c9a66`

Legenda

GAP (Generic Access Profile): jak se zařízení *najdou a spojí*, tedy kdo vysílá reklamy a kdo se připojuje
GATT (Generic Attribute Profile): jak si po spojení *vyměňují data*, ve formě služeb a charakteristik

Attestation: Samsung a Google



- Hodinky pošlou **challenge**, telefon vygeneruje EC klíč v Keystore s `setAttestationChallenge`
- Vrátí **řetězec certifikátů**, podepsaný až kořenem Googlu: dokazuje, že klíč je v hardwaru (TEE)
- Proč je těžké ho obejít: privátní klíč z TEE nejde vytáhnout, řetězec nejde padělat bez Googlova klíče, a challenge je pokaždé nová, takže ani replay nepomůže

V praxi: běžná komunikace (zařízení, baterie) atestaci nepotřebuje, hlídá jen vybrané citlivé operace.

Pairing má tři vrstvy

1. BLE bonding (SMP)

BlueZ + pairing agent, šifrované spojení

2. Autentizace výrobce

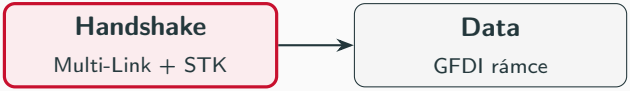
vlastní handshake, vlastní šifrování

3. Datový protokol

rámce, příkazy, zdravotní záznamy

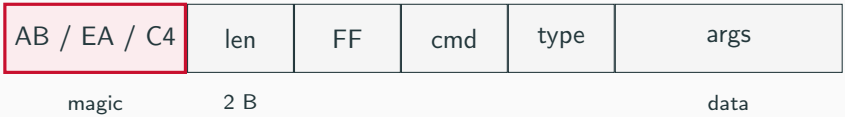
	Vrstva 2
WearFitPro	žádná, stačí bonding
Garmin	ML handshake + STK
Samsung	JSON setup; atestace jen u citlivých operací
Android Wear	mimo BLE (cloud)

Garmin: nejdřív handshake, jinak ticho



- Bez handshaku hodinky **přidělí handle** a teprve pak berou GFDI rámce jako data
- STK: obě strany si pošlou závazek, pak ho odhalí a ověří; klíč vznikne přes XXTEA

WearFitPro: nejjednodušší případ



- **Žádná autentizace** výrobce, stačí BLE bonding
- Nordic UART, rámce se po 20 B fragmentují
- Tep, kroky, stres, baterie: čitelné přímo z capture

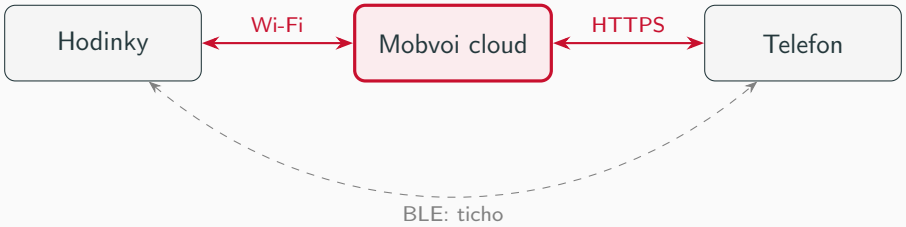
Samsung: čitelný JSON




```
{"msgId":"msg_popup_request","sender":"phone",
"data":{"category":2}}
```

- Po bondingu je komunikace **čistý JSON** přes GATT, v btsnoopu čitelná
- Hardwarová atestace (SAK) hlídá jen vybrané citlivé operace

Android Wear: TicWatch



- Část konfigurace jde přes **Wi-Fi** přímo do cloudu, Bluetooth se vůbec nepoužije
- Zdraví: REST API Mobvoi, na telefonu v lokálních databázích (Room)

Tady se vyplatí vlastní OS přímo na hodinkách, cesta přes BLE nestačí.

Co se dá vytáhnout

- **Vitální data**: tep, kroky, spánek – u některých protokolů hotové, u jiných rozpracované
- **Notifikace**: hovor, SMS, e-mail i aplikace jako Telegram nebo WhatsApp
- **Stav zařízení**: baterie, info o hodinkách
- U některých hodinek navíc seznam nainstalovaných aplikací a ciferníků

Jednotný model zdraví

- Rozdělení: jedna vrstva data z hodinek *stahuje*, druhá je jen *čte* – bez znalosti transportu
- Widgety znají jen tu čtecí vrstvu: tep, kroky, spánek za časový rozsah
- Výrobce dekóduje do společného úložiště, UI se nemění
- Skutečný příklad: WearFitPro už touto cestou dodává reálná data

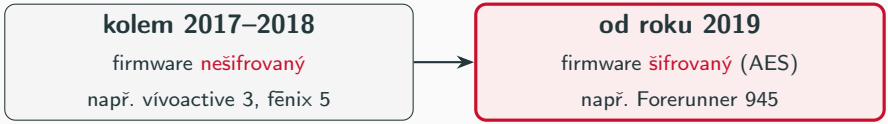
Aplikace na hodinkách: Garmin Connect IQ



- Garmin má vlastní jazyk **Monkey C** a vlastní **virtuální stroj**, na kterém běží aplikace
- Typy: watch face, widget, datové pole, aplikace, zdroj audia
- `.PRG` se kompiluje pro konkrétní zařízení; hodinky umí vypsat nainstalované aplikace (GFDI, protobuf)

VM je zajímavý cíl: v něm se našlo 13 zranitelností (Anvil Secure).

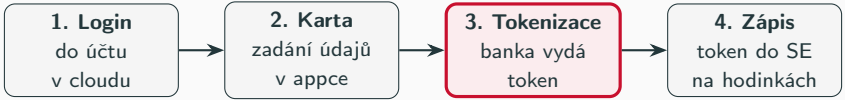
Firmware: starší nešifrovaný, novější šifrovaný



- Ze 78 prozkoumaných zařízení: **44 nešifrovaných**, **34 šifrovaných**
- Šifrované je od prvního firmwaru a nikdy se to dodatečně nezměnilo

Nešifrovaný firmware jde prohledávat přímo, šifrovaný ne.

Platba kartou z hodinek: nastavení



- **Secure Element (SE)**: samostatný čip na hodinkách, mimo hlavní MCU
- Místo čísla karty se do SE uloží jen token, vázaný na toto konkrétní zařízení
- Číslo karty samo nikdy neopustí banku a cloud výrobce

Platba kartou: zjednodušeně



- Teď už spolu mluví jen **hodinky a terminál** přes NFC, žádný telefon ani cloud
- SE pošle token a podepíše transakci, stejně jako platební karta nebo telefon
- Zbytek hodinek (MCU, aplikace) se zaplacení vůbec neúčastní

Platba kartou: nevýhody

- Stejný model jako **Google Pay**, **Apple Pay**, **Samsung Pay**: mezi vámi a bankou je vždy gatekeeper
- Nutný **účet** u výrobce platformy, bez něj se karta vůbec nenastaví
- Tokenizaci a zápis do **Secure Element** smí spustit jen jejich appka a jejich cloud
- Vyžadují **atestaci zařízení**: `Play Integrity` (Google), `Knox` (Samsung)
- Bez průchozí atestace appka nastavení karty odmítne, i na jinak funkčním telefonu
- Stejný problém mají i **alternativní Android ROM** (LineageOS, GrapheneOS): bez certifikované atestace Play Integrity appka nastavení odmítne stejně jako na Linuxu

Otevřený Linux telefon nebo hodinky: do tohoto systému nemají přístup.

Kdo vidí vaše transakce

- **Banka**: vidí každou transakci, to je nutné pro zúčtování
- **Platební síť** (Visa, Mastercard): směruje transakci, vidí částku, obchodníka, token
- **Gatekeeper** (Google, Apple, Samsung): vidí minimálně *kdy a kde* jste platili, i když ne vždy přesnou částku
- Vaše telefon/hodinky: žádná z těchto stran nad ním nemáte kontrolu, je to jejich platforma
- Se **Secure Element** čipem se smí přímo bavit jen ten, kdo dělal jeho provisioning (vložil klíče při výrobě/aktivaci); nikdo jiný do něj nezapíše ani nepřečte token

Diskuze: vadí vám, že platbu vidí i firma, která vám prodala hodinky?

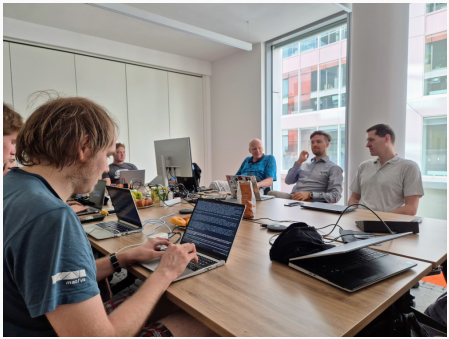
Co dál

- **Upload nového firmwaru**: na vlastní riziko, bez podpory výrobce
- **Downgrade**: obejít rollback prevention, nebo se smířit s tím, že nejde
- Co když se update nepovede: záchrana z bricknutých hodinek, nouzový režim
- Další hodinky: Suunto, Samsung, TicWatch
- Repozitář ciferníků a aplikací ve stylu F-Droid

Shrnutí

1. Mainline Linux na telefonu je použitelný, ekosystém ale chybí
2. Na Androidu sjednocuje přístup k hodinkám Gadgetbridge, na Linuxu nic takového není
3. Hodinky jdou spárovat a číst přímo, bez telefonu a cloudu; každé po svém, někdy s vlastní šifrovanou autentizací
4. Řešení: `gadgetd` nad BlueZem, WASM pluginy a jednotný model zdraví
5. S tímto modelem jde připojit cokoliv přes BT nebo BLE – stejný gadgetd, jen nový plugin

Děkuji!





Připojte se na Matrixu `#sdm845:ixit.cz`