AYI NEDJIMI Consultants

Hardware Hacking : JTAG, SWD, UART et Extraction de Firmware

Catégorie : Articles Techniques Lecture : 7 min Publié le : 15/02/2026 Auteur : Ayi NEDJIMI

Reverse engineering hardware, identification de debug ports JTAG/SWD/UART, dump de flash SPI pour audits IoT. Guide technique complet. Guide détaillé.

Cette analyse technique de Hardware Hacking : JTAG, SWD, UART et Extraction de Firmware s'appuie sur les retours d'experience d'equipes confrontees quotidiennement aux defis operationnels du domaine. Les methodologies presentees couvrent l'ensemble du cycle de vie, de la conception initiale au deploiement en production, en passant par les phases de test et de validation. Les recommandations sont directement applicables dans les environnements professionnels. Reverse engineering hardware, identification de debug ports JTAG/SWD/UART, dump de flash SPI pour audits IoT. Guide technique complet. Guide détaillé. Ce guide technique sur hardware hacking s'appuie sur des retours d'expérience terrain et des méthodologies éprouvées en environnement de production. Nous abordons notamment : table des matières, 1. introduction et 2. identification de ports debug (jtag, swd, uart). Les professionnels y trouveront des recommandations actionnables, des commandes prêtes à l'emploi et des stratégies de mise en œuvre adaptées aux environnements d'entreprise.

Table des matières

Auteur : Ayi NEDJIMI **Date :** 15 février 2026

1. Introduction

Le hardware hacking est l'art d'analyser, de comprendre et d'exploiter les composants physiques d'un système électronique. Dans le contexte de la cybersécurité, cette discipline est devenue incontournable avec l'explosion de l'Internet des Objets (IoT). En 2026, plus de 30 milliards de dispositifs connectés sont déployés dans le monde : caméras IP, routeurs, serrures connectées, contrôleurs industriels (PLCs), dispositifs médicaux, et systèmes embarqués automobiles.

Contrairement aux audits logiciels classiques, le hardware hacking nécessite un accès physique au dispositif, des connaissances en électronique, et un ensemble d'outils spécialisés. L'objectif principal est généralement l'extraction du firmware, qui contient le code applicatif, les clés de chiffrement, les credentials par défaut, et la logique métier du dispositif. Une fois le firmware extrait, l'analyse statique peut révéler des vulnérabilités exploitables à distance.

Cet article couvre l'ensemble de la méthodologie de hardware hacking : de l'identification visuelle des ports de debug (JTAG, SWD, UART) sur le PCB, jusqu'à l'analyse avancée du firmware avec Ghidra, en passant par le dump de flash SPI/NAND et les attaques par injection de fautes (glitching).

Cadre légal et éthique
Le hardware hacking ne doit être pratiqué que sur des équipements dont vous êtes propriétaire ou pour lesquels vous disposez d'une autorisation explicite d'audit. La rétro-ingénierie peut être encadrée par des lois spécifiques (DMCA, directive européenne sur les secrets d'affaires). Pour approfondir, consultez Evasion d'EDR/XDR : techniques.

---

2. Identification de ports debug (JTAG, SWD, UART)

Reconnaissance physique du PCB

La première étape consiste à ouvrir le boîtier du dispositif et examiner le PCB (Printed Circuit Board). Les fabricants laissent souvent des interfaces de debug accessibles, utilisées pendant le développement et la production :

- **Headers non peuplés** : Rangées de trous de soudure alignés (souvent 4, 5 ou 10 pins) sans composant soudé.
- **Test pads** : Points de soudure ronds ou carrés, isolés, souvent près du processeur principal.
- **Silk screen labels** : Inscriptions TX, RX, GND, TDI, TDO, TCK, TMS sur le PCB.
- **Connecteurs JST/molex** : Petits connecteurs blancs ou noirs avec 3 à 10 pins.

UART (Universal Asynchronous Receiver-Transmitter)

UART est l'interface de debug la plus courante sur les dispositifs IoT. Elle fournit généralement un accès console série, souvent avec un shell root sans authentification. UART nécessite seulement 3 fils : TX (Transmit), RX (Receive) et GND (Ground).

```
# Identification des pins UART
# 1. Identifier GND avec un multimètre (continuité avec le plan de masse)
# 2. Identifier VCC (3.3V ou 5V) - NE PAS CONNECTER
# 3. TX : pin qui montre de l'activité au boot (oscilloscope ou analyseur logique)
# 4. RX : pin restante

# Avec un analyseur logique Saleae Logic Pro
# Connecter toutes les pins suspectes, démarrer le device
# Le logiciel Saleae détecte automatiquement le protocole UART et le baud rate

# Avec JTAGulator (identification automatisée)
# Connecter jusqu'à 24 pins
# Mode UART :
uart
# Le JTAGulator teste toutes les combinaisons TX/RX
# et les baud rates courants (9600, 19200, 38400, 57600, 115200)

# Connexion via adaptateur USB-TTL (FTDI, CP2102, CH340)
# ATTENTION : vérifier la tension (3.3V vs 5V) avant connexion !
# Pin TX du device -> Pin RX de l'adaptateur
# Pin RX du device -> Pin TX de l'adaptateur
# GND du device -> GND de l'adaptateur

# Connexion avec minicom/screen
sudo minicom -D /dev/ttyUSB0 -b 115200
# ou
screen /dev/ttyUSB0 115200

# Résultat typique au boot :
# U-Boot 2023.01 (Sep 15 2025)
# DRAM: 256 MiB
# Loading kernel...
# [    0.000000] Linux version 5.15.0
# ...
# BusyBox v1.36.1 built-in shell (ash)
# / # whoami
# root
```

JTAG (Joint Test Action Group)

JTAG (IEEE 1149.1) est une interface de debug et de test standard. Elle permet le contrôle complet du processeur : lecture/écriture de la mémoire, placement de breakpoints, single-stepping du code. Les 5 signaux JTAG sont TDI (Test Data In), TDO (Test Data Out), TCK (Test Clock), TMS (Test Mode Select) et TRST (Test Reset, optionnel).

Notre avis d'expert

Le Security by Design est souvent invoqué, rarement pratiqué. Intégrer la sécurité dès la conception coûte 6 fois moins cher que de corriger en production. Nos audits d'architecture montrent que les choix techniques des premières sprints conditionnent la posture de sécurité pour des années.

Considerations pratiques avancees

```
# Identification JTAG avec JTAGulator
# Connecter les pins suspectes (jusqu'à 24)
# Commande d'identification :
idcode
# Le JTAGulator teste toutes les combinaisons possibles de TCK, TMS, TDI, TDO
# et affiche les IDCODE trouvés :
# TDI: CH2, TDO: CH5, TCK: CH1, TMS: CH3
# IDCODE: 0x4BA00477 (ARM Cortex-A/R Debug Port)

# Connexion JTAG avec OpenOCD (Open On-Chip Debugger)
# Configuration pour un processeur ARM (exemple : STM32)
cat > openocd.cfg << 'EOF'
source [find interface/ftdi/ft2232h-module-target.cfg]
transport select jtag
source [find target/stm32f4x.cfg]
adapter speed 1000
EOF

openocd -f openocd.cfg
# Open On-Chip Debugger 0.12.0
# Info : JTAG tap: stm32f4x.cpu tap/device found: 0x4ba00477
# Info : stm32f4x.cpu: hardware has 6 breakpoints, 4 watchpoints

# Connexion GDB pour debug interactif
arm-none-eabi-gdb
(gdb) target remote localhost:3333
(gdb) monitor reset halt
(gdb) monitor flash read_image firmware.bin 0x08000000 0x100000
# Dump de 1MB de flash à l'adresse 0x08000000

# Dump de la RAM
(gdb) monitor dump_image ram_dump.bin 0x20000000 0x00040000
```

SWD (Serial Wire Debug)

SWD est une alternative à JTAG spécifique aux processeurs ARM, nécessitant seulement 2 fils (SWDIO et SWCLK) plus GND. C'est l'interface de debug standard sur les microcontrôleurs STM32, nRF52, ESP32-C3 et la plupart des MCUs ARM Cortex-M :

```
# SWD avec ST-Link V2 (ou clone)
# Connexions : SWDIO, SWCLK, GND, (optionnel: 3.3V, NRST)

# Utilisation d'OpenOCD en mode SWD
cat > swd.cfg << 'EOF'
source [find interface/stlink.cfg]
transport select hla_swd
source [find target/stm32f1x.cfg]
EOF

openocd -f swd.cfg

# Dump complet de la flash via SWD
# Méthode 1 : OpenOCD
openocd -f swd.cfg -c "init; reset halt; flash read_image firmware.bin 0x08000000
0x80000; exit"

# Méthode 2 : st-flash (STMicroelectronics)
st-flash read firmware.bin 0x08000000 0x80000

# Méthode 3 : pyOCD (multi-cible)
pyocd flash --target stm32f103rc -r firmware.bin

# Vérification de la protection de lecture (RDP)
openocd -f swd.cfg -c "init; stm32f1x options_read 0; exit"
# Si RDP Level 1 : lecture bloquée, mais bypass possible via glitching
# Si RDP Level 2 : protection permanente (irréversible)
```

---

3. Outils (Bus Pirate, JTAGulator, Saleae)

Arsenal du hardware hacker

Outil	Fonction	Protocoles	Prix
Bus Pirate v5	Protocole universel	UART, SPI, I2C, JTAG, 1-Wire	~50 EUR
JTAGulator	Identification automatique	JTAG, UART, SWD	~180 EUR
Saleae Logic Pro 8	Analyseur logique	Tous (décodage)	~500 EUR
ST-Link V2	Debug ARM	SWD, JTAG	~5 EUR (clone)
CH341A	Programmeur flash	SPI, I2C (EEPROM)	~5 EUR
Tigard	Multi-protocole FT2232H	UART, SPI, I2C, JTAG, SWD	~50 EUR
ChipWhisperer	Glitching / Side-channel	Power analysis, fault injection	~300 EUR

```
# Bus Pirate v5 - Lecture SPI Flash
# Connexion : CS, MOSI, MISO, CLK, GND, 3.3V
# Mode SPI dans la console Bus Pirate :
m 5  # Mode SPI
W    # Activer l'alimentation
[0x9F r:3]  # Lire JEDEC ID
# Réponse : 0xEF 0x40 0x18 = Winbond W25Q128 (16MB)

# Dump avec flashrom via Bus Pirate
flashrom -p buspirate_spi:dev=/dev/ttyUSB0 -r firmware_dump.bin
# Taille détectée automatiquement : 16777216 bytes (16 MB)
# Lecture complète : ~30 minutes via Bus Pirate (lent)

# Alternative rapide : CH341A + clip SOIC-8
# Clip directement sur la puce SPI sans dessouder
flashrom -p ch341a_spi -r firmware_dump.bin
# Lecture : ~2 minutes pour 16 MB

# Saleae Logic - Capture et décodage
# 1. Connecter les sondes sur les signaux SPI/UART/I2C
# 2. Démarrer la capture pendant le boot du device
# 3. Le logiciel décode automatiquement les protocoles
# 4. Exporter les données décodées en CSV/binaire
```

Cas concret

L'attaque sur SolarWinds Orion (2020) a illustré les limites des architectures de sécurité traditionnelles. L'insertion d'une backdoor dans le processus de build du logiciel a contourné toutes les couches de défense, rappelant que la supply-chain logicielle est un vecteur de menace de premier ordre.

Votre processus de patch management couvre-t-il l'ensemble de votre parc applicatif ?

4. Dump de Flash SPI/NAND

Flash SPI NOR (la plus courante)

La majorité des dispositifs IoT stockent leur firmware dans une puce flash SPI NOR (Winbond W25Qxx, Macronix MX25Lxx, GigaDevice GD25Qxx). Ces puces sont facilement identifiables : boîtier SOIC-8 (8 pins), située près du processeur principal. Pour approfondir, consultez Top 10 Solutions EDR/XDR.

```
# Identification de la puce flash sur le PCB
# 1. Repérer le boîtier SOIC-8 (marqué W25Q128, MX25L64, etc.)
# 2. Vérifier la datasheet pour le pinout :
#    Pin 1: /CS (Chip Select)
#    Pin 2: DO (Data Out / MISO)
#    Pin 3: /WP (Write Protect)
#    Pin 4: GND
#    Pin 5: DI (Data In / MOSI)
#    Pin 6: CLK
#    Pin 7: /HOLD
#    Pin 8: VCC (3.3V)

# Méthode 1 : Lecture in-circuit avec clip SOIC-8
# (le processeur doit être maintenu en reset ou éteint)
# Clip Pomona 5250 + CH341A
flashrom -p ch341a_spi -r dump1.bin
flashrom -p ch341a_spi -r dump2.bin
# Vérifier l'intégrité : les deux dumps doivent être identiques
md5sum dump1.bin dump2.bin
# Si différents : problème de signal, refaire le dump

# Méthode 2 : Dessouder la puce (station à air chaud)
# Température : 350°C, flux no-clean
# Avantage : lecture fiable sans interférence du processeur
# Inconvénient : risque d'endommager la puce

# Méthode 3 : Lecture via JTAG/SWD (si le processeur est accessible)
# Le processeur accède à la flash SPI via un bus interne
openocd -f swd.cfg -c "init; flash read_image dump.bin 0x0 0x1000000; exit"
```

Flash NAND et eMMC

```
# Les dispositifs plus complexes (routeurs haut de gamme, NAS, caméras)
# utilisent des flash NAND ou eMMC

# eMMC : lecture via adaptateur eMMC-USB (Easy JTAG, Medusa Pro)
# ou mode ISP (In-System Programming) via les pads du PCB

# NAND : lecture avec programmeur universel (TNM5000, XGecu T56)
# Attention aux bad blocks et à l'ECC (Error Correction Code)

# Lecture eMMC via les pads CMD, CLK, DAT0 du PCB
# Avec un adaptateur SD-eMMC et un lecteur SD USB
dd if=/dev/sdX of=emmc_full_dump.bin bs=1M status=progress
# ou
sudo dd if=/dev/mmcblk0 of=emmc_dump.bin bs=512 count=30535680

# Parsage des partitions eMMC
fdisk -l emmc_dump.bin
# Device         Boot   Start      End  Sectors  Size Id Type
# emmc_dump.bin1         2048   526335   524288  256M 83 Linux
# emmc_dump.bin2       526336 30535679 30009344 14.3G 83 Linux

# Extraction de la partition rootfs
dd if=emmc_dump.bin of=rootfs.img bs=512 skip=526336 count=30009344
mount -o loop rootfs.img /mnt/rootfs/
```

---

5. Analyse de firmware (binwalk, Ghidra)

Extraction avec binwalk

```
# binwalk : outil de référence pour l'analyse de firmware
# Scan des signatures (headers de fichiers, systèmes de fichiers)
binwalk firmware_dump.bin
# DECIMAL       HEXADECIMAL     DESCRIPTION
# 0             0x0             uImage header, "Linux Kernel"
# 64            0x40            LZMA compressed data
# 1048576       0x100000        Squashfs filesystem, little endian, version 4.0
# 14680064      0xE00000        JFFS2 filesystem, little endian

# Extraction automatique
binwalk -e firmware_dump.bin
# Crée un répertoire _firmware_dump.bin.extracted/
# Contient le système de fichiers décompressé

# Extraction récursive (décompresse les archives imbriquées)
binwalk -eM firmware_dump.bin

# Analyse de l'entropie (détection de chiffrement/compression)
binwalk -E firmware_dump.bin
# Si entropie proche de 1.0 partout : firmware chiffré
# Si entropie variable : sections compressées identifiables

# Alternative : jefferson pour JFFS2
jefferson firmware_dump.bin -d output_jffs2/

# Alternative : sasquatch pour SquashFS non-standard
# (firmwares modifiés avec compression propriétaire)
sasquatch -f firmware_dump.bin -d output_squashfs/
```

Analyse du système de fichiers

```
# Une fois le rootfs extrait, recherche de vulnérabilités
cd _firmware_dump.bin.extracted/squashfs-root/

# 1. Credentials hardcodées
grep -rn "password\|passwd\|secret\|key\|token" etc/
cat etc/shadow
# root:$1$abc123$...:0:0:99999:7:::
# admin:$1$xyz789$...:0:0:99999:7:::
# Cracker avec John the Ripper ou hashcat

# 2. Clés privées SSH/SSL
find . -name "*.pem" -o -name "*.key" -o -name "id_rsa" -o -name "*.crt"
# Souvent dans etc/ssl/ ou usr/share/

# 3. Configuration réseau et services
cat etc/init.d/rcS  # Script de démarrage
cat etc/inittab     # Services lancés au boot
grep -rn "telnetd\|dropbear\|sshd\|httpd" etc/init.d/

# 4. Binaires custom (logique métier)
file usr/bin/* usr/sbin/*
# Identifier l'architecture : ARM, MIPS, x86
# Chercher les binaires non-standard (pas BusyBox)

# 5. Émulation avec QEMU pour analyse dynamique
# Pour ARM :
cp $(which qemu-arm-static) .
sudo chroot . ./qemu-arm-static /usr/sbin/httpd
# Le serveur web embarqué tourne localement pour analyse

# Firmware Analysis Toolkit (FAT) - automatisation
git clone https://github.com/attify/firmware-analysis-toolkit
cd firmware-analysis-toolkit
./fat.py firmware_dump.bin
# Émule le firmware complet avec QEMU + networking
```

Reverse engineering avec Ghidra

```
# Ghidra : outil de reverse engineering de la NSA (gratuit, open source)
# Import du binaire cible dans Ghidra
# File > Import File > sélectionner le binaire ARM/MIPS

# Configuration de l'architecture :
# ARM Little Endian 32-bit pour la plupart des IoT
# MIPS Big Endian 32-bit pour les routeurs (Broadcom, Mediatek)

# Analyse automatique (CodeBrowser > Analysis > Auto Analyze)
# Recherche de fonctions vulnérables :
# - strcpy, sprintf, gets (buffer overflow)
# - system, popen, exec (command injection)
# - memcpy avec taille non vérifiée

# Script Ghidra pour trouver les appels à system()
# Dans la console Python de Ghidra :
from ghidra.program.model.symbol import *
fm = currentProgram.getFunctionManager()
for func in fm.getFunctions(True):
    if 'system' in func.getName().lower():
        refs = getReferencesTo(func.getEntryPoint())
        for ref in refs:
            print(f"system() appelé depuis: {ref.getFromAddress()}")
            listing = currentProgram.getListing()
            inst = listing.getInstructionAt(ref.getFromAddress())
            print(f"  Instruction: {inst}")
```

---

6. Attaques Side-Channel (Glitching, Power Analysis)

Voltage Glitching

Le voltage glitching consiste à injecter une perturbation brève (quelques nanosecondes) sur la ligne d'alimentation du processeur au moment précis où il exécute une vérification de sécurité (vérification de signature, contrôle RDP, authentification). Le glitch provoque un saut d'instruction qui peut bypasser la vérification :

```
# ChipWhisperer - Plateforme de fault injection
# Installation
pip install chipwhisperer

# Script Python pour glitching sur STM32 (bypass RDP)
import chipwhisperer as cw

# Connexion au ChipWhisperer Lite
scope = cw.scope()
target = cw.target(scope)

# Configuration du glitch
scope.glitch.clk_src = "clkgen"
scope.glitch.output = "enable_only"
scope.glitch.trigger_src = "ext_single"

# Paramètres à balayer (brute-force du timing)
scope.glitch.width = 10      # Largeur du glitch (en % du cycle)
scope.glitch.offset = 25     # Offset par rapport au trigger
scope.glitch.repeat = 1      # Nombre de répétitions

# Boucle de glitching
for width in range(5, 50, 1):
    for offset in range(-40, 40, 1):
        scope.glitch.width = width
        scope.glitch.offset = offset

        # Reset du target et armement
        scope.arm()
        target.reset()

        # Attendre le résultat
        ret = scope.capture()
        response = target.read()

        if "SUCCESS" in response or len(response) > expected_len:
            print(f"[!] GLITCH REUSSI! width={width}, offset={offset}")
            print(f"    Response: {response}")
            break
```

Simple Power Analysis (SPA) et Differential Power Analysis (DPA)

L'analyse de la consommation électrique pendant l'exécution de fonctions cryptographiques permet d'extraire les clés de chiffrement. Chaque opération du processeur (multiplication, addition, accès mémoire) a une signature énergétique unique :

```
# Capture de traces de consommation avec ChipWhisperer
import chipwhisperer as cw
import chipwhisperer.analyzer as cwa

# Acquisition de traces pendant le chiffrement AES
scope = cw.scope()
target = cw.target(scope, cw.targets.SimpleSerial)

# Capturer 5000 traces avec des plaintexts aléatoires
traces = []
for i in range(5000):
    plaintext = bytearray(os.urandom(16))
    target.simpleserial_write('p', plaintext)
    scope.arm()
    target.simpleserial_wait_ack()
    trace = scope.get_last_trace()
    traces.append((plaintext, trace))

# Attaque CPA (Correlation Power Analysis) pour extraire la clé AES
attack = cwa.cpa(traces)
key_guess = attack.run()
print(f"Clé AES extraite: {key_guess.hex()}")
# Précision typique : 100% avec ~1000 traces pour AES-128
```

---

7. Cas Pratiques IoT

Audit d'une caméra IP

Scénario typique d'audit hardware d'une caméra IP grand public :

1. **Ouverture du boîtier** : 4 vis cruciformes sous l'étiquette. PCB unique avec SoC HiSilicon Hi3518EV300, flash SPI W25Q128 (16 MB), RAM DDR3 512 MB.
2. **Identification UART** : Header 4 pins non peuplé. JTAGulator identifie TX/RX à 115200 baud. Shell root BusyBox au boot sans mot de passe.
3. **Dump firmware via SPI** : Clip SOIC-8 + CH341A. Dump de 16 MB en 2 minutes. binwalk extrait un rootfs SquashFS.
4. **Analyse du firmware** : Credentials admin:admin123 dans /etc/passwd. Serveur RTSP sans authentification sur le port 554. Clé privée SSL commune à tous les modèles dans /etc/ssl/.
5. **Exploitation** : Command injection dans le paramètre "NTP server" de l'interface web : `; telnetd -l /bin/sh -p 4444 &`

Audit d'un routeur

```
# Méthodologie complète pour un routeur TP-Link / Netgear / Asus

# 1. Récupération du firmware (sans hardware)
# Télécharger depuis le site du constructeur
wget https://www.tp-link.com/res/down/soft/TL-WR841N_V14_firmware.bin

# 2. Analyse avec binwalk
binwalk -eM TL-WR841N_V14_firmware.bin
cd _TL-WR841N_V14_firmware.bin.extracted/squashfs-root/

# 3. Recherche de vulnérabilités
# Backdoors connues
grep -rn "TDDP\|tddp" usr/bin/
strings usr/bin/httpd | grep -i "backdoor\|debug\|test"

# 4. Si firmware chiffré (tendance 2025-2026)
# Extraire la clé de déchiffrement depuis un dump UART
# ou via une version antérieure non chiffrée du firmware

# 5. Émulation avec FirmAE (successeur de Firmadyne)
git clone https://github.com/pr0v3rbs/FirmAE
cd FirmAE
./init.sh
sudo ./run.sh -d netgear ./firmware/R7000-V1.0.11.116_10.2.100.chk
# Le routeur émulé est accessible à http://192.168.0.1
```

Pour approfondir ce sujet, consultez notre outil open-source log-analyzer qui facilite l'analyse automatisée des journaux de sécurité.

Questions frequentes

Comment ce sujet impacte-t-il la securite des organisations ?

Ce sujet a un impact significatif sur la securite des organisations car il touche aux fondamentaux de la protection des systemes d'information. Les entreprises doivent evaluer leur exposition, mettre en place des mesures preventives adaptees et former leurs equipes pour faire face aux risques associes a cette problematique.

Quelles sont les bonnes pratiques recommandees par les experts ?

Les experts recommandent une approche basee sur les risques, incluant l'evaluation reguliere de la posture de securite, la mise en place de controles techniques et organisationnels, la formation continue des equipes et l'adoption des referentiels de securite reconnus comme ceux du NIST, de l'ANSSI et de l'OWASP. Pour approfondir, consultez Agents IA pour la Cyber-Défense et le Threat Hunting Automatisé.

Pourquoi est-il important de se former sur ce sujet en 2026 ?

En 2026, la maitrise de ce sujet est devenue incontournable face a l'evolution constante des menaces et des exigences reglementaires. Les professionnels de la cybersecurite doivent maintenir leurs competences a jour pour proteger efficacement les actifs numeriques de leur organisation et repondre aux obligations de conformite.

Sources et références : MITRE ATT&CK · CERT-FR

8. Conclusion

Le hardware hacking est une composante essentielle de l'audit de sécurité IoT. La majorité des dispositifs connectés présentent des faiblesses matérielles exploitables : ports UART avec shell root, flash SPI lisible sans authentification, firmwares non chiffrés, et absence de mécanismes anti-tampering. Ces vulnérabilités permettent souvent d'obtenir un accès complet au système et d'identifier des failles exploitables à distance.

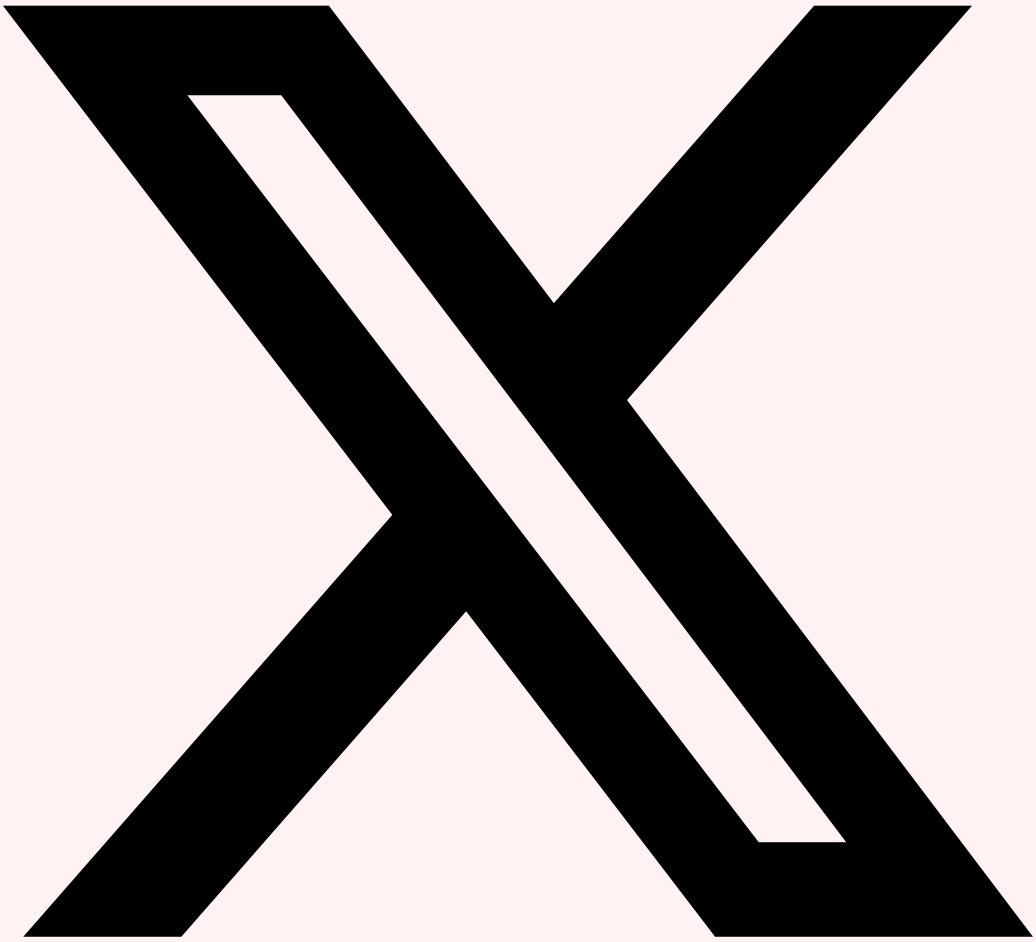
Les fabricants doivent implémenter des mesures de protection à chaque couche : désactivation des ports de debug en production (eFuse), chiffrement du firmware avec clé unique par device, Secure Boot avec chaîne de confiance matérielle (Root of Trust), protection anti-glitching (détecteurs de fault), et mécanismes anti-readout (RDP Level 2 sur STM32, Code Read Protection sur NXP).

Recommandations pour les fabricants IoT

- Désactiver JTAG/SWD/UART en production via eFuse ou configuration OTP
- Implémenter le Secure Boot avec vérification de signature du firmware
- Chiffrer le firmware avec une clé unique par device (device-specific key)
- Utiliser un Trusted Execution Environment (TEE) pour les opérations sensibles
- Implémenter un mécanisme de mise à jour OTA sécurisé (signature + rollback protection)
- Protéger la flash SPI contre la lecture externe (Secure Flash, encrypted XIP)
- Ajouter des contre-mesures anti-glitching (voltage/clock monitors, redundant checks)

Partagez cet Article

Partagez avec votre réseau professionnel ! Pour approfondir, consultez Secrets sprawl : collecte.



Partager sur X

Partager sur LinkedIn

Ayi NEDJIMI

Expert en Cybersécurité & Intelligence Artificielle

Consultant senior avec plus de 15 ans d'expérience en sécurité offensive, audit d'infrastructure et développement de solutions IA. Certifié OSCP, CISSP, ISO 27001 Lead Auditor et ISO 42001 Lead Implementer. Intervient sur des missions de pentest Active Directory, sécurité Cloud et conformité réglementaire pour des grands comptes et ETI.

LinkedIn Profil complet Tous ses articles

Ressources & Références

Binwalk
github.com
Ghidra (NSA)
ghidra-sre.org
OpenOCD
openocd.org

Références et ressources externes

- OWASP Testing Guide — Guide de référence pour les tests de sécurité web
- MITRE ATT&CK T1542 — Pre-OS Boot — Firmware
- PortSwigger Academy — Ressources d'apprentissage en sécurité web
- CWE — Common Weakness Enumeration — catalogue de faiblesses logicielles
- NVD — National Vulnerability Database — base de vulnérabilités du NIST

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

ayinedjimi-consultants.fr · ayi@ayinedjimi-consultants.fr