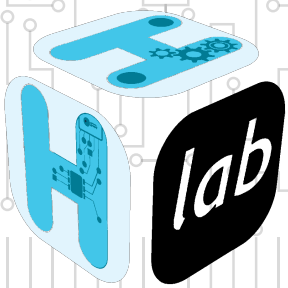
H2LAB-STIG-JTAG
07-2026

Embedded Systems: the JTAG interface



STIG H^{2}Lab

Information

Created in 2020, H2Lab is a non-profit organization dedicated to designing, developing, and sharing open-hardware and open-source solutions for experimentation around embedded systems. Its main areas of work are:

- Design of hardware platforms for hardware and software analysis
- Development of open alternatives to proprietary, often opaque tools
- Publication of technical guides and recommendations to promote bestpractices in security, privacy, and sustainability across the ecosystemof connected devices and embedded systems

Support for researchers, teachers, and students through the provision of tools, educational content, and training.
Academic partnerships to encourage sharing of resources and knowledge.

Release history

Version	Date	Author	Changes
1.0	July 2026	H2Lab	Initial version of the guide

Contents

1
Glossary and acronyms

1.1 Glossary

Trust anchor. Immutable data (public key, firmware hash, root certificate) used to establish an initial trust relationship.

Anti-probing. Hardware and design measures intended to make identification and instrumentation of sensitive test points more difficult.

Anti-rollback. Mechanism that prevents reverting to an earlier version that is cryptographically valid but vulnerable.

Brown-out. Transient supply voltage drop that can cause exploitable errors during fault-injection attacks.

Supply chain. Set of stakeholders, tools, and operations involved between product design and commissioning.

Challenge-response. Authentication mechanism where the target verifies possession of a secret based on a cryptographic challenge.

Debug in production. Analysis or reprogramming capability retained on a delivered product, generally for maintenance.

Exception debug. Temporary debug access, strictly controlled and traceable, enabled only with operational justification.

Deny by default. Security principle where all access is denied by default, unless explicitly authorized.

Forensic window. Set of exploitable traces used to reconstruct a security event after an incident.

OTP fuse. One-Time Programmable configuration bit, often used to lock sensitive functions.

Lifecycle. Lifecycle states of a component (development, production, service, end of life) that determine security privileges.

Lock bits. Configuration bits that enable access restrictions (read, write, debug) depending on component state.

Service mode. Operating mode that enables controlled, temporary, and traceable debugging on equipment in operation.

Hardware root of trust. Hardware element ensuring secure storage and use of cryptographic identifiers.

Supply-chain. Common English term used to designate the supply chain and its compromise risks.

Attack surface. Set of entry points exploitable by an attacker on a given system.

Traceability. Ability to reconstruct who did what, when, how, and with what level of authorization.

1.2 List of acronyms

AM. Attacker profile formalized in this guide (AM-01 to AM-05).

ANSSI. French national agency for information systems security.

ARM. *Advanced RISC Machines.*

BKPT. Software breakpoint instruction (*BreaK PoinT*).

BOM. *Bill Of Materials.*

CAP. Technical capability associated with an attacker profile (CAP-01 to CAP-08).

CPU. *Central Processing Unit.*

CSIRT. *Computer Security Incident Response Team.*

DFU. *Device Firmware Update.*

EM. *Electromagnetic.*

EMFI. *ElectroMagnetic Fault Injection.*

ENISA. *European Union Agency for Cybersecurity.*

ESD. *ElectroStatic Discharge.*

ETM. *Embedded Trace Macrocell.*

FDA. *Food and Drug Administration.*

FPGA. *Field-Programmable Gate Array.*

FS. Security function required in this guide (FS-01 to FS-09).

HSM. *Hardware Security Module.*

HW. *Hardware.*

IACS. *Industrial Automation and Control Systems.*

IEEE. *Institute of Electrical and Electronics Engineers.*

IEC. *International Electrotechnical Commission.*

ISO. *International Organization for Standardization.*

IoT. *Internet of Things.*

IR/DR. *Instruction Register / Data Register.*

ITM. *Instrumentation Trace Macrocell.*

JTAG. *Joint Test Action Group.*

MCU. *Microcontroller Unit.*

MDR. *Medical Device Regulation.*

NIST. *National Institute of Standards and Technology.*

NISTIR. *NIST Interagency/Internal Report.*

OEM. *Original Equipment Manufacturer.*

OS. Security objective in this guide (OS-01 to OS-08).

OT. *Operational Technology.*

OTP. *One-Time Programmable.*

PMP. *Physical Memory Protection.*

PCB. *Printed Circuit Board.*

RAM. *Random Access Memory.*

RDP. *Readout Protection.*

RMA. *Return Merchandise Authorization.*

SE. *Secure Element.*

SIEM. *Security Information and Event Management.*

SL. *Security Level* (IEC 62443).

SOC. *Security Operations Center.*

SP. *Special Publication* (NIST SP).

SRAM. *Static Random Access Memory.*

SSI. Information systems security.

STIG. *Security Technical Implementation Guide.*

SWD. *Serial Wire Debug.*

SWO. *Serial Wire Output.*

SVC. *Supervisor Call.*

TAP. *Test Access Port.*

TCK/TMS/TDI/TDO. Standard signals of the JTAG interface.

TEE. *Trusted Execution Environment.*

TRST. Optional TAP reset signal (*Test ReSeT*).

TPM. *Trusted Platform Module.*

TTY. *Teletype* (console-like terminal channel).

TVS. *Transient Voltage Suppressor.*

UART. *Universal Asynchronous Receiver-Transmitter.*

USB. *Universal Serial Bus.*

2
Introduction and scope

2.1 Guide objective

This document is a **STIG** (*Security Technical Implementation Guide*) dedicated to securing JTAG/SWD debug interfaces in critical embedded systems. The objective is to provide an implementation framework that is:

- technically actionable (concrete, testable requirements)
- operable in an industrial OT context (production, maintenance, incident)
- auditable (evidence, traceability, compliance criteria)

The guide covers the full lifecycle: design, integration, production, deployment, and maintenance. It explicitly addresses the tension between maintainability and cyber robustness, following a defense-in-depth and operational traceability approach [9, 10, 18, 1].

2.2 Target audience

This guide is designed for two-level reading: strategic level (architecture, risk governance, compliance) and implementation level (concrete settings, technical validation, audit evidence). This dual reading facilitates alignment between engineering decisions and business constraints.

The document is intended for the following profiles:

- Embedded cybersecurity architects (OT systems, industrial IoT, automotive, medical)
- Hardware/firmware engineers responsible for MCU/SoC, boot, and provisioning choices
- Developers of critical security products who must integrate controlled debug mechanisms
- Information security managers, technical auditors, and quality teams responsible for compliance evidence

Recommendations are written to be used both as a design baseline and as an internal or third-party audit reference.

2.3 Technical scope and assumptions

2.3.1 Equipment type and OT context

The scope covers connected or semi-connected embedded equipment, based on MCU/SoC/FPGA with JTAG/SWD-type debug interfaces. The targeted use cases include factory commissioning, field diagnostics, and corrective maintenance in OT environments with strong availability constraints.

2.3.2 Lifecycle considered

The document addresses debug security over the whole lifecycle:

- board and boot/debug architecture design
- industrialization (provisioning, fuses/lock bits, quality control)
- deployment in operation (supervision, access control)
- maintenance (temporary opening, closure, and post-intervention evidence)

2.3.3 Threat assumptions

The considered scenarios include a local attacker with opportunistic physical access, a malicious insider or subcontractor, and an equipped attacker able to combine debug access, firmware extraction, and transient faults. The central defensive assumption is that partial physical access is plausible during maintenance, and must therefore be controlled rather than assumed impossible.

2.3.4 Operational reading of the STIG

This document can be used in three complementary modes:

- **Architecture mode**: define security principles from the design phase
- **Industrialization mode**: translate principles into factory settings and batch tests
- **Audit mode**: verify effective compliance and operational resilience

This three-step reading facilitates dialogue between hardware, firmware, cybersecurity, and OT operations teams, and reduces the gap between theoretical requirement and field applicability.

3
Technical fundamentals of JTAG

3.1 Origin and standardization

JTAG historically emerged for electronic board interconnection testing (*boundary scan*) before becoming a preferred channel for debugging and programming. IEEE 1149.1 standardization defines the chain model, TAP controller states, and command/data registers [11].

In industry, legitimate low-level debug uses are numerous:

- End-of-line production testing
- Hardware bring-up and firmware validation
- Field fault diagnostics and product recovery
- Initial programming or controlled reprogramming

In practice, the same interface that strongly reduces qualification time can also reduce attack cost if it remains open in production. The STIG logic is therefore to preserve the operational value of debug while turning it into an exception capability rather than a permanent capability.

The security problem is therefore not the existence of JTAG itself, but its level of exposure outside controlled development and production contexts.

3.2 Physical and logical architecture

JTAG architecture is based on TAP, driven by TCK, TMS, TDI, TDO (and sometimes TRST) signals. TAP orchestrates state transitions that load instructions (IR) and then exchange data (DR). This mechanism can provide deep access to a component internal state.

From a security perspective, three properties are decisive:

- **Access universality**: register and memory read/write depending on implementation
- **Precision**: fine execution control (halt/step/breakpoint)

- **Partial bypass**: depending on the MCU, some software policies can be bypassed through the hardware debug plane

Concrete variability example: on STM32 families configured at RDP level 0, memory read via the debug interface may remain possible during development, whereas in strong lock configuration (for example, high RDP level), that same access is heavily restricted or blocked. Conversely, on some NXP platforms with a *secure debug mailbox* mechanism, access can be temporarily reopened after explicit authentication, which is not equivalent to permanently free debug [26, 20].

3.3 Variants and evolutions

In the ARM ecosystem, SWD often replaces JTAG on microcontrollers to reduce pin count while preserving strong debugging capabilities. Other extensions exist: embedded trace (ETM), semihosting, proprietary maintenance interfaces [2].

Semihosting. Semihosting is a mechanism by which target firmware delegates some input/output operations to the debug host (tooling PC), via a software breakpoint (often a *BKPT/SVC*-type instruction depending on architecture). Concretely, an application call such as *printf* can be redirected to the debugger console, without a network stack or UART driver on the target side. During development, this greatly improves observability. In production, however, semihosting left enabled can create an information leakage channel or execution disruption channel if an unauthorized debug tool is connected.

ETM (Embedded Trace Macrocell). ETM is a hardware trace block that exports fine-grained execution information (notably instruction flow, branches, events) with limited CPU overhead compared with traditional software instrumentation. This capability is valuable for temporal bug analysis and post-mortem investigation. From a security standpoint, trace can still reveal sensitive sequences (authentication paths, cryptographic states, control logic). A robust STIG must therefore control activation, scope, and trace retention.

TTY over JTAG/SWD port. The notion of TTY designates a console-like terminal channel (text input/output) exposed through debug infrastructure, often through mechanisms such as semihosting, SWO/ITM, or tooling bridges to a host pseudo-console. In practice, this amounts to obtaining a "virtual serial console" without a dedicated physical UART. The benefit is operational (fast diagnostics), but the risk is reintroducing a powerful interactive channel on equipment that is supposed to be locked down. In production, this channel must be explicitly disabled or strictly conditioned (authentication, limited duration, full logging).

Major evolutions to consider in a STIG are:

- Multiplication of debug channels (JTAG, SWD, backup UART, USB DFU)
- Chaining of multiple components (daisy-chain), with impact on trust-domain isolation
- Advanced debug functions that can expose sensitive data in cleartext (keys, session secrets, authentication states)

From a defensive viewpoint, these evolutions require an aggregated-surface approach: all maintenance paths must be secured, not only the main JTAG connector, otherwise the attacker bypasses the most visible measure through a less protected auxiliary channel.

4

Threats and attack vectors through JTAG

The JTAG threat must be understood as an *operational chain* threat. In observed incidents, the attack rarely succeeds because of a single technical weakness. It succeeds rather through chaining: plausible physical access, insufficient maintenance-mode control, heterogeneous security settings across batches, and incomplete supervision. This perspective is essential to avoid a purely "protection-bit" response.

4.1 Attacker interest

Uncontrolled JTAG/SWD access often turns a complex attack into a direct extraction and modification operation:

- Firmware dump from internal/external flash
- Arbitrary read/write of SRAM, peripherals, and security registers
- Code injection and bypass of integrity-control mechanisms
- Recovery of temporary secrets (derived keys, tokens, passwords in memory)

Concretely, these actions give the attacker very low-level visibility and control that bypass most application mechanisms. Strong application-side authentication does not protect a secret already exposed in RAM during an unauthorized debug session.

Strategic value is high: intellectual property theft, product cloning, targeted sabotage, compromise of maintenance and supply chains. Field analyses show that successful attacks frequently exploit a combination of weaknesses (active debug, insufficient logs, static secrets) rather than a single flaw [22, 16].

4.2 Attack surface

Contrary to common belief, a physical-access constraint is not a sufficient guarantee in industrial environments. Realistic scenarios include:

- Malicious intervention in a maintenance workshop
- Compromise of a repair subcontractor
- Opportunistic attack on exposed equipment (cabinet, vehicle, terminal)
- Access through undocumented test pads or internal connectors

In OT environments, the opportunity window is often tied to the maintenance cycle: chassis opening, module replacement, temporary board access. These are precisely the moments that must be protected by identity, traceability, and supervision controls.

Common targets are microcontrollers, SoCs, FPGAs, and associated memories. Modern pin-identification tools significantly reduce attack entry cost [7, 21].

4.3 Attacker capabilities with unprotected JTAG access

Observed technical capabilities include:

- Memory mapping and full exfiltration of firmware images
- Deactivation of application controls (authentication, anti-rollback, business checks)
- Persistence through binary patching and firmware reinjection
- Creation of backdoors difficult to distinguish from support operations
- Replay across equipment series when secrets are not individualized

The main risk is not only information extraction, but the ability to industrialize the attack: a method validated on a pilot device can then be reproduced at low cost across a whole fleet.

In multi-component architectures, unprotected debug on a single link may be enough to compromise the entire system.

4.4 Implications for industrial security

OT impacts are systemic:

- **Operational sabotage**: line stoppage, process drift, asset unavailability
- **IP theft**: extraction of algorithms, proprietary parameters, calibration
- **Supply chain compromise**: insertion of modified firmware before delivery
- **Safety impact**: alteration of critical functions in safety environments

In an industrial context, these impacts often propagate beyond the compromised asset: service unavailability, quality defects, contractual non-compliance, and even regulatory exposure. JTAG risk management must therefore be treated as a business continuity issue, not only as local hardening.

Academic feedback, embedded-security competitions, and specialized conferences confirm the operational feasibility of these scenarios when debug governance is insufficient [15, 3, 4, 30].

4.5 Reference attack context

The following cases provide operational context for debug security. The goal is not to establish strict equivalence between historical incidents, but to link observed attack patterns to testable technical requirements.

Case (date)	Attacker	Attack	Target asset	Associated gain
Firmware extraction IoT/OT (2018–2025)	Local attacker with opportunistic physical access	Access to debug pads, memory dump, binary analysis	MCU/SoC, embedded secrets, control logic	IP theft, clone preparation, vulnerability discovery [14]
STM32 readout-protection bypass (2017–2024)	Equipped physical attacker (lab)	Voltage/clock glitch, partial protection bypass	Internal flash, security states, boot chain	Secret extraction, serial-attack preparation [26, 17]
Field maintenance compromise (OT feedback)	Malicious insider/-subcontractor	Service-mode abuse, untracked debug opening	In-operation equipment, event logs	Stealth persistence, targeted sabotage, trace deletion [16]
MCU fault injection (2017–2024)	Advanced physical attacker	EMFI/glitch during critical checks	Boot controls, debug locks, lifecycle states	Control bypass, local capability escalation [5, 17, 25]
Diverted industrial debug tooling (ongoing)	Local attacker trained on vendor tools	OpenOCD scripts/debug probes, runtime modification	RAM, sensitive registers, active firmware	Backdoor injection, process-behavior alteration [21, 7]

Table 4.1: Reference attack context for JTAG/SWD

4.6 Attacker model and technical capabilities

The threat model is formalized along two dimensions:

- **AM-* profiles**: attacker position in the system (local, insider, advanced physical)
- **CAP-* capabilities**: technically achievable means

ID	Profile	Primary surface	Capability assumption
AM-01	Opportunistic local	External connectors, accessible test pads, open enclosure	CAP-01, CAP-02, CAP-03
AM-02	Malicious maintainer	Maintenance process, workshop tooling, service mode	CAP-02, CAP-03, CAP-04, CAP-06
AM-03	Ecosystem insider	Production chain, provisioning scripts, workshop secrets	CAP-04, CAP-05, CAP-06
AM-04	Physical lab attacker	PCB, power supply, clock, EM instrumentation	CAP-02, CAP-07, CAP-08
AM-05	Hybrid supply-chain + field attacker	Product batches, maintenance subcontracting, insufficient supervision	CAP-03 to CAP-08

Table 4.2: Attacker profiles considered for debug risk

ID	Capability	Operational description
CAP-01	Hardware reconnaissance	Identification of debug pins, signal mapping, TAP/SWD chain validation.
CAP-02	Unauthorized debug access	Halt/step, memory read/write, register control, tooling-script execution.
CAP-03	Firmware extraction and analysis	Flash/RAM dump, reverse engineering, search for secrets and control mechanisms.
CAP-04	Local persistence	Binary patch in memory/flash, insertion of dormant mechanism, lasting behavior alteration.
CAP-05	Workshop-secret compromise	Abuse of maintenance tokens/scripts, reuse of technical credentials.
CAP-06	Debug-governance bypass	Out-of-procedure opening, tooling-identity impersonation, trace deletion or absence.
CAP-07	Physical fault injection	Voltage/clock glitch, EMFI, disruption of critical boot checks.
CAP-08	Attack industrialization	Multi-batch reproducibility, automation of extraction/modification steps, large-scale propagation.

Table 4.3: Technical capabilities of the JTAG/SWD threat model

4.7 Security objectives related to debug governance

Security objectives are standardized below under `OS-*` naming.

ID	Objective	Verification criterion
OS-01	Confidentiality of firmware assets	Unauthorized debug access does not allow exploitable firmware or secret extraction.
OS-02	Execution integrity	Unauthorized debug access does not allow persistent alteration of software behavior.
OS-03	Access control to debug mode	Any debug opening is conditioned on strong identity and explicit authorization.
OS-04	Temporal and scope limitation	Debug sessions are bounded in duration and scope, and revocable at any time.
OS-05	Boot/debug consistency	Debug authorization remains consistent with Secure Boot state and anti-rollback policy.
OS-06	Resistance to basic physical attacks	Protections resist opportunistic attempts and detect simple disturbances.
OS-07	Forensic traceability of interventions	Any debug session produces auditable, timestamped, correlatable evidence.
OS-08	Secure continuity of operations	Maintenance remains possible in exception mode without creating a permanent attack channel.

Table 4.4: Standardized security objectives for JTAG/SWD governance

Required security functions (FS-*). Implementable security functions are identified under `FS-*` naming and linked to `OS-*` objectives.

ID	Security function	Implementation requirement
FS-01	Production hardware locking	Removal/protection of connectors, anti-probing, traceable DEV/VAL/PROD separation.
FS-02	Robust fuse/lock policy	OTP profile validated per component family, post-programming verification, per-batch evidence.
FS-03	Strong debug authentication	Challenge-response/certificates, default denial for unauthorized identities.
FS-04	Session time control	Short tokens, automatic expiration, operable and verified revocation.
FS-05	Coupling to boot state	Opening conditioned on boot integrity and active version policy.
FS-06	End-to-end logging	Recording of request/authorization/actions/closure, reliable timestamping, SIEM export.
FS-07	Maintenance exception mechanisms	RMA/service-mode procedure with dual approval and automatic relocking.
FS-08	Fault detection and response	Detection of physical/logical anomalies, graduated reaction, return to safe state.
FS-09	Technical-secret governance	Secret individualization, role separation, rotation/revocation of tool identities.

Table 4.5: Required security functions for debug use

4.8 Traceability threats → objectives → functions

The matrix below makes explicit the security-justification chain for the most representative JTAG/SWD attack scenarios.

Scenario	Dominant capabilities	OS objectives	Required FS functions
Firmware dump via test pads	AM-01 + CAP-01 / CAP-02 / CAP-03	OS-01, OS-03, OS-06	FS-01, FS-02, FS-03
Field maintenance-mode abuse	AM-02 + CAP-04 / CAP-05 / CAP-06	OS-03, OS-04, OS-07, OS-08	FS-03, FS-04, FS-06, FS-07, FS-09
Lock-bits bypass by fault injection	AM-04 + CAP-07 / CAP-08	OS-01, OS-02, OS-06	FS-02, FS-08
Backdoor injection through debug session	AM-02 / AM-03 + CAP-02 / CAP-04 / CAP-06	OS-02, OS-05, OS-07	FS-05, FS-06, FS-07
Workshop-secret compromise	AM-03 / AM-05 + CAP-05 / CAP-06 / CAP-08	OS-03, OS-04, OS-07	FS-03, FS-04, FS-06, FS-09
Industrialized multi-batch attack	AM-05 + CAP-03 to CAP-08	OS-01 to OS-08	FS-01 to FS-09

Table 4.6: Threat-objective-function traceability for JTAG/SWD debug

5
Protection mechanisms and locking

The following requirements are not isolated mandates. They must be read as a coherent system, where each control covers a different weakness: physical exposure, logical activation, actor identity, and post-event evidence capability.

5.1 Board-level hardware protections

Board protections are the first defense layer. They do not replace cryptographic mechanisms, but they significantly increase the cost and discretion required for a physical attack.

STIG-JTAG-01 (OS-03, OS-06 → FS-01)	Remove debug connectors on production versions, or enforce an exceptional physical-access mechanism (sequenced connector, dedicated tooling, sealing).
STIG-JTAG-02 (OS-01, OS-06 → FS-01)	Mask, scatter, or bury critical test points, and document an anti-probing strategy proportionate to the threat.
STIG-JTAG-03 (OS-06 → FS-01, FS-08)	Add passive countermeasures (pull resistors, filtering, ESD/TVS diodes, defensive routing) to limit unintended activation and increase instrumentation cost.
STIG-JTAG-04 (OS-07 → FS-01, FS-09)	Strictly separate board variants (development, validation, production) with nomenclature and configuration traceability.

This separation avoids a classic flaw: accidental migration of a "lab" configuration into a production series. In audit, configuration control is as important as technical locking itself.

Operationally, this section answers a simple question: *can an unauthorized actor physically and quickly access an exploitable debug point?* If the answer is yes, later logical layers will have to absorb much stronger attack pressure.

5.2 Locking through fuse bits and lock bits

Locking through fuses and lock bits is the hard core of the "deny by default" strategy. It must be defined very early, then validated on samples representative of real production (batch variation, silicon revisions, supply conditions).

Technical difference OTP vs eFuse. The term OTP designates a *functional property* (programmable only once), whereas *eFuse* designates a frequent *physical implementation* of that property.

- **OTP (logical concept)**: non-volatile area written once, which can be implemented in different ways (eFuse, anti-fuse, dedicated OTP cells, etc.)
- **Flash-based OTP**: on some platforms, the "one-time" effect is enforced by a security controller and a programming sequence driven by embedded logic (internal microcode/-firmware), not by intrinsic physical irreversibility of the memory cell
- **eFuse (silicon implementation)**: array of micro-fuses electrically "blown" to encode an irreversible state, then read at boot or through security registers
- **Lock bits**: policy bits that often use these OTP zones to permanently enable certain restrictions (flash read, debug, lifecycle transitions)

Impact on cyber resistance. From a defensive standpoint, OTP/eFuse provides strong resistance to remote software reconfiguration, but does not constitute absolute tamper-proofing against an advanced physical attacker.

- **Strength**: hardware irreversibility, strongly reducing accidental or opportunistic debug reopening risk in production
- **Flash OTP limitation**: when protection relies on an embedded controller (not a physical fuse), the sequence can be more sensitive to fault attacks (glitch/EMFI) on control logic
- **Hardware eFuse property**: internal programming/readback mechanism is physical and does not rely on the same software control path, reducing that specific risk (without removing other physical vectors at system level)
- **STIG consequence**: combine OTP/eFuse with strong authentication, logging, post-programming checks, and robustness tests in realistic conditions

STIG-JTAG-05 (OS-03, OS-06 → FS-02)	Define an OTP locking profile per component family and validate it on a test bench before industrialization.
STIG-JTAG-06 (OS-01, OS-02, OS-06 → FS-02)	Apply the highest protection levels compatible with maintenance and certification requirements.
STIG-JTAG-07 (OS-07 → FS-02, FS-06)	Verify effective security-bit state after programming and archive per-batch evidence (factory traceability).

Mechanisms vary by vendor (RDP, secure debug mailbox, life-cycle states), but the principle remains identical: reduce active debug surface to strict necessity and explicitly control state transitions [26, 27, 20, 13, 29].

Comparative example: some ST ranges can enforce very strict production debug locking, while some NXP ranges enable a conditionally open model through challenge-response. Likewise, on TI families, JTAG access may depend on a password scheme and component security state, while on some Microchip families lock-bit granularity differs from one series to another. In audit, this implies checking documentation for the exact reference (part number), not reasoning only at brand level.

A good practice is to document a "product state / debug state" matrix: for each lifecycle phase, specify what is allowed, who can allow it, and what evidence is retained.

This approach helps audit teams distinguish expected behavior (for example, controlled temporary opening in workshop) from unauthorized drift (for example, persistent active debug state after maintenance).

5.3 Authentication and access control

When debug must remain possible for maintenance, the objective is to turn a technical capability into a governed capability: strong identity, explicit authorization, minimal scope, limited duration, and full traceability.

STIG-JTAG-08 (OS-03 → FS-03)	Implement strong debugger-to-target authentication (challenge-response, certificates, asymmetric keys).
STIG-JTAG-09 (OS-04 → FS-04, FS-07)	Time-limit debug sessions (short tokens, automatic expiration, revocation).
STIG-JTAG-10 (OS-05 → FS-05)	Couple debug authorization to boot integrity level (valid Secure Boot, compliant anti-rollback state).
STIG-JTAG-11 (OS-07 → FS-06)	Log debug events (request, authorization, operation, closure), with reliable timestamping and SIEM export.

Without this governance layer, a "secure" debug mechanism remains difficult to audit and to use in incident response. Logging requirement is therefore both a security and operational requirement.

In OT environments, this governance must remain compatible with field constraints: short intervention windows, sometimes limited connectivity, and need for rapid recovery. Selected mechanisms must therefore be robust *and* operable.

5.4 Advanced solutions

For high-criticality environments (SL-3/SL-4), the following additional measures are recommended:

- Secure debug via encrypted tunnel or dedicated authentication protocol
- Restrict debug to non-sensitive memory regions (TEE/TrustZone partitioning, PMP)
- Physical intrusion detection with graduated response (alert, blocking, conditional wipe)
- Hardware root-of-trust chain (HSM/TPM/SE) for maintenance-secret management

These advanced measures are especially relevant when the targeted attacker has significant hardware means (lab, fault-injection tooling, reverse expertise). They are not reserved to regulated sectors, but become essential as soon as business impact of compromise is major.

It is recommended to deploy them progressively, following a maturity logic: first baseline governance and production locks, then strong debug authentication, then advanced detection/response functions.

6
Limits of existing protection mechanisms

6.1 Bypass of fuses and lock bits

OTP locks strongly reduce risk but are not absolute proof of tamper-proofing. Glitch attacks (voltage/clock), transient faults, or implementation flaws can open bypass windows on some component generations [5, 17, 25].

Consequence for the STIG: fuses must be complemented with detection, impact-limitation, and operational-supervision controls.

6.2 Architecture flaws

Recurring causes observed in audits are:

- Debug interface activatable without hardware authentication
- Permanent maintenance mode due to absence of terminal lifecycle state
- Undocumented fallback paths retained in production
- Misalignment between boot security and debug security

These situations often result from organizational decoupling: hardware, firmware, and operations teams apply different threat assumptions. The STIG should precisely serve as common language to avoid these divergences.

These defects are system-architecture issues, not a simple local misconfiguration.

6.3 Limits of obfuscation-only approaches

Physical obfuscation alone is not sufficient. Signal-discovery tools, routing analysis, and reverse-engineering methods can reconstruct debug interfaces at decreasing cost.

Use of AI models to assist PCB reverse engineering (pad classification, netlist hypotheses) should be considered an acceleration factor for attackers, not a distant hypothesis.

6.4 Constraints of protection mechanisms

Implementing secure debug introduces practical constraints:

- Protection and renewal of access secrets
- Per-device key individualization at large scale
- CPU/memory cost of embedded cryptographic protocols
- Power consumption and thermal impact in edge context
- Compatibility with real-time and safety requirements

A common error is to treat these constraints too late, during industrialization. It is better to integrate them from architecture choices to avoid weak last-minute compromises (shared secrets, excessive token durations, incomplete logging).

The STIG recommends arbitrating these constraints during architecture phase, with explicit modeling of residual risks.

6.5 Industrial operational constraints

In OT, total debug prohibition can be unrealistic for field maintenance. Good practice is to organize **exception debug**, strictly controlled:

- temporary activation,
- dual approval,
- logged session,
- automatic revocation and relocking,
- archived intervention evidence

This model addresses operational need without normalizing permanent access. It strongly reduces risk of insider abuse, maintenance error, and compromise of technical accounts.

7

State of the art of attacks and tooling

7.1 Research and exploitation tools

Available open-source and commercial tools lower the entry barrier:

- **OpenOCD** and compatible probes for memory-access scripting and execution control [21]
- **JTAGulator** for automatic identification of interfaces and pinouts [7]
- Vendor debug suites (ST-LINK, Lauterbach, Segger, etc.) often very powerful
- Firmware fuzzing and binary-analysis tools combinable with debug-obtained dumps [6, 24]
- Fault-injection (glitching) platforms to test real lock robustness [17]

From the defender viewpoint, these same tools must be integrated into product-security testing strategy. The objective is not only to verify that a protection bit is enabled, but to demonstrate that an equipped attacker cannot obtain useful access in realistic field conditions.

7.2 Recent academic publications

Useful literature for an embedded OT program notably includes:

- analyses of embedded competitions (MITRE eCTF) highlighting frequent design gaps, notably in secret management and maintenance modes [15]
- case studies on firmware extraction and debug-protection bypass, useful to understand attack paths actually used [14]
- survey work on embedded-object security and physical attacks, helping prioritize countermeasures by threat level [30, 28]
- technical-conference feedback (Black Hat, DEF CON, Hardwear.io, REcon), valuable to track emerging techniques and feasibility evidence [3, 4, 8, 23]

For STIG use, these sources must be read through an operational lens: which attack assumptions are relevant to your OT context, which measures are testable, and which evidence is auditable.

7.3 Documented attack scenarios

The following scenarios are representative of critical OT products:

- **Connected medical**: firmware extraction to understand protocol and prepare fleet attack
- **Automotive**: recovery of secrets in peripheral ECUs then pivot toward trust functions
- **Industrial IoT**: equipment cloning and insertion of modified firmware into maintenance chain
- **Readout-protection bypass**: attempts combining debug access, transient faults, and configuration errors

Each scenario generally combines three dimensions: physical access, governance weakness, and exploitable technical flaw. Effective treatment therefore requires a cross-response spanning architecture, process, and supervision.

For product teams, these scenarios should be replayed as internal validation exercises. The objective is not to reproduce the full sophistication of an offensive lab, but to verify that key controls resist credible, repeatable, and documented attempts.

These scenarios justify a default *deny-debug* posture, with contractualized exceptions.

8

Security best practices for industrial equipment

This chapter provides an arbitration framework. In practice, OT teams must reconcile availability, maintainability, and cyber requirements. A good security practice reduces risk without blocking normal system operation.

8.1 Alignment with industrial standards

Security framing should align with:

- IEC 62443-4-1 for secure development practices [9]
- IEC 62443-4-2 for component technical requirements [10]
- NIST SP 800-193 for firmware platform resilience [18]
- institutional guidance (ANSSI, NIST) on embedded-security hygiene [1, 19]

For constrained environments, the recommended minimum target is SL-2 with trajectory toward SL-3 for critical assets.

8.2 Defense-in-depth strategies

A robust strategy combines at minimum:

- production hardware locking,
- Secure Boot and anti-rollback,
- strong authentication of maintenance operations,
- OT network segmentation,
- SOC/CSIRT supervision of debug events

The value of this approach comes from layer complementarity: if one measure fails (example: lock-bit configuration error), other layers should still prevent the attack or at least enable rapid detection.

This controlled redundancy is especially important in multi-site programs, where production and maintenance practices may vary from one factory to another.

No single measure is sufficient. Risk level is a function of overall architecture consistency and operational execution quality.

8.3 Managing the security/maintainability trade-off

The trade-off is organized around an industrialized process:

- secure recovery mode, activatable with justification
- approved and timestamped debug-access requests
- session limited to an explicit technical scope
- automatic closure and post-intervention verification
- retention of evidence (log, firmware hash, action report)

This process must be tested regularly, like a crisis exercise: access lead time, quality of produced evidence, revocation capability, and return to a verifiable locked state.

In a continuous-improvement logic, incidents and near-incidents related to debug should feed a formalized feedback loop: procedure updates, hardening of authorization thresholds, and tooling adaptation.

This approach preserves maintainability without trivializing debug privilege.

8.4 Recommendations by industrial sector

Energy and utilities. Prioritize permanent locking, with an offline exception process and reinforced control (SL-3/SL-4 target).

Automotive. Strictly separate workshop diagnostic channels and customer functions; apply rigorous tool-identity management.

Medical. Ensure regulatory compliance (MDR/FDA), auditability, and firmware-change control.

Manufacturing. Protect process intellectual property and prevent clones through secret individualization and integrity attestations.

A

Comparative table of vendor JTAG implementations

Vendor	Main mechanism	Debug-control level	STIG attention points
STMicro	RDP, lifecycle states, debug authentication on recent families	From read blocking to authenticated conditional access	Verify factory configuration, boot/debug consistency, and exception unlock procedure [26, 27]
NXP	Secure debug mailbox, lifecycle, provisioning keys	Controlled access through authentication protocol	Protect provisioning chain and log authorizations [20]
Microchip	Memory-protection bits, lock/debug modes, variation by family	From simple locking to secure maintenance modes	Example: some families provide strict memory lock, others finer service modes; test by exact reference [13]
Texas Instruments	JTAG password protection, security states, and component-specific options	Conditional access depending on configuration and device	Example: depending on family, JTAG opening may be tied to password/security state; validate under real conditions [29]

Table A.1: Comparative synthesis of JTAG/SWD mechanisms by vendor

B
Normative and regulatory references

The following texts should be considered the documentary basis of this STIG:

- IEC 62443-4-1: secure component development process
- IEC 62443-4-2: technical security requirements for IACS components
- ISO/SAE 21434: automotive cybersecurity (relevant for mobile/transport equipment)
- NIST SP 800-193: platform resilience and firmware protection
- Complementary references: NISTIR 8259 and ANSSI recommendations on IoT/embedded security

These references must be translated into verifiable internal requirements (specification, testing, audit, operations).

C

Implementation checklist by lifecycle phase

The following checklist serves as an internal qualification baseline. Each item should be marked *compliant / non-compliant / not applicable* and linked to evidence.

For "expert-audit level" use, this checklist should be coupled with depth criteria: evidence quality, test reproducibility, verification independence, and ability to detect configuration drifts across batches.

Design phase

- Debug connectors absent from production version or protected by documented physical mechanism
- Minimal-exposure strategy for test pads (routing, masking, tooling access)
- Selection of components supporting secure debug and robust lifecycle states
- Secure boot chain with trust anchor and anti-rollback

This phase determines most of the achievable security level. A poor component or debug-architecture choice is difficult to recover at end of cycle.

Production phase

- Fuse/lock programming according to validated and reviewed profile
- Automated, traceable post-programming verification per batch
- Secure management of factory secrets (HSM, role separation, audit)
- Quality control including unauthorized memory-read attempt

Real robustness is decided here: a well-designed but poorly industrialized protection becomes a systemic risk across the whole series.

Deployment phase

- Integrity control of received equipment and their lock state
- Supervision of maintenance and debug events
- Revocation procedure for tool and technician identities

- Incident-escalation policy in case of suspected physical compromise

Deployment must include a continuous-monitoring logic. Absence of debug events is not proof of security if no reliable telemetry is collected.

Maintenance phase

- Exception debug only on approved and timestamped ticket
- Temporary, individual access tokens with minimal scope
- Full session logging and evidence retention
- Automatic relocking and state verification at end of intervention

This phase is the most exposed to human error. Procedures must be simple enough for field conditions, but strict enough to remain auditable.

JTAG security-validation test procedure (per batch)

- Test 1: debug-access attempt without authentication (expected failure)
- Test 2: protected-memory read attempt (expected failure)
- Test 3: robustness test against state errors (reboot, brown-out, reset)
- Test 4: authorized maintenance session, full traceability, and clean closure
- Test 5: post-maintenance verification of lock state and firmware integrity

Results should be historized per batch, hardware version, and firmware version to quickly identify any security regression.

C.1 Expert audit matrix for requirements STIG-JTAG-01 to STIG-JTAG-11

The following matrix formalizes, for each requirement, the triplet *compliance criterion / test method / expected evidence*. It can be used as-is in an audit-plan appendix.

This matrix can be enriched with internal quantitative thresholds (acceptable failure rate, maximum revocation delay, verification coverage rate per batch) to drive maturity over multiple quarters.

Requirement	Compliance criterion	Test method	Expected evidence	Frequency	Severity
STIG-JTAG-01	No exploitable debug connector in production or documented exceptional physical access	Physical inspection, BOM/PCB review, access test with standard tooling	Sample photos, drawing review, access-attempt report	At each HW revision	Critical
STIG-JTAG-02	Sensitive test points not accessible without specific disassembly/instrumentation	Layout review, visual inspection, pin-identification trial	PCB review report, signed anti-probing checklist	At each HW revision	High
STIG-JTAG-03	Passive countermeasures present and compliant with reference design	Schematic check, spot electrical measurement, debug-activation stability test	Approved schematics, measurement records, validation report	Initial batch + changes	High
STIG-JTAG-04	DEV/VAL/PROD variants clearly separated and traceable	Configuration audit, nomenclature and marking verification	Configuration register, manufacturing traceability	Monthly + each release	Critical
STIG-JTAG-05	Fuse/lock profile defined and validated on component family	Execution of security-state read script on test bench	Signed automated results, configuration baseline	Each batch	Critical
STIG-JTAG-06	Maximum protection level applied according to approved policy	Policy vs actual component-state review	Compliance report + validated waivers	Each batch	Critical
STIG-JTAG-07	Post-programming verification archived	End-of-line control + random sampling	Timestamped factory logs, sampling evidence	Each batch	Critical
STIG-JTAG-08	Strong authentication required for any debug opening	Access test with valid/invalid identities, replay test	Authentication traces, rejection logs	Quarterly + releases	Critical
STIG-JTAG-09	Debug-session duration and scope limited and revocable	Token-expiration test, constrained-time revocation test	Expiration logs, revocation proof	Quarterly	High
STIG-JTAG-10	Debug authorization conditioned on boot integrity	Test on valid/invalid/rollback firmware	Access-decision logs, hashes, and boot states	Each firmware release	Critical
STIG-JTAG-11	Complete, SOC/SIEM-usable logging	Log review, correlation test, incident simulation	SIEM extracts, correlation rules, end-to-end audit trail	Monthly + exercises	High

Table C.1: Expert audit matrix for requirements STIG-JTAG-01 to STIG-JTAG-11

D
Reference resources and tools

The resources below should be viewed as a working baseline, not an exhaustive list. The challenge is to maintain a continuous capability to evaluate attacks and countermeasures, adapted to the pace of product evolution.

Useful open-source tools for defense and audit

- OpenOCD (JTAG/SWD control and test scripting)
- JTAGulator (debug-pin identification)
- GDBFuzz (assisted firmware fuzzing)
- Binwalk and binary-analysis chains for post-extraction inspection

These tools make it possible to reproduce attack chains in a controlled way and objectively measure product robustness.

Commercial tools frequently encountered

- High-end professional debuggers (Lauterbach, Segger, etc.)
- Fault-injection test platforms and hardware-evaluation systems
- Key-management solutions and HSM for secure industrialization

Their main value is analysis depth and test-campaign automation, particularly useful for high-volume programs or programs with high certification requirements.

Communities and conferences to follow

- Black Hat, DEF CON, Hardwear.io, REcon
- NIST, ANSSI, ENISA publications and industrial feedback
- eCTF-type competitions to observe attack/defense trends

This corpus helps keep the STIG living and aligned with the technical state of the art.

Bibliography

[1] ANSSI. Recommandations de sécurité relatives à l'iot. Guide ANSSI, 2020.

[2] Arm Ltd. *ARM Debug Interface Architecture Specification (ADIv5/ADIv6)*, 2023.

[3] Black Hat. Black hat conference archives. Conference archives, 2026.

[4] DEF CON. Def con talks and archives. Conference archives, 2026.

[5] Jean-Max Dutertre et al. *Fault Injection Attacks and Countermeasures*. Springer, 2021.

[6] GDBFuzz Contributors. Gdbfuzz. Projet open-source, 2026.

[7] Joe Grand. Jtagulator: Assisted discovery of on-chip debug interfaces. Projet open-source, 2026.

[8] Hardwear.io. Hardwear.io conference archive. Conference archives, 2026.

[9] IEC. Security for industrial automation and control systems – part 4-1: Secure product development lifecycle requirements (iec 62443-4-1). Norme IEC, 2018.

[10] IEC. Security for industrial automation and control systems – part 4-2: Technical security requirements for iacs components (iec 62443-4-2). Norme IEC, 2019.

[11] IEEE. Ieee standard for test access port and boundary-scan architecture (ieee 1149.1). Standard IEEE, 2013.

[12] ISO/SAE. Road vehicles – cybersecurity engineering (iso/sae 21434). Norme ISO/SAE, 2021.

[13] Microchip Technology. *Microchip MCU Security and Debug Access Control Documentation*, 2026.

[14] A. Miller et al. Practical firmware extraction from secure iot devices. *Proceedings of Embedded Security Workshop*, 2021.

[15] MITRE. Embedded capture the flag (ectf). Competition and technical resources, 2026.

[16] MITRE. Mitre att&ck for ics. Knowledge base, 2026.

[17] NewAE Technology. Chipwhisperer platform. Open hardware / open-source platform, 2026.

[18] NIST. Platform firmware resiliency guidelines. Technical Report SP 800-193, National Institute of Standards and Technology, 2018.

[19] NIST. Foundational cybersecurity activities for iot device manufacturers. Technical Report NISTIR 8259, National Institute of Standards and Technology, 2020.

[20] NXP Semiconductors. *NXP Secure Debug and Device Lifecycle Management Documentation*, 2026.

[21] OpenOCD Project. Open on-chip debugger. Projet open-source, 2026.

[22] OWASP. Firmware security testing guide. OWASP Project, 2024.

[23] REcon. Recon conference archive. Conference archives, 2026.

[24] ReFirmLabs. Binwalk firmware analysis tool. Projet open-source, 2026.

[25] Riscure. Fault injection knowledge base and training material. Industry resources, 2026.

[26] STMicroelectronics. *STM32 Reference Manuals and Readout Protection (RDP) documentation*, 2026.

[27] STMicroelectronics. *STM32H5 Debug Authentication Application Notes*, 2026.

[28] Mark Tehranipoor and Cliff Wang. *Introduction to Hardware Security and Trust*. Springer, 2011.

[29] Texas Instruments. *Texas Instruments Device Security and JTAG Lock Documentation*, 2026.

[30] X. Zhou et al. Security analysis of embedded devices: Threats, attacks, and defenses. *Sensors*, 2023.

License