

Assume Capture. Engineer Resilience.

A Security Guide for Autonomous Systems



Summary

Autonomous and semi-autonomous defence systems increasingly carry critical data, software, algorithms and decision-making capabilities directly on deployed platforms. When those systems operate in hostile environments, physical loss or capture has to be considered alongside remote compromise as part of the security model.

The challenge is not simply whether protections such as encryption, secure boot, readout protection or zeroization are present. It is whether those protections still hold when an adversary has physical access to the device and can deliberately challenge the assumptions behind them. Side-channel analysis, fault injection and reverse engineering can expose weaknesses that are not visible through conventional remote security testing alone.

A captured platform can expose far more than cryptographic keys. Firmware, communication protocols, mission configuration, proprietary algorithms, sensor-processing logic, AI models and operational know-how may all become part of the attack surface. Because hardware and software are closely connected, an attack can move from one layer to another: physical manipulation may expose firmware, while software analysis may reveal where a physical attack should be focused.

This white paper proposes a practical capture-aware approach to security assessment: identify the assets that matter, map where they are stored and processed, understand the assumptions behind their protection, test those assumptions under adversarial conditions, and feed the results back into engineering.

The objective is not to claim that a system is impossible to attack. It is to build evidence about which protections have been tested, against which realistic techniques, with what results and residual risk, then use that evidence to strengthen the system throughout its lifecycle.

The threat model is changing

Autonomous and semi-autonomous systems are increasingly operating at the edge, carrying onboard the computing, software and data needed for navigation, communications, sensing and decision-making. In defence environments, those systems may also be lost, intercepted, jammed or recovered. Physical possession of the platform therefore needs to be considered as part of the threat model, not as an exceptional scenario.

This raises a practical set of questions. What information and capabilities remain on the platform if it is captured? Where are those assets stored and processed? Which mechanisms protect them? And what happens when those mechanisms are tested under conditions chosen by an adversary rather than by the product specification?

Addressing physical capture requires looking across hardware and software, from the assets carried by the platform to the assumptions behind their protection. The goal is to understand which attack paths are realistic, which protections need to be challenged, and where validation should begin.

Start with your own system

Before going further, open the Captured Device Threat-Model Checklist and begin with your own platform. Define what capture could look like, identify the assets you would most want to protect, and note where the main unknowns are. As you move through the following sections, use the concepts and examples to refine that first assessment and identify which attack paths deserve deeper validation.

[Start your threat-model checklist.](#)

Assume capture

Editorial contribution by Ido Govrin

For autonomous defence systems, physical possession should be considered from the beginning. Drones crash, are intercepted, jammed or forced to land. Components can be recovered from battlefields, while commercial and dual-use platforms can also be acquired and studied. The relevant cybersecurity question is therefore not only whether an adversary can compromise the platform remotely, but what they can learn once they have unrestricted physical access to it.

A captured system may contain far more than a single secret key. Depending on the platform, it can expose cached maps, flight logs, waypoints, communication settings, cryptographic keys, credentials, firmware, proprietary protocols, algorithms, sensor-processing logic, AI models and mission configuration. Taken together, these elements can reveal how the system communicates, authenticates, updates, senses and makes decisions.

This is why the starting point for a capture-aware threat model should be the assets themselves. Rather than beginning with a catalogue of attack techniques, engineering teams can ask a simpler question:

If an adversary obtained one of our systems tomorrow, what would we most want them not to learn?

From keys to operational know-how

Cryptographic keys and credentials may protect communications, authenticate components, validate software or control access to services. Firmware can reveal boot logic, authentication checks, update paths, hardware configuration and undocumented behaviour. Reverse engineering can expose proprietary communication protocols, command-and-control mechanisms and relationships between the platform and ground-control systems.

Autonomous platforms may also contain proprietary algorithms, sensor-processing logic and AI models representing substantial R&D investment and operational know-how. Their surrounding software can reveal how sensor inputs are processed, how classifications or decisions are made and how the system responds to particular conditions. As intelligence moves to the edge, intellectual property and mission-relevant know-how move with it.

When protective mechanisms do not trigger as expected

Physical capture also raises questions about mechanisms designed to protect these assets after a platform is lost.

Automated key erasure, or zeroization, is an important line of defence, but it

depends on the system correctly recognising that a compromise is taking place. Loss of communication may leave operators unable to determine whether the platform is temporarily out of range or being actively compromised. Recovery logic may interpret signal loss or interference as a normal operational condition and execute Return-to-Home or an emergency landing instead.

Physical damage can interrupt the process before erasure is completed, while manipulated sensor or telemetry data may prevent the platform from recognising that it has been compromised. If the event is not detected, zeroization may never trigger, leaving sensitive information intact in memory.

The practical objective is therefore not to assume that every possible physical attack can be prevented. It is to identify what must remain protected even after the platform is lost, determine where those assets exist across hardware and software, and understand the mechanisms intended to protect them.

That asset map provides the foundation for the next question: **do those protections still hold when the implementation itself is challenged?**

From security mechanisms to implementation security

Security architecture often relies on established mechanisms such as encryption, authentication, secure boot, signed firmware and memory readout protection. These mechanisms are essential, but their presence alone does not demonstrate that the complete implementation will resist an attacker with physical access.

The distinction lies in implementation security: whether the hardware and software continue to behave securely when the attacker deliberately moves outside normal operating conditions.

A cryptographic algorithm, for example, may be mathematically secure while its implementation produces observable variations in power consumption, electromagnetic emissions, timing or other signals. Side-channel analysis uses these physical effects to infer information that the algorithm was designed to keep secret. The attack does not necessarily break the mathematics. It exploits the physical behaviour of the implementation.

This changes the validation question from **“Are we using a strong cryptographic algorithm?”** to **“Can this implementation use the secret without leaking exploitable information?”**

The same principle applies to mechanisms such as secure boot and firmware protection. These protections depend on a sequence of processor operations being executed correctly. Fault injection deliberately challenges that assumption. Voltage or clock glitches, electromagnetic pulses and related techniques can cause operations to be skipped, corrupted or executed differently. Depending on the target, this can contribute to bypassing security checks or exposing firmware that was intended to remain inaccessible.

The objective of testing is not to assume that every protection will fail. It is to determine the conditions under which the security claim remains true.

The security property belongs to the final system

A component may provide security functions, but the final security property emerges from the interaction between silicon, board design, firmware, cryptographic implementation and system architecture.

This is why component-level protections still need to be evaluated in the context of the system in which they are used. eShard security research has included cryptographic-key recovery on a UAV through side-channel analysis, as well as reproduction of an STM32 readout-protection bypass using fault injection.

Both illustrate the difference between the presence of a protection and the behaviour of its implementation under adversarial conditions.

For engineering teams, this means identifying the assumptions behind critical security mechanisms and testing those assumptions directly. Does a cryptographic implementation resist leakage? Does secure boot still enforce the intended policy when execution is disturbed? Does readout protection remain effective against realistic physical attack techniques?

These questions turn security requirements into evidence. They also shift the focus from verifying that a mechanism exists to understanding whether it continues to provide the intended protection when the implementation itself becomes the target.

One system, one attack surface

Security teams often separate responsibilities across hardware security, firmware security, application security and reverse engineering. An attacker examining a captured embedded system has no reason to preserve those boundaries.

A hardware attack may expose firmware. Firmware analysis may identify a security decision worth attacking physically. A fault-injection experiment may reveal unexpected software behaviour that then requires deeper reverse engineering.

Hardware can become the route into software

Physical access creates opportunities that are not available through remote testing alone. Interfaces can be probed, electrical behaviour can be observed and execution can be manipulated.

If a protection mechanism can be bypassed, previously inaccessible firmware or configuration may become available for analysis. At that point, the investigation moves directly from hardware into software.

Software can guide the physical attack

The relationship also works in the opposite direction.

Reverse engineering can reveal cryptographic implementations, authentication logic, secure-boot decisions, update mechanisms, proprietary protocols and anti-tamper protections. This information can help identify which operations, data or processor states are worth observing through side channels or disturbing through fault injection.

For sophisticated embedded systems, side-channel analysis, fault injection and software reverse engineering are therefore not independent disciplines. They are different perspectives on the same target.

Follow the complete attack path

A useful assessment follows the evidence across layers rather than stopping when one technical domain ends. The objective is to understand how far a capable adversary can actually get with the device in their hands.

This full-system approach also reflects how eShard combines hardware and software security analysis: the investigation follows the attack path rather than treating each technical domain independently.

Understanding that complete path also helps determine where remediation belongs. The most effective fix may sit in the cryptographic implementation, firmware logic, hardware configuration, system architecture or a combination of these layers. This becomes particularly important when changes to silicon or board design are costly.

AI at the edge extends the security perimeter

Editorial contribution by Ido Govrin

Autonomous and semi-autonomous defence systems increasingly use AI directly on embedded platforms, close to sensors and mission activity. As more processing and decision-making move onto the platform, valuable information and know-how move with them. AI models, parameters, preprocessing logic and surrounding software can represent substantial R&D investment and operational knowledge.

Protecting the communication channel does not fully address this exposure. If the computing platform itself is recovered, an adversary may seek to understand how sensor inputs are processed, what features matter to the model, how decisions are made, how the system responds to particular conditions or how proprietary algorithms are implemented.

AI depends on the security of the platform

AI does not sit outside the rest of the system. Models and related software depend on the same firmware, credentials, secure-boot mechanisms and interfaces that support the platform as a whole.

If hardware security is bypassed, firmware and model-related information may become more accessible. If software is reverse engineered, it may reveal where models are stored, how they are loaded and which protections are applied. Protecting AI on the platform is therefore closely connected to embedded implementation security.

The model may not be the crown jewel

The AI model itself is not necessarily the most valuable information on the device.

Engineering teams may need to consider model files, parameters, proprietary preprocessing, sensor fusion, decision logic, configuration and training-derived know-how. Decision logic can be particularly important because it may represent not only engineering investment, but also operational knowledge.

At the same time, many deployed platforms do not carry highly sophisticated decision logic. Other information stored locally, such as reference libraries, mission parameters and configuration, may be equally or more important. These elements may be present precisely because the system needs to operate without a permanent connection to ground control.

This means AI should not be treated as an isolated security problem. The relevant question remains the same as for the rest of the platform: what information matters,

where is it stored and processed, how does it move through the system, and which mechanisms are intended to protect it?

As more intelligence moves from centralized infrastructure to the edge, more valuable information is physically deployed into potentially hostile environments. AI models, decision logic, mission data and the software around them therefore need to be included in the same capture-aware threat model as keys, firmware and communications.

From assessment to resilience

Physical security testing is most useful when it becomes part of engineering rather than a one-time exercise immediately before deployment. Weaknesses that involve the interaction between silicon, board design, firmware and cryptographic implementation can be much harder to correct once the hardware architecture is frozen and production has started.

Where possible, teams can evaluate cryptographic implementations, firmware protections and attack hypotheses during development, then repeat relevant testing on the final hardware. Early testing does not replace validation of the production platform. It gives engineering teams more time and more design options when weaknesses are found.

An iterative security cycle

A practical lifecycle can be expressed through five activities:

Assess → Attack → Understand → Harden → Retest

Assess the assets, architecture and threat model. Attack the implementation under realistic adversarial conditions. Understand the complete path across hardware and software. Harden the system at the most appropriate layer. Retest to verify that the mitigation changes the security outcome.

This cycle can be repeated as hardware, firmware, cryptography, AI functionality and mission requirements evolve.

Build evidence, not assumptions

The purpose of implementation security is not to claim that a device is impossible to attack. It is to understand which protections have been tested, against which realistic techniques, with what results and what residual risk remains.

That evidence is particularly important for systems expected to operate in hostile environments where physical capture is credible.

As autonomy increases, AI moves onto embedded platforms and systems are deployed at scale, cybersecurity engineering needs to evolve with them. The question is no longer only whether someone can remotely compromise a platform, but also what happens after they capture one.

Ready to assess your own system?

Use our **Captured Device Threat-Model Checklist** to turn the principles in this white paper into a first set of capture scenarios, attack hypotheses and validation priorities.

Download the checklist [↓](#)

ESHARD | THREAT-MODEL CHECKLIST

Captured Device Threat-Model Checklist

A practical framework for understanding what an adversary could access, extract or learn after physically capturing your autonomous system. Use it with your architecture, hardware, firmware/software and security teams.

If an adversary obtained one of our systems tomorrow, what would we most want them not to learn?

0 / 54 checked

Print

Reset

1. Define the capture scenario

Agree what 'captured' means. The attacker's opportunities depend on what was recovered, its condition and the assumed level of access.

WHAT WAS CAPTURED?

☐ Complete autonomous platform

☐ Subsystem / payload

☐ PCB or electronic module

☐ Removable storage

☐ Ground-control equipment or associated software

Other:

HOW DID CAPTURE OCCUR?

☐ Crash / battlefield recovery

☐ Interception

☐ Jamming / forced landing

☐ Theft or loss

☐ Commercial / dual-use acquisition

Other:

ATTACKER ASSUMPTIONS

☐ Device still boots or can be repaired

☐ Repeated reset and manipulation are possible

☐ Laboratory access is available

Assume Capture. Engineer Resilience.

Autonomous defence systems are carrying more computing power, intelligence and sensitive information into environments where physical capture is a credible scenario. This changes what security assessment needs to consider. The question is no longer only whether a platform can be compromised remotely, but what an adversary could learn or access once the device itself is available for analysis.

The objective is not to make a device impossible to attack. It is to understand which protections have been tested, against which realistic techniques, with what results and what residual risk remains.

That requires looking beyond individual security mechanisms and considering the complete system: the assets it carries, how they move across hardware and software, the assumptions behind their protection, and what happens when those assumptions are deliberately challenged.

Physical security assessment is most valuable when this evidence feeds back into engineering. As hardware, firmware, cryptography, AI functionality and mission requirements evolve, the same cycle can be repeated: **Assess, Attack, Understand, Harden and Retest.**

Authors



Guillaume Ballu

Defence & Aerospace Sales Manager, eShard

Following studies in business school and international relations, Guillaume's career has been marked by various experiences in the defense and cybersecurity sectors; these enable him to work today at eShard, focusing on embedded systems cybersecurity for the defense and aerospace industries.

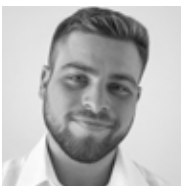


Guillaume Vilcocq

COO, eShard

With more than two decades of experience in embedded systems and semiconductors, Guillaume has worked in hands-on technical roles as well as in product strategy and commercialization. This background gives him a practical understanding of how complex technologies move from engineering and integration to real-world deployment.

Editorial Contributor



Ido Govrin

Program Coordinator, Israel Innovation Institute

Ido Govrin is a Program Coordinator at the Israel Innovation Institute, responsible for InnovaCyberBridge, a cybersecurity startup program run in partnership with the Institute. He served as a UAV systems operator, giving him firsthand operational insight into drone platforms and how they're used in the field.

