Cycuity

3 Steps to Capturing Effective Hardware Security Requirements

As hardware vulnerabilities continue to rise, it's increasingly crucial for those developing semiconductors to reduce consumer and business risk by establishing comprehensive security programs. These should include a systematic process for developing security requirements, verifying them at scale throughout the design process, and producing final documentation for security sign-off before tapeout.

Proactively Mitigating Risk Before Manufacturing

Organizations should start by developing and documenting security requirements that meet the specific needs of all stakeholders of semiconductor security. This set can include marketing, engineering, product security, legal and compliance, and more. Pre-defined security requirements will be the central driver for your security verification plan and are essential in the final signoff.

Clearly defining the requirements and starting early with your comprehensive hardware security program will help you minimize the cost of remediation and avoid expensive security surprises later.

1 Specify a Threat Model

Since new vulnerabilities are disclosed daily, maintaining a detailed, up-to-date listing of known and unknown attacks and their mitigations is infeasible. A threat model specifies the capabilities of a hypothetical attacker at a high level of abstraction. Developing a high-level threat model will help you create a future-proof security specification and serve as a boundary on the scope and quantity of the security needs.

So, before starting on requirements, answer the following questions:

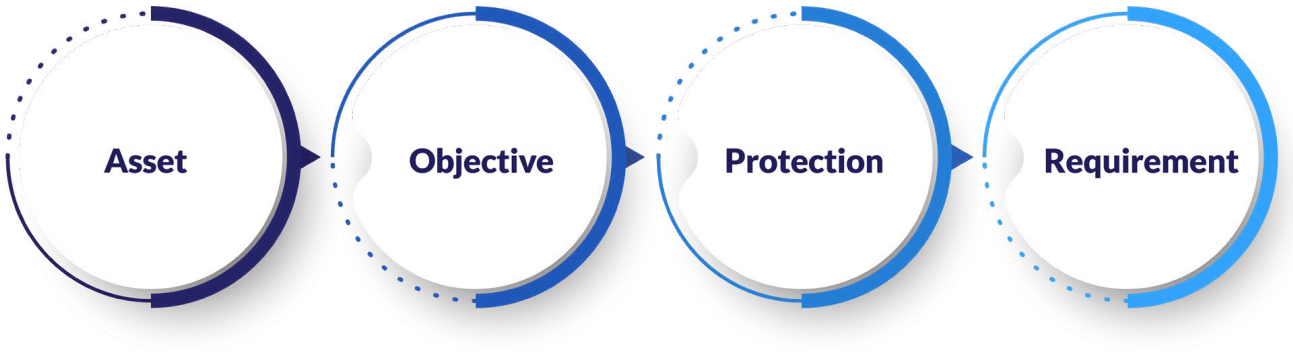
1. Who is the likely attacker, and what are their motivations and resources?

2. What access will an adversary have to the design under test (DUT)?

3. How long will the DUT be actively deployed in the field?

2 Identify the Key Components

Next, to effectively derive security requirements from a specific threat model, isolate each key component so that they can be built systematically from the bottom up.



1. Security-Sensitive Assets

An asset is anything on the device that must be protected to preserve the device's security intent. These are what an adversary would either want to get access to, modify, or make inaccessible during a successful attack.

2. Security Objectives

Security objectives underpin the intent of the protections that must be implemented and verified. They should include of one of the following:

- **Confidentiality:** Specifies that the asset should not be accessed, read, or observed by an adversary either directly or indirectly
- **Integrity:** States that the adversary should not be able to modify the asset
- **Availability:** States that the asset should always be responsive and accessible

At least one security objective must be identified for each asset.

3. Protection Mechanisms

A protection mechanism is a design feature or quality of a device's deployment environment that is intended to preserve the security objectives of an asset. The specified threat model should bound a protection mechanism and prevent it from adding protections against irrelevant threats.

You must pair every asset's security objectives with one or more known protection mechanisms in the microarchitecture. Otherwise, it will be considered trivially vulnerable.

3 Derive The Security Requirement

The final step is to derive the security requirement. A security requirement is a statement about the DUT that must always be true for the associated protection mechanisms to be effective. Each security requirement must be falsifiable, or possible to be shown false with data through security verification.

The Common Weakness Enumerations (CWE™) provides a comprehensive list of over 100 weaknesses that the industry views as the root cause of all known vulnerabilities. When creating security requirements, referencing the CWE can help you define what is in scope and identify gaps in protection.

Examples of Effective and Ineffective Security Requirements

A missing, incorrect, ambiguous, or incomplete security requirement is a security risk, and, depending on the asset, the consequences could be severe.

Examples:

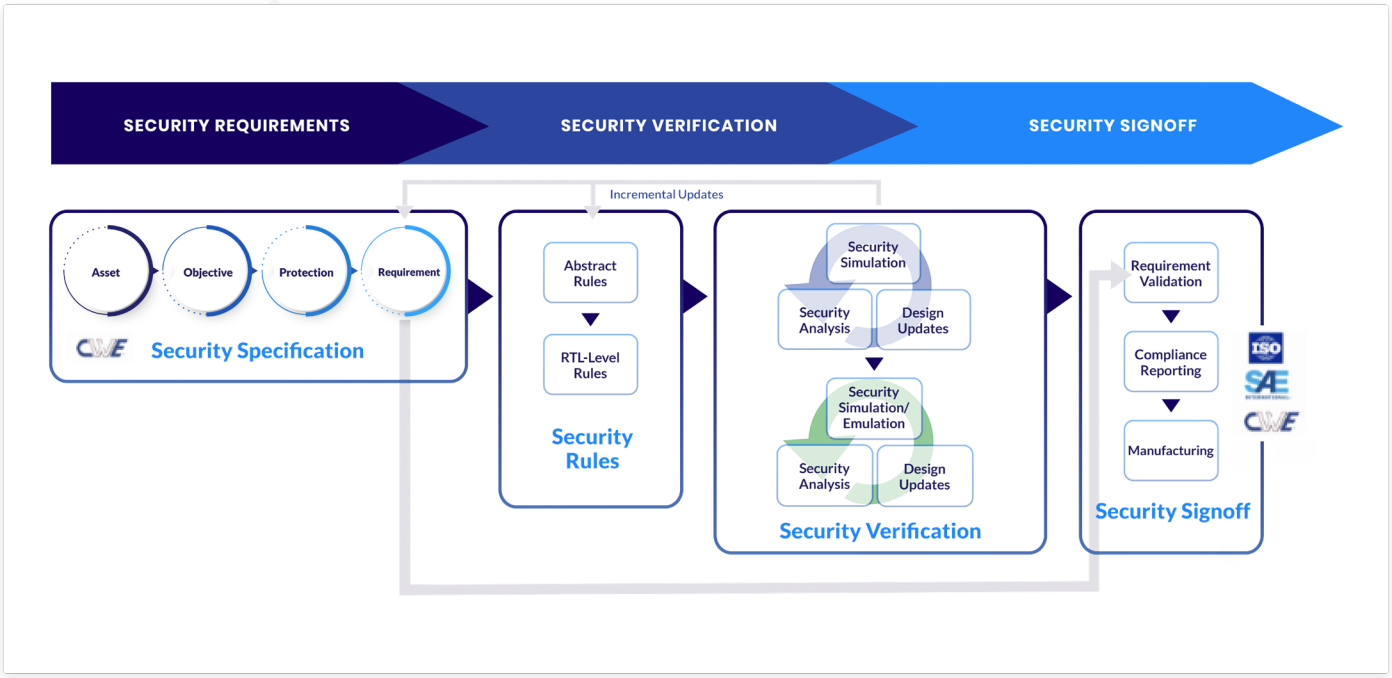
- **Ineffective:** No third-party IP can access system SRAM until secure boot is complete
- **Effective:** It must not be possible for any third-party IP to read or write system SRAM until secure boot is complete.

The ambiguity and imprecise language led to an untestable claim in the first example. The second rule, however, is clear and can be tested when verifying requirements.

Next Step: Security Verification

Next, you must perform security verification using the requirements you created to identify weaknesses in the implementation and validate whether protection mechanisms work in the actual design. Cycuity can help organizations move through this stage efficiently and achieve a successful signoff.

The Cycuity approach compiles the natural language security requirements into Radix security rules. This is performed first on an abstract level where design elements are expressed symbolically. Next, it is performed on the Register Transfer Level (RTL) level where the rule elements are instantiated with the corresponding RTL names.



The Radix rules, combined with the simulation and emulation tests, establish the security verification plan.

This plan is applied at every step of the design process, from block-level to system-level, to detect known and unknown vulnerabilities.

Once your security requirements are ready, Cycuity is here to help you verify your security at scale.