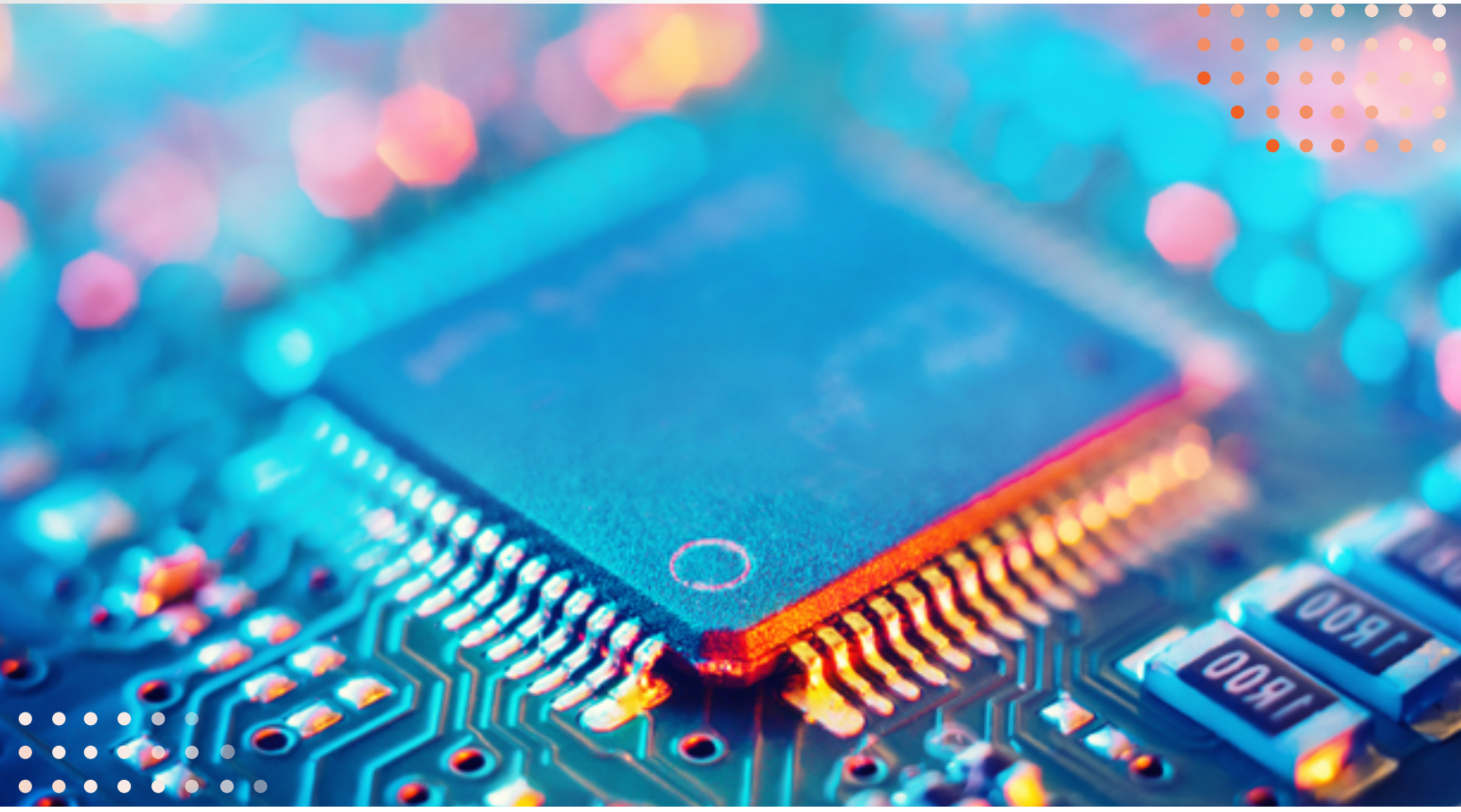
WHITEPAPER

Best Practices to Ensure Firmware Security



Security Compass

When we consider software security, we often tend to think only about cloud and mobile applications. What's often missing in our discussions is the software that exists within hardware components, namely firmware. In today's world, where attacks can occur at the software or hardware layers, we need to extend our traditional software security models to include product firmware.

There are some trends that indicate product security is highly relevant today. We see a proliferation of IoT devices, along with the integration of OT and IT in critical infrastructure or industrial environments, and the impact of software layer breaches traced to product firmware vulnerabilities. All of this points to the need for us to consider the creation of secure products that encompass both software and firmware.

From a guidance perspective, industry standards and frameworks like NIST 800-53 and ISA 62443 have long advocated the need for both secure software and firmware. Even in the most recent revision of NIST 800-53, there is continued emphasis on software, hardware, and firmware. Our security controls, therefore, should extend beyond software into firmware.

From an operational standpoint, within organizations, accountability for security software and firmware rests with two roles. The Chief Information Security Officer (CISO) and the Chief Product Security Officer (CPSO) collaborate together to ensure that security concerns at the software and firmware layers are properly addressed.

In order to prioritize product security concerns appropriately, we need a repeatable process that drives out the requirements and priorities to focus on. This means understanding an attacker's motivations and current attack vectors to determine the likelihood and impact of a compromise.

Collecting all of this information together and flowing it through a risk assessment process that feeds into an overall risk management framework with thresholds provides the necessary information to make an informed decision. By taking a risk-based approach, we can focus on areas that are most relevant to the business and manage software and firmware security risks. In this way, we balance the need for speed and security in line with business priorities

Understanding the motivations of a cyberattacker

Before considering an approach to manage software and firmware security, it's important to take a step back and consider why an attacker might be interested in the firmware layer. According to the Eclypsium 2020 report, there are several motivations of an attacker:

- They want to achieve persistence of their malicious code by bypassing higher levels of operating system and application security. They want to gain the highest levels of privilege and thereby gain access to all parts of the system.
- They want to execute undetected attacks that bypass many of the logging systems that exist on top of the firmware layer.
- They want to damage the device so that it becomes necessary to perform a prolonged reset or look for a physical replacement.

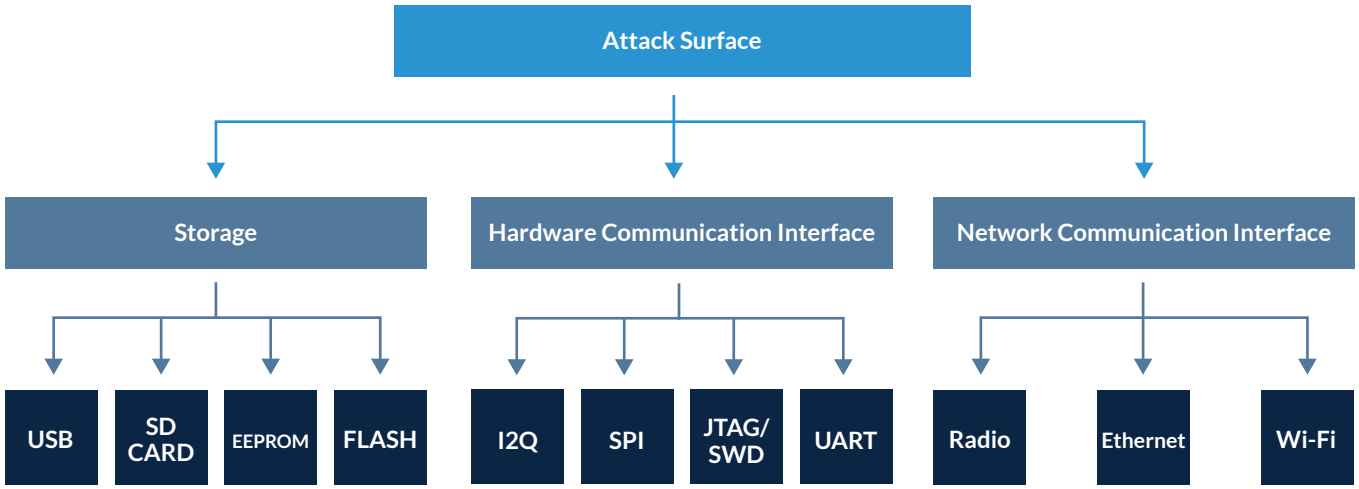
Given these motivations, we can better understand the attack vectors. According to Raju et al in their 2019 paper, Chip off IoT Devices: Attacks and Mitigations, attacks can be targeted at one of these three layers:

- Storage
- Hardware Communications Interface
- Network Communications Interface

As per the Microcontroller Based IoT System Firmware Security: Case Studies by Gao et al, there are two attack vectors:

- **Internal:** There is no authentication or encryption for firmware upgrades.
- **External:** Access to the hardware through common external interfaces like WiFi or Bluetooth low energy (BLE).

Understanding an attacker's motivations and possible attack vectors can help us better understand the scope of the problem and look for proactive ways to manage the corresponding risk.



Source: Chip off IoT Devices: Attacks and Mitigations, 2019

Can the vulnerable firmware layer lead to cyberattacks?

Firmware development is often done using low-level programming languages like C or Assembly. These languages provide a small footprint and enable precise control over the hardware. Unfortunately, this also has its risks.

According to the Building Secure Firmware book by Jiewen Yao et al, "...the C code can be susceptible to the class of attacks that afflict higher-level software. These attacks include memory safety issues, involving the variants of buffer overflow, such as stack overflow, heap overflow, and integer overflow. In addition, control flow attacks against C code in the application or OS space can be repurposed against system firmware."

MITRE has also identified several CWEs related to managing buffers during software development:

- CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
- CWE-121 Stack-based Buffer Overflow
- CWE-122 Heap-based Buffer Overflow
- CWE-124 Buffer Underwrite ('Buffer Underflow')
- CWE-131 Incorrect Calculation of Buffer Size

The good news is there are several guidance documents and standards available to guide the development of C programs. Operationalizing secure C firmware development in a way that is repeatable and consistent is an important step in reducing the number of vulnerabilities.

Challenges with integrating software and hardware life cycles

When we talk about extending the secure software development life cycle with hardware life cycle, the challenge is trying to bridge these two life cycles. On the software side, change is rapid and deploying changes is relatively easy. On the firmware side, the changes take longer to propagate.

We interviewed some experts on comparing the two lifecycles and found the following common themes:

- Product security might involve assessing the security of the whole system, not just the software.
- It includes hardware components as well as the interconnecting devices, like the network interfaces and communication channels.
- Software security has often been prioritized over hardware security.
- Software update life cycles are relatively easy compared to hardware update life cycles.

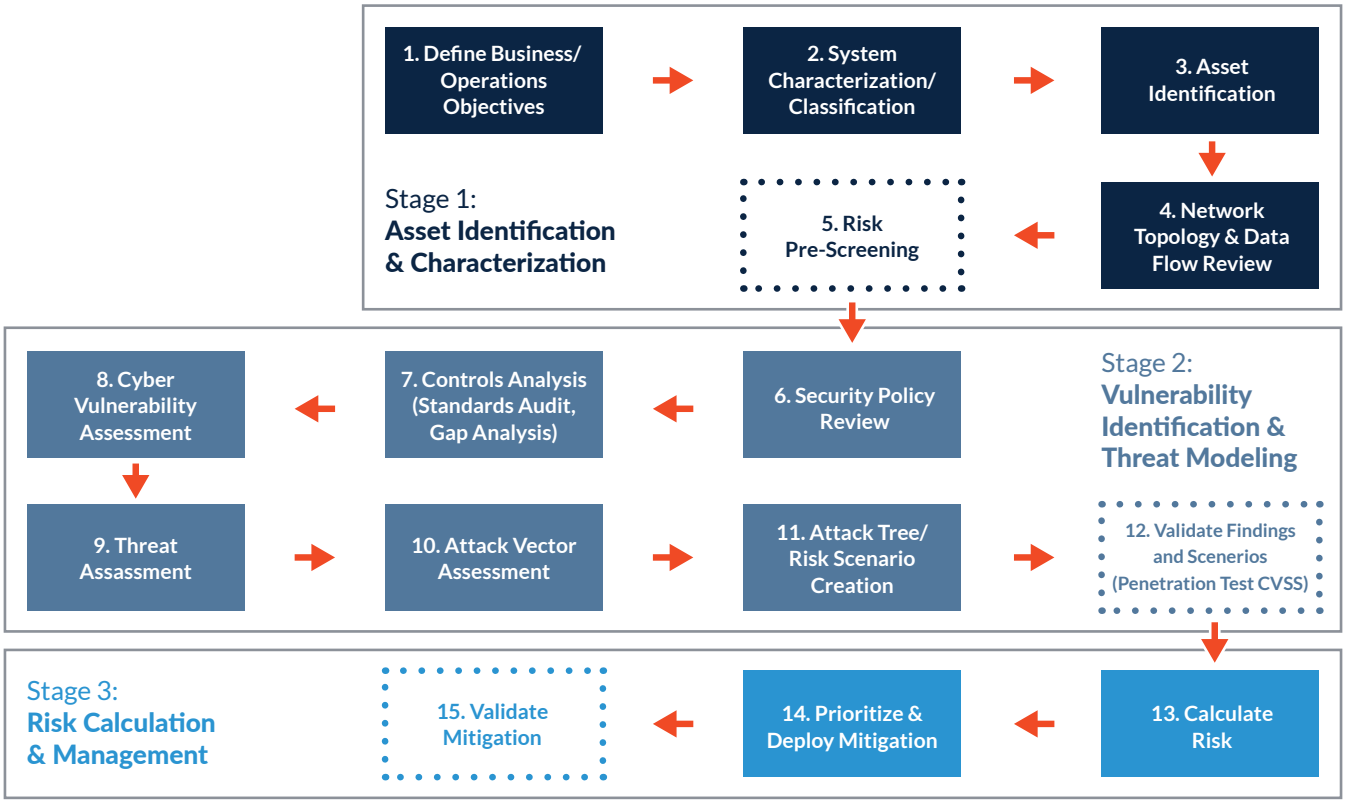
Since both domains contain some differences in their approach, we need to consider a higher level abstraction that can tie these life cycles together. The motivation that drives both lifecycles should come from the business. In business discussions, the conversation often turns toward balancing speed to market and security risks. So both the software and hardware lifecycles need to focus on a balanced risk approach that enables the business to get to market quickly and efficiently without introducing risk beyond an acceptable threshold.

Starting with a risk-based approach

A discussion on risk should start by looking at different standards or frameworks that are already available. It turns out there are many different risk assessment standards that can be used as a starting point.

For example, ISO 31010, NIST 800-30, or SEI/CERT OCTAVE. The goal in a risk-based approach is to create a repeatable assessment process that scales.

You can see a sample risk assessment process below sourced from the book, Hacking Exposed Industrial Control Systems, ICS and SCADA Security Secrets & Solutions by Wilhoit et al.



Source: Hacking Exposed Industrial Control Systems, ICS and SCADA Security Secrets & Solutions, 2016

ISO 31010 suggests the following process:

Ultimately, we need the ability to assess risk of a given asset (our software and firmware, in this case) and determine the right controls to implement in order to minimize the impact of high-probability, high-risk outcomes. It is important that a process be followed in order to deliver consistency. Risk assessments based on different processes makes comparisons difficult.

Establishing policies to guide firmware development

Once an assessment is complete, the creation of appropriate security policies that are actionable becomes important. It is not enough to say that a vulnerability exists. There must be a remediation that is understandable to firmware developers.

According to Yao et al, there are several secure development principles and techniques to help build more secure firmware:

Principles

- Prevent buffer overrun
- Prevent arbitrary buffer access and execution
- Avoid arithmetic error
- Eliminate banned functions
- Be aware of race conditions
- Take care of information leaks
- Know bad compiler optimizations
- Use ASSERT the right way
- Check input across trust boundary
- Fail intelligently
- Reduce attack surfaces
- Use least privilege
- Defense in depth
- Open design
- Remove backdoors
- Keep code simple

Techniques

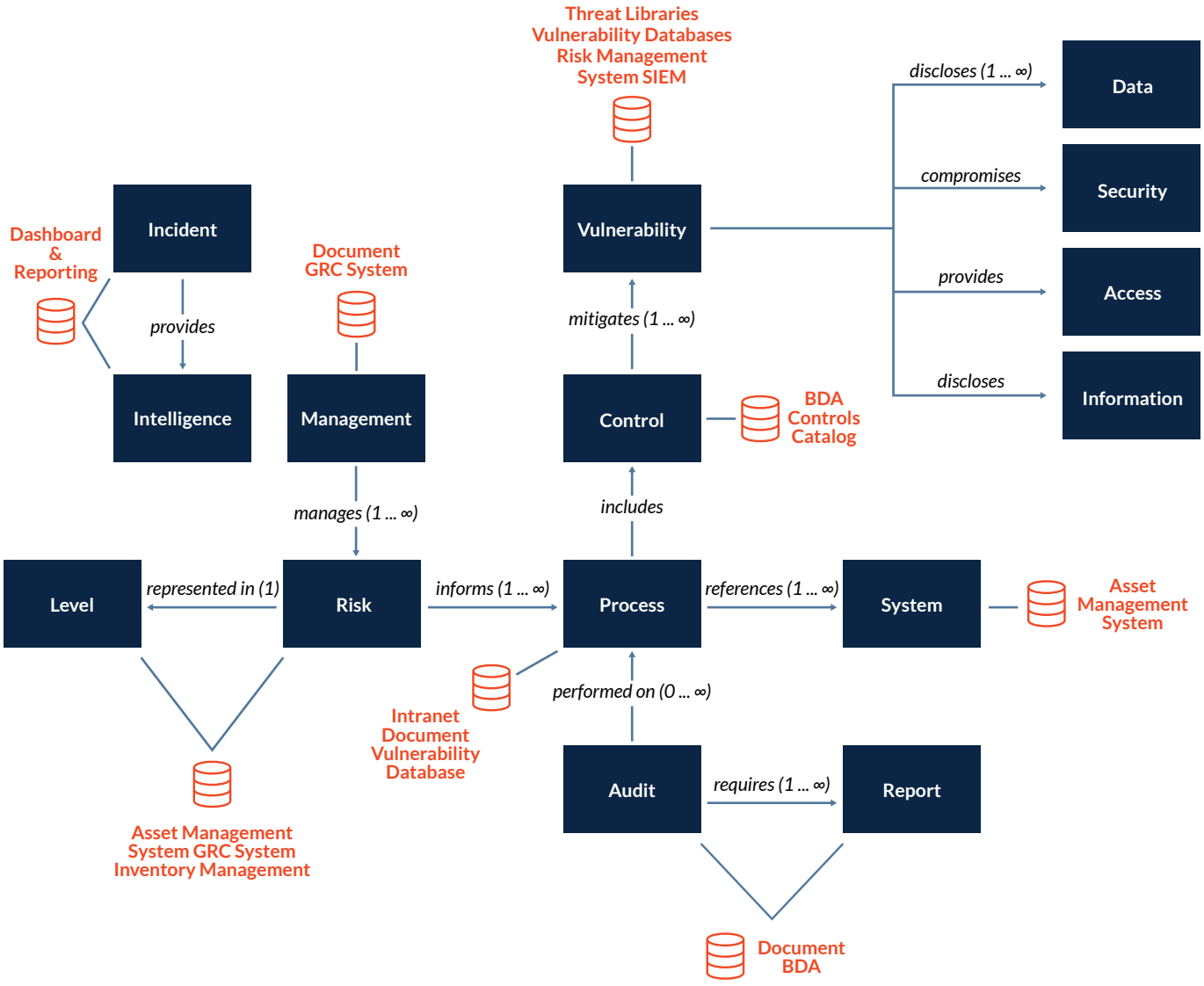
- Static code analysis
- Dynamic code analysis
- Fuzzing
- Symbolic execution
- Formal verification when possible

When creating firmware code, it is essential to provide a mechanism for developers to check their work early and often. Contextual guidance, along with relevant code samples can aid developers in gaining greater confidence in their code. Furthermore, this will help to reduce false positives from downstream code scanning activities.

Automating the risk assessment process

To be effective in supporting the business need for speed and risk management, it is important to establish a set of tool and system integrations to support this. The goal of the integrations is to facilitate consistency and speed. These integrations can trigger an alert when any risk threshold is exceeded and notify relevant stakeholders to pursue the right course of action.

A sample reference architecture for achieving an integrated software and firmware security fabric might look something like this:



Ensuring security by design for firmware

We need to ensure that firmware in our products are created with security in mind. This means examining relevant attack vectors and addressing corresponding vulnerabilities based on a sound risk management approach. Part of this risk management will involve risk assessments to help with proactive mitigation before product development proceeds too far.

In an era of rapid product development, integrated software and firmware lifecycles, a balanced development approach that manages both speed and security risk is an essential component to providing business value.

Security Compass

Additional resources:

- Gupta et al, “IoT Penetration Testing Cookbook,” 2017
- ISA, “Security for Industrial Automation and Control Systems”
- NIST, “SP 800-53: Security and Privacy Controls for Information Systems and Organizations,” 2020
- The Balancing Act Podcast by Security Compass, The Difference Between Product and Software Security
- The Balancing Act Podcast by Security Compass, Build a Product Security Program
- Software Engineering Institute, SEI CERT C Coding Standard

Security Compass

Go Fast. Stay Safe.

Security Compass, a leading provider of cybersecurity solutions, enables organizations to shift left and build secure applications by design, integrated directly with existing DevSecOps tools and workflows. Its flagship product, SD Elements, allows organizations to balance the need to accelerate software time-to-market while managing risk by automating significant portions of proactive manual processes for security and compliance. SD Elements is the world's first Balanced Development Automation platform. Security Compass is the trusted solution provider to leading financial and technology organizations, the U.S. Department of Defence, government agencies, and renowned global brands across multiple industries. The company is headquartered in Toronto, with offices in the U.S. and India. For more information, please visit www.securitycompass.com.

1.888.777.2211
info@securitycompass.com
www.securitycompass.com

@SECURITYCOMPASS
SECURITY COMPASS

OFFICES

GLOBAL HEADQUARTERS
1 Yonge Street
Suite 1801
Toronto, Ontario
Canada M5E 1W7

TORONTO
390 Queens Quay W
2nd Floor
Toronto, Ontario
Canada M5V 3A6

NEW JERSEY
621 Shrewsbury Avenue
Suite 215
Shrewsbury, New Jersey
USA 07702

CALIFORNIA
600 California Street
San Francisco, California
USA 94108

INDIA
#4.07
4th Floor, Statesman House
Barakhamba Road, New Delhi
India 110001