Product & Supply Chain Security for Connected Vehicles

Rising Product & Supply Chain Risk for ECUs and Connected Vehicles

Typical embedded system firmware, including the firmware used in connected vehicles, is composed of 80-95% third-party and open source components. The supply chain for these components is complex and opaque, leaving vehicle manufacturers in the dark when it comes to security issues inherited from these components. In an environment with long product lifecycles and limited opportunities for firmware updates, today's connected vehicles must be secure by design.

You can't protect what you can't see

Finite State's automated platform helps connected vehicle manufacturers break through these opaque supply chains and gain deep visibility into each component, providing your team with the data and guidance to rapidly address security issues and supply chain risks. With Finite State's robust firmware analysis capabilities you can quickly analyze products across your portfolio and business units, including each new security patch and firmware update. By scanning final firmware images before they are released, you can ensure the continued security of your products throughout their development lifecycle.

Key challenges:

- **Regulatory risk** due to the ever-changing landscape of cybersecurity compliance.
- **Time-to-market challenges** caused by costly, time consuming, and unscalable manual testing.
- **Third-party and open source risk**, including legal risk from unknown, undisclosed, or expired licenses.
- **Lack of device-specific security tooling.** AppSec tools don't have the ability to analyze and support embedded system architectures, tools, and binary formats.
- **Protecting brand and reputation** from increasing high profile attacks and breaches.

The Finite State Platform: Benefits at a Glance

Manage supply chain risk

See where your code is coming from and how it could expose you to risk, allowing you to make informed component sourcing decisions.

Enhance product cyber resilience

Discover and remediate security issues such as hard-coded credentials, known open source vulnerabilities, configuration errors, and crypto materials.

Create comprehensive SBOMs

Know the composition of every firmware version, so that components containing newly-identified zero-day vulnerabilities can be traced and patched quickly.

Reduce time-to-market

Drastically reduce or even eliminate the need for costly, time consuming, and unscalable manual testing.

Resolve security issues early

See every firmware version for easy comparison, to ensure new ECU firmware versions are secure before production line FOTA updates.

Prioritize remediation

Learn which security issues could have the biggest impact so fixes can focus on the problems most likely to pose major business and customer risks.

Comply with evolving standards

Maintain compliance with UNECE WP.29, EO 14028, and other standards and regulations for connected vehicles.

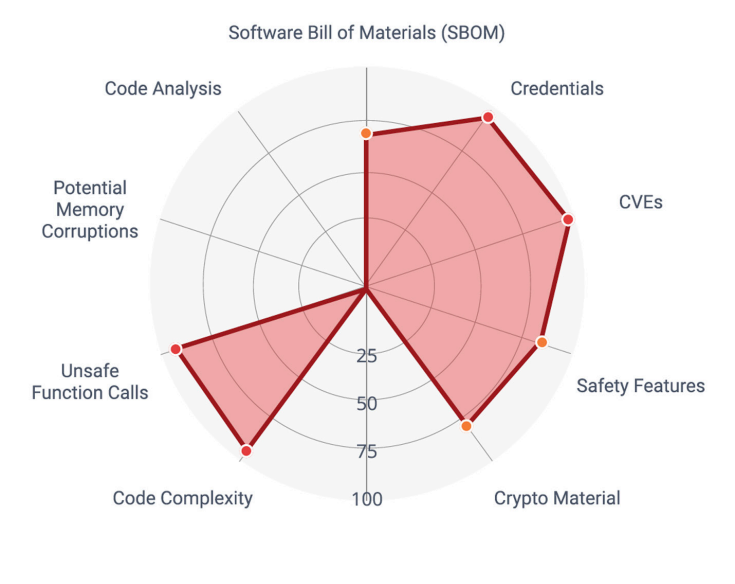
FINITE STATE

Overview: Automated Firmware Analysis

Today, most firmware security testing is done at the time of development of first-party code, and typically looks only at this code. This completely ignores the 80-95% of third party and open source code found in connected vehicle components.

Finite State takes a different approach to firmware analysis, examining the fully compiled code in the form of binaries for each firmware version and identifying security issues present from first party, third-party, and open source components.

By analyzing binaries after they are compiled and ready to be flashed onto a device, Finite State is capable of detecting security issues that would typically be missed during the development phase, including:



Hard-coded credentials

When passwords or other authentication is hard-coded in firmware, attackers can use these credentials to gain trust and access to key systems.

CVEs

Known vulnerabilities often exist for years in firmware code, leaving potential backdoors open to black hats or even nation-state level attackers.

Unsafe function calls

Code containing unsafe function calls can leave firmware vulnerable to injection attacks that can cause denial of service, privilege escalation, or full takeover of the device.

Cryptographic material

Poorly configured systems may contain files such as private keys that may serve as a backdoor or let attackers impersonate the system, and expired or self-signed certificates can present weaknesses attackers can use to compromise the system.

Insecure configurations

Some security issues are generated during the build process, such as not using compiler flags for exploit mitigations. Binary analysis allows these weaknesses to be identified and remediated. Other security issues may be introduced after the build process, such as network facing service configurations that leave the device open to attackers.

The Billion Dollar Question for Vehicle Manufacturers

If a zero-day vulnerability is discovered tomorrow in an open-source library that is used in your product firmware or in a vendor supplied component, how long would it take you to determine which devices and firmware versions were impacted?