A well-known Fortune 500 company was developing a new mission-critical device in 2022. It was to be an extension of its existing product line, one of the main promises of which was the ability to secure sensitive data.

To make good on this undertaking, the product team invested heavily in researching best-of-breed embedded security solutions. It sought a comprehensive solution that included these requirements:

- **Protection for sensitive information** being kept in TrustZone (an ARM security foundation)
- **Minimal performance impact**, and easy integration with Zephyr-operated devices
- **0-day protection** enabled by hardened firmware, with security controls to ensure memory and control flow integrity
- Remote access option for **real-time device information** (e.g., security alerts, analytics, logs, performance metrics)

- **Compliance with regulatory demands**, calling for advanced and demonstrable security controls with post-production vulnerability and exploit visibility

The team identified the Sternum platform as the best match based on its ability to provide comprehensive security and monitoring solutions with near-zero overhead.
Preventing TrustZone Data Theft

During the POC, Sternum demonstrated how a common software vulnerability in the application process could lead to a total compromise of the TrustZone environment and theft of sensitive information. We also showed how the same vulnerability could be automatically blocked by its security controls.

In addition, Sternum confirmed protection against other vulnerabilities that bypass Zephyr’s native memory protection, such as heap buffer overflow, double-free, use-after-free, and return-oriented programming (code reuse) attacks.

The POC emphasized gaps in the company’s hardware security, native OS, and/or compiler security features against software vulnerabilities and in-memory attacks—gaps that Sternum can easily address.

Fixing a Memory Leak During Deployment

Once its engineers integrated Sternum’s embedded integrity verification technology, they were quickly alerted about a memory integrity violation discovered by EIV™ as it began profiling memory during runtime.

The alert was triggered upon processing a cloud server request. It identified a non-null-terminated JSON string used with a `strlen()` function.

This presented security issues; in the absence of termination, `strlen()` would continue to traverse memory until it encounters an arbitrary NULL character further down the line, resulting in a length value much larger than the interned string size.

Using a non-null-terminated character sequence could result in buffer overflow or data corruption. This would render the system in an undefined state, thus making it prone to crashing and remote code execution attacks. (For more information, see CWE-170: Improper Null Termination.)

### Improper Null Termination

In the following code, the `readlink()` function expands the name of a symbolic link stored in `pathname` and places the absolute path into `buf`, with the resulting value length being larger than that calculated using `strlen()`.

```c
char buf[MAXPATH];
...
readlink(pathname, buf, MAXPATH); // does not null terminate
int length = strlen(buf);
...
```

In the absence of a NULL character, `strlen()` would continue to traverse memory, which could lead to several security and/or performance issues.
As for the device in question, when string termination didn’t occur, EIV™ immediately issued an automatic alert. This provided engineers with precise details, including identification of the exact lines of problematic code and memory addresses. They were able to quickly make the fix, using code similar to this:

```c
char buf[MAXPATH] = { 0 };
...
readlink(pathname, buf, MAXPATH - 1);
int length = strnlen(buf, MAXPATH);
...
```

---

**Baking Security Into Your Product Lifecycle**

Because the `buf` contents and its trailing memory was accidentally `NULL`, the `strlen()` function appeared to be behaving correctly from a static analysis point of view. Since it has no visibility to runtime program execution, this illustrates the gaps rooted in any static and passive approach.

Instantly surfacing the problem, this case study illustrates how EIV’s use of dynamic memory profiling and runtime memory protection helps with both active mitigation for deployed devices and the identification of runtime issues that evade static analysis.

Moreover, in-memory attacks can bypass hardware security, secure boot, and naive memory protections of some operating systems, resulting in fully compromised sensitive data, secure elements, and the device itself.

Even if detected post-deployment, patching such issues at scale for hundreds of thousands of devices across the globe would be a much slower and more costly affair. Detecting bugs after a device is already in the field makes it significantly more costly to fix—not counting the hidden business cost of shifting your engineers’ focus away from innovation.
About EIV Runtime Protection

EIV runtime protection is part of Sternum's full-stack IoT platform, offering a wide range of on-device security, threat detection, and monitoring capabilities.

Using patented technology, EIV leverages binary instrumentation to deploy verification checks across all exploitation paths, thus deterministically preventing all code and memory manipulation attempts— Including third-party code—to ensure system integrity at all times.

Deployed as part of your build, EIV seamlessly integrates with all development, testing, and deployment processes. It enhances your devices with agentless, in-code security with near-zero overhead and no reliance on external communication.

More about EIV →  More about Sternum →