Embedded C/C++ Static Code Analysis with Fortify Static Code Analyzer
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>AUTOSAR Software Development Platform</td>
<td>1</td>
</tr>
<tr>
<td>Required Customization</td>
<td>2</td>
</tr>
<tr>
<td>Findings</td>
<td>3</td>
</tr>
<tr>
<td>GENIVI Software Development Platform</td>
<td>3</td>
</tr>
<tr>
<td>Required Customization</td>
<td>4</td>
</tr>
<tr>
<td>Findings</td>
<td>5</td>
</tr>
<tr>
<td>Conclusion</td>
<td>5</td>
</tr>
</tbody>
</table>
Introduction

More and more applications running on microcontrollers are based on embedded C/C++ code. This holds for automotive, aviation, nautical, and medical markets. It also applies to IoT, for example, in the areas of home automation and sports. However, microcontrollers tend to a certain scarcity of resources (especially memory), so special compilers are often used instead to produce optimized binary code—compilers that might even be for specific microcontroller chips, for example the Greenhill Renesas v850 compiler.

Whereas Micro Focus ® Fortify Static Code Analyzer (Fortify SCA) offers a good integration with IDEs such as Visual Studio, Eclipse, or IntelliJ, embedded C/C++ programmers seem to live more in a Linux or Unix-based world that is more prone to command line toolsets. This makes the scanning of embedded C/C++ code a challenge in this environment. But even with this challenge, far more is possible than the standard System Requirements document promises for Fortify Static Code Analyzer.

For example, developers in Germany have been quite successful with scanning embedded C/C++ applications in the automotive industry. A prerequisite is that the code must be compliant with ANSI-C, so that Fortify Static Code Analyzer parser can understand the code. Most car manufacturers request that their 3rd-party coding partners support this standard.

These capabilities could potentially be used in all of the markets mentioned in the introduction to this paper, so these capabilities could be a door opener to these markets and use cases.

The automotive industry is mainly using two software development environments: AUTOSAR and GENIVI.

AUTOSAR Software Development Platform

AUTOSAR (AUTomotive Open System ARchitecture) is a worldwide development partnership of vehicle manufacturers, suppliers, and other companies from the electronics, semiconductor, and software industry. AUTOSAR is also a software development platform that literally all vehicle manufacturers worldwide contribute to and use to develop embedded C applications running on the many microcontrollers built into their vehicles.

AUTOSAR enables vehicle manufacturers to focus on the development of their software components and lets the AUTOSAR platform take care of the rest. Embedded C applications developed for running on microcontrollers require highly optimized code for the microcontrollers’ processors (which usually lack resources, especially memory). This code optimization is performed by special compilers, often from Green Hills or Wind River.
These compilers are not officially supported by Micro Focus® Fortify SCA yet. However, we have successfully scanned one AUTOSAR Application Software Component and the complete AUTOSAR technology stack, with only minor customizations. This includes an Application Software Component running on a microcontroller using the Renesas V850 chip and the Green Hills ccv850 compiler. This white paper presents the required tweaks to get this working.

We have also successfully scanned code compiled with the Wind River Diab compiler, using a very similar setup. And we are continuing our efforts to support even more compilers on the AUTOSAR platform, which opens up a whole new set of use cases for Fortify SCA.

**Required Customization**

Fortify SCA Properties Add the following entries to the compilers section in fortify.sca.properties:

```
# Support for Green Hills ccv850 compiler
com.fortify.sca.compilers.ccv850 = com.fortify.sca.util.compilers.GccCompiler
com.fortify.sca.compilers.ccv850 = com.fortify.sca.util.compilers.GppCompiler
```
**PATH Environment Variable**
Add the absolute paths to sourceanalyzer. Then make and ccv850 to the PATH variable from where the build is later executed. Check whether you can call these executables directly from there, for example, sourceanalyzer --help.

**Modify the Makefile**
Locate the CC statement in the makefile and replace the invocation of ccv850 with a sourceanalyzer wrapped invocation of ccv850. For example:

```
CC = sourceanalyzer -b ccv850
```

**Execute the Build and Run the Scan Step**
Execute the build as normal and then run the scan step:

```
sourceanalyzer --b <nameofautorsarapplication> -scan --f <nameof fprofile>
```

**Findings**
Fortify SCA’s engines identified various issues in embedded C source code. (Since these individual vulnerabilities contain information about customer-specific code, no findings can be shared publicly.)

Developers are then able to review these findings and fix these issues in the source code.

**GENIVI Software Development Platform**
“GENIVI is a nonprofit industry alliance” of more than 140 companies “committed to driving the broad adoption of open source, In-Vehicle Infotainment (IVI) software and providing open technology for the connected car.”

In stark contrast to AUTOSAR, which is completely proprietary, GENIVI is completely open source.
For this white paper, we have selected one of the applications on the GENIVI Github. The VSOMEIP application code is initially coming from BMW AG and uses CMAKE as the build tool.

**GENIVI Technical Deliverables**

**Software Development Environment**

**GENIVI® Development Platform (GDP)**

**GENIVI Baseline**
- Individual Software Components
- Standard Interfaces (APIs)
- Architecture & Compliance Specification

**Development & Operations Tooling**

*Figure 2. GENIVI Technical Deliverables*

GENIVI enables vehicle infotainment manufacturers to focus on the development of their software components and lets the GENIVI platform take care of the rest.

Embedded C applications developed to run on microcontrollers require highly optimized code for the microcontrollers’ processors, which usually lack resources (especially memory).

For this white paper, we have selected one of the applications on the GENIVI Github. The VSOMEIP application code is initially coming from BMW AG and uses CMAKE as the build tool.

**Required Customization**

CMAKE is one of the most difficult challenges because you will rarely find a configuration file where you can interfere, in order to include Fortify translation into the compile step (wrap the compiler with sourceanalyzer).

In our case, we had to go down to the very end of the build tool food chain and replace the compiler binary with a shell script, wrapping the compiler calls with sourceanalyzer, as follows:

**Shell Script**

```
#!/bin/bash
sourceanalyzer -b g++-4.8 "$@
```
I checked in fortify-sca.properties file and there was already a compiler entry for g++4*, so I was all set:

```java
com.fortify.sca.compilers.g++4* = com.fortify.sca.util.compilers.GppCompiler
```

After that, I ran Cmake and Make, and the code compiled and linked snuggly. Then I performed the required Fortify translate step required for the static analysis.

## Findings

### Analyzers

We had results from the following analyzers:

- Control Flow
- Semantic
- Structural

### Categories

**High:** Memory leaks, Type mismatches—Functions meant to deliver unsigned results delivering signed results and vice versa; Unreleased resources

**Low:** Dead Code—Code that can never be executed; Variables defined but never used; Variables never read; Variables used but never initialized; Missing check against null; Redundant null checks; Process control; Unchecked return values

For code that requires high optimization, even the low findings are of interest. Every dead code and every variable defined requires additional memory, which on microcontrollers is a rare resource.

## Conclusion

With only minimal customization, we were able to prove that Fortify SCA can scan AUTOSAR applications using Make and GENIVI applications using Cmake as their build tools. We recommend extending this research to additional applications on the AUTOSAR and GENIVI platforms. And keep in mind that many IoT devices are also built on microcontrollers, so we could also extend into that direction.

Learn More At

www.microfocus.com/fortifysca