Evaluation of Software Vulnerabilities in Vehicle Electronic Control Units

Jesse Edwards, Ameer Kashani, Gopal Iyer
1. Technology Trends and Research Motivation
2. Code Quality: Existing Considerations
3. Static Code Analysis (SCA) Background
4. Main Purpose and Approach
5. Selection of System, SCA Tool, and Secure Coding Compliances
6. Results
   • Vulnerabilities Comparison
   • Most Common Vulnerabilities
   • Greatest Threats
7. Recommendation for Fixing Security Vulnerabilities
8. Conclusion, Limitations, and Future Work
Technology Trends

- Modern architectural design paradigms are increasing the complexity of vehicle platforms
  - Software features (DbW, electrification, etc.)
  - Increased number of computing modules/smart sensors
  - Increased external connectivity and data transactions
  - Interoperability of components with untrusted devices

- The rising complexity increases the potential for critical threats
  - Unauthorized access of private OEM APIs
  - Loss of personally identifiable user data
  - Remote attacks (DoS, control manipulation, espionage, etc.)

- Attack feasibility demonstrations are continuously published by the security research community
Research Motivation

- The largest class of security vulnerabilities stems from coding errors made by developers
  - 64 percent of the nearly 2,500 vulnerabilities in the National Vulnerability Database were caused by programming errors [16]

- Existence of minor, preventable bugs in a device’s software simplifies efforts needed by adversaries to exploit the available attack vectors and compromise the system
  - Information disclosure
  - Buffer overflow
  - Use after free

- Many classes of vulnerabilities caused by coding errors can be entirely mitigated prior to production release
  - Early bug detection in SDLC
OEM Security Requirements

- Static Analysis
  - Static Code Analysis (SCA)
- Dynamic Analysis
  - Behavioral Analysis
  - Dynamic Code Analysis (DCA)
  - Fuzzing
- Interactive Analysis
  - Penetration Testing
  - Threat Analysis
- Hardware Testing
Static Code Analysis (SCA) Background

Advantages
• Finding potential bugs, with references to CWE
• Comparing projects by severity
• Breaking down results by category
• Monitoring trends

Challenges
• Consistency in reports (SCA scope and output is not the same)
• False Positives: A warning that is not a bug
• False Negatives: Tool did not find a bug that was in the code
• Cost
Main Purpose and Approach

Main Purpose
Finding the most common and dangerous security bugs using static code analysis with secure coding compliance checks to achieve systematic elimination of bugs

Approach
1. Select target systems
2. Select SCA tools
3. Cross-reference secure coding standards
4. Gather insights
5. Bug elimination

Result

Secure Coding
- SANS Top 25
- CWE
- CERT C

SCA Tools
- Tool x
- Tool y

System
- RTOS
- Bare Metal
Select Target Systems

- Architecture varies by vehicle model and only an example.

**Infotainment**
- RTOS
- Linux

**Cluster**
- RTOS
- AUTOSAR OS
- Bare metal

**BCM**
- AutoSAR OS
- Bare metal
RTOS and Bare Metal

RTOS
• Complex preemption
• Common libraries
• Middleware (Filesystem, USB, TCP/IP)
• Heavy resources
• Complex computing

Bare Metal
• Light resources
• Low memory
• Simplistic
Evaluate SCA Tools

- Developed evaluation framework
- Benchmarked eight leading SCA tools using framework and selected two competing finalists

<table>
<thead>
<tr>
<th>Benchmark Item</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality/coverage</td>
<td>50%</td>
</tr>
<tr>
<td>Tool PC Performance</td>
<td>30%</td>
</tr>
<tr>
<td>Integrability</td>
<td>20%</td>
</tr>
<tr>
<td>Usability</td>
<td>20%</td>
</tr>
<tr>
<td>Upgradeability</td>
<td>20%</td>
</tr>
<tr>
<td>Cost</td>
<td>20%</td>
</tr>
</tbody>
</table>
Secure Coding Compliance

- Secure coding check used in the SCA tool is SEI CERT C
- CERT C has a relationship with Common Weakness Enumeration (CWE)
  - CWE’s contain high level design issues and are independent of programming language
  - There are over 1000 CWE’s

[Image: Secure coding guidelines covered by MISRA C 2012]

- Covered 37%
- Partially Covered 19%
- Contradicting 1%
- Not Covered 43%

[17]
Vulnerabilities Comparison

RTOS vs. Bare Metal

• Hypothesis: higher warnings for the RTOS-based system primarily due to the greater code size and software module complexity

• Hypothesis was incorrect
  ➢ Bare Metal system contained ~10 times the amount of CWE warnings
Which Vulnerabilities Were Most Common in RTOS & Bare Metal?

Most Common CWE’s (RTOS & Bare Metal)

- CWE-192 Integer Coercion Error
- CWE-197 Numeric Truncation Error
- CWE-457 Use of Uninitialized Variable
- CWE-466 Return of Pointer Value Outside of Expected Range
- CWE-587 Assignment of a Fixed Address to a Pointer
- CWE-665 Improper Initialization
- CWE-704 Incorrect Type Conversion or Cast
- CWE-758 Reliance on Undefined, Unspecified, or Implementation-Defined Behavior
- CWE-908 Use of Uninitialized Resource
Which Coding Errors Posed the Greatest Threat?

SANS Top 25: a list of the top 25 most dangerous software errors.

1. CWE-120 Buffer Copy Without Input Size Check
2. CWE-134 Uncontrolled Format String
3. CWE-190 Integer Overflow or Wraparound
4. CWE-676 Use of Potentially Dangerous Function

- Buffer overflow is still the leading weakness of embedded software
Recommendations for Fixing Security Vulnerabilities

• Immediate action of elimination of security vulnerabilities
• Use SCA with security compliance checks
• Avoid the noise (false positives). Filter for high risk warnings first
• Long term action of elimination of security vulnerabilities
• Provide training to developers for secure coding practices
• Work SCA into the environment
  • Include SCA results in code reviews
  • Always perform SCA before checking in code
  • Secure coding standard and SCA checks should align
  • Benchmark tools and find out which is best for your organization
• Revisit projects and compare SCA results to monitor potential issues
Conclusion, Limitations, and Future Work

Conclusion
• After performing SCA with secure coding checks we can immediately reduce security vulnerabilities early in the lifecycle.

Limitations
• SCA cannot conclude that a warning translates to a vulnerability and requires human expert intervention to investigate the code
• Can only catch coding errors (e.g. design logic flaws)

Future Work
• Using data aggregator with machine learning to eliminate false positives
• Automate SCA tool data collection and track quality improvement trend.
References

[14] Society of Automotive Engineers (SAE), http://www.sae.org/
[16] https://www.cert.org/secure-coding/products-services/scale.cfm?