

# 嵌入式软件静态测试方法研究

赵正旭 梅成芳 张 强

(石家庄铁道大学 复杂网络与可视化研究所 河北 石家庄 050043)

**摘 要:** 嵌入式软件的特点决定了嵌入式软件的测试重点是运行时检测、内存、安全等方面。而且 C/C++ 语言作为嵌入式软件的主流编程语言,其本身具有典型的代码缺陷:缓冲区溢出、数组越界、空指针引用异常、内存泄漏等。为了在检测出 C/C++ 代码缺陷的同时兼顾嵌入式软件的测试重点,针对嵌入式软件的静态测试方法进行研究。掌握 C/C++ 语言的常用静态测试工具的主要功能侧重点及使用方法,使用 C++Test、PC-Lint 两种测试工具对同一嵌入式软件源码进行静态测试。两者测试结果形成对比和补充,降低了代码缺陷的漏报、误报的可能性,提高了测试效率。实现了在代码编写阶段检测出软件中隐藏的代码缺陷,进而保障嵌入式软件安全可靠的运行,验证了该嵌入式软件静态测试方法的有效可行性。

**关键词:** 嵌入式软件; 静态测试; C++Test; PC-Lint

中图分类号: TP301

文献标识码: A

文章编号: 1673-629X(2019)03-0064-05

doi: 10.3969/j.issn.1673-629X.2019.03.013

## Research on Static Testing Method of Embedded Software

ZHAO Zheng-xu, MEI Cheng-fang, ZHANG Qiang

(Institute of Complex Networks and Visualizations, Shijiazhuang Tiedao University, Shijiazhuang 050043, China)

**Abstract:** The characteristics of embedded software determine its test focus is run-time detection, memory, security and so on. Moreover, C/C++ as the mainstream programming language of embedded software, has its own typical code defects: buffer overflow, array out of bounds, null point exception, memory leak and so on. In order to detect the defects of C/C++ code and take the test emphasis of embedded software into consideration, the static test method of embedded software is studied. After mastering the main functions and methods of the static testing tools of C/C++ language, C++Test and PC-Lint are used to test the source code of the same embedded software. The results of the two tests are contrasted and supplemented. The possibility of false negatives and false positives of code defects is reduced, and the testing efficiency is improved. The code defects hidden in the software are detected at the code writing stage, and the secure and reliable operation of the embedded software is ensured. The feasibility of the proposed static test method is verified.

**Key words:** embedded software; static testing; C++Test; PC-Lint

## 0 引 言

嵌入式软件广泛应用于航天、通信、轨道交通等领域,其自身有着实时性、专用性、与硬件紧密关联等特点,因此嵌入式软件对软件可靠性、安全性要求极高。静态测试是指不运行程序源代码,通过人工或者借助工具来发现源代码中隐藏的空指针、变量未初始化、数组越界、内存泄漏、嵌套错误等问题<sup>[1]</sup>。人工检测代码缺陷受测试人员经验的影响,且难度高、工作强度大,一般采取工具检测和人工干预的方式来相互配合提高工作效率。针对嵌入式软件自身特点,文中提出一种在嵌入式软件初期编码阶段进行静态测试的方法,以

保障高质量代码编写以及嵌入式软件的安全性、可靠性。

## 1 概 述

### 1.1 嵌入式软件特点

国内普遍认同的嵌入式系统定义为:以应用为中心,以计算机技术为基础,软硬件可裁剪,适应应用系统对功能、可靠性、成本、体积、功耗等严格要求的专用计算机系统<sup>[2]</sup>。

嵌入式软件具有实时性。例如航天系统中的嵌入式软件要求软件能够实现实时运行,在特定时间内需

收稿日期: 2018-04-24

修回日期: 2018-08-28

网络出版时间: 2018-12-20

基金项目: 河北省第三批创新团队及领军人才“巨人计划”(冀办字[2018]33号)

作者简介: 赵正旭(1960-),男,博士,教授,研究方向为小世界网络系统、虚拟现实技术和应用;梅成芳(1991-),女,硕士研究生,研究方向为软件测试。

网络出版地址: <http://kns.cnki.net/kcms/detail/61.1450.TP.20181219.1542.074.html>

完成接收、处理、发送信息等一系列任务,对处理时间、处理任务的时序性,软件运行速度以及避免运行时内存遗漏都有严格要求<sup>[3]</sup>。

嵌入式软件要求高可靠性。嵌入式软件一般有别于通用软件适用环境,其工作环境常常伴随着强辐射、强磁场、真空、远距离操控等复杂问题。

嵌入式软件具有专用性。嵌入式软件的开发一般是针对特定用途、特定场景,具有很强的专用性。

嵌入式软件与硬件关系密切。嵌入式系统由软件和硬件组成,其中软件部分正是为硬件部分设计的。嵌入式软件的编写还涉及到嵌入式系统的外围设备,例如通信接口、输入/输出设备、外设接口等。

## 1.2 嵌入式软件静态测试

由于嵌入式系统自身具有软硬件结合的特点,其软件的代码编写过程中会涉及到对硬件的操作。由于C语言本身可以对硬件操作,又具有易学、灵活、可移植、高效率等特点,因此日渐成为嵌入式软件的主要编程语言。C++是C语言的增强版,也在一定程度上继承了C语言的缺陷,C/C++语言其本身都属于弱类型语言,允许变量类型隐式转换,在开发效率高的同时可靠性较弱。而且C/C++编译器不进行强制类型检查,也不做任何边界检查,很容易存在代码安全隐患<sup>[4]</sup>。

大多数静态测试工具都支持静态测试规则检查。对于嵌入式软件测试规则的制定,国内外已经有很多成熟的案例。汽车工业软件可靠性联合会(The Motor Industry Software Reliability Association, MISRA)在1998年发布了汽车行业著名的安全性C语言编程规范MISRA-C:1998,旨在协助开发安全、高可靠性的嵌入式软件。随后又出版了MISRA-C:2004,该版本把适用领域从汽车领域扩展到所有高安全性系统。MISRA系列编码规则通过不断完善和更新,之后相继出版了MISRA-C++:2008、MISRA-C:2012<sup>[5]</sup>。类似的规则还有由航天科工集团出版的《航天型号软件C语言安全子集》(简称GJB5369-2005)<sup>[6]</sup>,由总装电子信息基础部出版的《C/C++语言编程安全子集》(简称GJB8114-2013)和《军用软件安全性设计指南》(简称GJB/Z 102A-2012)等。这些规则大多分为强制执行和推荐执行两类,需要根据测试的嵌入式软件灵活选择。

嵌入式软件静态测试和通用软件静态测试的方法大致相同,区别之处在于针对其软件自身特点有针对性地进行检测<sup>[7]</sup>。例如:关注运行时检测、内存遗漏问题;针对嵌入式软件的专用性有侧重点地去筛选检测项;关注嵌入式软件主流编程语言C/C++的自身缺陷;适当采用成熟的静态测试规则。尽可能地在代码编写过程中解决软件中隐藏的安全隐患,进而确保嵌

入式软件在特定硬件环境下安全可靠的运行。

## 2 代码缺陷

由于C/C++语言本身风格自由易出错,程序员在编写过程中要格外注意防止安全漏洞的产生。通过静态测试工具辅助程序员排除缓冲区溢出、数组越界、指针引用错误、内存泄漏等安全隐患,消除代码缺陷的同时保障程序的安全。

### 2.1 缓冲区溢出

缓冲区溢出指程序接收的数据长度超出了缓冲区的容量,而程序中没有对数据长度的限制操作,导致部分数据覆盖了临近内存段的数据。而被覆盖的数据可能是与程序运行相关的重要数据,也有可能是某一个操作指令的指针,这些都可能造成程序运行失败、拒绝服务、系统异常等安全隐患。最为严重的是导致堆栈溢出,攻击者可以实施攻击,改变程序返回地址。这个被改变的地址有可能就指向任意恶意代码,使攻击者可以对系统进行非法操作<sup>[8]</sup>。因为缓冲区溢出漏洞普遍存在以及缓冲区攻击易于实现,缓冲区溢出攻击已成为远程网络攻击的主要方式。在C/C++语言中可能导致缓冲区溢出的语句有:printf()、vsprintf()、scanf()、gets()、strcpy()、strcat()、strcmp()等<sup>[9]</sup>。

### 2.2 数组越界

数组占用一段连续的内存空间,通过数组的下标来访问某一个数组成员。例如数组int a[len]的合法访问范围是0到len-1,当数组索引不在其正常范围内时,程序运行时就很可能出错<sup>[10]</sup>。因为C/C++的编译器在编译时遇到数组访问越界代码是不会提示错误的,但是当运行时可能会改变其他内存的数据,此种错误极有可能是灾难性的,而且调试时不易查找。

### 2.3 空指针引用异常

当声明一个指针变量并没有对它实例化时,是不指向任何合法内存的。发生空指针引用异常导致程序运行时抛出Null Point Exception的情况可能是:调用空指针对象的本身、属性或者方法;获取空指针对象的长度;试图修改空指针对象的数据成员。

### 2.4 内存泄漏

内存泄漏指程序没有释放已经不再使用的内存,使程序对这些未释放的内存无法控制,导致程序可用内存空间减少,直接导致性能不良。更为严重的是程序运行时过多的内存泄漏可能会导致程序运行停止。短暂运行过程中的内存泄漏对计算机影响不大,因为导致内存泄漏的进程终止时,这个进程使用的内存会被释放。前提是系统有足够的内存空间,比如PC机。但是嵌入式系统硬件不同于计算机系统硬件拥有大容量的硬件存储器,嵌入式系统使用像EPROM和EEP-

ROM 之类的闪存。嵌入式软件设计中存在的内存泄漏有可能会系统崩溃等严重后果。所以对嵌入式软件中的内存泄漏的检测是必不可少的。但是由于内存泄漏难以定位查找,此时静态测试工具就提供了很大的便利。导致内存泄漏的情况有:堆里用 new 创建了对对象,但是没有手动 delete 释放(编译器不会自动释放堆内 new 分配的内存);new 和 delete 没有一一对应,有遗漏的内存空间未得到释放或者多次释放同一个内存;不正确释放二维数组;不正确释放指向对象的

指针数组,应该是先释放对象再释放指针;释放和分配函数不对应,例如 malloc/new/new[] 分别对应 free/delete/delete[];释放未被分配的内存等。

### 3 静态测试工具

成熟的测试工具有很多,例如 Fortify SCA、Coverity、C++Test、PC-Lint、CppCheck 等。这些静态测试工具都有其测试的侧重点,它们的主要功能特点如表 1 所示。

表 1 五种工具的主要功能特点

工具	支持语言	编码规则检查	商业软件	侧重点/特点
C++Test	C/C++	是	是	运行时错误检测
Fortify SCA	C/C++、Java、.NET 等	是	是	安全漏洞
PC-Lint	C/C++	是	是	代码逻辑分析
CppCheck	C/C++	是	否	非语法错误
Coverity	C/C++、C#、java 等	是	是	质量缺陷

#### 3.1 C++Test

C++Test 是 Parasoft 公司的基于 C/C++ 的测试工具,涵盖静态测试、单元测试、回归测试。C++Test 静态测试部分包含了很多典型行业规范,有 BugDetective、Effective C++、GJB5369、MISRA C 2004、MISRA C++ 2008 等。C++Test 还提供 RuleWizard 用户自定义规则功能。C++Test 的静态测试具有基于数据流和基于模式的两种分析技术。基于数据流的静态测试分析技术也被称为 BugDetective。BugDetective 模拟和识别代码中的复杂路径,搜索并定义可疑点,进而暴露可能触发运行时缺陷的路径。特别是针对遗留代码库和嵌入式代码的运行检测效果较差的情况,BugDetective 可以无需测试用例和执行代码就发现除数为零、内存和资源泄漏、空指针引用等运行时缺陷,而这些软件错误很多是基于模式的静态分析技术不易检测到的<sup>[11]</sup>。C++Test 提供的可视化、易操作的静态测试完全可以胜任嵌入式软件的静态测试工作。

#### 3.2 Fortify SCA(Fortify Source Code Analysis)

Fortify SCA 是 Fortify 360 系列产品中的静态代码分析器,旨在检测源代码中存在的安全漏洞。Fortify SCA 有全面的安全编码规则,提供最新的安全编码规则包下载来应对新的安全漏洞,用户还可以自定义安全规则。具有数据流、语义、结构、控制流、配置流等分析引擎<sup>[12]</sup>。支持 C/C++、Java、JavaScript、.NET、PHP 等 20 多种语言,支持 Windows、Linux、Unix 等操作平台和多种编译器,以及提供 Eclipse、Visual Studio 等 IDE 的插件。该工具分析源代码分为三个步骤:转换、扫描与分析、校验。转换:源代码关联一个 Build ID,

创建中间文件。扫描与分析:扫描中间文件,分析源代码,将检测结果写入 FPR 文件。校验:验证所有源文件依据正确的规则包被扫描。但是其本身是商业软件,价格昂贵。

Fortify SCA 提供扫描方式:IDE 插件扫描、命令行、Audit Workbench(Fortify SCA 图形用户界面)扫描。例如想要扫描一个 Visual Studio 项目 FreeBird,此时就可以用下列语句:

```
sourceanalyzer -b FreeBird devenv FreeBird.sln /
REBUILD
```

```
sourceanalyzer -b FreeBird -scan -f FreeBird.fpr
```

FreeBird 是项目名,devenv 表示 visual studio 的可执行程序,/REBUILD 是 devenv 命令参数。

#### 3.3 PC-Lint

PC-Lint 是 GIMPEL SOFTWARE 公司开发的 C/C++ 软件代码静态分析工具,侧重于代码的逻辑分析<sup>[13]</sup>。PC-Lint 的历史可以追溯到 1979 年贝尔实验室开发的静态代码分析工具 Lint(早期最著名的 C 语言工具之一)。而 Lint 起初是 UNIX 系统工具,后来衍生了 Linux 系统发行版本 Splint 和 Windows 系统发行版本 PC-Lint。PC-Lint 支持大多数流行编译环境和编译器。PC-Lint 采用命令行和开发环境插件集成两种方式。PC-Lint 的基本命令行的形式为:

```
c:\lint\lint-nt.exe -i"c:\lint" -u std.lnt "d:\
FreeBird\*.cpp"
```

lint-nt.exe 是 PC-Lint 在 Windows 下的可执行程序,-i"c:\lint"表示查找到配置文件\*.lnt 的路径,-u std.lnt 指明使用的那些配置文件,"d:\FreeBird\\*.

cpp"表示要测试的目标文件。

PC-Lint 检测出来的错误列表如表 2 所示。

表 2 PC-Lint 警告列表

警告类型	C	C++	警告级别
语句错误	1-199	1001-1199	1
内部错误	200-299		0
致命错误	300-399		0
警告	400-699	1400-1699	2
消息	700-899	1700-1899	3
可选部分	900-999	1900-1999	4

每个编号对应信息可以通过 PC-Lint 安装包中 msg.txt 文档进行查阅。语句错误: 普通编译器也能检测出的语法错误, 必须改正; 内部错误: 程序内部不应该出现的错误, 必须改正; 致命错误: 超过某个限制而导致的系统致命错误, 必须改正; 警告: 指出程序中很可能存在的错误; 消息: 可能存在的错误, 也有可能是个人编码风格导致被检测(其本身属于合法编程); 可选部分: 不会被默认检查, 需要在选项中指定。

### 3.4 CppCheck

CppCheck 是针对 C/C++ 的检测非语法错误的开源静态检测工具。一般作为编译器或者其他静态检测工具的补充。支持命令行、插件版、独立版三种检测代码缺陷方式。其中独立版操作较其他工具简单。报告类别为: 错误、警告、风格警告、可移植性警告、性能警告、信息消息。CppCheck 的基本命令行的形式为:

```
cppcheck --enable=all "d:\FreeBird\*.cpp"
```

--enable=all 表示全部启用以上 6 种报告类型, "d:\FreeBird\\*.cpp" 表示要测试的目标文件。

### 3.5 Coverity

Coverity 擅长精确查找源代码中最严重以及最难检测到的缺陷, 例如在检测 C/C++ 源代码时能发现并发、性能低下、导致崩溃的缺陷、不正确的程序行为、安全编码缺陷、隐含的缺陷等问题。支持 Windows、Linux 等操作系统, 支持 C/C++、C#、java、JavaScript、PHP、Python 在内的大多数编程语言。Coverity 拥有一套基础技术: 检测质量缺陷、潜在安全漏洞、测试冲突以及 java 运行时缺陷。分析源代码可以使用基于 GUI 的 Coverity Wizard, 也可以使用插件集成和命令行。Coverity Wizard 分析源代码的标准工作流为: 简介、编译器配置、捕获、分析、提交缺陷、查看结果。简介: 填写项目名称, 勾选分析选项(一般缺陷、安全缺陷、报告不充分的测试、确定测试优先级); 编译器配置: Coverity 内置的配置文件 coverity\_config.xml 自动配置通用编译器, 包含 gcc、javac、clangcc、javascript、py-

thon、php、ruby、swiftc、cl、csc, 可添加其他编译器; 捕获: 设置包含命令行构建(推荐)和 IDE 构建。分析: 设置分析选项, 运行分析; 提交缺陷: 将包含源代码和缺陷在内的分析结果提交给 Coverity Connect 服务器。查看结果: 使用网页浏览器查看已提交的缺陷, 并进行管理和分类<sup>[14]</sup>。Coverity 作为静态源代码分析领域的领导者, 在全球拥有很多用户。早在 2010 年底, 华为公司就部署了 Coverity 作为静态分析工具, 并且在一年内实现从小范围使用到全公司覆盖。

## 4 测试案例

文中静态测试实验是依据嵌入式开源的 Visual Studio 项目 FreeBird 展开的, 选用静态测试工具 C++Test、PC-Lint 进行检测。

### 4.1 C++Test

C++Test 静态测试实验选择 C++Test 的 Visual Studio 2008 插件集成方式。考虑到嵌入式软件对运行时检测的要求, 选择 C++Test 内置的 BugDetective 测试配置, 来识别运行时错误。测试结果如图 1 所示, 共显示 17 个测试结果, 可展开节点查看严重度、位置等具体信息(由于图中信息比较清晰, 在这里不进行列举)。

17 测试结果, 0 代码审查

- [7] >确保资源已释放 (BD-RES-LEAKS-1)
- [1] >避免在检查 null 之前解引用 (BD-PB-DEREF-2)
- [2] >避免始终判断为相同值的条件 (BD-PB-CC-2)
- [7] >避免空指针解引用 (BD-PB-NP-1)

图 1 BugDetective 测试结果

选择 Parasoft's Recommended Rules 测试配置, 来识别最容易造成严重结构错误的代码缺陷, 测试结果如图 2 所示。

75 测试结果, 0 代码审查

- [3] >为动态分配内存的类声明复制构造函数 (MRM-38-1)
- [7] >为动态分配内存的类定义了赋值运算符 (MRM-37-1)
- [10] >使用引用而非数值传递对象 (OPT-14-3)
- [4] >在构造函数中初始化所有成员变量 (INIT-06-1)
- [1] >如果类中含有虚拟函数, 那么它也应该包含析构函数
- [11] >将圈复杂度限制在 10 之内 (METRICS-18-3)
- [1] >异常抛出值, 捕获异常 (EXCEPT-02-1)
- [15] >绝不要从析构、释放分配以及 swap 函数抛出异常 (EXC)
- [2] >释放数组内存时 delete 必须使用括号 (MRM-36-3)
- [16] >降序声明成员变量 (OPT-13-3)
- [5] >饰构造函数允许使用 explicit 进行显示转换 (CODSTA-C)

图 2 Parasoft's Recommended Rules 测试结果

### 4.2 PC-Lint

PC-Lint 的静态测试同样选择了 Visual Studio 2008 插件集成方式。PC-Lint 作为一个命令行工具, 其检查结果的可视化体验比 C++Test 略差, 但代码缺陷信息描述很清晰, 可作为 C++Test 检测结果的核实与补充, 这里只展示一部分结果, 如图 3 所示。

```
.\UserLink.cpp(5): error 1927: (Note -- Symbol 'CUserLink::cpl
head=new MemberNode;
.\UserLink.cpp(6): error 1732: (Info -- new in constructor for
.\UserLink.cpp(6): error 1733: (Info -- new in constructor for
Clear();
.\UserLink.cpp(16): error 1551: (Warning -- Function may throw
```

图 3 PC-Lint 部分检测结果

error1927: 构造函数应该初始化所有成员变量。

error1732: 没有定义赋值运算符。

error1733: 没有声明复制构造函数。

error1551: 这样的异常需要在 try 块中被捕获,析构函数不应该抛出异常。

### 4.3 测试结果分析

考虑到 C++Test 和 PC-Lint 软件的测试重点和嵌入式软件的测试重点,在这里只对部分测试项进行统计对比,如表 3 所示。

表 3 实验结果

检测项	C++Test	PC-Lint
缓冲区溢出	5	0
数组越界	12	0
空指针引用	8	40
内存泄漏	13	9
判断值不变	2	0
非法异常抛出	16	27
不安全的线程	2	0
无法到达的函数	0	2
强制类型转换	0	4
变量隐式转换	0	2
变量未初始化	4	16

C++Test 选择运行时检测以及推荐规则两个测试项,PC-Lint 选择警告级别为 0、1、2 的测试项。从这里可以看出 C++Test 侧重运行时错误、资源泄漏等缺陷,PC-Lint 侧重非法异常抛出、无法到达的函数等代码逻辑问题。两者测试结果既形成对比又相互补充,减少漏报、误报的可能性,提高了测试效率,完善了嵌入式项目 FreeBird 的测试结果。

## 5 结束语

针对嵌入式软件的特点,得出嵌入式软件静态测

试的测试重点。对常出现的代码缺陷进行分析,总结能有效检测出这些代码缺陷的五种静态测试工具的功能侧重点和使用方法。通过实验分析得出适用于嵌入式软件静态测试的测试方案:使用 C++Test 和 PC-Lint 两种静态测试工具测试嵌入式软件,两者侧重点不同形成互补,能更全面、有效地检测出嵌入式软件中隐藏的代码缺陷。

### 参考文献:

- [1] MYERS G J. The art of software testing [M]. Canada: Word Association 2004.
- [2] 原义盈. 嵌入式软件堆栈溢出的静态测试方法研究 [D]. 北京: 北京交通大学 2011.
- [3] 林晨. 嵌入式箭载计算机控制软件测试关键技术研究 [D]. 上海: 上海交通大学 2014.
- [4] 吕文晶. 基于规则的嵌入式软件系统静态测试 [D]. 天津: 天津大学 2012.
- [5] 孟云秀. 基于 C/C++ 代码的静态检测技术分析与研究 [D]. 石家庄: 石家庄铁道大学 2015.
- [6] 侯成杰. 航天器 C 语言软件常见编程错误分析及检测方法研究 [J]. 空间控制技术与应用 2013 39(6): 53-57.
- [7] 张蕾. 嵌入式软件测试技术及工具的研究 [J]. 中国新技术新产品 2017(21): 36-37.
- [8] 孟云秀, 赵正旭. 基于源代码分析的软件静态测试 [J]. 河北省科学院学报 2013 30(2): 16-21.
- [9] LHEE K S, CHAPIN S J. Buffer overflow and format string overflow vulnerabilities [J]. Software Practice & Experience, 2010 33(5): 423-460.
- [10] 褚蕾. 基于静态源码分析的软件安全测试技术研究与实践 [D]. 成都: 电子科技大学 2010.
- [11] 杨颖, 周志飞, 钟理, 等. 嵌入式软件静态测试技术 [J]. 机车电传动 2017(1): 61-64.
- [12] 李昊. 基于 LLVM-Clang 的软件静态检测工具研究与实现 [D]. 西安: 西安理工大学 2017.
- [13] GIMPEL J. Software that checks software: the impact of PC-lint [J]. IEEE Software 2014 31(1): 15-19.
- [14] 姜文, 刘立康. 基于 Linux 环境的 C/C++ 软件重量级静态检查 [J]. 微型电脑应用 2016 32(5): 12-15.