C++与 Java 软件重量级静态检查

姜 文,刘立康
(西安电子科技大学 通信工程学院,陕西 西安 710071)

摘 要:为了保证 Windows 环境下 C/C++和 Java 软件产品质量,对 C/C++代码和 Java 代码进行静态检查非常重要。以 SVN 作为软件配置管理工具,将重量级静态检查工具 Coverity 和 Fortify 集成到持续集成工具 ICP-CI 上,分别对 C/C++代码和 Java 代码进行重量级静态检查。详细叙述了 Windows 环境下软件配置管理工具 SVN 客户端安装,SVN 版本库的代码更新,对 C/C++和 Java 代码分别进行 Coverity 和 Fortify 编译器的配置、编译脚本编写和在 ICP-CI 任务管理页面上配置检查任务。介绍了静态检查处理过程,分析了出现各种常见问题的原因并提出了相应的解决方案。给出了一个软件产品中 C/C++代码模块和 Java 代码模块的重量级静态检查应用案例。工作实践表明,进行重量级静态检查有助于及时发现并解决 C/C++和 Java 软件源代码的各种缺陷和安全漏洞,从而提高软件产品的质量和安全性。

关键词:Windows 操作系统;静态检查;持续集成;安全漏洞
中图分类号:TP311.56 **文献标识码**:A **文章编号**:1673-629X(2016)08-0017-07
doi:10.3969/j.issn.1673-629X.2016.08.004

Heavy-weight Static Checking of Software in C/C++ and Java

JIANG Wen,LIU Li-kang
(School of Telecommunication Engineering,Xidian University,Xi'an 710071,China)

Abstract:In order to ensure the quality of code in language C/C++ and Java based on Windows,it is very important for the static checking to code of C/C++ and Java. With SVN as configuration management tool,the heavy-weight static checking tools Coverity and Fortify are integrated into continuous integration tool ICP-CI,and static checking is to be done for both C/C++ and Java. The SVN client installation and the code updating of the SVN repository under Windows is described in details,and the Coverity and Fortify compiler configuration,the compiler scripts writing,and the inspection tasks configuration on ICP-CI task management page are conducted for both C/C++ and Java. The process of the static checking is introduced and the causes of problems are analyzed and the corresponding solutions are given. Finally the heavy-weight static checking applied in a case for both modules of code in C/C++ and Java in software product is introduced. Practice shows that the static checking is helpful to discover and solve all kinds of flaws of the code in C/C++ and Java timely,improving quality and safety for software.

Key words:Windows operating system;static checking;continuous integration;security vulnerabilities

0 引 言

Windows 操作系统是具有图形界面的操作系统,是完全的多任务操作系统,是使用最多且排名第一的操作系统。Windows 操作系统提供功能强大的应用程序编程接口。目前在 Windows 环境下流行的软件开发语言主要有 C++、C#和 Java。

在 Windows 环境下通常采用集成开发环境 Visual Studio 进行 VC/C++、C#程序设计。VC 编译器[1-2]通常位于 bin 目录中。cl.exe 是 VC/C++编译器。编译器产生通用对象文件格式(COFF)对象(.obj)文件。LINK.EXE 是将通用对象文件格式(COFF)对象文件和库链接起来以创建 32 位可执行(.exe)文件或动态链接库(DLL)的 32 位工具。C#编译器是 csc.exe。

在 Windows 环境下通常采用 Eclipse 进行 Java 程序设计。Java 编程需要下载安装 Java 的软件开发工具 JDK[3-4],JDK 目录的 bin 文件夹中包含 Java 编译器 javac.exe 和 Java 解释器 java.exe。Java 源程序都是扩展名为.java 的文本文件,编译后生成可执行代码,文

收稿日期:2015-12-12 **修回日期**:2016-04-05 **网络出版时间**:2016-08-01
基金项目:国家部委基础科研计划:国防预研基金项目(A1120110007)
作者简介:姜 文(1986-),女,工程师,硕士,CCF 会员,研究方向为图像处理与分析、数据库应用和软件工程;刘立康,副教授,研究方向为数字通信、图像传输与处理、图像分析与识别等。
网络出版地址:http://www.cnki.net/kcms/detail/61.1450.TP.20160801.0907.054.html

件名与源文件名相同,扩展名为. class。运行 java. exe 来执行该 Java 程序。

在 Windows 环境下的软件开发大部分都是基于 C/C++语言和 Java 语言。为了保证基于软件产品代码质量,检测软件源代码中存在的缺陷和安全漏洞[5-7]非常重要。文中采用 CodeCC(Code Check Center)工具压缩包进行检测。工具压缩包中包括静态检查工具 Coverity[8-11] 和 Fortify[12-15],采用静态检查方法检测源代码的缺陷和安全漏洞,将检查结果反馈给开发人员及时处理,从而提高软件的质量和安全性。

1 重量级静态检查工具

静态测试工具直接对代码进行分析,不需要运行代码,也不需要生成可执行文件。静态测试工具一般是对代码进行语法扫描,找出不符合编码规范的地方,根据某种质量模型评价代码的质量,生成系统的调用关系图等。通常把 Coverity 和 Fortify 称为重量级的静态检查工具。

1.1 Coverity Prevent

Coverity Prevent 是由美国 Coverity 公司开发的一款高性能静态检查软件。Coverity 公司提供最先进的、可配置的、用于检测软件缺陷和安全隐患的静态源代码分析解决方案。将基于布尔可满足性验证技术应用于源代码分析引擎,分析引擎利用其专利的软件 DNA 图谱技术和 meta-compilation 技术,综合分析源代码、编译构建系统和操作系统等可能使软件产生的缺陷。通过对构建环境、源代码和开发过程给出一个完整的分析,建立了高质量软件的标准。Coverity Prevent 是第一个能够快速、准确分析当今大规模(几百万甚至几千万行的代码)、高复杂度代码的工具。Coverity 解决了影响源代码分析有效性的很多关键问题:构建集成、编译兼容性、高误报率、有效的错误根源分析等。

Coverity Prevent 的主要功能包括源代码分析、缺陷管理、扩展工具(Coverity extend)、绘制软件 DNA 图谱。Coverity Prevent 是检测和解决 C、C++、Java 和 C# 源代码中严重缺陷的领先自动化方法。在 Windows 环境下,Coverity Prevent 支持集成开发环境 Visual Studio,支持 VC/C++编译器和 C#编译器 CSC. exe;支持集成开发环境 Eclipse,支持 Java 编译器 javac. exe。

1.2 Fortify SCA

HP Fortify 是目前全球最大的静态源代码检测厂商。静态代码分析器 Fortify SCA(Static Code Analyzer)是该公司开发的一款软件源代码缺陷静态测试工具。采用数据流分析引擎、语义分析引擎、结构分析引擎、控制流分析引擎、配置分析引擎和特有的 X-Tier 跟踪器,从不同的方面查看代码的缺陷和安全漏洞。

Fortify SCA 通过调用语言的编译器或者解释器把源代码转换成一种中间媒体文件 *. nst(Normal Syntax Tree),对应用软件的源代码进行静态分析。在分析过程中与它特有的软件安全漏洞规则集进行全面的匹配、查找,从而将源代码中存在的缺陷和安全漏洞扫描出来,并整理报告。扫描结果中不但包括详细的安全漏洞的信息,还会有相关的安全知识的说明,同时提供修复意见。

Fortify SCA 支持的编程语言多达 17 种,包括 C、C++、C#、Java 等,基本涵盖了绝大多数的应用,具有相当广泛的适用性。Fortify SCA 广泛的适用性使其适用于横跨多种语言的开发和测试。在 Windows 环境下,Fortify SCA 支持集成开发环境 Visual Studio,支持 VC/C++编译器和 C#编译器 CSC. exe;支持集成开发环境 Eclipse,支持 Java 编译器 javac. exe。

2 软件配置管理和持续集成工具

通常软件产品在开发阶段每天都要把源代码合入版本库中。为了保证软件源代码的质量,每周至少对合入版本库的源代码进行 2 ~ 3 次 Coverity 和 Fortify 检查。手动执行脚本调用 Coverity 和 Fortify 工具生成编译中间文件后,再上传到分析中心耗时长、效率低,不能及时提供检查结果。

2.1 基本的构建系统模型

现在通常采用持续集成工具来完成静态检查任务。持续集成[16]工具可以提供方便的集成平台,可以设置定时任务,使 Coverity 和 Fortify 检查任务能够充分利用非工作时间完成编译和静态分析等比较耗时的执行过程,保证及时得到分析结果,方便开发人员根据检查结果报告修改产品源代码缺陷。图 1 显示了一个基本的构建系统视图。

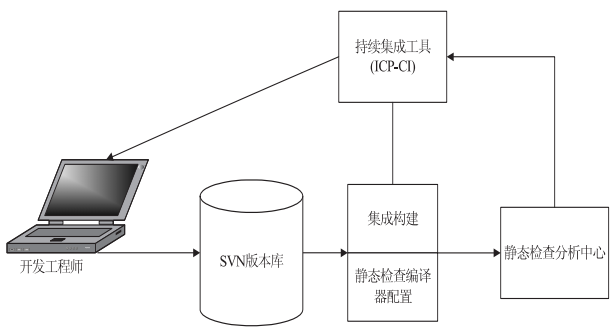

图 1 构建系统视图

构建流程如下:

(1)软件开发工程师编写或修改代码。

(2)开发工程师向软件配置管理(SCM)系统的版本库提交验证通过的源代码。

(3)持续集成工程师(CIE)编写脚本和静态检查编译器配置。

(4)持续集成工程师在持续集成工具的任务管理页面上配置构建任务和静态检查任务。

(5)持续集成工程师制定构建定时任务(通常是非工作时间)自动进行集成构建。

(6)编译中生成的中间文件上传到静态检查分析中心并运行静态分析工具。

(7)静态检查结果发送到持续集成工具。

(8)持续集成工具通过电子邮件向软件产品经理和开发工程师及时反馈构建结果和静态检查结果。

(9)软件开发工程师下载检查报告及时处理存在的各种问题。

文中采用的软件配置管理工具是 SVN,持续集成工具是 ICP-CI。由于使用的编译环境是 Windows,首先将集成了 Coverity 和 Fortify 工具的 CodeCC 工具的压缩包 CodeCC_Win32. zip 拷贝到持续集成主控服务器与代理服务器的 plugin 目录下解压,再分别对 Coverity 和 Fortify 工具进行相关的编译器配置,然后完成搭建构建工程。构建工程命名为:“产品名_版本号_CodeCheck”。

2.2　软件配置管理工具 SVN

2.2.1　Windows 环境下的 SVN 客户端安装

ICP-CI 服务器通常需要安装 SVN 命令行工具和图形 SVN 客户端工具 TortoiseSVN。安装完成命令行工具和图形客户端工具之后,可以选择安装支持中文操作的软件语言包。

完成 SVN 工具的安装之后,可以根据配置管理工程师提供的产品代码配置库路径,使用 SVN 客户端工具的“SVN 检出(check out)”功能从配置库下载代码到代理服务器的指定路径下。

2.2.2　SVN 版本库的代码更新

持续集成工具 ICP-CI 需要在版本库锁库之后完成源代码更新,ICP-CI 工具执行代码更新时,需要编写批处理脚本 Code_Update. bat,把脚本配置在任务中。更新代码的批处理脚本内容如下:

```
::配置需要更新代码的视图路径
SET CodePath=D:/Code
::解除代码视图中的文件锁定
svn cleanup % CodePath %
::更新代码到视图路径
svn update % CodePath %  --accept postpone
```

3　基于 C/C++的静态检查

Windows 环境下的 C/C++程序设计通常采用 Microsoft Visual Studio2010 集成工具,但是也有一些老的软件产品 C/C++编程采用 VC6.0 集成工具。对 C/C++代码的静态检查需要处理两种情况。

(1)Coverity 编译器配置。

软件产品模块进行 Coverity 检查时,需要在模块源代码进行编译的时候调用 Coverity 工具中的 cov_configure 命令。为了在编译过程中成功调用 cov_configure 命令,需要完成编译器的配置。

对于 Windows 环境下的 C/C++程序,编译器配置在 tqeconfig. ini 文件中。

编程使用的 VC6.0 编译器配置内容如下:

msvc = D: \Tools \Microsoft Visual Studio \VC98 \bin\cl. exe

编程使用的 Microsoft Visual Studio2010 编译器配置内容如下:

msvc=D: \Tools \Microsoft Visual Studio10. 0 \VC \bin\cl. exe

在配置过程中需要注意的是,cl. exe 为工具的安装路径,实际在配置过程中可以根据 Microsoft Visual Studio 工具的实际安装路径进行配置。

完成编译器配置后,执行 tqeconfig. bat 文件,该文件如果执行成功,则将生成相应的配置文件保存在 CodeCC\tool\coverity\config 目录下。在配置编译器时需要注意 tqeconfig. bat 脚本的执行结果,当执行窗口中提示执行成功,配置的编译器才能在 CodeCC\tool\coverity\config 目录下生成对应编译器的配置文件。

完成编译器配置后,需要将持续集成主控服务器和代理服务器上的 coverity 工具路径:plugin\CodeCC\tool\coverity\bin 路径添加到环境变量 path 中。

(2)Coverity 编译脚本编写。

Coverity 任务配置到 ICP-CI 上之后,通常由主控服务器将任务下发到代理服务器执行。执行 Coverity 任务需要编写相应的编译脚本,使 Coverity 任务执行时能够调用 cov-configure 来编译 Microsoft Visual Studio 的工程文件。

编程使用 VC6.0,编译脚本内容如下:

```
cd/d “D:\Code\Security_Versions\alm_src\Make PC”
“D:\Tools\Microsoft Visual Studio\Common\MSDev98\Bin\MSDEV. exe” “Make PC. dsw” /make “Make PC - win32 RELAESE” /clean
“D:\Tools\Microsoft Visual Studio\Common\MSDev98\Bin\MSDEV. exe” “Make PC. dsw” /make “Make PC - win32 RELAESE” /rebuild
```

编程使用 Microsoft Visual Studio2010,编译脚本内容如下:

```
cd/d “D:\Code\Security_Versions\CZLog”
“D:\Tools\Microsoft Visual Studio 10.0\Common7\IDE\devenv” “CZLog. sln” / clean
“D:\Tools\Microsoft Visual Studio 10.0\Common7\IDE\devenv” “CZLog. sln” /Rebuild “RELAESE | Win32”
```

(3)Fortify 编译器配置。

在 Windows 环境下,进行 Fortify 检查也需要区分两种不同的情况。

编程使用 Visual Studio 2010 时,首先需要在 Visual Studio 2010 IDE 上安装 HP Fortify 插件 HP_Fortify_SCA_and_Apps_4.00_windows_x86.exe,安装过程如下所示:

· 将 D:\ICP_CI_Windows_agent\plugins\CodeCC\tool 下的 fortify 文件夹备份为 fortify_bak,然后新建 fortify 目录。

· 双击 HP_Fortify_SCA_and_Apps_4.00_windows_x86.exe 进行安装,安装时将安装路径设置到在 ICP_CI 的 CodeCC 插件的如下路径:D:\ICP_CI_Windows_agent\plugins\CodeCC\tool\fortify。

· 在安装过程中,需要选择对应的 VS 版本,此处选择“VS2010”。

· 在安装过程中,还需要选择 license,license 文件需要选择 D:\ICP_CI_Windows_agent\plugins\CodeCC\tool\fortify_bak\fortify.license,其他安装弹出框选择“next”即可,安装完成之后将 fortify_bak 覆盖到 fortify 文件夹下。

· 安装完成之后需要打开 Visual Studio 2010 软件,查看插件是否已经安装(Visual Studio 2010 的菜单栏上有 HP Fortify 的选项即可)。

为了使 Fortify 工具可以识别和使用 C/C++程序的编译器,需要修改 Fortify 的配置文件,将 C/C++程序的编译器配置到 CodeCC\tool\fortify\core\config 目录下的 fortify-sca.properties 文件中。

编程使用 VC6.0 和使用 Visual Studio 2010 的编译器配置相同,如下所示:

```
com.fortify.sca.compilers.msdev = com.fortify.sca.util.compilers.DevenvAdapter
com.fortify.sca.compilers.devenv = com.fortify.sca.util.compilers.DevenvNetAdapter
com.fortify.sca.compilers.cl = com.fortify.sca.util.compilers.MicroSoftCompiler
com.fortify.sca.compilers.link = com.fortify.sca.util.compilers.MicrosoftLinker
```

编译器配置完成之后,需要将持续集成主控服务器与代理服务器上的 fortify 工具路径:plugin\CodeCC\tool\fortify\bin 添加到环境变量 path 中。

(4)Fortify 编译脚本编写。

进行代码编译时,需要将以前编译生成的过程文件与目标文件全部删除。Fortify 工具通过跟踪编译器生成中间文件 *.nst,如果编译过程中以前的过程文件与目标文件没有删除,Fortify 工具无法跟踪编译器生成正确的 *.nst 文件。

为了使 Fortify 工具通过跟踪的方式编译生成中间文件 *.nst,需要根据软件模块重新编写编译脚本和 makefile 文件,在编译脚本中嵌入 fortify 命令。在编译过程中需要调用钩子函数 sourceanalyzer.exe 文件,将编译器和链接器都挂在钩子上,从而生成中间文件 *.nst。

编程使用 VC6.0 时 fortify 编译脚本如下所示:

```
cd/d “D:\Code\Security_Versions\alm_src\Make PC”
“D:\Tools\Microsoft Visual Studio\Common\MSDev98\Bin\MSDEV.exe” “Make PC.dsw” /make “Make PC - win32 RELAESE” /clean
sourceanalyzer - b alm_help_tool_build - clean
sourceanalyzer - b alm_help_tool_build touchless“D:\Tools\Microsoft Visual Studio\Common\MSDev98\Bin\MSDEV.exe” “Make PC.dsw” /make “Make PC - win32release” /rebuild
```

其中,sourceanalyzer 表示 Fortify 工具的执行命令主体;build_id 表示 Fortify 的工程名(不能与 Fortify 的关键字相同)。

编程使用 Visual Studio 2010 时 fortify 编译脚本如下所示:

```
cd/d “D:\Code\Security_Versions\CZLog”
“D:\Tools\Microsoft Visual Studio 10.0\Common7\IDE\devenv” “CZLog.sln” / clean
sourceanalyzer - b CZLog - Xmx1024M
sourceanalyzer - bCZLog touchless - Xmx1024M “D:\Tools\Microsoft Visual Studio 10.0\Common7\IDE\devenv” “CZLog.sln” /Rebuild “release | Win32”
```

由于 Visual Studio 提供的编译器对 fortify 工具而言属于标准编译器,因此在编写 fortify 脚本时需要对调用 sourceanalyzer 这个钩子函数采用“无侵入”式集成[1-2]。“无侵入”集成对钩子函数在调用时有固定格式,具体形式如下:

```
sourceanalyzer - b <build_id> touchless make - f xxx.mak
```

(5)ICP-CI 的任务管理页面配置基于 C/C++的检查任务。

在 ICP-CI 的任务管理页面的构建工程上配置 CodeCC 检查任务,通常 Coverity 任务和 Fortify 任务同时配置。以基于 C/C++的软件模块 Alm 为例来描述集成过程。配置 Alm 模块的 CodeCC 任务时,在任务栏上选择“CodeCC”任务。

对于 Coverity 任务,将 Alm 模块的编译脚本 alm_help_update_coverity_build.bat 和 Alm 模块编译脚本路径配置到 CodeCC 任务类型页面下的编译脚本、编译路径中,选择编译类型为 CL,并在任务选项栏添加“Coverity”任务。对于 Fortify 任务,将 Alm 模块的 Fortify 编译脚本 alm_help_update_fortify_build.bat 以及 Alm 模块编译脚本路径配置到 CodeCC 任务的 forti-

fyexecutable 配置项中。最后在任务类型中添加“Fortify”任务。CZLog 模块在 ICP-CI 页面上的检查任务配置同上。

完成任务配置之后需要配置检查结果的发送人列表,主送人为该模块的开发工程师与持续集成工程师,抄送人为产品经理、各开发组长。

4　基于 Java 的静态检查

(1)Coverity 编译器配置。

对于 Windows 环境下的 Java 程序,将编译器配置在 tqeconfig. ini 文件中。配置内容如下:java = D:\Tools\jdk\bin\javac. exe。

配置过程也需要根据实际安装路径来完成。

完成编译器配置后,执行 tqeconfig. bat 文件,该文件如果执行成功,则将生成相应的配置文件保存在 CodeCC\tool\coverity\config 目录下。在配置编译器时需要注意 tqeconfig. bat 脚本的执行结果,当执行窗口中提示执行成功,配置的编译器才能在 CodeCC\tool\coverity\config 目录下生成对应编译器的配置文件。

完成编译器配置后,需要将持续集成主控服务器和代理服务器上的 coverity 工具路径:plugin\CodeCC\tool\coverity\bin 路径添加到环境变量 path 中。

(2)Coverity 编译脚本编写。

执行 Coverity 任务时,需要编写相应的编译脚本,使 Coverity 任务执行时能够调用 cov-configure 来编译 Java 的工程文件。脚本内容如下:

```
<? xml version="1.0" encoding="UTF-8"? >
<project name="Nao" default="run" basedir=".">
<property name="src" value="src"> </property>
<property name="dest" value="classes"> </property>
<property name="NaoJar" value="NaoJar"> </property>
<property name="libdir" value="."> </property>
<target name="clean">
<delete dir="${dest}"/>
</target>
<target name="init" depens="clean">
<mkdir dir="${dest}"/>
</target>
<path id="compile.classpath">
<fileset dir="./lib">
<include name="*.jar"/>
<include name="startuploader.jar"/>
</fileset>
<fileset dir="./plugins/com.cn.ao/lib">
<include name="*.jar"/>
</fileset>
</path>
<target name="compile" depends="init">
<javac includeantruntime="on" srcdir="${src}" destdir="${dest}" fork="true">
<classpath refid="compile.classpath"/>
</javac>
</target>
<target name="build" depends="compile">
<jar jarfile="${NaoJar}.jar" basedir="${dest}"/>
</target>
<target name="run" depends="build">
</target>
</project>
```

(3)Fortify 编译器配置。

在 Windows 环境下,为了使 Fortify 工具可以识别和使用 Java 程序的编译器,需要修改 Fortify 的配置文件,将 Java 程序的编译器配置到 CodeCC\tool\fortify\core\config 目录下的 fortify-sca. properties 文件中。

Java 语言的编译器配置如下所示:

```
com.fortify.sca.compilers.javac=com.fortify.sca.util.compilers.javacCompiler
```

编译器配置完成之后,需要将持续集成主控服务器与代理服务器上的 fortify 工具路径:plugin\CodeCC\tool\fortify\bin 添加到环境变量 path 中。

(4)Fortify 编译脚本编写。

进行代码编译时,需要将以前编译生成的过程文件与目标文件全部删除。Fortify 工具通过跟踪编译器生成中间文件 *. nst,如果编译过程中以前的过程文件与目标文件没有删除,Fortify 工具无法跟踪编译器生成正确的 *. nst 文件。

为了使 Fortify 工具通过跟踪的方式编译生成中间文件 *. nst,需要根据软件模块重新编写编译脚本和 makefile 文件,在编译脚本中嵌入 fortify 命令。在编译过程中需要调用钩子函数 sourceanalyzer. exe 文件,将编译器和链接器都挂在钩子上,从而生成中间文件 *. nst。

fortify 编译脚本如下所示:

```
cd/d "D:\Code\Security_Versions\nao_src"
sourceanalyzer - b Nao - Xmx1024M ant -f "D:\Code\Security_Versions\nao_src\build.xml"
```

其中,在调用钩子函数 sourceanalyzer 时,加上 -Xmx1024M 指令是为了避免分析中间文件时因内存不足导致报内存溢出的错误,在实际操作过程中可以根据编译生成中间文件的代理服务器的系统内存,自行设置这个值。如果代理服务器的内存不能满足,可以考虑增加内存、更换内存较大的代理服务器来完成中间文件编译。

(5)ICP-CI 的任务管理页面配置基于 Java 的检查任务。

以基于 Java 的软件模块 Nao 为例来描述集成过程。配置 Nao 模块的 CodeCC 任务时,在任务栏上选择“CodeCC”任务。对于 Coverity 任务,将 Nao 模块的编译脚本 build. xml 和 Nao 模块编译脚本路径配置到 CodeCC 任务类型页面下的编译脚本、编译路径中,选择编译类型为 JAVAC,并在任务选项栏添加“Coverity”任务。对于 Fortify 任务,将 Nao 模块的 Fortify 编译脚本 nao_fortify_build. bat 以及 Nao 模块编译脚本路径配置到 CodeCC 任务的 fortifyexecutable 配置项中。最后在任务类型中再添加“Fortify”任务。

完成任务执行脚本配置之后需要配置检查结果的发送人列表,主送人为该模块的开发工程师与持续集成工程师,抄送人为产品经理、各开发组长。

5 检查结果分析和处理

5.1 CodeCC 检查处理过程

ICP-CI 工具通常先对模块做 Coverity 检查,生成的中间文件压缩包上传到指定的分析服务器;接着对模块做 Fortify 检查,同样将生成的中间文件压缩包上传到同一个分析服务器。此时 ICP-CI 的执行窗口显示 CodeCC 任务成功并处于等待分析结果状态。等待分析结果的时间长短取决于生成的中间文件的大小以及分析服务器的忙碌程度。

当分析服务器分析完毕,将模块的分析结果回传到 ICP-CI 工具,在 ICP-CI 工具的页面上可以看到 Coverity 和 Fortify 工具各自的检查结果。检查结果包括模块的缺陷数以及总缺陷数,缺陷级别。缺陷级别分为高、中、低 3 个级别。同时根据检查模块任务配置的邮件主送人和抄送人,给这些人发送邮件。邮件内容为该检查模块的 Coverity 和 Fortify 检查日志与检查结果下载路径。

对检查出来的各种问题,需要开发人员下载检查结果文件,并对检查结果与模块源代码进行分析。确认是源代码问题,修改源代码后重新合入版本库,启动新一轮的 CodeCC 检查;根据新的检查结果确认代码缺陷是否已经被解决,已经解决掉的缺陷呈现的状态为“Fixed”,呈现状态为“new”缺陷数量减少。分析之后,确认是误报的缺陷,从 ICP-CI 上显示的 Coverity 和 Fortify 工具检查结果页面的“Ignore defects”链接进入由分析服务器指定的缺陷库完成误报缺陷的屏蔽,屏蔽之后的缺陷呈现为“Dismissed”状态。当执行下一次 CodeCC 检查时,分析结果的缺陷数也会减少。

5.2 CodeCC 检查失败的分析和处理

进行 CodeCC 检查时,难免会出现失败的情况。CodeCC 检查失败需要及时发现处理,根据已经失败的模块、构建工程页面上提示失败的信息和构建工程的详细日志文件,来确定该模块 CodeCC 检查失败的原因,并确定解决问题的方案。

5.2.1 Coverity 或 Fortify 检查在编译阶段出错

在编译阶段出错,查看对应的编译日志可以发现各种问题(编译器配置问题、编译脚本问题或源代码编译错误等),导致在编译阶段 Coverity 或 Fortify 检查报错。

解决方法如下:

(1)在 tqeconfig. ini 中配置相应编译器,并执行 tqeconfig. bat 文件完成配置,或在 fortify-sca. properties 文件中完成配置。

(2)根据日志所报编译问题,重新编写编译模块的编译脚本。

(3)开发工程师定位模块编译错误,修改后的源代码合入版本库之后重新执行模块的 CodeCC 检查任务。

5.2.2 Coverity 或 Fortify 检查在分析阶段出错

在分析阶段出错,查看对应的分析日志可以发现大部分是分析服务器问题导致的执行失败,通常表现为上传中间文件压缩包失败、分析结果回传失败等。根据分析日志发现此类问题,需要联系 CodeCC 工具运维工程师和相关人员解决分析服务器问题。

5.2.3 CodeCC 检查文件的比例问题

此类问题需要查看 Coverity 检查的编译日志文件 build. log,在该文件的结尾部分查看模块编译检查文件的百分比,如果模块编译的文件都完成检查,那么检查文件的百分比应该是 100%。如果这个百分比值小于 100%,需要查找错误。通常这些错误不是产品模块的代码编译错误,而是 Coverity 检查时产品模块源代码生成中间文件过程中与编译器冲突导致的。为了提高检查的文件比例,需要开发工程师与系统工程师进行分析,尽可能通过修改的源代码进一步提高检查文件的比例。

6 典型案例

某公司有一个软、硬件结合的大型开发项目。采用的软件配置管理工具为 SVN,持续集成工具为 ICP-CI。1 台主控服务器和 8 台代理服务器参与 CodeCC 检查。Windows 操作系统的版本是 Windows 7。采用 Visual Studio 2010 进行 C++程序设计。采用 Eclipse 3.6 进行 Java 程序设计。采用的 Coverity 工具版本为 6.5.3;Fortify 工具版本为 6.00.0096。集成到 ICP-CI 工具后,使用 Shell 脚本和 ANT 脚本完成对软件模块的 Coverity 检查和 Fortify 检查。

以 C++的 Alm 模块与 CZLog 模块为例,对该模块进行检查的扫描结果如表 1 所示。

以 Java 的 Nao 模块为例,对该模块进行检查的扫　　描结果如表 2 所示。

表 1　对 Alm 工具进行 Coverity 检查和 Fortify 检查的扫描结果

模块名称	告警总数	Coverity 告警			Fortify 告警		
		高级别告警	中级别告警	低级别告警	高级别告警	中级别告警	低级别告警
Alm	951	245	170	123	118	202	93
CZLog	1 065	303	200	115	213	56	178

表 2　对 Nao 工具进行 Coverity 检查和 Fortify 检查的扫描结果

模块名称	告警总数	Coverity 告警			Fortify 告警		
		高级别告警	中级别告警	低级别告警	高级别告警	中级别告警	低级别告警
Nao	688	23	46	12	12	240	355

检查结果报告中分别给出了 Coverity 检查和 Fortify 检查的告警级别、告警数量,并详细列举了缺陷的类型与种类。开发工程师通过下载检查结果报告,对产品源代码进行分析和处理。经过对 Alm 模块和 Nao 模块的代码优化、重构以及误报告警屏蔽,最终将 Alm 模块和 Nao 模块的告警数目清零,修正了模块源代码存在的缺陷,改进了模块的源代码质量。

工作实践表明,重量级静态检查有助于及时发现并解决 C/C++与 Java 软件源代码的各种缺陷,便于产品主管了解工作进度和解决存在的问题,进一步提升产品质量。

7　结束语

长期的工作实践表明,在 Windows 环境下进行 C/C++和 Java 软件开发,Coverity 和 Fortify 组成的重量级静态检查工具在开发过程中发挥了重要作用。检查工具集成到持续集成工具 ICP-CI,可以自动完成检查,快速地向软件开发人员反馈检查结果,使软件开发人员能够及时修复源代码的缺陷,同时也给项目管理提供了很好的保证。在软件开发过程中做好静态检查工作将在很大程度上提高产品的质量,降低软件开发的成本。

参考文献:

[1] 白　乔,左　飞. 把脉 VC++[M]. 北京:电子工业出版社,2009.

[2] 张立勇,刘　坚,陈　平. C/C++程序的静态安全分析[J]. 系统工程与电子技术,2008,30(6):1155-1158.

[3] 张　峰. Java 程序设计与项目实战[M]. 北京:清华大学出版社,2011.

[4] 赵　平. JAVA 源代码静态分析系统的设计与实现[D]. 长春:吉林大学,2013.

[5] Stiehm T,Gotimer G. Building security in using continuous integration[J]. The Journal of Defense Software Engineering,2010,23(2):24-27.

[6] 路晓波. 软件开发过程中白盒测试方法和工具的研究及应用[D]. 南京:南京邮电大学,2012.

[7] 吴世忠,郭　涛,董国伟,等. 软件漏洞分析技术[M]. 北京:科学出版社,2014.

[8] Almossawi A,Lim K,Sinha T. Analysis tool evaluation:coverity prevent[R]. [s. l.]:Carnegie Mellon University,2006.

[9] Coverity® 6. 6deployment guide planning, installation, tuning,and supported platforms[R]. [s. l.]:Coverity Inc,2013.

[10] Coverity® scan:2013 open source report[R]. [s. l.]:Coverity Inc,2014.

[11] Coverity® scan open source report 2014[R]. [s. l.]:Synopsys Inc,2015.

[12] Fortify source code analyzer 用户指南(版本 5. 1)[M]. [s. l.]:Fortify Software Inc,2008.

[13] Fortify SCA user guide fortify 360 version 2. 6[M]. [s. l.]:Fortify Software Inc,2010.

[14] Blay P,Corlett S. Fortify SCA performance guide[M]. [s. l.]:Fortify Software Inc,2014.

[15] Fortify source code analysis suite tutorial[M]. [s. l.]:Fortify Software Inc,2010.

[16] 罗时飞. 敏捷持续集成(CruiseControl 版):高效研发之道[M]. 北京:电子工业出版社,2008.