

基于错误传播分析的软件脆弱点识别方法研究

李爱国¹⁾ 洪炳镕¹⁾ 王 司²⁾

¹⁾(哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

²⁾(哈尔滨工业大学航天学院 哈尔滨 150001)

摘 要 在太空环境中,软件系统经常受到各种辐射现象的影响.在此类环境下,寻找软件脆弱点主要是考虑环境扰动对该软件的影响.文中提出了一种由环境扰动引入的软件脆弱点的分析方法.首先在对软件系统模块化的基础上,通过在两个层面上分析错误在软件中的生成及其传播过程,给出寻找软件脆弱点的理论框架,随后进一步给出该框架中一些参数的实验估计方法,最后给出该框架在某卫星光纤陀螺捷联航姿控制系统上的应用.应用结果令人满意.

关键词 软件脆弱点;错误传播;故障注入;环境扰动;单粒子效应

中图法分类号 TP302

An Approach for Identifying Software Vulnerabilities Based on Error Propagation Analysis

LI Ai-Guo¹⁾ HONG Bing-Rong¹⁾ WANG Si²⁾

¹⁾(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001)

²⁾(School of Astronautics, Harbin Institute of Technology, Harbin 150001)

Abstract For the software system operating in space environment where radiation phenomenon exists widely, identifying the vulnerabilities emphasizes particularly on the analysis about the effect of environment on the software. This paper presents a methodology for analyzing the vulnerabilities in software subjected to environment perturbation. Based on the premise that the software has been modularized, this methodology analyzes the error-generation and error-propagation process in software from signal and module level each, as a result of giving a theory framework for identifying software vulnerabilities. Whereafter, a fault-injection-based method for estimation of the various measures in the framework is described and the software of a real embedded control system used in a satellite is analyzed to show the type of results obtained by the methodology.

Keywords software vulnerability; error propagation; fault injection; environment perturbation; single event upset

1 引 言

广义上,软件脆弱点可被定义为软件系统中一

个不好的特性或部分,该特性可能导致潜在威胁,这种威胁可能是恶意的或是能对计算机系统相关资源产生不期望的影响的^[1].该定义可从两个方面来理解:(1)在软件系统中存在着某些固有的弱点或缺陷,

它们的执行能显式或隐式地造成对系统的破坏; (2) 软件系统中的各个部分或单元对环境扰动的抵御能力各不相同, 对环境扰动最敏感的部分也称之为软件脆弱点。

第一种情况是指软件代码内部自身的缺陷。该类缺陷可被一些恶意的用户利用来引发各种安全问题, 如获取非授权权限, 干扰关键进程的执行等。这一类问题在互联网应用程序上更加突出。对该类脆弱点, 目前已发现了多种类型, 如缓冲区泄露^[2]、格式字符串^[3]、条件竞争^[4]、SQL 注入^[5]等。针对该类脆弱点, 国内外许多学者也提出了很多定位和识别方法^[6-9], 其中包括侵入分析^[10]、静态分析^[11-12]、模型校验等^[13-14], 同时对脆弱点分类也作了大量工作^[15-16]。

对于运行于恶劣环境中的软件系统, 第二种情况是一种相当严重和普遍的现象, 这种环境包括电磁干扰、电压波动、超高/低温环境、辐射或高能粒子等。在此种环境下, 即使软件系统本身没有任何缺陷(基本不可能), 也可能出现用户和设计者意想不到的情况, 其中大多数都是由于环境扰动所引入的错误造成的。而且由于深亚微米技术的应用, 电子系统具有非常高的操作频率和集成度以及较低的供电电压, 这也导致此类系统对环境的敏感度随之增加, 环境影响不仅可使存储单元产生各种瞬态故障, 而且电路的组合逻辑部分也会受到瞬态故障的影响^[17], 此种现象称为“单粒子瞬变”(Single Event Transient)。环境影响的作用不仅仅局限在太空环境中, 在地球表面也能观察到这种效应^[18]。

此种影响产生的故障一般都为瞬态故障(又称软故障), 它与电子产品的生产或设计故障不同, 它不连续发生, 由于是由外部事件引起, 所以具有一定的随机性。它一般不会对处理器或其它单元造成永久的物理伤害, 但却可以改变信号的传输或存储单元的值, 致使程序的执行发生错误。瞬态故障已经造成了很多的严重后果。我国在 1995 年 2 月 8 日发射升空的“实践四号”卫星上搭载的两套用于单粒子事件测量的监测装置, 在入轨后的 19 天内共发生了 65 次翻转。“风云一号(B)”气象卫星也因多次 SEU 事件导致姿态控制系统失控而过早的失效^[19]。2000 年, Sun 微系统公司声明: 由于宇宙射线干扰缓存而使一些主要的用户站点的服务器发生了崩溃, 其中包括美国在线(America Online), eBay 和 10 个其它站点^[20]。

为解决环境干扰对软件执行的影响, 特别是在

高可靠性的航空航天领域, 要使软件系统在有故障或错误存在的条件下仍能良好地运行, 我们必须为软件系统装配容错机制, 如果能事先找到软件系统中的脆弱点, 则可使容错机制的“成本效率”达到最高。

与第一种情况比, 在解决环境扰动对软件影响方面的工作还相对较少。Du 等人在文献[7]中提出了一种使用环境扰动来探测软件脆弱点的方法, 他们把外界环境扰动看成故障, 如果产生了安全隐患, 则看成是软件系统对该故障的容错失败, 并据此提出了环境-应用程序相互作用故障模型(Environment-Application Interaction, EAI)。但该方法的目的还是为寻找软件系统内部固有的缺陷。在文献[21]中, Hiller 等人提出了一种用于分析软件系统中数据错误传播的方法。他们在定义了错误渗透率的基础上提出了一套相应的度量参数, 这些参数可用于分析软件中的脆弱点。但它仅针对模块间信号上数据流的错误传播, 而没有考虑模块本身由于外界干扰因素所引入的错误特性。

本文提出了一种用于寻找软件系统中脆弱点的新方法, 该方法通过分析错误在软件中的生成及其传播来发现系统中最“脆弱”的部分, 它可以用于软件开发的后期或用于软件的可靠性评测过程中。该方法不仅考虑了错误在信号级别上的传播过程, 还考虑了软件本身由于环境干扰而引入的错误生成特性, 这通常包括数据流错和控制流错两种错误类型, 这一点和文献[21]有所不同。

2 软件系统与环境模型

软件系统通常包括多个相互作用的模块, 一个软件模块可以看作是拥有多个输入和输出的广义黑箱, 模块之间通过各种不同的方式进行联接, 如消息传递、参数传递、内存共享等。一个软件模块使用所提供的输入进行计算来产生输出, 在较低级的层次上, 一个模块可以是一个过程或一个函数, 也可以是程序中一个基本模块或一个过程和函数内部的特定的代码片段。多个这样的模块组成一个系统并通过联接器(通常称之为信号)互相联接, 如图 1 所示。当然, 此系统也可以看作是一个更大系统的子模块。信号可从模块内部产生(如作为计算结果)或从外部硬件系统产生(如从寄存器读取数据的传感器)。信号的目的地可以是系统内部(如作为模块输入集合的一部分)也可以是外部(如放置在硬件寄存器上的值)^[21]。

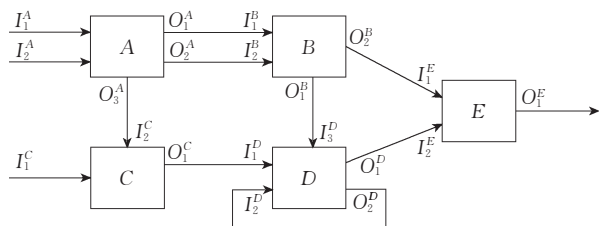


图 1 一个 5 模块的软件系统示例

此种模块化的软件系统中,当输入信号或中间信号发生错误时,该错误很可能会经各模块传递,进而导致系统输出信号出现错误.虽然模块以及它们之间联接器的形式多种多样,但对于本文讨论的问题,模块以及它们之间联接器的具体形式可以不予考虑,而只把模块建模成可把输入激励和内部状态映射为一新的状态和输出的函数集,联接器建模成两模块之间可传递的信号集合.

通常,环境对软件系统的执行过程的影响有两种方式,一是软件系统从环境中接受输入,则环境中存在的错误可能进入软件系统的输入信号中,该错误又可能进一步引起软件内部状态发生变化.由于环境中的错误可通过软件内部状态进行传播,如果软件系统不对此错误进行正确处理,就可能导致系统失效或产生安全隐患.例如,如果一程序从传感器接收输入,则传感器中的任何错误都可能经由对应的输入传递到程序内部状态中,当此程序无校验地使用该输入进行计算时,就可能引发像“超出范围”这样的问题.

环境影响软件系统的另一种方式是环境直接改变软件的内部状态(如辐射),当软件不加处理地在此种环境中运行时,通常都会引发安全问题.例如在某一程序中,假设该程序对某一变量进行了一个赋值,当程序再次使用该变量时有可能出现以下情况:一是该值还是先前赋予的值,没有变化,在这种情况下,变量读操作和其后的程序运行是安全的.另一种可能是由于环境影响了程序的内部状态而导致该值已经不是先前所赋予的值(此时此程序运行者还假设该值没有变化),如果程序不对此错误进行处理就会由于运行状态错误导致安全问题的出现.

3 错误传播特性

本节主要从两个层面对错误传播进行描述:

(1) 信号级上的错误传播.它主要考虑变量或信号中数据错误的传播过程,和环境影响软件系统的第一种方式相对应;(2) 模块级上的错误传播.它不仅

要考虑数据错误的传播问题,还要考虑模块自身的错误生成问题,它对应环境影响软件系统的第二种方式.

3.1 信号级上的错误传播

定义 1(信号错误渗透率). 在模块 A 的输入信号 i 发生错误的条件下,致使模块 A 对模块 B 的输出信号 k 出现错误的概率称为信号 i 对信号 k 的错误渗透率,记为 $PEP_{i,k}^{A \rightarrow B}$.

$$PEP_{i,k}^{A \rightarrow B} = Prob(\text{输出信号 } k \text{ 时发生的错误} |$$

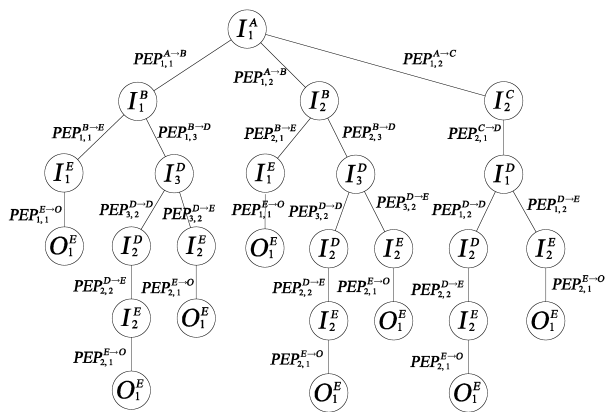
$$\text{输入信号 } i \text{ 时发生的错误}) \quad (1)$$

此定义主要表示某一模块的输入/输出对对发生在输入上错误的可透过性,它与输入端错误出现概率的大小无关,而与模块的负载以及在输入端可能出现的错误类型有关.如果一输入/输出对的错误渗透率为 0,这并不说明错误没有引起任何破坏,而是它可能在模块的内部状态中引发了一个潜伏的错误,只不过还没有在输出端表现出来.

基于信号错误渗透率,可以分析某一信号对系统输出的影响.由于信号中的错误可沿着许多不同的路径传播至系统输出,所以为了计算某一信号 S 对系统输出 O^{sys} 的影响,必须先构造该信号的传播树.传播树的生成算法如下^[21]:

1. 选择一个系统输入信号或中间信号,把它作为传播树的根节点.
2. 确定该信号进入哪个模块,并把该模块的每个输出都作为根节点的子节点.
3. 对于每个子节点,如果对应的信号不是系统输出信号,则跟踪该信号进入它的射入模块并确定该模块的输出信号,然后从步 2 开始分别构建这些子信号的子树;如果对应的信号是系统输出信号,则该节点就成为此传播树的叶子节点.如果某一模块有反馈输入,则只考虑一次反馈(不考虑此反馈带来的递归作用),并产生其余输出的子树.
4. 如果有其它的系统输入信号,返回步 1.

使用该算法可以为每个输入信号各生成一对应的传播树,其中根节点代表系统输入,叶子节点代表系统输出,中间节点代表系统内部输入,每一条边都对应一个信号错误渗透率值,这样从根到叶子之间的具有最高权重的路径就是错误最有可能的传播路径.对于图 1 中的软件系统,输入 I_1^A 的传播树如图 2 所示,其中每一条路径的错误传播概率就是把沿该路径的所有错误渗透率值相乘,如出现在 I_1^A 中的错误由 I_1^B 传至模块 E 的输入 I_1^E 再到输出的概率就是 $P = PEP_{I_1^A \rightarrow I_1^B}^{A \rightarrow B} \cdot PEP_{I_1^B \rightarrow I_1^E}^{B \rightarrow E} \cdot PEP_{I_1^E \rightarrow O_1^E}^{E \rightarrow O}$.如果已知错误在输入 I_1^A 中出现的概率为 $Pr(I_1^A)$,那么可以调整 P 为 $P' = Pr(I_1^A) \cdot P$.



式(9)所示.

$$\begin{aligned}
 M \rightarrow O^{\text{sys}} &= 1 - \prod_i (1 - (M \rightarrow O_i^{\text{sys}})) \\
 &= 1 - \prod_i (1 - (1 - \prod_k (1 - PEL_k^M \cdot \\
 &\quad PA^M \cdot (O_k^M \rightarrow O_i^{\text{sys}}))) \\
 &= 1 - \prod_i \prod_k ((1 - PEL_k^M \cdot PA^M \cdot \\
 &\quad (O_k^M \rightarrow O_i^{\text{sys}}))) \quad (9)
 \end{aligned}$$

$$\begin{aligned}
 M \rightarrow O^{\text{sys}} &= 1 - \prod_i \prod_k ((1 - PEL_k^M \cdot PA^M \cdot \\
 &\quad C_{O_i^{\text{sys}}} \cdot (O_k^M \rightarrow O_i^{\text{sys}}))) \quad (10)
 \end{aligned}$$

和信号影响因子类似,如果考虑系统输出信号的重要程度不同,则式(9)可变为式(10).其中, PEL_k^M 为模块 M 对第 k 个输出 O_k^M 的泄漏率, PA^M 为模块 M 的活动率, $O_k^M \rightarrow O_i^{\text{sys}}$ 为模块 M 的第 k 个输出 O_k^M 对系统的第 i 个输出 O_i^{sys} 的影响因子, $C_{O_i^{\text{sys}}}$ 为系统输出 O_i^{sys} 的重要度参数.

我们讨论的模块级的错误传播是在信号级传播分析的基础上,又考虑了软件本身由于环境因素所引入的错误特性(包括控制流错误及数据错误).在已知信号传播和模块泄漏率及活动率的前提下,通过该错误传播分析可得出如下信息:(1)可计算得出错误渗透率较大的模块;(2)可知自身容易产生错误的模块(高泄漏率);(3)通过计算影响因子可得出对系统输出影响较大的模块.这些信息对于识别软件系统的“脆弱点”及对软件系统进行“加固”具有重要的指导意义.

3.3 识别软件脆弱点

根据上面得到的错误传播分析结果,我们可以识别出软件系统中较“脆弱”的部分.下面给出一些识别规则或建议:

(1)模块的影响因子越大,从该模块流出的错误造成系统危害的可能性就越大.这样,给影响因子较高的模块装配错误探测和恢复机制的代价就会较高.对信号影响因子的分析与之类似.

(2)模块的泄漏率越高,在环境扰动存在的情况下该模块产生错误的概率就越大.这样,对高泄漏率模块进行容错处理比对低泄漏率模块进行容错处理的成本效率要高.

(3)模块的错误渗透率越高,该模块包容错误的能力就越低,这就导致后继模块暴露给错误的概率增大.所以对高渗透率模块增加容错能力就会有较高的成本效率.当应用这些规则来确定软件的脆

弱点时,它们各自得出的结果并不完全吻合.例如一个模块可能有较低的泄漏率,但却有很高的影响因子.低泄漏率表明该模块产生错误的概率较低,而高影响因子表明一旦错误发生,引起破坏的概率较大,所以可以视低泄漏率、高影响因子的模块为系统脆弱点.

通常,一个软件项目可能会规定一些必须满足的条件,这会帮助我们确定软件中的哪一部分是“脆弱”的,需要得到加强.如可能规定软件所有模块的泄漏率不能超过某一上限,如果某一模块超过了这个限制,那就表明该模块需要增强错误处理能力.影响因子和渗透率与之类似.

本文中的错误传播和错误影响分析可作为其它系统可靠性分析活动,如故障模式影响分析(Failure Mode and Effects Analysis, FMEA)或故障模式影响及危害性分析(Failure Mode, Effects and Criticality Analysis, FMECA)的一种补充.在一个重复的开发过程中,每次循环过程中的这种软件剖面分析可能会指出哪些模块和信号在本次循环过程中需要特别的重视,这样错误传播与影响分析作为系统分析方法和资源管理手段在一些系统的开发过程阶段就会成为一个有用的工具.

本文中的脆弱点分析与 FMEA 和 FMECA 分析也有所不同. FMEA 是分析系统中每一产品所有可能产生的故障模式及其对系统造成的所有可能影响,并按每一个故障模式的严重程度、检测难易程度以及发生频度予以分类的一种归纳分析方法. FMECA 是故障模式影响分析(FMEA)和危害性分析(Criticality Analysis, CA)的组合分析方法.它们实际上是一组系列化的活动,其过程包括:找出产品/过程中潜在的故障模式;根据相应的评价体系对找出的潜在故障模式进行风险量化评估;列出故障起因/机理,寻找预防或改进措施.但 FMEA 一般都是针对组件的单个失效进行风险评估,而且都是针对某一具体的故障模式.本文中的方法考虑某一组件所有可能的故障综合在一起的效应,且对某一具体的故障模式的影响不作分析,而且是从组件对错误的传播和生成两个角度来分析脆弱点的.

本节在引入错误传播分析框架基本概念的基础上,阐述了错误在软件中的传播过程及定量的分析方法.只要已知信号的错误渗透率、模块的泄漏率及活动率,就可以通过该错误传播框架获得我们需要的信息.

4 参数的实验估计方法

对于错误传播框架中的几个基本参数,如渗透率、泄漏率和活动率,由于影响因素多且复杂(如错误类型、运行环境等),我们很难从理论上计算得到它们的精确值,这里我们给出基于故障注入的各参数的实验估计方法.为了分析原始的实验数据,我们使用黄金运行结果,黄金运行是系统在没有任何注入的情况下的运行,其结果被认为是正确的并作为一种参考基准.下面给出错误渗透率(EPE)和错误泄漏率估计算法(ELRE),它们动态执行程序,在执行过程中自动注入故障并决定估算过程何时结束.

信号错误渗透率估计算法(EPE)的基本原理就是根据信号渗透率的定义(定义 1),通过故障注入工具往模块某一输入信号中注入特定的故障(实际可能发生的故障),同时观察该模块的某一输出信号是否与黄金运行结果相同,如果不同,则认为该注入故障发生了传播.经过反复大量地注入实验来统计故障发生传播的次数占总的注入次数的百分比,即该信号对的错误渗透率值.模块错误泄漏率估计算法(ELRE)的原理与 EPE 算法相似,也是根据模块泄漏率定义(定义 4),通过故障注入实验来得到统计结果,只不过故障是注入进模块的状态空间中,在本文中,模块的状态空间特指在执行该模块的过程中所有 CPU 寄存器、堆栈空间以及该模块所占用的数据空间(包括代码段部分和数据段部分)的内容.

假设目标程序为 P , M 代表 P 中的所有模块的集合, I 代表某一模块 M' 的输入信号集, O 代表模块 M' 的输出信号集, S 代表模块 M' 的状态空间, E 代表可能被注入的故障集合.

算法. EPE.

1. 对 M 中的每个模块 M' , 执行步 2~步 10;
2. 进行目标程序的黄金运行,并在运行期间记录模块 M' 的输出信号集 O ,我们称此值集为 G ;
3. 把 $count$ 和与模块 M' 的每个输出信号 y 对应的 $errorNumber_y$ 计数器初始化为 0;
4. 对 I 中的每个输入信号 x , 执行步 5~步 10;
5. 执行目标程序,并当程序运行至模块 M' 的入口处时,从 E 中随机选择一故障并把它注入 x ;当程序运行至 M' 的出口时,记录 M' 的输出集 O ,我们称此值集为 R ,同时递增 $count$;
6. 对于 M' 的每个输出信号 y , 比较它在 G 和 R 中的值,如不等则递增对应的 $errorNumber_y$;
7. 重复执行步 5~步 6 n 次,这里 n 为需要注入错误的

最小次数;

8. 计算对各输出信号的渗透率值 $PEP_{x,y} = errorNumber_y / count$;

9. 再额外执行步 5~步 6 100 次,然后跳转至步 8 产生另一个 $PEP'_{x,y}$. 如果 $PEP_{x,y}$ 和 $PEP'_{x,y}$ 之间的变化不超过 0.001,则执行下一步骤,否则重复执行步 9;

10. 各个 $PEP_{x,y}$ 即为我们得到的信号渗透率的实验估计值.

EPE 是基于模块化软件的一个算法,所以目标软件系统必须先进行模块化.步 1 表明先考虑哪一个模块,然后得到该模块的黄金输出结果作为下面的参考.步 3 把实验次数计数器和每个输出错误次数计数器设为 0,为实验进行做准备.步 5 开始执行程序,并进行故障注入过程,同时记录实验次数,该过程中注入的故障可能引起模块的输出信号出现错误,所以步 6 进行比较来判断错误是否发生,并用计数器记录错误次数.在完成规定的最小实验次数之后,计算某一输出出现错误的次数占总的注入次数的比率就可得到对应的渗透率值.步 9 主要是为了保证实验结果的精确度,以 100 为单位对实验次数进行累积,直到精度波动小于 1%.

ELRE 算法和 EPE 算法基本相同,不同之处在于故障注入的时刻与位置. EPE 是当程序运行至模块入口时开始注入,同时把故障注入进模块输入信号中;而 ELRE 是在程序执行某一模块的时间范围内随机选择一时刻进行注入,注入位置为该模块的状态空间.限于篇幅,ELRE 算法这里不再列出.

错误传播特性会由于系统负载的变化而不同,所以在参数的实验估计过程中,要尽可能使用接近真实运行情况的负载.同时所注入的故障类型对估计值也有影响,所以在参数估计之前,我们必须确定该软件系统主要处理什么类型的故障,并在实验过程中采用与实际故障尽可能接近或相似的故障类型.

5 实验分析与讨论

为进一步阐述上面提出的软件脆弱点识别方法,我们在某卫星的光纤陀螺捷联航姿控制系统上进行了实例研究,目标系统结构如图 3 所示.为研究

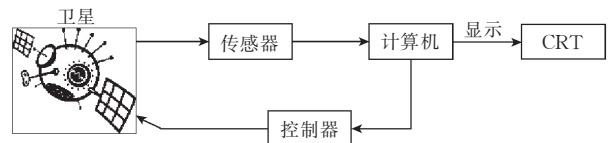


图 3 光纤陀螺捷联航姿系统结构

需要,我们把此软件移植到了 Linux 系统平台上,系统的传感器输入和控制器输出由环境仿真器模拟。

本实验的实验策略如下:首先在分析目标软件系统的基础上,通过故障注入得出各信号对的错误渗透率值以及各模块的泄漏率和活动率;然后分别在考虑环境扰动(考虑模块泄漏率和活动率的影响)和不考虑环境扰动(不考虑模块泄漏率和活动率的影响,如文献[21])的情况下,根据上面给出的影响因子(模块和信号)的计算方法,计算出各信号和各模块对系统输出的影响因子,并对两种结果进行比较;接着再通过故障注入实验,直接实验得出各信号和各模块对系统输出的影响因子;最后比较理论结果和实验结果的差别。

文献[22]中提出的软件可靠性实验方法的 6 个步骤在宏观上具有一定的指导意义,本文中的实验方法与之基本吻合,限于篇幅,这里只给出本文中具体的实验过程,对宏观的实验方法感兴趣的读者可参考文献[22]。

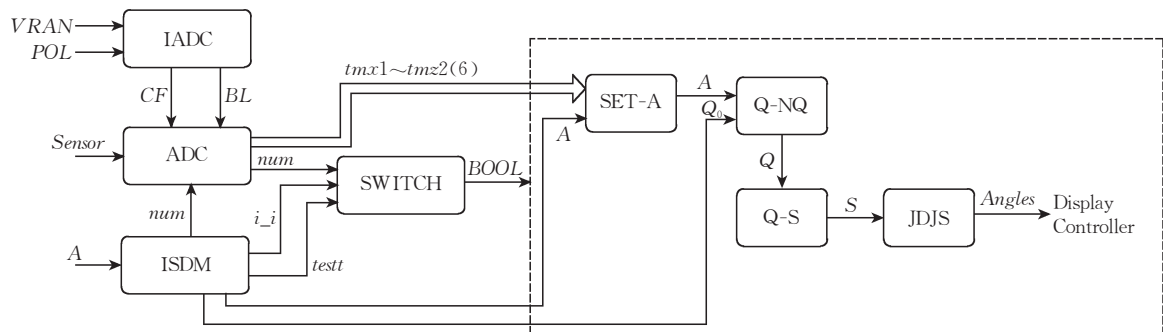


图 4 目标系统软件结构

(1) ADC 中断采样子程序,负责从外部传感器采集原始数据,输出 6 个采集数据 $tmx1, tmx2, tmy1, tmy2, tmz1, tmz2$,同时还输出另一信号 num 用于记录中断次数。

(2) IADC 采样板初始化子程序,输入为电压范围和极性,输出为在 ADC 采样过程中非常重要的两个参数 CF 和 BL 。

(3) ISDM 捷联矩阵 S 初始化,同时也对 num 和另两个控制参数 $i_i, testt$ 初始化。

(4) SWITCH 它利用 num, i_i 和 $testt$ 这几个参数来控制航姿程序的执行,所以在图 4 中,此模块的输出为一 $BOOL$ 并且并不指向具体的模块,它的输出结果决定虚线框内的模块下面是否要被执行。

(5) SET-A 利用采样得到的各个值对 A 进行设置。

(6) Q-NQ 此模块表示四元数的递推,由某一时刻

在程序运行过程中,错误的引入和数据的回收由故障注入工具来完成。注入工具采用我们自主开发的软件实现的故障注入工具 SAVIE(SoftWare Vulnerabilities Identifying Environment),它通过使用 Linux 操作系统的 Ptrace 系统调用或 Solaris 操作系统的 /proc 虚拟文件系统,在被注入程序运行时,故障可被随机地注入到各种位置,包括程序的代码段、数据段、堆栈段和各种寄存器中。故障注入以后,被注入程序的运行可被监视并回收必要的数据。由于采用了软件探针技术,此工具还可以在程序运行过程中对一些变量进行跟踪。

此软件系统主要处理太空中由辐射引起的单粒子翻转故障(Single Event Upsets, SEU),所以在本实验中,我们使用单位翻转(Single bit-flips)故障模型,因为该模型已被证明可很好地模拟 SEU 现象。

5.1 目标系统简介

目标系统软件结构如图 4 所示,包括 8 个模块和模块间的输入输出信号集。各个模块的功能如下:

刻已知的四元数推出下一采样时刻的四元数,输入变量是 A (A 结构中的角速度采样值)及四元数 $Q(i)$,输出变量是四元数 $Q(i+1)$ 。

(7) Q-S 此模块表示由四元数计算出捷联矩阵,输入变量是 Q ,输出变量是 S 。

(8) JDJS 此模块表示即时计算出的欧拉角,输入变量是捷联矩阵 S ,输出变量是欧拉角 $Angles$ 。

5.2 实验设置与过程

在错误传播分析中,故障通常要在限定的地址空间(某一模块内部)和时间内进行注入,或者要求对特定的变量(模块输入和输出)进行注入和监测,这主要是通过软件探针技术来实现的。在系统执行过程中,当遇到探针时就会触发注入或数据记录过程。

在信号级别上,错误主要被注入进各个模块的每个输入中,并监测该模块产生的输出信号;而对于模块级,错误主要被注入到各个模块的状态空间中,

包括各种寄存器和程序堆栈中,当模块运行结束时记录它的输出信号.在每一次注入运行中,一次只对一个输入信号或在状态中注入一个错误.在每一次注入运行之前,我们首先进行一次系统的黄金运行并记录黄金结果,用来作为系统运行的一个参考基准.然后进行注入运行并把运行数据与黄金运行数据进行比较,对于每一个被记录的信号如发现和黄金运行数据不相等就标记为发生了一个错误.具体过程可参考第 4 节中的 EPE 和 ELRE 算法.

对于模块输入信号(包括普通变量和结构体),注入主要是当执行到该模块始端时,在某一输入信号的地址范围内随机选择一个地址并随机选择其中一位进行翻转,每个信号的注入次数都高于 2000 次.对于模块体,需要确定注入时间和注入位置.对于第一个问题,我们采取在该模块的代码段地址空

间内随机选择一个地址(或指令),当模块执行至该地址(或指令)时进行注入;而对于第二个问题,主要是在通用寄存器、段寄存器、指令指针和标志寄存器中随机选择一个寄存器并随机选择其中一位进行翻转以及注入时刻的程序堆栈空间内(由 EBP 和 ESP 寄存器确定)随机选择一地址并随机选择其中一位进行翻转.对每个模块的注入次数都大于 10000 次.

5.3 实验结果与讨论

根据目标软件的结构对软件进行模块划分并确定相互之间的信号联接,如图 4 所示.由图 4 可知目标软件系统共有 8 个模块和 44 个输入/输出对,使用第 4 小节提出的算法可实验得出各个输入/输出对的错误渗透率以及各模块泄漏率的估计值,分别示于表 1 和表 2 中,这些数据构成了计算其它参数(如影响因子等)的基础.

表 1 各输入/输出对的错误渗透率值

输入→输出			模块	值	输入→输出			模块	值
VRAN→CF			IADC→ADC	1.000	POL→CF			IADC→ADC	0.000
VRAN→BL			IADC→ADC	0.000	POL→BL			IADC→ADC	1.000
CF→tmx1~tmz2			ADC→SET-A	1.000	BL→tmx1~tmz2			ADC→SET-A	1.000
CF→num			ADC→SWITCH	0.000	BL→num			ADC→SWITCH	0.000
num→tmx1			ADC→SET-A	0.929	num→tmx2			ADC→SET-A	0.899
num→tmy1			ADC→SET-A	0.929	num→tmy2			ADC→SET-A	0.899
num→tmz1			ADC→SET-A	0.929	num→tmz2			ADC→SET-A	0.899
num→num			ADC→SWITCH	0.867	A→num			ISDM→ADC	0.000
A→i_i			ISDM→SWITCH	0.000	A→testt			ISDM→SWITCH	0.000
A→A			ISDM→SET-A	0.214	A→Q ₀			ISDM→Q-NQ	0.098
num→BOOL			SWITCH	0.085	i_i→BOOL			SWITCH	0.031
testt→BOOL			SWITCH	1.000	A→A			SET-A→Q-NQ	0.571
tmx1~tmz2→A			SET-A→Q-NQ	1.000	A→Q			Q-NQ→Q-S	0.314
Q ₀ →Q			Q-NQ→Q-S	0.252	Q→S			Q-S→JDJS	1.000
S→Angles			JDJS→Display	0.635					

表 2 模块泄漏率

故障位置		寄存器	堆栈	寄存器 & 堆栈
模块	输出	泄漏率	泄漏率	模块泄漏率
IADC	CF	0.105	0.109	0.083
IADC	BL	0.105	0.013	
ADC	num	0.392	0.001	
ADC	tmx1	0.392	0.042	0.231
ADC	tmx2	0.392	0.075	
ADC	tmy1	0.393	0.010	
ADC	tmy2	0.392	0.150	
ADC	tmz1	0.395	0.004	
ADC	tmz2	0.392	0.208	
ISDM	num	0.093	0.002	0.050
ISDM	i_i	0.093	0.002	
ISDM	testt	0.092	0.002	
ISDM	A	0.101	0.003	
ISDM	Q ₀	0.103	0.007	
SWITCH	BOOL	0.095	0.000	0.048
SET-A	A	0.401	0.713	0.557
Q-NQ	Q	0.146	0.168	0.157
Q-S	S	0.116	0.064	0.090
JDJS	Angles	0.148	0.271	0.210

表 3 模块错误渗透率

模块(A→B)	渗透率(PEP ^{A→B})	总渗透率(PEP ^A)
IADC→ADC	0.500	0.500
ADC→SET-A	0.971	0.874
ADC→SWITCH	0.289	
ISDM→ADC	0.000	
ISDM→SWITCH	0.000	0.062
ISDM→SET-A	0.214	
ISDM→Q-NQ	0.098	
SWITCH	0.372	
SET-A→Q-NQ	0.939	0.939
Q-NQ→Q-S	0.283	0.283
Q-S→JDJS	1.000	1.000
JDJS	0.635	0.635

根据定义 3,我们使用信号的错误渗透率值可计算出各模块的渗透率值,示于表 3 中.其中中间一列为前一个模块对后一个模块的渗透率值,而最右侧一列为前一个模块的总的渗透率值,它没有考虑模块输出的具体后继模块.

有了信号之间的错误渗透率,就可以计算某信号对系统输出的影响因子,为此我们必须生成各信号的传播树,由于篇幅有限,在此仅以模块 ADC 的输入信号 CF 为例说明该过程.使用第 3 节中描述的传播树生成算法可产生如图 5 所示的传播树,其中每个节点都是传播路径上的不同信号,而边上的值则为始节点和终节点之间的错误渗透率,此值即为表 1 中的实验数据.由图 5 可知,信号 CF 有 7 条错误传播路径,左侧第一条路径的权重为 $\omega_1=1.0\times 1.0\times 0.314\times 1.0\times 0.635=0.1994$,其它各条路径权重计算与之类似,最右侧路径权重为 0,表明该路径不传播错误.根据式(2)可得信号 CF 的影响因子:

$$\begin{aligned} CF \rightarrow Angles &= 1 - \prod_{k=1}^6 (1 - \omega_k) \\ &= 1 - (1 - \omega_1) \cdots (1 - \omega_6) \\ &= 1 - (1 - 0.1994)^6 \\ &= 0.7367. \end{aligned}$$

与上述计算过程类似,可得出其它信号对系统输出的影响因子,示于图 7 中.这些影响因子是根据本文提出的错误传播框架使用输入/输出对之间的错误渗透率计算得到的,我们称之为理论结果.

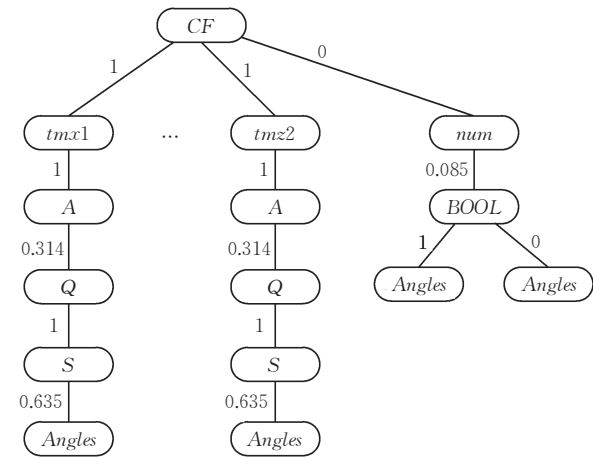


图 5 信号 CF 的传播树

在本文中,模块活动率是通过计算某一模块的运行时间占程序总的运行时间的比率得到的,其值列于表 4 中.对于模块泄漏率,由表 2 可知,不同位置的错误(寄存器和堆栈)使得模块的泄漏率并不相等,这里模块对某一输出的泄漏率采用寄存器和堆栈泄漏率的算术平均值,当然,在实际中可根据不同的情况采用不同权重的加权平均值.

下面分别从两个角度来计算模块影响因子:
(1)考虑环境扰动;(2)不考虑环境扰动.分别以模

块 Q-S 为例说明两种影响因子的计算过程,其它模块的影响因子计算与之类似,结果列于图 6 中.

表 4 各模块的活动率

模块(M)	活动率(PA ^M)
IADC	0.01
ADC	0.18
ISDM	0.25
SWITCH	0.01
SET-A	0.02
Q-NQ	0.23
Q-S	0.09
JDJS	0.21

考虑环境扰动:

$$\begin{aligned} Q-S \rightarrow Angles &= 1 - (1 - PEL^{Q-S} \cdot PA^{Q-S} \cdot S \rightarrow Angles) \\ &= 1 - (1 - 0.09 \times 0.09 \times 0.635) = 0.005. \end{aligned}$$

不考虑环境扰动:

$$\begin{aligned} Q-S \rightarrow Angles &= 1 - (1 - S \rightarrow Angles) \\ &= 1 - (1 - 0.635) \\ &= 0.635. \end{aligned}$$

由于模块影响因子的大小是一个相对的度量,所以比较由不同的计算方法计算得到的模块影响因子绝对值大小没有意义,但其大小的总体趋势是有可比性的.从图 6 中可看出,考虑环境扰动以后,各个模块影响因子的大小走势变化很大.如果根据 3.3 节中的脆弱点确定原则,则据此得出的系统脆弱点完全不同,这就说明环境扰动对错误传播分析以及脆弱点识别有很大的影响,如果有环境扰动存在的条件下,软件系统的可靠性分析中不能忽略环境的影响.

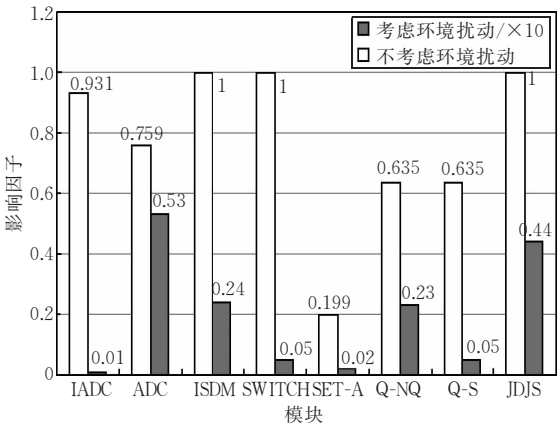


图 6 环境因素对模块影响因子的影响

为了验证本章中的理论框架的有效性与正确性,与信号和模块影响因子的理论结果相对应,本章又得出了其实验结果.信号影响因子的实验结果确定过程如下:首先在某一信号中注入错误,然后观察

系统输出,如果和黄金输出结果不同,则认为错误传播到了系统输出端,最后由系统输出错误次数除以实验总的次数来得到某一信号的影响因子.例如,在输入信号 *POL* 中注入了 10000 次错误,而在系统输出 *Angles* 上观察到 9889 次错误,则 *POL* 的影响因子为 0.9889.

各信号影响因子的理论结果和实验结果都示于图 7 中.从图 7 中可看出,实验结果和理论结果并不完全相等,而是稍大于理论结果,但信号影响因子总的趋势是不变的,如理论结果较大,则实验结果也较大,各个信号影响因子大小的理论结果排序基本可以反应实验结果.对于相等的理论结果,实验结果之间的差别最大不超过 0.06,这对于通过错误传播寻找软件系统中的“脆弱点”来讲,理论结果的精确度已经足够.

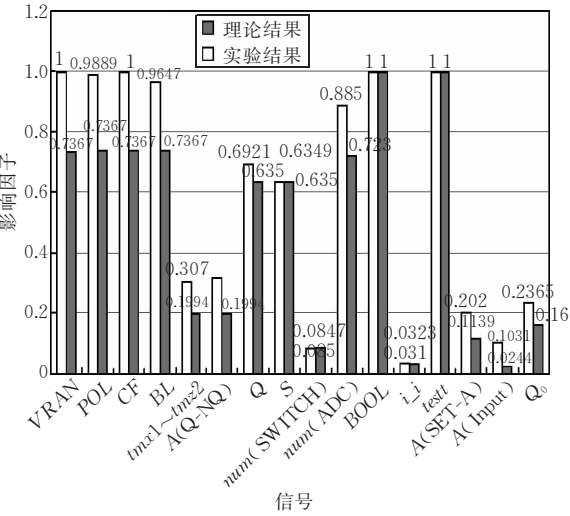


图 7 信号影响因子的理论与实验结果

在本文的实验中往某一模块注入错误的次数并不是根据模块运行时间的长短(如果按照活动率比例来注入故障的多少,则实验得出的模块影响因子的值可能不能反应真实的值),而是尽可能多地注入(让影响因子的值达到稳定),所以在与模块影响因子实验结果进行比较时,此时理论结果的计算中模块活动率都假设为 1,而不采用表 4 中的活动率值.但在实际的过程中,模块活动率对模块影响因子可能影响很大.

和信号影响因子类似,有了理论结果以后,又和实验结果进行了比较,实验结果的获得与信号实验结果获得类似,只是把错误注入到模块内部的寄存器和堆栈中而不是输入信号中.由于模块影响因子不仅包括模块泄漏率还包括错误信号的传播,是两者的综合作用,所以实验结果的特点与信号级类似:

实验结果一般大于理论结果(模块 ADC 例外),但趋势相同,如图 8 所示.

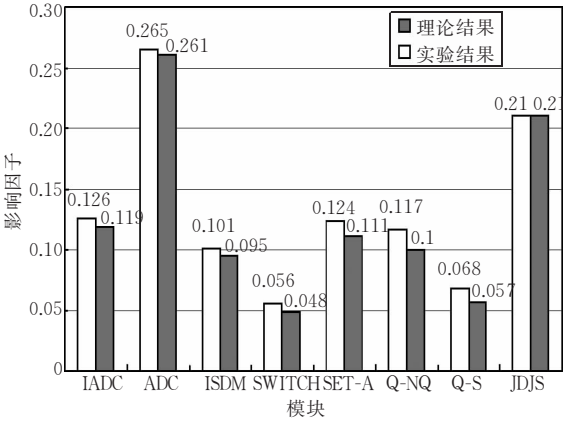


图 8 模块影响因子的理论与实验结果

从图 7 和图 8 中可看出,信号和模块影响因子的理论值变化趋势可以反应实验值变化趋势,这对于通过故障传播分析来识别系统脆弱点来讲,根据理论结果和实验结果得出的系统脆弱点应该是一致的.这说明本文中的理论框架是有效的,可以用于识别软件系统的脆弱点.

根据 3.3 节给出的脆弱点选取规则,我们可以确定目标软件的薄弱环节,但不同的规则选取出的脆弱点是各不相同的.如对于模块 Q-S,根据错误渗透率分析可得到 100% 的错误渗透率值(如表 3),表明此模块对传播到它的错误没有任何包容能力,应该是系统脆弱点.但从影响因子角度分析,该模块的影响因子却很低,这表明如果从该模块输出错误,此错误对系统输出影响很小,可能不会造成严重的系统失效,从这个角度来讲,该模块又不被认为是系统的脆弱点.

实际上,本文在错误传播过程中提出的各个参数分别侧重于两种不同的错误模型(对应环境影响软件系统的两种方式):(1) 错误仅从系统输入信号上引入;(2) 错误从程序内存空间和寄存器的随机位置引入.如果系统的存储区和代码部分以某种方式被保护或屏蔽,外部干扰影响不到这些部分,但传感器输入仍然可受到影响,这时就会出现第一种错误模型.在此种情况下,错误的传播过程起主导作用,所以此时对信号和模块的渗透率分析及信号影响因子分析应起到主要作用.而对于第二类错误模型,错误生成和泄漏特性是主要特性,同时考虑后继模块的错误传播问题,所以此时模块影响因子分析(考虑了泄漏率)应起主要作用,同时兼顾模块的渗透率.而对于两种错误模型都存在的情况下,各规则

得出的结果应相互补充.

对于我们的目标系统,由于没有对任何部分进行保护,所以根据上面的原则很容易得出系统的脆弱点,如模块 ADC、Q-S、SET-A、JDJS 和信号 *BOOL*、*testt*. 这些信息对后继的软件加固具有指导意义.

6 结 论

本文提出了一种基于错误传播分析的软件脆弱点的识别方法,该方法首先对软件系统进行模块化,然后根据环境影响软件系统的不同方式,分析了错误在软件系统中的生成及传播过程,在此过程中首次提出了模块泄漏率和活动率的概念并给出了其中一些参数的实验估计方法,据此结果描述了软件脆弱点的确定原则. 最后把该软件脆弱点识别框架应用于实际的导航软件系统上,并通过故障注入实验对该框架进行了参数估计及目标软件的脆弱点确定,结果令人满意.

参 考 文 献

- [1] Amoroso E G. Fundamentals of Computer Security Technology. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1994
- [2] Cowan C, Wagle P, Pu C et al. Buffer overflows: Attacks and defenses for the vulnerability of the decade//Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX). Oakland, CA, USA, 1999: 154-163
- [3] Talwar S K, Foster J, Wagner D. Detecting format-string vulnerabilities with type qualifiers//Proceedings of the 10th USENIX Security Symposium. Washington, DC, 2001: 201-218
- [4] Bishop M, Dilger M. Checking for race conditions in file accesses. Computing Systems, 1996, 9(2): 131-152
- [5] Halfond W, Viegas J, Orso A. A classification of SQL-injection attacks and countermeasures//Proceedings of the IEEE International Symposium on Secure Software Engineering (ISSSE 2006). Arlington, VA, USA, 2006
- [6] Ghosh A K, O'Connor T, McGraw G. An automated approach for identifying potential vulnerabilities in software//Proceedings of the IEEE Symposium on Security and Privacy. Oakland, CA, USA, 1998: 104-114
- [7] Du W, Mathur A P. Testing for software vulnerability using environment perturbation. International Conference on Dependable Systems and Networks (DSN 2000). New York, USA, 2000: 603-612
- [8] Lin Z, Mao B, Xie L. A practical framework for dynamically immunizing software security vulnerabilities//Proceedings of the 1st International Conference on Availability, Reliability and Security (ARES2006). Vienna, Austria, 2006: 348-357
- [9] Lu Yu-Liang, Xia Yang. Research on target-computer secure quantitative fusion model. Chinese Journal of Computers, 2005, 28(5): 914-920(in Chinese)
(陆余良, 夏阳. 主机安全量化融合模型研究. 计算机学报, 2005, 28(5): 914-920)
- [10] Pfleeger C, Pfleeger S, Theofanos M. A methodology for penetration testing. Computers and Security, 1989, 8(7): 613-620
- [11] Puchkov F M, Shapchenko K A. Static analysis method for detecting buffer overflow vulnerabilities. Programming and Computer Software, 2005, 31(4): 179-189
- [12] Wu Chun-Mei, Xia Nai, Mao Bing. A comparison of static analysis technology for intrusion prevention. Computer Engineering, 2006, 32(3): 174-176(in Chinese)
(吴春梅, 夏耐, 茅兵. 防范入侵的静态分析技术比较. 计算机工程, 2006, 32(3): 174-176)
- [13] Chen H, Wagner D. MOPS: An infrastructure for examining security properties of software. UC Berkeley: Technical Report UCB//CSD-02-1197, 2002
- [14] Huang Guang-Hua, Duan Chuan, Jiang Fan. Analysis of system vulnerabilities based on Model Checking. Computer Engineering, 2005, 31(4): 148-151(in Chinese)
(黄光华, 段川, 蒋凡. 基于 Model Checking 的系统脆弱性分析. 计算机工程, 2005, 31(4): 148-151)
- [15] Krsul I. Software vulnerability analysis [Ph. D. dissertation]. Department of Computer Sciences, Purdue University, 1998
- [16] Wang H, Wang C. Taxonomy of security considerations and software quality. Communications of the ACM, 2003, 46(6): 75-78
- [17] Anghel L, Nicolaidis M. Cost reduction of a temporary faults detecting technique//Proceedings of the Design, Automation, and Test Europe Conference. Paris, France, 2000: 591-598
- [18] Goloubeva O, Rebaudengo M, Reorda M S et al. Soft-error detection using control flow assertions//Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'03). Boston, MA, USA, 2003: 581-588
- [19] Wang Tong-Quan, Dai Hong-Yi, Shen Yong-Ping et al. Calculation of cosmic high energy proton induced single event upset rate. Journal of National University of Defense Technology, 2002, 24(2): 11-13(in Chinese)
(王同权, 戴宏毅, 沈永平等. 宇宙高能质子致单粒子翻转率的计算. 国防科技大学学报, 2002, 24(2): 11-13)
- [20] Baumann R C. Soft errors in commercial semiconductor technology: Overview and scaling trends//Proceedings of the IEEE 2002 Reliability Physics Tutorial Notes. Reliability Fundamentals. Dallas, Texas, 2002: 1-14
- [21] Hiller M, Jhumka A, Suri N. EPIC: Profiling the propagation and effect of data errors in software. IEEE Transactions on Computers, 2004, 53(5): 512-530
- [22] Cai K Y. Software reliability experimentation and control. Journal of Computer Science and Technology, 2006, 21(5): 697-707



LI Ai-Guo, born in 1977, Ph. D. candidate. His research interests include design and assessment of dependable software, fault injection and error detection.

HONG Bing-Rong, born in 1937, professor, Ph. D. supervisor. His current research interests include distributed artificial intelligence, multi-agent robot system.

WANG Si, born in 1963, Ph. D. , professor. His research interests include inertial and integrated navigation technology.

Background

In the special field, where very high dependability level is needed, the software is required to run smoothly in the presence of errors, which spurs us to find out the characteristic of software under these exceptions, especially to know how the errors propagate through software and affect its execution. Furthermore, in order to increase the software dependability, we must find such vulnerable parts for further analysis and possibly retrofitting with fault-tolerant mecha-

nisms. Error propagation analysis may be used to find the modules and signals which are most exposed to errors in a system and to ascertain how different modules affect each other in the presence of errors.

This work is supported by the National Aeronautical Research Program under grant No.417010402, and by the Aeronautical Innovation Fund under grant No. CASC0409.