第 42 卷 第 12 期 西 安 交 通 大 学 学 报 Vol. 42 №12
2008 年 12 月 JOURNAL OF XI′AN JIAOTONG UNIVERSITY Dec. 2008

利用构造类别代数的协议安全测试方法

陈伟琳[1,2,3]，周颢[1,2,3]，赵保华[1,2,3]

(1. 中国科学技术大学计算机科学与技术系，230027，合肥；2. 网络与交换技术国家重点实验室，100876，北京；
3. 安徽省计算与通讯软件重点实验室，230027，合肥)

摘要：针对协议安全测试中数据流的描述问题，在构造类别代数的基础上引入变异分析，由此提出了一种协议安全测试方法. 基于协议的描述设计了变异算子，然后对一致性公式集进行变异分析，从而得到安全变异项集. 将变异项转换为实际的安全测试例集，并描述由此进行的协议安全测试步骤. 测试实践表明，所提方法能与一致性测试良好地结合，在一定程度上覆盖已知安全漏洞，并具有发现潜在问题的能力.

关键词：协议安全测试；构造类别代数；变异分析

中图分类号：TP311 **文献标志码**：A **文章编号**：0253-987X(2008)12-1481-05

Constructed Type Algebra Based Protocol Security Testing Method

CHEN Weilin[1,2,3], ZHOU Hao[1,2,3], ZHAO Baohua[1,2,3]

(1. Department of Computer Science and Technology, University of Science & Technology of China, Hefei 230027, China;
2. State Key Laboratory of Networking and Switching Technology, Beijing 100876, China; 3. Anhui Province Key Laboratory of Software in Computing and Communication, Hefei 230027, China)

Abstract: Considering the data flow description in protocol security testing, mutation analysis is introduced based on constructed type algebra, and a new protocol security testing method is proposed. Mutant operators are designed based on protocol specification, security mutation formulas are generated from existing conformance testing expressions, and then practical security testing cases are transformed from these formulas. The steps of protocol security testing using the method are described. The method is well integrated with conformance testing to cover some known protocol security vulnerabilities, and has the ability to reveal potential problems.

Keywords: protocol security testing; constructed type algebra; mutation analysis

网络安全不仅仅依靠外部设备和安全软件的防护，也依赖于网络设备支持软件的自身安全. 传统的协议一致性测试[1]作为一种功能性测试，关注于协议实现需要达成的目标或功能，却不能检测出协议实现中存在的安全问题[2].

协议的安全性与控制流均与数据流相关. 测试中，最常用于描述协议的有限状态机(FSM)模型[3]善长于描述控制流，而对数据流的描述能力较差. 虽然该模型得到了扩展 EFSM[4]，但也只能表示一些简单的变量，很难用于描述具体报文. 在协议安全测试中，对可能引起安全问题的报文描述是很重要的. 构造类别代数方法[5]可形式化地描述协议，再根据描述中的公理生成测试序列，所以它能够有效地描述协议数据及协议行为，也适用于描述协议数据流.

变异分析最初是一种基于代码的单元测试方法[6]，目前在安全测试领域多有应用. 本文在协议构造类别代数描述的基础上，对一致性抽象公式集进行变异分析，得到安全测试例集，从而探寻协议可能

收稿日期：2008-01-04. 作者简介：陈伟琳(1982—)，女，博士生；赵保华(联系人)，男，教授，博士生导师. 基金项目：国家自然科学基金资助项目(60602016，60872009)；国家重点基础研究发展规划资助项目(2003CB314801)；国家高技术研究发展计划资助项目(2007AA01Z428)；华为基金资助项目(YJCB2006062WL，YJCB2007061WL).

存在的安全问题.

1 方法概述

构造类别代数定义为一个四元组 $D=\langle S,C,E,Q\rangle$,其中 S 是类别名的有限集合,C 是构造函数集合,E 是延拓函数集合,Q 是等式公理集合. 一个协议可以通过构造类别代数来描述,用类别定义协议的数据结构,用函数描述协议的行为,用公理描述行为应当满足的要求. 文献[5]也提出了相应的构造类别代数描述语言(CDL),协议描述由规范部分 D 和可观察性部分 O 组成. 从协议的 CDL 描述出发,可以通过一定的算法生成抽象公式集,进而得到测试例集.

传统的变异分析主要为被测程序产生一组变异体,以衡量已有测试集的充分性. 安全漏洞常涉及一些未知的多余行为或附加效应,为了在测试中尽量覆盖这些未知因素,可将变异分析引入到安全测试之中[7-9]. 一般的做法是:对相关参数给出一系列变异值,再观察被测系统能否正确地操作这些值而不产生异常.

本文在考虑协议的构造类别代数描述的基础上,使用变异分析产生安全测试例集,并进行安全测试. 具体过程可简述为:首先基于协议描述并按照一致性测试例生成方法得到抽象公式集,然后施加变异算子得到变异项集,再将变异项映射为实际测试例,最后执行并获得结果.

2 基于构造类别代数的变异算子

现实中的很多协议攻击都是由于对协议的正常行为进行歪曲而形成的,所以可以通过定义变异算子来模拟这些歪曲. 根据构造类别代数规范的结构特点,变异算子集 M 可划分为几类.

2.1 数据变异算子

在构造类别代数中,类别 S 代表协议从外界接收到的数据,而由构造函数 C 得到的类型是协议内部使用的数据结构. 数据变异只施加在 S 上,分为单值变异和复合变异. 单值变异:每个单值类别都有对应的合法取值范围. 现实中一些协议安全问题往往是由于没有正确处理边界值和非法值而引起的,单值变异算子的设计就是为了产生包含非法值的测试例. 复合变异算子:是针对某些具有内部结构的复合类别(如地址)的,这些复合类别存在着规整格式,可以通过改变内部域来引起变异(删除域、交换位置等),复合变异算子的作用是构造畸形的报文. 实现中,对畸形报文的处理不当往往容易导致安全问题.

2.2 函数变异算子

在构造类别代数中,可操作函数是实现与外界交互的接口,其参数代表着协议对外界数据的要求. 当外界提供非规范参数时,此类算子意在观测函数的行为是否会发生偏差. 函数变异算子包括:①参数个数变异算子,即对函数的参数个数进行改变,提供少于/多于函数要求的参数;②参数类型变异算子,即对函数参数类型进行变异,提供与函数参数类型不匹配的参数;③参数顺序变异算子,即对函数声明的参数顺序产生一个违背,提供失序的参数序列. 算子③限制在给定抽象公式之内,同时为了简化变异过程、利于错误定位,在对包含多个函数的一条抽象公式变异时每次只考虑单个函数的单次变异,也就是,若抽象公式 q 包含 m 个函数,每个函数有 n_i 个对应的变异算子,则最终将生成 $K=\sum_{i=1}^{m}n_i$ 个变异项 $q_1,q_2,\cdots,q_K$.

2.3 深度变异算子

一致性测试例生成算法在产生抽象公式集合时限制了函数的调用次数[5],降低了对某些重复操作出错的检测能力. 深度变异算子允许测试者加大函数的重复调用次数,即对形如 $g(f(f()))$ 的出现,产生类似 $g(f(f(f(f()))))$ 的变异项. 该算子根据深度增加方式可分为:①常量增加算子,即在函数当前调用次数上增加一个常数 d_a,其适用于深度缓和调整的情况;②加倍算子,即将函数当前调用次数乘上一个常数 d_m,其适用于深度需大幅度快速调整的情况;③并行算子,适用于某些协议在同时面对大量不同来源的重复操作时易产生问题的情况.

2.4 测试过程

基于协议 P 的 CDL 描述 $\langle D,O\rangle$ 和变异算子集合 M,对 P 的安全测试过程描述如下.

(1)根据 CDL 描述 $\langle D,O\rangle$,生成一致性抽象公式集 E.

(2)对 E 施加 M 中的变异算子,得到变异项集 M_E.

(3)将 M_E 中的每个变异项映射为实际测试例,得到测试例集 T,并执行获得的结果. 其中,E 的生成利用了一致性测试例生成算法[5].

2.5 变异项的生成

一致性测试例生成算法产生的抽象公式均为等式,等号左边是代数项(表示操作序列),右边是类别值(表示期待的执行结果). 将变异算子施加在代数

项上就能形成变异的操作序列.

变异项生成算法(算法 1)如图 1 所示.

输入:抽象公式集E

输出:普通变异项集深度变异项集并集$M_E \cup M_D$(初始时均为空)

(1)对每个类别进行类别变异,S_1是D中需要变异的类别集合:

对S_1中的每个s_i,

求得$E_i=\{e_{i1},e_{i2},\cdots,e_{iu}\}$,它是包含$s_i$类别变量或常量的抽象公式集;

对E_i中各个e_{ik}施加s_i的类别变异算子,得到变异项集M_{Ei},并加入M_E;

(2)对公式进行函数变异,F是D中带有函数变异算子的函数集合.对任意$f \in F$,f的变异算子集合为M_f,E_1是E中需要进行函数变异的抽象公式集合:

对E_1中的每个e_j,假定e_j包含函数调用$f_1,f_2,\cdots,f_n \in F$,

对每个f_i,将M_{fi}中的变异算子依次作用在e_j上,

得到变异项集M_{Fi},并加入M_E;

(3)对相关函数进行深度变异,G是D中带调用次数限制的函数集合,对任意$g \in G$,包含g的抽象公式为$T_g \subset T_q$,$T_g=T_{g1} \cup T_{g2} \cup,\ldots,\cup T_{gn}$,其中$T_{gi}$的公式中$g$调用次数为$i$;

对G中的每个g_k、对T_g中的每个抽象公式t_h施加g的深度变异算子,得到变异项d_{th}集M_{Dg},并加入M_E;

(4)整理得到的M_E和M_D集合,删除可能的重复项;

(5)建立M_E中项之间包含关系.

图 1　变异项生成算法

算法 1 最后一步提到的项之间的包含关系定义如下.

定义 1　项 t_1 包含项 t_2,记为 $t_1 < t_2$,当且仅当 t_1 的函数调用形式是 t_2 的函数调用形式的子调用,即 $t_2 = f(\cdots,t_1,\cdots)$,其中 f 是某个函数.

这是一个偏序关系,如 $f() < g(f())$,$f() < p(f(),h())$,但 $g(f())$、$p(f(),h())$之间不存在包含关系.

定义 2　不包含其他任何项的项,称为元项.

如 $t=f(a)$,a 是 int 类别变量,t 就是一个元项,而 $t'=g(f(f()),h())$则不是元项,且有 $t<t'$.

2.6　测试例的生成

根据可操作或可观察函数的定义,可以直接将一个抽象公式的左项转换为测试序列,具体步骤为:从内至外地递归记录相应函数对应的操作.测试例生成算法(算法 2)如图 2 所示.

用一个简单的例子来解释算法 2,即在栈描述中一个代数项为 Top(Push(1,Push(0,ε))),ε 为空

输入:变异项集合M_E

输出:测试例集合T_E

对M_E中的每个项t,初始测试序列$t_s=\varnothing$;

if(项t具有形式 $f(p_1,\ldots,p_k)$,其中所有的p_i都是值或无参构造函数)

$t_s \leftarrow$函数 f 对应的操作;

else{// 项t具有形式 $f(p_1,\ldots,p_k)$,其中存在p_i是对其他函数的调用

for(所有的p_i)

if(p_i是值或无参构造函数)　$t_s \leftarrow$函数 f 对应的操作;

else递归地对p_i生成测试序列;

$t_s \leftarrow$函数 f 对应的操作;}

将t_s加入T_E;

图 2　测试例生成算法

栈,则对应的测试序列为{向 ε 压入整数 0,向得到的栈中继续压入整数 1,使用 TOP 函数观察栈顶}.

由于项和测试例一一对应,所以可以类似地得到测试例的包含关系和元测试例的定义.

定义 3　若项 t_1 包含 t_2,则称由 t_1 得到的测试例 t_{s1} 包含由 t_2 得到的测试例 t_{s2},记为 $t_{s1} < t_{s2}$.

定义 4　由元项转化得到的测试例,称为元测试例.

在上述定义下,变异测试例集合可根据包含关系抽象为有向图,其中存在若干入度为 0 的结点(元测试例),同时有如下结论.

命题 1　从一个元测试例 t_0 出发,按包含关系能到达的所有结点,构成以 t_0 为根的广义树.

2.7　测试例的执行

由于各个变异项之间具有包含关系,测试例的执行(见图 3)也是相互关联的,所以可以据此进行优化.

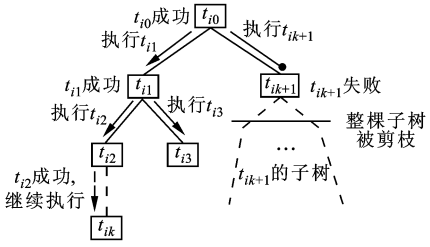

图 3　测试例执行示意图

对普通变异测试例,命题 1 说明在以 t_i 为根的子树中的结点都是包含 t_i 的,当 t_i 出错时,它们也会出错,因此可以沿命题 1 中的广义树执行.如果在某个子树根处执行后得到错误结果,则对该子树进行剪枝,不再继续执行.

对于深度变异测试例,则按照变异算子类型不断增加相关函数的调用,直到达到上限或出错.若存

在某个深度变异测试例 $t_s<t_s'$，且 t_s 已经出错，则 t_s' 直接标为出错.

2.8 示 例

本文限于篇幅，仅以栈这个简单例子来说明变异算子的设计和使用. 栈的构造类别代数规范参见文献[5]，它包含惟一的类别 $S=\{a\}$（a 为字母）和构造函数集合 $C=\{\varepsilon,\text{Push}\}$、延拓函数集合 $E=\{\text{Top},\text{Pop},\text{Equ}\}$，可观察集合为 $O=\{\text{Top}(s)\}$，且规定递归深度为 2，而 Push 和 Pop 函数的调用限制为 2. 假定 a 为整数[0,1]集合，则设计变异算子如表 1 所示.

抽取一致性测试算法，产生的一个抽象公式 e_1：Top (Pop (Push (0, Push (0, ε))))=0，对其进行变异操作得到的变异项如表 2 所示.

表 1 变异算子

算子种类	子类	算子描述		目的
类别变异算子	单值算子	S_1：小于 0 的整数	S_2：大于 1 的整数	检验实现对超过取值范围的值的处理是否正确
函数变异算子	Push 函数参数变异算子	F_1：第 1 参数为浮点数	F_2：第 1 参数为字符	检验 Push 函数实现对参数类型是否进行了鉴别
		F_3：不提供第 1 参数	F_4：提供 3 个参数，前 2 个都是合法整数	检验 Push 函数对参数的顺序和个数是否进行了验证
深度变异算子	Push 函数深度变异算子	D_1：调用次数每次加倍		检验 Push 函数是否正确处理了入栈次数超限的情况
	Pop 函数深度变异算子	D_2：调用次数每次加 2		检验 Pop 函数是否正确处理了连续出栈中遇空栈

表 2 公式 e_1 的普通变异项

	算子 S_1		算子 S_2
$e_1^{S_1}$	Top (Pop (Push (−1, Push (0, ε))))	$e_1^{S_2}$	Top (Pop (Push (2, Push (0, ε))))
$e_2^{S_1}$	Top (Pop (Push (0, Push (−2, ε))))	$e_2^{S_2}$	Top (Pop (Push (0, Push (3, ε))))
	算子 F_1		算子 F_2
$e_1^{F_1}$	Top (Pop (Push (1.2, Push (0, ε))))	$e_1^{F_2}$	Top (Pop (Push ('a', Push (0, ε))))
$e_2^{F_1}$	Top (Pop (Push (0, Push (0.5, ε))))	$e_2^{F_2}$	Top (Pop (Push (0, Push ('b', ε))))
	算子 F_3		算子 F_4
$e_1^{F_3}$	Top(Pop(Push(Push(0,ε))))	$e_1^{F_4}$	Top(Pop(Push(0,1,Push(0,ε))))
$e_2^{F_3}$	Top(Pop(Push(0,Push(ε))))	$e_2^{F_4}$	Top(Pop(Push(0,Push(0,1,ε))))

深度变异算子 D_1 产生变异项 t_{11}：Top (Pop (^{D_1}Push (0, Push (0,ε)))),而在测试过程中，它要执行所产生的若干个项（及对应测试例），直到得到期望结果或实现出错，即

t_{12}：Top (Pop (Push (0,Push(0,Push(0, Push (0, ε)))))) // Push 调用次数为 4

⋮

t_{1k}：Top(Pop(Push(0,Push(0,…,Push(0, ε))))) // Push 调用次数为 $2k$

而深度变异算子 D_2 产生变异项

t_{21}：Top (^{D_1}Pop (Push (0, Push (0,ε))))

它会产生 k 为$\frac{1}{2}$的 Push 调用次数，亦即等于 1 个变异项（及对应测试例），即

t_{22}：Top (Pop(Pop(Pop(Push (0, Push (0, ε)))))) // *Pop* 调用次数为 3

3 结 论

本文提出了一种基于构造类别代数的协议安全测试方法：将变异分析引入到了构造类别代数规范之中；描述了变异算子的设计；将变异算子作用于抽象公式，得到了安全测试例的过程和协议安全测试

的基本步骤;根据变异项之间的关系,组织测试,减少了冗余并提高了测试效率.协议测试实践表明,该方法能与一致性测试良好地结合,并覆盖相当数量的已知漏洞,具有发现潜在问题的能力.

参考文献:

[1] CCITT. 290-ISO/IEC 9646-1991 Information technology open systems interconnection conformance testing methodology and framework [S]. Geneva, Switzerland: CCITT, 1991.

[2] ANDY WANG Ju′an. Security testing in software engineering courses [C]// 34th ASEE/IEEE Frontiers in Education Conference. Piscataway, NJ, USA: 2004: 13-18.

[3] 龚正虎. 计算机网络协议工程 [M]. 长沙,国防科技大学出版社,1993.

[4] HUANG Chung-Ming, LIN Chih-Hao. An EFSM-based formal model for providing VCR-like functions in multimedia systems[C]// Proceedings of the 12th International Conference on Software Engineering Workshop on Multimedia Software Engineering. Los Alamitos, CA,USA: IEEE Comp. Soc., 1998:12-19.

[5] 孙宇霖,屈玉贵,赵保华.一种通信协议测试序列生成的新方法[J].通信学报,2001,22 (6): 122-127.
SUN Yulin,QU Yugui,ZHAO Baohua. A new method of generation of communication protocol test sequence [J]. Journal of China Institute of Communications, 2001,22 (6): 122-127.

[6] DEMILLO R A, LIPTON R J, SAYWARD F G. Hints on test data selection: help for the practicing programmer [J]. Computer, 1978,11(4): 34-37.

[7] AGRAWAL H, DEMILLO R A. Design of mutant operators for the c programming language, SERC-TR-41-P [R]. West Lafayette, IN, USA: Software Engineering Research Center, Purdue University, 1989.

[8] SPAFFORD E H. Extending mutation testing to find environmental bugs [J]. Software Practice and Experience 1990,20(2): 181-189.

[9] JIANG Ying, HOU Shanshan. Contract-based mutation for testing components [C]// Proceedings of 21st IEEE International Conference on Software Maintenance. Los Alamitos, CA, USA: IEEE Comp. Soc., 2005: 483-492.

(编辑　苗凌)