

# 基于静态分析的强制访问控制框架的正确性验证

吴新松<sup>1),2)</sup> 周洲仪<sup>1),2)</sup> 贺也平<sup>1)</sup> 梁洪亮<sup>1)</sup> 袁春阳<sup>3)</sup>

<sup>1)</sup>(中国科学院软件研究所 北京 100190)

<sup>2)</sup>(中国科学院研究生院 北京 100049)

<sup>3)</sup>(国家计算机网络应急技术处理协调中心 北京 100029)

**摘 要** 现阶段对操作系统的强制访问控制框架的正确性验证的研究主要集中于对授权钩子放置的验证. 文中基于 TrustedBSD MAC 框架对强制访问控制框架的正确性验证问题进行了研究, 在授权钩子放置验证的基础上, 提出了安全标记的完全初始化验证和完全销毁验证. 为了实现上述验证, 文中提出了一个路径敏感的、基于用户自定义检查规则的静态分析方法. 该方法通过对集成于编译器的静态分析工具 mygcc 进行扩展来验证强制访问控制框架的钩子放置的准确性和完备性. 该方法具有完全的路径覆盖性, 且具有低的误报率和时间开销.

**关键词** 正确性验证; 静态分析; 强制访问控制框架; 钩子放置; mygcc

中图法分类号 TP301

DOI 号: 10.3724/SP.J.1016.2009.00730

## Static Analysis Based Correctness Verification for Mandatory Access Control Framework

WU Xin-Song<sup>1),2)</sup> ZHOU Zhou-Yi<sup>1),2)</sup> HE Ye-Ping<sup>1)</sup> LIANG Hong-Liang<sup>1)</sup> YUAN Chun-Yang<sup>3)</sup>

<sup>1)</sup>(Institute of Software, Chinese Academy of Sciences, Beijing 100190)

<sup>2)</sup>(Graduate University of Chinese Academy of Sciences, Beijing 100049)

<sup>3)</sup>(National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing 100029)

**Abstract** Current researches on correctness verification for the mandatory access control framework of operating systems mainly focus on authorization hooks placement verification. Based on TrustedBSD MAC framework, this paper analyses the correctness verification problem for mandatory access control framework, and proposes complete initialization and complete destruction for security labels, as well as complete authorization for access. In order to enforce these verifications, this paper also presents a path-sensitive and user-defined-rule based static analysis approach. This approach verifies the accuracy and completeness of hooks placement of the mandatory access control framework through extending the compiler integrated static analysis tool-mygcc. It achieves high path cover, low false positive rate and time overhead.

**Keywords** correctness verification; static analysis; mandatory access control framework; hooks placement; mygcc

收稿日期: 2008-12-01; 最终修改稿收到日期: 2009-01-17. 本课题得到国家自然科学基金(90818012)、国家“八六三”高技术研究发展计划项目基金(2007AA010601)、中国科学院重要方向项目(KGCX2-YW-125)资助. 吴新松, 男, 1974 年生, 博士研究生, 主要研究方向为系统软件、信息安全. E-mail: xinsong@nfschina.com. 周洲仪, 男, 1975 年生, 博士研究生, 主要研究方向为系统软件、信息安全. 贺也平, 男, 1962 年生, 研究员, 博士生导师, 主要研究领域为信息安全、可信计算. 梁洪亮, 男, 1972 年生, 副研究员, 主要研究方向为系统软件、信息安全. 袁春阳, 男, 1979 年生, 博士, 主要研究方向为系统软件、网络与信息安全.

## 1 引言

强制访问控制框架(Mandatory Access Control Framework,以下简称 MAC 框架)是操作系统内核中的一组用于访问控制的通用钩子函数的集合.其作用是使操作系统同时提供对多种安全策略的支持,如 BLP 机密性多级安全策略<sup>[1]</sup>、BIBA 严格数据完整性策略<sup>[2]</sup>以及 LOMAC<sup>[3]</sup>低水标数据完整性策略等.目前,重要的开源操作系统均提供强制访问控制框架,如 Linux 操作系统的 Linux 安全模块(Linux Security Module, LSM)<sup>[4]</sup>,FreeBSD 操作系统及 Mac OS X 操作系统的 TrustedBSD MAC 框架<sup>[5]</sup>①(以下简称 TMAC)等.

MAC 框架的钩子函数是由内核开发人员插入到操作系统内核源代码中的.内核开发人员首先对内核源代码进行分析以确定钩子函数放置点,然后在放置点插入对应的钩子函数.钩子函数放置的位置准确性和位置完备性完全依赖于内核开发人员对内核源代码的分析和判断.而操作系统内核代码庞大,内核子系统之间存在复杂的关系.所以,难免会出现钩子函数放置位置不准确以及放置位置不完备等问题.由于钩子函数分布在操作系统内核的各个子系统中,是内核代码的一部分,因此,钩子函数放置出现问题可能会导致违反安全策略的行为发生.所以,需要对钩子函数的放置进行验证,以保证钩子函数放置位置的准确性和完备性.这里,我们把钩子函数放置的准确性和完备性称为 MAC 框架的正确性.

MAC 框架的钩子函数分布在操作系统内核的各个子系统中,通过人工方式对其进行正确性验证是非常困难的,需要采用自动化分析方法. IBM 研究院 Watson 研究中心的研究人员最先对 MAC 框架的正确性验证进行了研究,提出了验证授权钩子函数放置准确性和完备性的分析方法<sup>[6-7]</sup>.他们使用 CQUAL<sup>[8]</sup>对 LSM 的完全仲裁(complete mediation)和完全授权(complete authorization)进行了基于静态分析的验证,并成功发现了一些钩子函数放置错误.

在我们开发基于 FreeBSD 的满足 GB17859—1999<sup>[9]</sup>第三级的安全操作系统(相当于 TCSEC<sup>[10]</sup>的 B1 级,已通过公安部计算机信息安全产品质量监督检验中心测评)过程中,我们发现仅验证授权钩子函数的放置是不够的,还需要对安全标记的初始

化钩子函数和销毁钩子函数进行验证.根据我们的研究和实验,安全标记初始化钩子函数放置不准确或不完备会造成空指针引用,而安全标记销毁钩子函数放置不准确或不完备会造成内核空间内存泄漏,最终导致系统崩溃.同时,我们研究发现安全标记销毁钩子函数放置验证是路径敏感的,而 Watson 研究中心使用的 CQUAL 是路径不敏感的,因此,无法直接使用他们的方法验证该类钩子函数放置的准确性和完备性(实例见 3.1 节),也就无法实现对 MAC 框架进行全路径覆盖的正确性验证.

本文对 MAC 框架的正确性验证问题进行了分析,提出了 MAC 框架的安全标记完全初始化合验证、安全标记完全销毁验证和访问操作完全授权验证.为了实施这三类验证,本文提出了一个路径敏感的基于用户自定义检查规则的静态分析方法.该方法通过扩展集成于编译器的静态分析工具 mygcc<sup>[11]</sup>实现全路径覆盖的正确性验证.本文第 2 节分析强制访问控制框架的正确性验证问题;第 3 节介绍正确性验证方法及对 mygcc 的扩展;第 4 节介绍运用本文方法对 TMAC 的正确性验证;第 5 节介绍相关工作;最后是结论.

## 2 正确性验证问题

MAC 框架的作用是实施安全管理员定义的安全策略,而安全策略的正确实施依赖于主体(subject)与客体(object)安全标记的正确标识和访问操作的正确授权.因此,安全标记和访问授权是保证 MAC 框架正确性的两个重要方面,需要对其相关的钩子函数的放置进行正确性验证.

我们首先给出受控对象和受控操作的定义,然后分析 MAC 框架安全标记钩子函数放置验证和访问授权钩子函数放置验证的重要性和必要性.

### 2.1 受控对象与受控操作

操作系统中存在大量对象,如文件、索引节点(inode)、进程(task)、套接字(socket)以及共享内存(shared memory)等,为了实施安全管理员定义的安全策略,这些对象需要得到 MAC 框架的保护,即对这些对象的访问需要得到 MAC 框架的授权.我们称这些对象为受控对象.

受控操作是指作用于受控对象并且会引发系统安全敏感事件的系统操作,如导致信息流动的读操

① <http://www.trustedbsd.org/>

作和写操作. 有些操作虽然作用于受控对象,但并不会导致信息流动,如索引节点的加锁和解锁操作,则不属于受控操作. 为了实施安全管理员定义的安全策略,必须对所有的受控操作进行仲裁.

2.2 安全标记钩子函数验证

尽管有的安全模型可以借助已有的主体和客体信息实现,如 UNIX 信任状(credential)数据、文件访问控制列表(Access Control List,ACL)等,但很多强制访问控制策略需要更多的主体和客体标记信息. 例如,BLP 模型需要为主体和客体关联机密性标记(包括安全级别 level 及安全类别 category),而 BIBA 模型需要为主体和客体关联完整性标记(包括完整性级别 level 及安全类别 category). 在内核中,每个受控对象由一个结构体表示,如 VFS 节点用 *vnode* 结构体表示,每个结构体包含一个策略无关(policy-agnostic)的安全标记结构,用于存储安全标记.

安全标记相关的钩子函数主要包括安全标记初始化(存储空间分配)函数、安全标记赋值函数和安全标记销毁(存储空间释放)函数等,它们必须放置在宿主对象的初始化位置 *object\_initilization* 和销毁位置 *object\_destruction* 之间,并满足如图 1 所示的位置关系.

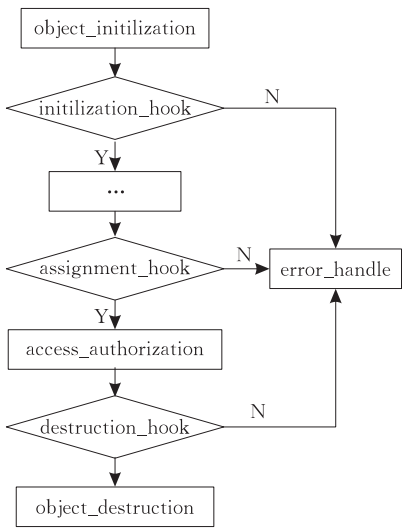


图 1 安全标记钩子函数的位置关系

这里,有 3 个位置关系非常重要,需要进行正确性验证:

- (1)安全标记赋值 *access\_authorization* 放置在安全标记初始化 *initilization\_hook* 之后;
- (2)安全标记销毁 *destruction\_hook* 放置在对象销毁 *object\_destruction* 之前;

(3)访问授权 *access\_authorization* 放置在安全标记赋值 *assignment\_hook* 之后.

如果位置关系(1)不满足,将会出现空指针引用问题,导致内核出错;位置关系(2)不满足,将会出现内核空间内存泄漏问题,并最终导致内核出错;位置关系(3)不满足,将会出现受控对象安全标记不确定问题,影响安全策略的正确实施. 显然,除了这 3 个位置关系外还要满足其它位置关系,如 *initilization\_hook*、*assignment\_hook* 和 *destruction\_hook* 必须放置在 *object\_initilization* 和 *object\_destruction* 之间等,通过我们对 MAC 框架的分析,我们发现上述 3 个位置关系较其它位置关系更加重要,因此,本文仅考虑上述 3 个位置关系的验证.

为了方便描述,我们称位置关系(1)验证为安全标记完全初始化合验证,称位置关系(2)验证为安全标记完全销毁验证.

2.3 访问授权钩子函数验证

访问授权钩子函数分散在内核各个子系统中,如进程管理子系统、文件子系统、进程间通信(Inter Process Communication,IPC)子系统以及网络协议栈等. 通常,访问授权钩子函数具有相似的形式:接收访问授权上下文(请求者信任状、目标对象及相关的安全标记等参数),并依次调用注册的安全策略模块进行授权决策. 根据访问监控器的不可旁过要求<sup>[12]</sup>,MAC 框架的访问授权钩子函数必须满足不可旁过性原则,即访问授权钩子函数放置位置的正确性和完备性要求,如图 2 所示. 为了保证访问授权钩子函数放置的正确性和完备性,我们需要验证:

- (1)受控对象的所有受控操作之前必须放置 *access\_authorization\_hook*;
- (2)所有参数包括受控对象引用的受控操作必须在其前放置 *access\_ authorization\_hook*.

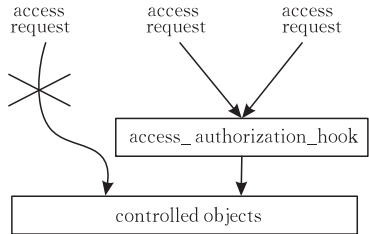


图 2 访问授权钩子函数的关系

如果位置关系(1)和(2)得不到保证,将会出现访问请求旁过授权钩子函数的问题,导致违反安全策略的非预期信息流的产生. 位置关系(1)和

(2)的区别在于,位置关系(1)通常是过程内(intra-procedure)问题,而位置关系(2)通常是过程间(inter-procedure)问题,需要分别进行验证.

为了方便描述,我们称位置关系(1)验证和位置关系(2)验证为受控操作完全授权验证.

### 3 正确性验证方法

#### 3.1 工具选择

在评估了各种可能的 MAC 框架验证工具之后,我们选择对基于 gcc 编译器的静态分析工具 mygcc 进行扩展.主要出于以下考虑:

(1) MAC 框架的安全标记完全初始化、安全标记完全销毁和受控操作完全授权的正确性验证与路径信息密切相关.例如图 3 违背了安全标记完全销毁原则,而非路径敏感分析却无法验证该错误(如果执行路径从 201 行到 227 行,inp 会被释放,而其相关的安全标记没有释放),因此,选用的工具必须是路径敏感的,而 mygcc 支持路径敏感分析.并且,在支持 TMAC 框架的 FreeBSD 内核中,绝大部分访问授权钩子函数与受控操作位于同一个函数中,适合 mygcc 这类路径敏感的过程内分析工具.

```
/* $FreeBSD: src/sys/netinet/in_pcb.c,v 1.197 ... Exp $ */
175 int
176 in_pcballoc(struct socket *so, ...)
177 {
    ...
189 #ifdef MAC
190 error=mac_inpcb_init(inp, M_NOWAIT);
191 if (error != 0)
192     goto out;
193 SOCK_LOCK(so);
194 mac_inpcb_create(so, inp);
195 SOCK_UNLOCK(so);
196 #endif
197
198 #ifdef IPSEC
199 error=ipsec_init_policy(so, &inp->inp_sp);
200 if (error != 0)
201     goto out;
    ...
224 #if defined(IPSEC) || defined(MAC)
225 out;
226 if (error != 0)
227     uma_zfree(pcbinfo->ipi_zone, inp);
228 #endif
229 return (error);
230 }
```

图 3 违反安全标记完全销毁要求的代码段

(2)类似于 CQUAL 和 SPLINT<sup>[13]</sup>的工具是基于源代码注解的(annotation),为开发者带来较大的额外负担.并且,开发者在添加注解过程中可能会引入新的错误.而 mygcc 使用独立的配置文件来指导

检查,在常规的编译过程中执行静态分析.

(3) mygcc 主要针对一般性内核编码错误如死锁和空指针引用等检查,而 MAC 框架的正确性验证有特殊的要求(详见 3.3 节),所以我們必須在 mygcc 原有工作基础上进行扩展.

#### 3.2 验证场景分析

在第 2 节中,我们分析了 MAC 框架正确性验证问题.为了对 mygcc 进行扩展以实施验证,还需对 MAC 框架的典型验证场景进行分析.图 4 给出了 4 个典型的验证场景.

验证安全标记的完全初始化,就是验证当对象创建类受控操作创建一个受控对象后,在程序执行路径上须存在一个安全标记初始化钩子函数(如图 4(a)所示).如果不存在此类钩子函数,则说明违反了安全标记完全初始化要求.

验证安全标记的完全销毁,就是验证受保护对象被销毁之前,在到达该销毁操作的路径上须存在一个安全标记销毁钩子函数.如果不存在此类钩子函数,则说明违反了安全标记完全销毁要求.图 3 给出了违反安全标记完全销毁要求的代码实例.

验证访问完全授权,就是验证实际访问操作类受控操作实施于一个受控对象之前,在程序执行路径上须存在一个或多个访问授权钩子函数(如图 4(c),(d)所示).在现有的 MAC 框架中,某些授权钩子函数的参数并非直接是受控对象,所以需要对我 mygcc 进行扩展以匹配授权钩子函数的参数和受控对象,如图 4(c),(d)的点划线框所示.

由于 mygcc 是过程内分析工具,所以对于一些间接的授权检查,我们需要把分析过程拆分为几个阶段.如图 4(b)所示,我们先检查受控操作 sched\_nice 在函数内执行路径之前是否存在对函数 p\_cansched 的调用,再检查在 p\_cansched 函数中 mac\_proc\_check\_sched 的返回值是否决定 p\_cansched 函数的返回值.

#### 3.3 Mygcc 相关扩展

根据对 MAC 框架的正确性验证问题分析和典型验证场景分析,我们对 mygcc 进行扩展,使扩展后的 mygcc 分析框架满足:能够从函数入口进行检查,能够匹配占位符的类型以及能够建立占位符之间的联系.

(1)能够从函数的入口开始检查.

原有的 mygcc 必须从一个指定的匹配结点也就是某个指定的受控操作开始检查,对函数的抽象语法树进行遍历,直到遇到符合条件的豁免操作成

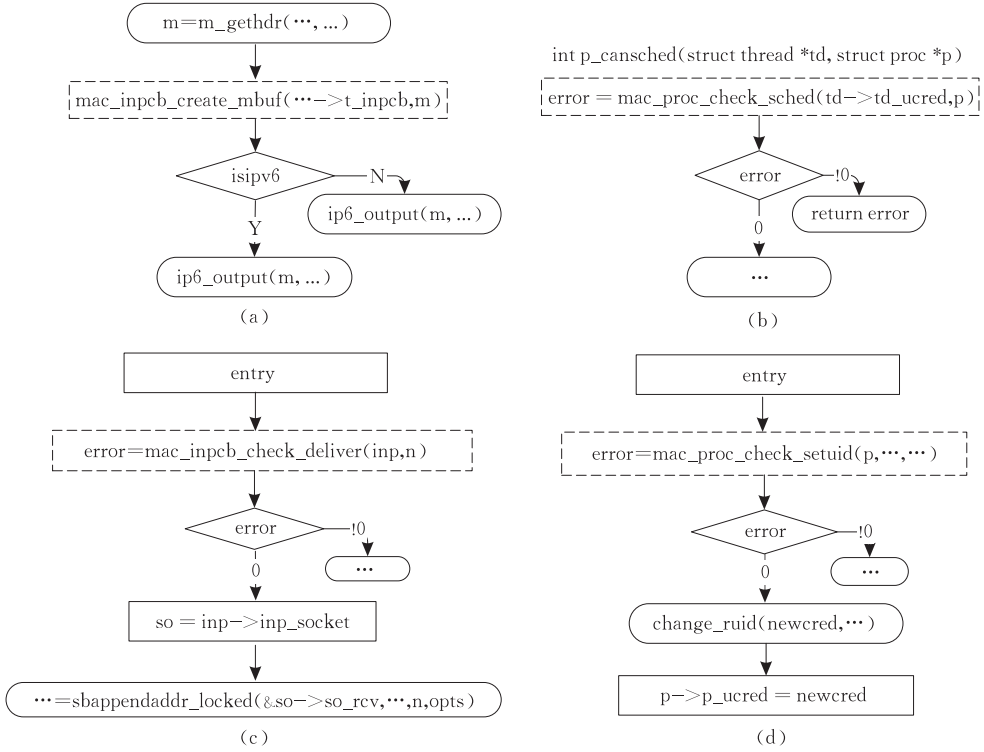


图 4 典型验证场景

功结束,或者遇到指定的另一个受控操作报警. 然而, MAC 框架的静态分析情景在很多情况下只匹配唯一的一个受控操作,如图 4(d)所示. 为了满足这种要求,本文通过扩展 mygcc 实现了从函数入口开始检查,扩展 mygcc 的基本流程,如图 5 所示.

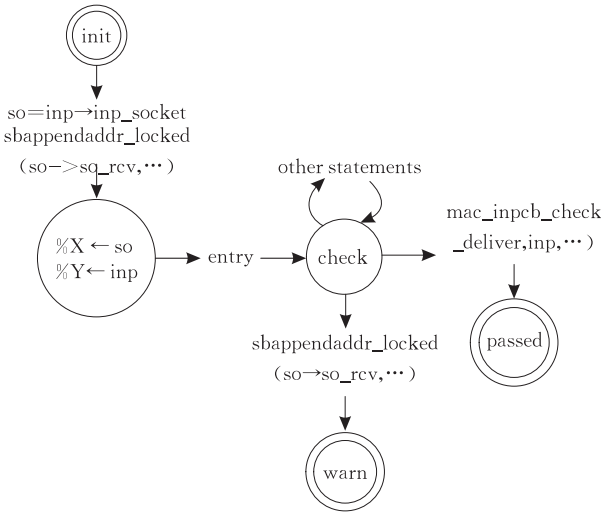


图 5 扩展的 mygcc 分析流程

函数入口检查扩展算法如下:

```
proc check_entry(CFG, condte)
  substs ← ∅;
  foreach node t ∈ CFG do
    global_store ← ∅;
    if condte.from.format_spec = "entry"
```

```
    then if match(t, condte.to)
        then
          substs ← substs ∪ {global_store};
          fi // match(t, condte.to)
        fi // condte.from.format_spec = "entry"
    end // for
  for substs ← substs do
    global_store ← substs;
    list ← [];
    if condte.from.format_spec = "entry"
      then
        foreach node t ∈ CFG.entry.succs do
          list ← [t | list];
        end
        fi // condte.from.format_spec = "entry"
    Do depth first check starting from the element of list;
  end // for
end // proc
```

在上面的算法中,在占位符替换收集阶段,如果遇到配置项 entry,则以配置项中的 to 为占位符替换的收集基准. 在深度优先检查阶段,如果遇到配置项 entry,则深度优先检查的起始结点为函数的入口.

(2) 能够匹配占位符的类型.

原有的 mygcc 检查不匹配占位符的类型. 然而 MAC 框架的正确性验证需要对类型的匹配. 图 3 需要对 MAC 框架的完全销毁进行检查,也就是说

在系统销毁网络访问控制块之前,MAC 框架必须销毁对应的安全标记.仅靠函数 `uma_zfree` 来标识受控操作是不够的,我们必须依靠函数 `uma_zfree` 的第 2 个参数的类型来标识受控操作,即如果 `uma_zfree` 的第 2 个参数的类型是 `inpcb` 的结构指针,则其匹配网络控制块受控操作.占位符的类型匹配算法如下:

```

proc match(t, pattern)
  arglist ← ∅
  parmlist ← ∅
  foreach arg ∈ t.args do
    arglist ← [arg | arglist]
  end //for
  foreach parm ∈ pattern.parms do
    arglist = [arg | rest]
    arglist ← rest
    if parm.type = arg.type
      then global_store ← [global_store | parm ← arg]
      else goto fail;
    fi
  end //for
  goto out;
fail:
  global_store ← ∅;
out:
end //proc

```

(3) 能够建立占位符之间的联系.

在 MAC 框架中,某些访问授权钩子函数的参数并非直接是受控对象,本文通过建立占位符之间的联系来匹配授权钩子函数参数和受控对象.例如在图 4(c)中,受控操作 `sbappendaddr_locked(&so→so_rcv, ..., n, opts)` 中的参数 `&so→so_rcv`,即套接字的接收缓冲区进行添加操作,而现有的 MAC 框架钩子函数检查参数为网络控制块,所以我们通过子图中的点划线框在套接字和网络控制块之间建立联系.下面是基于占位符匹配的深度优先检查算法:

```

proc check_match(CFG, conddate)
  for subst ← substs do
    global_store ← subst
    match_store ← global_to_match[subst]
    list ← construct the depth first check starting
      node list
    while list = [t | rest] do
      list ← rest
      if ¬ visited(t)
        then
          visited(t) ← true
          if matched(t, conddate.to, global_store)

```

```

      then warning “reached t”
      else foreach edge e = t ← t' do
        if ¬ matched(e, avoid, match_store)
          then list ← [t' | list]
        fi
      end
    fi // matched(t, conddate.to, global_store)
  fi // ¬ visited(t)
end //while
end //for
end //proc

```

除了以上扩展,为了适应 MAC 框架正确性验证的需要,我们还对 `mygcc` 进行了其它扩展,具体扩展详见我们对 `mygcc-1.0.0` 提供的补丁<sup>①</sup>.

### 3.4 精确度问题

静态代码分析的最大优势在于完全的路径覆盖性,但是由于静态分析不能获取运行时的上下文信息,误报率非常高,一般的静态分析工具都有 90% 以上误报率,需要人工介入将真正的报错整理出来.因此,当前围绕静态代码分析的大量工作集中在提高分析的精确度上.另一方面,精确度的提高依赖于具体的应用场景,例如,面向缓冲区溢出缺陷的静态分析和面向 MAC 框架的静态分析的精确度提高就有不同的方法.本文对 `mygcc` 的扩展有 60% 左右的代码用于提高分析的精确度.如图 4(c),MAC 框架的钩子函数 `mac_inpcb_check_deliver` 检查的输入参数为互联网控制块指针 `inp` 和网络包的指针 `n`,然而受控操作 `sbappendaddr_locked` 的输入参数是套接字 `so` 的一个字段和网络包的指针 `n`,为了提高精确度,我们在路径分析到达受控操作之前把套接字 `so` 与互联网控制块指针 `inp` 建立联系,如图 4(c) 的虚线框所示.

## 4 TMAC 正确性验证

在开发基于 FreeBSD 的满足 GB17859—1999<sup>[9]</sup> 第三级的安全操作系统过程中,我们运用第 3 节的方法对 TMAC 框架进行了验证,并成功发现了几处钩子函数放置错误.对 MAC 框架进行正确性验证,首先要标识需要验证的受控操作,然后编写静态分析规则.这两项工作完成后,就可以在内核编译过程中执行对强制访问控制框架的正确性验证.

### 4.1 受控操作标识

这里,我们针对 TMAC 给出 3 类典型的受控操

① <http://wiki.freebsd.org/ZhouyiZHOU?action=AttachFile&do=get&target=mygcc.patch>

作的标识方法,其它子系统的受控操作的标识与这里类似。

(1) 跨文件系统的 VFS 的受控操作通过内核对象接口模板和模式文件生成. 涉及的安全敏感函数位于 arch/compile/KERNEL/vnode\_if.h 文件中, 例如 VOP\_READ 和 VOP\_WRITE 等。

(2) 设备文件系统特定的受控操作在文件/src/sys/fs/devfs/devfs\_vnops.c 和 src/sys/fs/devfs/devfs\_devs.c 中手工标识. 例如 devfs\_delete 和 devfs\_vmkdir 等。

(3) 套接字相关的受控操作以如下方式标识: 函数指针调用的参数之一具有 socket 指针结构类型, 函数调用包含 sockbuf 指针结构类型和 mbuf 指针结构类型的两个参数, 我们使用修改了的 gcc 来标识, 相关伪代码如下:

```
proc controlled_socket_op(CFG)
  foreach node  $t_1 \in \text{CFG}$  do
    if  $\text{Type}(t_1) = \text{CALL}$ 
      then foreach node  $t_2 \in \text{Params}(t_1)$  do
        if  $\text{Type}(t_2) = \text{pointer to structure socket}$ 
          then controlledsocketops  $\leftarrow t_1$ ;
```

```
1 condade inpcb_destory {
2 types "%X struct inpcb *"
3 from "entry"
4 to "uma_zfree(%-, %X)"
5 avoid "mac_inpcb_destroy(%X)" or
6 + "%w=mac_inpcb_init(%X, %-); %w != 0"
7 } warning("must call mac_inpcb_destory to destroy inpcb");
```

(a)

```
1 condade inpcb_check_deliver {
2 types "%Y struct socket *; %Z struct inpcb *"
3 from "entry"
4 to "%- = sbappendaddr_locked(&.%Y->so_rcv, %-,
   %X, %-)"
5 match "%Y = %Z->inp_socket"
6 avoid + "%w=mac_inpcb_check_deliver(%Z, %X);
   %w = 0" or
7 - "%w=mac_inpcb_check_deliver(%Z, %X); %w != 0"
8 } warning("must call mac_inpcb_check_deliver before deliver ...");
```

(c)

```
1 condade change_ruid {
2 from "entry"
3 types "%X struct ucred *; %Y struct proc *"
4 to "change_ruid(%X, %-)"
5 follow "%Y->p_ucred = %X"
6 avoid ...
7 + "%W=mac_proc_check_setuid(%Y, %-, %-); %W = 0" or
8 - "%W=mac_proc_check_setuid(%Y, %-, %-); %W != 0"
9 } warning("must call something to check set real uid");
```

(e)

```
goto nextnode;
else if  $\text{Type}(t_2) = \text{pointer to structure}$ 
      mbuf/sockbuf
  then if exists  $t_3 \in \text{Params}(t_1)$  and
         $\text{Type}(t_3) = \text{pointer to structure}$ 
          sockbuf/mbuf
    then controlledsocketops  $\leftarrow t_1$ ;
  goto nextnode;
fi
fi// $\text{Type}(t_2) = \text{pointer to structure socket}$ 
fi// $\text{Type}(t_1) = \text{CALL}$ 
nextnode;
end //for
endproc
```

而套接字初始化的相关受控操作和销毁的相关受控操作分别由 uma\_zalloc(socket\_zone, ...) 和 uma\_zfree(socket\_zone, ...) 来标识。

## 4.2 静态分析规则

我们针对 TMAC 的安全敏感操作编写了 81 个基于扩展 mygcc 的 TMAC 静态分析规则<sup>①</sup>, 这些规则覆盖了 FreeBSD 主要功能模块. 图 6 给出了 5 个典型的静态分析规则。

```
1 condade mbuf_create {
2 types "%X struct mbuf *"
3 from "%X=m_gethdr(%-, %-)" or
4 "%X=m_getcl(%-, %-, %-)"
5 to "ip_output(%X, %-, %-, %-, %-, %-, %-)" or
6 "ip6_output(%X, %-, %-, %-, %-, %-)"
7 avoid "mac_inpcb_create_mbuf(%-, %X)"
8 } warning("uninitialized mbuf send");
```

(b)

```
1 condade p_cansched {
2 functions "p_cansched(%Y, %Z)"
3 types "%Y struct thread *; %Z struct proc *"
4 from appearatleastonce
5 from + "%X=mac_proc_check_sched(%Y->td_ucred, %Z);
   %X != 0" or
6 - "%X=mac_proc_check_sched(%Y->td_ucred, %Z);
   %X = 0"
7 to "return %X"
8 avoid "%X = %-"
9 } warning("must call mac_proc_check_sched to sched other proc");
```

(d)

图 6 静态分析规则示例

① <http://wiki.freebsd.org/ZhouyiZHOU?action=AttachFile&do=get&target=ex5.chk>

图 6(a)给出的规则检查网络控制块对应安全标记的完全销毁,图 6(a)的第 1 行给出规则的名称,第 2 行匹配占位符的类型,第 3 行表明深度优先检查从函数入口开始,第 4 行给出了报警的条件,即销毁类受控操作:对网络控制块执行 `uma_zfree` 函数,第 5 行给出在深度优先检查过程中不报错的条件:

- (1)在程序执行路径上,TMAC 销毁钩子函数 `mac_inpcb_destory` 位于受控操作 `uma_zfree` 之前;
- (2)TMAC 安全标记初始化钩子函数执行失败,因为安全标记未曾分配,所以无须销毁.

图 6(b)给出的规则检查 `mbuf` 对应的安全标记的完全初始化(对应于图 4(a)).即受控操作 `m_gethdr`或 `m_getcl` 产生了一个 `mbuf` 后(第 3,4 行),在程序执行路径上对此 `mbuf` 执行另一个受控操作之前(第 5,6 行),必须存在一个安全标记初始化钩子函数:`mac_inpcb_create_mbuf`(第 7 行).

图 6(c)给出的规则检查对套接字读缓冲区的添加受控操作的完全授权(对应于图 4(c)).即在程序执行路径上对套接字读缓冲区的添加受控操作(第 4 行)之前,TMAC 必须对此操作进行授权(第 6,7 行).如前所述,因为 TMAC 的此类授权参数为此套接字对应的网络控制块,所以还应将这两个结构对应起来(第 5 行).

图 6(d)给出的规则检查 TMAC 访问控制授权钩子函数 `mac_proc_check_sched` 的返回值决定 `p_cansched` 函数的返回值,第 4 行的关键字

`fromappearatleastonce` 说明该钩子函数须在函数 `p_cansched` 内至少出现一次,此规则对应于图 4(b).

图 6(e)给出的规则对应图 4(d),在程序执行路径上,受控操作 `change_ruid`(第 4 行)和信任状赋值(第 5 行)发生之前,TMAC 应该配备访问控制授权钩子函数 `mac_proc_check_setuid`.

### 4.3 验证结果

依据标准的 FreeBSD 内核配置,我们编译并检查了 976 个 C 文件,验证工具共给出了 81 个警告.经进一步分析,其中的 38 个为误报,另外的 43 个为有效警告并已得到 FreeBSD 社区的确认.在已确认的有效警告中,34 个与 NFS 和 FIFOFS 相关,而当前 FreeBSD 的 NFS 和 FIFOFS 缺乏 TMAC 的覆盖,我们的结果为 TMAC 的完备性完善提供了有力的帮助;另外 3 个有效警告分布在 IPv4 和 IPv6 等子系统中,并且已经在 FreeBSD CVS 库中修改,而其余 6 个有效警告的修正方案正在 TrustedBSD 邮件列表中讨论.

通过对验证结果进行分析,我们的方法存在 47%的误报率,还需要对相关算法和配置规则进行进一步完善.但相对于其它静态代码检查工具 95%以上的误报率(如文献[14]等),该方法有较高的精确度.

### 4.4 性能分析

我们使用 `-pg` 选项编译我们的验证工具,并使用 `gprof` 分析执行验证带来的开销(见表 1).

表 1 性能分析

	total time /s	execute_tree_check/(calls/s)	tree_check/(calls/s)	overhead/%
aic7xxx.c	30.56	190/7.61	15390/7.54	24.9
pf.c	60.17	125/20.32	10125/20.17	33.77
kern_prot.c	5.65	63/1.28	5103/1.26	22.65

静态分析花费的时间与文件中匹配的受控操作的有关.我们列举了 3 个例子:`aic7xxx.c`、`pf.c` 和 `kern_prot.c`,文件 `aic7xxx.c` 没有匹配的受控操作,`pf.c` 匹配一个受控操作,而 `kern_prot.c` 匹配 46 个受控操作.

以上测试在 IBM T30 笔记本(P4M 1.8GHz 处理器,512KB L2 高速缓存和 512MB 存储器)上执行.在所有 976 个 C 文件上,正确性验证执行的静态分析带来的平均额外开销是 27.46%.

## 5 相关工作

文献[6]首次提出了针对强制访问控制框架钩

子函数放置的静态分析方法.他们首先使用修改后的 gcc 输出 Linux 内核中受控操作发生的位置,然后利用一个 perl 脚本对 Linux 内核源码按照 CQUAL 的要求进行标注,最后使用 CQUAL 进行非路径敏感的过程间的 LSM 钩子函数完备性分析.由于 CQUAL 是基于类型的静态检查工具,因此文献[6]的方法是路径不敏感的,不能执行路径敏感分析.如该方法无法发现图 3 中的错误.

文献[7]给出了 Linux 内核 LSM 钩子函数完备性检查的运行时验证工具,他们审计系统调用的进入、离开和参数,函数的进入、离开、受控操作和授权.他们利用过滤规则对审计结果进行过滤,然后人工检查由过滤后的审计结果生成的授权图.文献[7]



相比文献[6]易于实施,但是由于动态分析的性质,缺乏对内核的完全覆盖,而且无法做到分析的自动化。

文献[15]提出了一种在 Linux 内核上自动插入授权钩子函数的方法.他们首先对授权钩子函数的源代码进行分析,推导出每个钩子函数所关联的授权;然后他们对 Linux 内核源码进行静态分析来收集受控操作的位置;最后,他们利用一个融合算法来自动插入钩子函数.由于已有的 MAC 框架的正确性依赖于路径特征,所以文献[15]不能用于 MAC 框架的正确性验证.

在面向 C 语言的静态分析工具中,Coverity<sup>[16]</sup>是目前最好的静态分析工具之一. Coverity 使用一种基于状态机的语言 metal 来匹配在事先定义的变量上执行的敏感操作. SPLINT<sup>[13]</sup>是一种主要针对安全漏洞检查的流敏感的静态分析工具,但是需要对被分析的代码事先进行标注. Bell 实验室的轻量级分析工具 Uno 通过遍历抽象语法树来检查常见的代码缺陷,然而 Uno 缺乏自带的预处理器,不能直接处理操作系统内核源代码。

## 6 结 论

强制访问控制框架位于内核中,是安全策略实施的基础,其正确性非常重要.现有的对强制访问控制框架的正确性验证的研究主要集中于对授权钩子函数放置的验证.我们在开发基于 FreeBSD 的满足 GB17859—1999<sup>[9]</sup>第三级的安全操作系统的过程中,通过研究和分析发现强制访问控制框架的正确性不仅依赖于授权钩子函数放置的正确性,还依赖于安全标记初始化钩子函数和销毁钩子函数放置的正确性,否则会导致系统在运行过程中出现空指针引用和内存泄露等问题,最终导致内核崩溃.本文对强制访问控制框架的正确性验证问题进行了分析,在受控操作完全授权验证的基础上,提出了安全标记完全初始化验证和安全标记完全销毁验证.针对现有验证方法的不足,我们给出了一个路径敏感的、适用于强制访问控制框架正确性验证的静态分析方法.该方法通过对静态分析工具 mygcc 进行扩展,实现了对强制访问控制框架授权钩子函数、安全标记初始化钩子函数和安全标记销毁钩子函数放置准确性和完备性的验证.本文的方法具有一般静态方法所具备的完全路径覆盖性,同时具有高的精确度和性能.通过本文的方法,我们成功发现了 Trusted-

BSD MAC 框架的几处钩子函数放置错误,并已得到 FreeBSD 社区的确认.

**致 谢** 感谢 TrustedBSD MAC 框架作者 Robert N. M. Watson 和 mygcc 作者 Nic Volanschi 对我们工作提供的帮助.感谢审稿专家和编辑提出的修改意见!

## 参 考 文 献

- [1] Bell David E, LaPadula Leonard J. Secure computer system: Unified exposition and MULTICS interpretation. The MITRE Corporation, Bedford, MA, USA: MTR-2997 Rev. 1, 1976
- [2] Biba K J. Integrity considerations for secure computer systems. Electronic Systems Division, Air Force Systems Command, Hanscom Air Force Base, Bedford, MA, USA: ESD-TR-76-372, 1977
- [3] Fraser Timothy. LOMAC: Low water-mark integrity protection for COTS environments. NAI Labs Report 0775, 2000
- [4] Wright C, Cowan C, Morris J, Smalley S, Kroah-Hartman G. Linux security modules: General security support for the Linux kernel//Proceedings of the Usenix Security Symposium. Usenix Assoc., 2002: 17-31
- [5] Watson R, Morrison W, Vance C, Feldman B. The TrustedBSD MAC framework: Extensible kernel access control for FreeBSD 5.0//Proceedings of the USENIX Annual Technical Conference. San Antonio, TX, 2003: 285-296
- [6] Zhang Xiao-Lan, Edwards Antony, Jaeger Trent. Using CQUAL for static analysis of authorization hook placement//Proceedings of the 11th Usenix Security Symposium. San Francisco, California, 2002: 33-48
- [7] Edwards Antony, Jaeger Trent, Zhang Xiao-Lan. Runtime verification of authorization hook placement for the Linux security modules framework//Proceedings of the ACM Conference on Computer and Communications Security. Washington, DC, USA, 2002: 225-234
- [8] Foster Jeffrey S, Fähndrich Manuel, Aiken Alexander. A theory of type qualifiers//Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'99). Atlanta, Georgia, 1999: 192-203
- [9] DoD 5200.28-STD, Department of defense standard. Department of Defense Trusted Computer System Evaluation Criteria. National Computer Security Center, Ft. Meade, MD, USA, 1985
- [10] GB 17859—1999, National standards of the People's Republic of China. The protection of computer information system security classification criteria. National Bureau of Quality and Technical Supervision of the PRC, published at September 1999, Implemented at January 2001(in Chinese)

(GB 17859—1999, 中华人民共和国国家标准. 计算机信息系统安全保护等级划分准则. 中国国家质量技术监督局, 1999年9月13日发布, 2001年1月1日实施)

- [11] Volanschi Nic. A portable compiler-integrated approach to permanent checking//Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering. Tokyo, Japan, 2006: 103-112
- [12] Anderson J P. Computer security technology planning study. Air Force Electronic Systems Division, Hanscom AFB, Bedford, MA: Technical Report ESDTR-73-51, 1972
- [13] Larochelle David, Evans David. Statically detecting likely buffer overflow vulnerabilities//Proceedings of the 10th USENIX Security Symposium. Washington, DC, USA, 2001: 177-190
- [14] Meng Ce, He Ye-Ping, Luo Yu-Xiang. Value equality analysis in C program API conformance validation. Journal of Software, 2008, 19(10): 2550-2561(in Chinese)  
(孟策, 贺也平, 罗宇翔. C代码API一致性检验中的等值分析. 软件学报, 2008, 19(10): 2550-2561)
- [15] Ganapathy Vinod, Jaeger Trent, Jha Somesh. Automatic placement of authorization hooks in the Linux security modules framework//Proceedings of the 12th ACM Conference on Computer and Communications Security. Alexandria, Virginia, USA, 2005: 330-339
- [16] Ashcraft Ken, Engler Dawson. Using programmer-written compiler extensions to catch security holes//Proceedings of the IEEE Symposium on Security and Privacy. Oakland, California, USA, 2002: 143-159



**WU Xin-Song**, born in 1974, Ph. D. candidate. His research interests include system software, network and information security.

**ZHOU Zhou-Yi**, born in 1975, Ph. D. candidate. His research interests include system software, network and information security.

**HE Ye-Ping**, born in 1962, Ph. D., professor, Ph. D. supervisor. His research interests include secure protocol design, system security and trusted computing.

**LIANG Hong-Liang**, born in 1972, Ph. D., associate professor. His research interests include software security, system software, network and information security.

**YUAN Chun-Yang**, born in 1979, Ph. D., engineer. His research interests include system software, network and information security.

## Background

This work attributes to the project “FreeBSD-based Trusted Operating System Research and Development”, which is supported by the National Science Foundation of China under grant No. 90818012, the National High Technology Research and Development Program of China (863 Plan) under grant No. 2007AA010601, and Key Project of Chinese Academy of Sciences under grant No. KGX2-YW-125.

Mandatory access control frameworks are important mechanisms of operating systems to support multi-security policies. Security label management hooks and authorization hooks, which are manually inserted into the operating system kernel source code, are essential parts of the mandatory access control framework. Therefore, the correctness of these hooks placement is crucial for the framework. There have

been many researches on the correctness verification of the hooks placement of the LSM (Linux Security Module). However, current researches mainly focus on the verification of the authorization hooks placement. In this work, based on the TrustedBSD MAC framework, the authors analyze the correctness verification problems of the security label management hooks placement and the authorization hooks placement. To address these correctness verifications, this paper also presents a path-sensitive and user-defined-rule based static analysis approach. This approach verifies the accuracy and completeness of hooks placement through extending the compiler integrated static analysis tool-mygcc. It achieves complete coverage of execution paths and has low false positive rate. The verified TrustedBSD MAC framework has been implemented in the NFSARK trusted operating system.