面向PaaS云的信息流控制框架设计与实现

邵 婧[1,2] 陈左宁[2] 殷红武[2] 许国春[2]
(解放军信息工程大学 郑州 450001)[1] (江南计算技术研究所 无锡 214083)[2]

摘 要 分布式信息流控制(DIFC)是实施端到端数据保护的一种有效方法。现有DIFC方法存在信息流控制粒度单一、需要修改语言运行时环境等问题,不能很好地满足PaaS平台的数据安全需求。基于最典型的PaaS云平台GAE,提出了一个信息流控制框架GIFC,其结合了对象级、消息级和SQL级3种控制粒度。组件内基于Python库来控制调用对象的方法中所涉及的实体间的信息交互;组件间消息代理根据消息安全标记来过滤消息,以此限制组件可以接收的消息集;组件与数据库之间扩展GAE中的数据模式支持标记信息在datastore中的持久化存储。实验表明,多种IFC粒度相结合有效平衡了信息流控制精度和运行性能。

关键词 Google应用程序执行引擎,信息流控制,组件,Python,中间件
中图法分类号 TP309 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.12.055

Design and Implementation of Information Flow Control Framework for PaaS

SHAO Jing[1,2] CHEN Zuo-ning[2] YIN Hong-wu[2] XU Guo-chun[2]
(PLA Information Engineering University, Zhengzhou 450001, China)[1]
(Jiangnan Institute of Computing Technology, Wuxi 214083, China)[2]

Abstract Decentralized information flow control is an effective method for end-to-end data protection. The existing DIFC methods have some shortages, for example, information flow tracking granularity is too simplex and language runtime environment has to be modified, which cannot satisfy the data security requirements of PaaS platform. An information flow control framework for GAE was proposed. The framework GIFC combines three granularities of objects, message and SQL. In the component, the information interactions of the entities are controlled with the Python library. The entities are those involved in the method calling for objects. Between the components, message proxies filter the messages with the security labels, in order to restrict the messages received by the component. Between the components and datastore, the data models of GAE are extended, supporting the persistent storage of labels in the datastore. The evaluation shows that the combination of multi IFC granularities effectively balances the precision and performance.

Keywords Google app engine, Information flow control, Component, Python, Middleware

1 引言

目前,云计算正以其便利、廉价、扩展性高等特点,得到了日益广泛的关注和应用。根据服务类型,云计算服务通常可以分为3种模式:基础设施即服务、平台即服务、软件即服务。其中,平台即服务(Platform as a Service, PaaS)通常以可编程容器的形式交付一组软件和基础设施,使得云租户有能力使用供应商支持的编程语言和工具来开发、托管及部署应用程序或服务[1]。PaaS平台自身负责维护资源的动态扩展,可极大降低应用程序的开发成本和开发负担。典型的PaaS平台包括Google的应用程序执行引擎(Google App Engine, GAE)、Microsoft的微软云计算服务平台Windows Azure等。

然而,PaaS平台同样面临着无法回避的数据安全问题[2]。用户将数据存储在云端,不再拥有对自己数据的完全控制能力,只能信任云服务商来提供数据安全保护。但是,现有的PaaS平台只提供了简单的租户隔离机制,缺乏细粒度的污点数据跟踪和信息流控制机制。同时,PaaS平台的应用程序通常由许多组件组合构成。但是,组件代码本身不可避免地会存在一些缺陷或脆弱点,组件之间的交互也可能会存在错误配置。这些都可能会威胁到整个平台和用户数据的安全,造成隐私数据泄露、平台完整性破坏等严重安全后果。因此,需要对PaaS平台实施以数据为中心的端到端的安全防护,保证用户数据的机密性和完整性安全。

分布式信息流控制是实施端到端数据保护的一种有效方法[3]。现有的DIFC通常是针对某一特定的信息流跟踪粒度来设计的,如进程级[4]、变量级[5]、消息级[6,7]等。显然,粒度越细精度越高,但性能损失也越大。而PaaS平台中的信息流动涉及到组件内、组件间、数据库等多个层次,任一特定的控

到稿日期:2014-11-02 返修日期:2015-01-30 本文受核高基项目(2013ZX01029002-001)资助。
邵 婧(1986-),女,博士生,主要研究方向为安全操作系统、云计算安全,E-mail:shaojingfox@163.com;陈左宁(1957-),女,研究员,博士生导师,中国工程院院士,主要研究方向为信息安全、计算机体系结构等;殷红武(1973-),男,高级工程师,硕士生导师,主要研究方向为安全操作系统等;许国春(1979-),男,研究员,主要研究方向为安全操作系统等。

制粒度都不能很好地满足整体安全需求，因此，应该同时结合多种粒度来实施信息流控制。同时，现有 DIFC 的实施大多需要修改语言运行时环境[8-10]。而 PaaS 平台要求开发者必须使用特定的编程环境并遵照特定的编程模型，可以提供给租户或客户对安全的控制较少[11]。因此，PaaS 平台的 DIFC 设计应结合现有平台结构、编程语言和运行时环境，尽可能减小对开发者的影响。此外，标记信息的持久化存储也是 PaaS 平台实施 DIFC 需要特别考虑的问题[12]。

针对上述安全需求，本文结合最典型的 PaaS 云平台 GAE，分析其应用程序开发模式，将应用解耦合为组件，提出了一个面向 GAE 的信息流控制框架(Information Flow Control framework for GAE，GIFC)。该框架设计并实现了 3 种粒度相结合的信息流跟踪和控制：组件内实施对象级的 IFC，基于 Python 库控制对象的方法调用所涉及实体间的信息交互；组件间实施消息级的 IFC，消息代理依据消息头中封装的标记控制组件的信息交互；组件与数据库之间实施 SQL 级的 IFC，扩展 GAE 中的数据模式支持污点标记信息在 datastore 中的持久化存储。GIFC 实现了面向 PaaS 云的端到端的细粒度信息流控制，通过 Python 库和中间件实施 IFC 的方式尽可能减小了对语言运行时环境的影响，实验结果表明 GIFC 所带来的性能开销较小。

2 GIFC 设计

2.1 GAE 安全性分析

Google App Engine(GAE)是一个基于 Google 数据中心的用于开发、托管 Web 应用程序的 PaaS 云平台，目前主要支持基于 Python、Java 和 Go 编程语言的程序开发。GAE 的架构主要分成前端、应用服务器、应用管理节点和数据库 4 个部分[11]。如图 1 所示，前端主要负责路由转发和负载均衡，将请求转发给负载较轻的应用服务器；应用服务器启动应用实例运行来处理请求，生成响应信息并返回给前端；应用管理节点管理前端及应用服务器的运行，实施应用调度及计费等；分布式数据库 datastore 提供可动态扩展的对象级数据存储和检索服务。

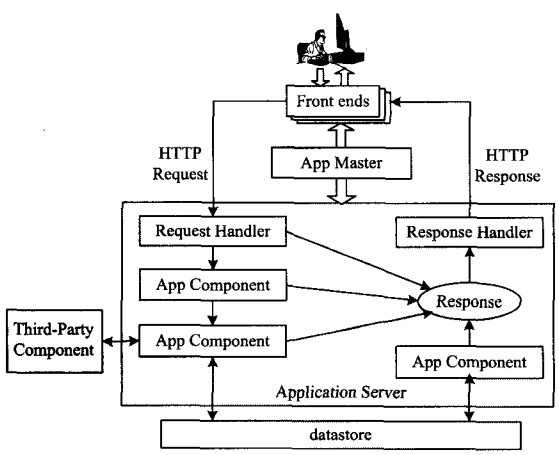

图 1 GAE 信息处理架构

用户通过 HTTP 请求和响应与 GAE 进行信息交互，获取信息或将信息存储到云端。GAE 中 Python Web 应用响应用户 HTTP 请求的完整工作流程为：1)应用服务器根据应用配置文件创建并初始化一个应用实例，配置好语言解释器、库等运行时环境；2)调用应用实例中与 URL 对应的请求句柄，请求句柄初步处理后调用各个应用组件代码，组件处理具体的业务逻辑，访问数据库及 GAE 提供的其他服务，计算并生成响应；3)响应阶段，应用服务器销毁句柄及应用实例回收资源，将生成的 HTTP 响应返回给前端和用户。

通过上述分析可知，用户信息经 HTTP 请求进入 GAE 平台的应用后，就将信息的控制权完全交给了平台。信息通过前端进入组件内部，组件内的一切数据都是以对象的形式出现，通过操作对象来实现程序操作逻辑，信息会伴随组件内代码的运行在对象之间进行流动。同时，任务的完成通常需要多个组件的相互协同合作，组件之间通过消息进行通信，因此隐私信息会通过消息传递从该组件流向应用的其它组件，或者第三方组件及 GAE 平台本身提供的服务组件。另外，为了实现数据持久化和信息共享，隐私信息也可能根据应用需求存储在 datastore 中。由此可知，用户信息会在构成应用的组件内以及各个组件之间流动，也可能持久化存储在数据库中。

随着应用复杂性的增加，组件代码本身不可避免地会存在一些缺陷或脆弱点，组件之间的交互也可能会存在错误的授权策略配置，这些都可能会威胁到整个平台和用户数据的安全。因此，用户数据在进入云平台之后的整个生命周期内都需要实施端到端的、以数据为中心的安全保护，防止隐私数据非授权泄露、平台完整性破坏等安全威胁。GAE 目前提供的安全机制主要是“沙盒”，可以限制应用对基础操作系统的访问权限，并提供各个开发者(租户)的相互隔离。但是，该机制无法满足这种端到端的细粒度信息流控制需求。同时，由于 GAE 强制要求开发者使用特定的编程环境并遵照特定的编程模型，因此可以提供给开发者或用户对安全的控制较少。此外，datastore 的规范化存储使得安全标记无法伴随数据存储到 datastore 中。

综上所述，为了增强终端用户对 GAE 云平台数据保护的信心，需要确保即使平台本身或组件存在安全漏洞，也不会导致用户数据的机密性和完整性安全被破坏。对此，需要结合上述特点，设计满足以下要求的信息流控制框架。首先，需要结合多种信息流控制粒度，以此实现组件内、组件间及数据库这 3 个层次的信息流动安全；其次，由于无法修改语言运行时环境，因此不能采用修改编译器的信息流控制方法，需要设计针对特定语言的库或采用基于中间件的方法来实施信息流控制；第三，必须与 GAE 的数据存储方式相结合，以实现标记信息的持久化存储；最后，该框架的实施对开发者的影响及引入的额外性能开销应尽可能小。

2.2 GIFC 框架设计思想

针对 GAE 平台的上述安全需求及 Python 应用程序开发模式的特点，本文将应用解耦合为多个组件，设计了一个信息流控制框架 GIFC，以实施端到端的数据保护。该框架集成了消息级、对象级及 SQL 级 3 种信息流控制粒度，有效平衡了控制过程的准确性和性能开销。

GIFC 框架的组成如图 2 所示，组件内实施对象级的 IFC，为 Python 对象赋予相应的标记信息，将信息流控制转化为方法调用所涉及实体间的信息交互控制。组件间采用 SOAP 协议实现端到端的消息通信，组件通过消息代理来实

施消息级的 IFC；消息代理负责构造、解析 SOAP 消息及具体的消息分发，将污点标记信息封装在消息头中，以此控制组件的信息交互。组件与数据库之间支持污点标记信息在 datastore 中的持久化存储，采用基于 Python 装饰器的污点存储技术，扩展了 GAE datastore 的数据模式实施 SQL 级的 IFC。

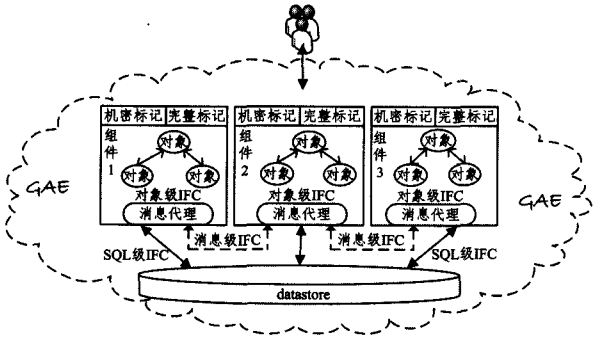

图 2　GIFC 框架

3　GIFC 实现

GIFC 通过将组件内敏感对象和组件间消息与标记关联起来，实施消息级、对象级及 SQL 级 3 种信息流控制粒度相结合的安全保护。GIFC 中的标记设计借鉴文献[4]的思想，分为机密性标记和完整性标记两类，机密性标记防止敏感数据的非授权泄露，完整性标记防止高完整性数据的非法篡改。标记的实施通过相关权限来管理，机密性标记包括两类特权，密级用来访问由机密性标记保护的信息，降级特权可以将高密级的信息进行降密处理，使得敏感信息更加公开。类似地，完整性标记也包括两类特权：完整级和升级特权。GIFC 通过特定的库和消息代理执行特权分配和检查，支持运行时的组件授权，以实施动态特权的相关安全策略。下面分别详细介绍 GIFC 中 3 种粒度的信息流控制。

3.1　SQL 级信息流控制

3.1.1　datastore 分析

GAE 上开发的应用程序使用 datastore 数据库，与传统的关系型数据库不同，datastore 是一个功能强大的分布式数据库。该数据库支持对象级的数据存储和检索服务，且规模能随着应用访问量的上升动态增长。GAE datastore 中数据对象通常称为实体，每个实体包含一个或者多个属性。

如图 3 所示，通过继承数据模型 Model 来定义实体类 Story，该实体类包含 3 个属性字段：字符串类型属性字段 title、text 类型属性字段 body 以及 datetime 类型属性字段 ctime。然后创建 Story 类实例数据对象 story，从客户端用户的 HTTP GET 请求中获取数据来填充 story 实体，并调用 put()方法将新生成的 story 实体存储于数据库中。为了方便从 datastore 中检索实体对象，GAE 提供查询对象接口和 Google Query Language(GQL)两种查询接口，其中 GQL 是一种简化的 SQL 查询语言。用户可以按需指定实体属性值区间和特定类别等查询过滤条件，实体数据查询结果以实例对象的形式返回。图 3 中 10—14 行分别使用两类查询接口从数据库中检索 Story 实体。

```
1. from google. appengine. ext import db
2. class Story(db. Model):
3.       title=db. StringProperty()
4.       body=db. TextProperty()
5.       ctime=db. DateTimeProperty(auto_now_add=True)
6. story=Story()
7. story. title=self. request. get('title')
8. story. body=self. request. get('body')
9. story. put()
10. # object-oriented interface
11. stories=Story. all(). order('-ctime'). fetch()
12. # GQL-based interface
13. gql_query=db. GqlQuery('SELECT * FORM Story ORDER BY
    created')
14. stories=gql_query. fetch()
```

图 3　datastore 实体操作

3.1.2　污点信息持久化存储

GAE 强制要求 datastore 中实体的属性字段必须符合系统规定的特定数据类型。因此，需要扩展 GAE 中的实体模型实现污点信息在 datastore 中的持久化存储。通过研究发现，datastore 数据库操作均位于 GAE SDK 中的 google. appengine. ext. db 包，数据库查询与存取相关的 API 共有 17 个，涉及 Model、Query、GqlQuery 和 Key 4 个类，如表 1 所列。

表 1　datastore 数据对象操作 API

类名	数据库 API	API 描述
Model	get(keys)	返回指定 key 对象的 Model 实例
	get_by_id(ids,parent=None)	返回指定数字 ID 的 Model 实例
	get_by_key_name(key_names,parent)	返回指定键名的 Model 实例
	get_or_insert(key_name, ** kwds)	使用事务获取/创建指定键名 Model 实例
	get_async(keys, ** kwargs)	异步获取多个 Model 实例
	put()	将 Model 实例存储进数据库中
	put_async()	异步存储多个 Model 实例
	parent(model, ** kwargs)	返回父实体 Model 实例
Query	filter(property_operator,value)	返回满足特定属性过滤条件的实体
	order(property)	返回排序后的实体
	ancestor(ancestor)	返回满足祖先过滤条件的实体
	get()	执行查询，返回第一个结果
	fetch(limit,offset=0)	执行查询，返回满足参数要求的结果
GqlQuery	get()	执行查询，返回第一个结果
	fetch(limit,offset=0)	执行查询，返回满足参数要求的结果
key	get(keys)	返回任意 Model 的指定键的实体
	put(models)	将多个 Model 实例批量存储进数据库

针对上述 GAE datastore 中存储和检索实体对象的特点以及对实体属性类型的约束，本文设计并实现了一种基于 Python 装饰器的污点标记存储技术，扩展了 GAE datastore 的机制以支持标记信息的存储，为后续信息流跟踪和控制提供基础支撑。如图 4 所示，装饰器函数 taintEntity()首先为实体增加两个 Text 类型属性字段(conf_tags/integ_tags)，对应机密性标记集和完整性标记集。然后重载数据库操作相关 API 函数实施具体的标记存储。重载的原理大致可分为两类，当调用 put()方法存储实体时，扫描实体所有带标记的属性，生成属性和标记对{property:tag}并保存进 conf_tags 和 integ_tags 中；与之对应，当调用 get()方法从数据库中取数据时，从实体机密性和完整性标记集的属性/标记对中恢复出来。例如，图 4 中 get_wrapper()对 Model 类中的 get()方法进行了重载。与传统的面向关系型数据库的污点标记存储技术相比(如创建镜像表或增加额外列字段存储污点位)，这是一种更轻量级的解决方案，对上层应用透明，且有效避免了传统方式修改底层数据库来实现的缺陷。

```
功能:datastore 污点标记存储          参数:datastore Model
返回值:带标记数据对象
def taintEntity(superModel):
  class innerModel(superModel):
    conf_tags=db.TextProperty()
    integ_tags=db.TextProperty()
    def __init__(self,*args,**kwargs):
      conf_tags={}
      integ_tags={}
    def get_wrapper(*args,**kwargs):
        entity=superModel.get(*args,**kwargs)
        if hasattr(entity,'conf_tags'):
      conf_tags=pickle.loads(str(entity,conf_tags))
          for property,ctag in conf_tags.iteritems():
            superModel.__setattr__(entity,property,
              taint_class(getattr(entity,property),ctag))
      if hasattr(entity,'integ_tags'):
          integ_tags=pickle.loads(str(entity,integ_tags))
          for property,itag in integ_tags.iteritems():
              superModel.__setattr__(entity,property,
                taint_class(getattr(entity,property),itag))
      return entity
  return innerModel
```

图 4　datastore 污点信息存储装饰器函数

3.2　组件间消息级信息流控制

目前，GAE 中的 Web 应用主要采用端到端的消息传递模式。各个组件之间通过消息机制实现通信，每个组件对应一个独立的消息代理负责具体的消息分发，可以根据实际应用需求设计多种类型的消息传递机制。这种基于组件的松耦合结构设计具有通用性，可以适用于多种应用场景。GIFC 扩展了现有消息代理的功能，增加了标记信息的相关处理能力，以此实施组件间的信息流控制。

GIFC 中的消息代理采用简单对象访问协议 SOAP(Simple Object Access Protocol)，它是一种平台无关且独立于特定编程语言的、基于 XML 的可扩展轻量级协议，可用于在云环境下交换结构化信息。SOAP 消息由顶级元素信封、子元素消息头和消息主体组成。消息头主要包含与请求相关的元数据，消息主体封装了具体的消息数据，携带服务调用及其相关的参数信息。GIFC 对 SOAP 消息进行安全扩展，将污点标记信息作为元数据封装在消息头中，与 SOAP 消息一起发送，如图 5 所示。

```
〈?xml version="1.0" encoding="UTF-8"?〉
〈SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"〉
    〈SOAP-ENV:Header〉
      〈GIFC:confidential_label〉
        label:confidential:http://www.restfulwebservices.net/ServicePayment/225634
      〈/GIFC:confidential_label〉
      〈GIFC:integrity_label〉
        label:integrity:http://www.restfulwebservices.net/ServicePayment/213481
      〈/GIFC:integrity_label〉
    〈/SOAP-ENV:Header〉
    〈SOAP-ENV:Body〉
      〈ns1:getOrderID xmlns:ns1="urn:xmethods-Orders"〉
      〈v1 xsi:type="xsd:string"〉27502〈/v1〉
      〈/ns1:getOrderID〉
    〈/SOAP-ENV:Body〉
〈/SOAP-ENV:Envelope〉
```

图 5　支持污点标记的 SOAP 消息设计

本文设计的消息代理基于一个开源 Python 软件 SOAPPy，支持组件间的信息流控制。SOAPPy 中的 SOAPProxy 类代理了组件所有的 SOAP 内部事务，包括：根据函数名和参数列表调用 SOAPBuilder 类创建 SOAP 消息，并将请求消息通过 HTTP 发送到另一端的 SOAP 代理服务器；SOAPServer 接收消息后，解析并提取 SOAP 消息携带的数据，调用本地方法生成并返回新的 SOAP 响应消息。GIFC 通过派生类扩展 SOAPProxy 的功能，重写 SOAP Header，增加新的用于标识污点标记信息的元素实现安全性扩展，如图 5 所示。GIFC 中的代理类根据消息安全标记来过滤消息，以此限制组件可以接收的消息集。当且仅当消息的安全性标记是某个组件对应特权标记的子集时，该消息才可以被代理分发给该组件。

3.3　组件内对象级信息流控制

组件内的信息流控制主要利用 Python 面向对象的特点及动态分发机制，设计了一个污点函数库，其核心功能是为组件内对象提供污点标记及信息流控制的支持。Python 对象可划分为不可变性对象和可变性对象两类。不可变性对象是指该对象创建后值不能更改，相应地，可变性对象是指创建后可以动态修改的对象。

标记信息总是依附于对象，并随着对象间的信息流动自动调整，因此，需要扩展对象内容以提供对象标记和动态调整的支持。针对不可变性对象创建后值无法修改的特点，采用

继承不可变性对象所属的类对象来创建新的子类，通过在子类中添加污点标记信息的属性字段、并重载父类方法的方式来支持信息流控制。对于用户自定义的可变性对象，可以直接在类定义语句外通过赋值变量名给该类动态增加标记属性，同时增加污点信息的相关处理方法。

通过跟踪和控制对象的方法调用来实施组件内的信息流控制。方法的整个执行周期分为两个阶段：方法调用阶段和返回阶段，对方法执行时涉及到的调用者、被调用者、参数与返回值之间任何可能的信息流动进行跟踪和控制。

4 示例分析及性能测试

4.1 示例分析

本节以 GAE 为实验平台，分析如何应用 GIFC 框架来改进 GAE 中云应用的安全性。Cartridge 是一个使用 Python 语言编写的、基于 Mezzanine 网站内容管理平台的开源购物系统，它建立在 Django web 框架之上，旨在为设计和开发电子商务网站提供简洁精巧的核心代码[13]。

文献[14]通过适当修改将 Cartridge 移植到 GAE 上运行，扩展后的 GAE Cartridge 由购物(Shop)、支付(Payment)、递交(Delivery) 3 个功能模块组成。Shop 组件主要负责提供商品和购物车信息，并提供接口以供客户浏览和订购商品；Payment 组件依据 GAE datastore 中存储的个人账户信息验证 Shop 组件传递的客户支付信息的有效性，并根据客户账单信息进行支付；Delivery 组件根据客户地址和商品信息将商品最终递交给客户。

组件之间的信息流动和交互如图 6 所示。从图中可知，客户只能直接访问 Shop 组件提供的相关服务，一旦用户将其订单信息提交给购物系统后，将无法控制其个人隐私信息(通信地址、联系电话、商品清单等)的流向，无法阻止组件的安全漏洞所导致的信息泄露，或者商店对用户隐私信息的滥用。

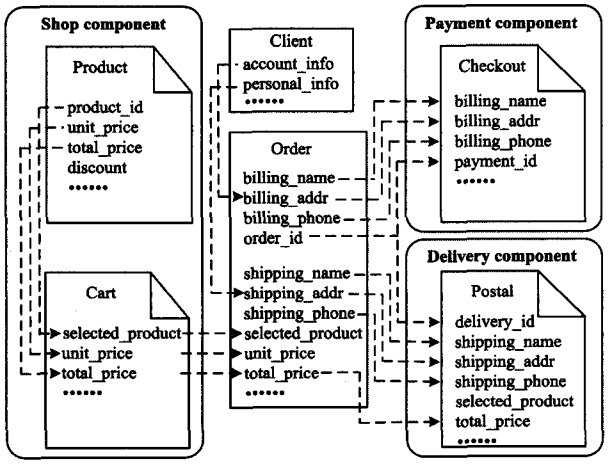

图 6 Cartidge 信息流图

对此，本文将 GIFC 框架应用于该购物系统。Shop 组件标记为具备客户信息读写权限，Payment 组件具备账户信息读写权限，Delivery 组件具备通讯地址读写权限。Shop 组件可以从 datastore 中读取 account_info 类和 personal_info 类等客户信息对应的类对象，组件内随着敏感对象的信息流动标记会自动调整，同时 Shop 组件可以将敏感信息发送给其他组件。当 Payment 组件尝试发送账户信息时，由于发送信息已被污染，组件代理的信息流控制将自动终止发送过程；类似地，Delivery 组件向外发送通讯地址信息时也会失败。通过上述分析可知，GIFC 框架的应用可极大缓解用户的安全隐私威胁。

4.2 性能测试

本节通过对比 GIFC 应用前后 GAE Cartridge 处理请求的响应速度，来测试 GIFC 框架对系统性能的影响。重复多次执行商品信息浏览和订单查询等操作，分别记录 Cartridge 的消息处理时间以及数据库操作的时间开销，并进行数据统计与分析。如表 2 所列，GIFC 信息流控制方法调用的时间开销为 5.38%～9.06%，数据库操作的平均延迟为 3.41%，总延迟率为 8.79%～12.47%，优于变量级的信息流控制系统 FlowR[5] 的平均开销(12%～15%)。测试结果表明，GIFC 引入的开销较小，满足设计需求。

表 2 datastore 性能开销

数据库操作	单个实体平均时间		延迟率
	GIFC 关闭	GIFC 开启	
put	3.16ms	3.26ms	3.21%
get	0.73ms	0.76ms	4.13%
update	2.52ms	2.59ms	2.90%
delete	2.69ms	2.78ms	3.42%

结束语 PaaS 平台在给应用开发带来极大便利的同时，也面临着无法回避的数据安全问题。对此，本文分析了 GAE 平台上基于组件的应用程序开发模式以及分布式数据库的存储机制，提出了一个面向 GAE 的信息流控制框架 GIFC。该框架扩展了 GAE datastore 的机制，实现了污点标记信息的持久化存储；结合 Python 语言特点设计污点函数库，实现了组件内对象级的污点标记及信息流控制；组件间扩展消息代理，对消息头中标记相关元数据进行处理。实验结果表明，GIFC 很好地满足了 GAE 平台的数据隐私安全需求，3 种粒度相结合的信息流控制所带来的额外性能开销较小。

下一步将继续完善该框架的实施细节，设计相应的信息流控制模型，并采用相关的形式化方法来分析和验证框架设计的正确性和安全性。

参 考 文 献

[1] 温克勒. 云计算安全：架构、战略、标准与运营[M]. 刘戈舟，等译. 北京：机械工业出版社，2013
Winkler V J R. Securing the Cloud: Cloud Computing Security Technologies and Tactics[M]. Liu Ge-zhou, et al. Beijing: China Machine Press, 2013

[2] Fernandes D A B, Soares L F B, Gomes J V, et al. Security issues in cloud environments: a survey[J]. International Journal of Information Security, 2014, 13(2): 113-170

[3] Bacon J, Eyers D, Pasquier T, et al. Information Flow Control for secure cloud computing[J]. IEEE Transactions on Network and Service Management, 2014, 11(1): 76-89

[4] Krohn M, Yip A, Brodsky M, et al. Information flow control for standard OS abstractions[C]// 21th ACM SIGOPS Symposium on Operating Systems Principles. New York, ACM, 2007: 321-334

[5] Pasquier T F J M, Bacon J, Shand B. FlowR: aspect oriented programming for information flow control in ruby[C]// 13th International Conference on Modularity. New York, ACM, 2014: 37-48

[6] Hosek P, Migliavacca M, Papagiannis I, et al. SafeWeb: A middleware for securing Ruby-based Web applications[C]// Proceedings of the 12th International Middleware Conference. International Federation for Information Processing. 2011: 480-499

[7] Migliavacca M, Papagiannis I, Eyers D M, et al. DEFCON: High-Performance Event Processing with Information Security[C]// USENIX Annual Technical Conference. Boston, MA, 2010: 88-102

[8] Enck W, Gilbert P, Chun B G, et al. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones[C]// OSDI. Berkeley, CA, USA: USENIX Association, 2010: 255-270

[9] Rodero-Merino L, Vaquero L M, Caron E, et al. Building safe PaaS clouds: A survey on security in multitenant software platforms[J]. Computers & Security, 2012, 31(1): 96-108

[10] Pappas V, Kemerlis V P, Zavou A, et al. CloudFence: Data Flow Tracking as a Cloud Service[M]// Research in Attacks, Intrusions, and Defenses. Springer Berlin Heidelberg, 2013: 411-431

[11] 刘鹏. 云计算[M]. 北京: 电子工业出版社, 2011
Liu Peng. Cloud Computing [M]. Beijing: Publishing house of electronic industry, 2011

[12] Bello L, Russo A. Towards a taint mode for cloud computing Web applications[C]// 7th Workshop on Programming Languages and Analysis for Security. New York, ACM, 2012, 7: 1-7, 12

[13] McDonald S. [EB/OL]. (2012-11-18)[2014-01-20]. http://bitbucket.org/ stephenmcd/cartridge/

[14] Johnson N. [EB/OL]. (2010-03-12) [2014-03-25]. http://googleappengine.blogspot.com/2010/03/app-engine-community-update.html

(上接第 228 页)

Liu Tian. Research and Application of Software Distribution Mechanism[D]. Shijiazhuang: Shijiazhuang Tiedao University, 2014

[3] 王华, 刘焕敏, 冯朝阳, 等. 一种分布式软件发布部署系统[J]. 计算机系统应用, 2012, 21(2): 1-4
Wang Hua, Liu Huan-min, Feng Zhao-yang, et al. Distributed Software Release and Disposition System[J]. Computer Systems & Applications, 2012, 21(2): 1-4

[4] 陈伟, 魏峻, 黄涛. W4H: 一个面向软件部署的技术分析框架[J]. 软件学报, 2012, 23(7): 1669-1687
Chen Wei, Wei Jun, Huang Tao. W4H: An Analytical Framework for Software Deployment Technologies [J]. Journal of Software, 2012, 23(7): 1669-1687

[5] Malek S, Medvidovic N, Mikic-Rakic M. An extensible framework for improving a distributed software system's deployment architecture[J]. IEEE Trans. on Software Engineering, 2012, 38(1): 73-100

[6] 张娴. 基于可信计算的软件安全下载设计与实现[D]. 西安: 西安电子科技大学, 2010
Zhang Xian. Design and Implement of Software Secure Downloading Based on Trusted Computing[D]. Xi'an: Xidian University, 2010

[7] Virvilis N, Gritzalis D. Trusted Computing vs Advanced Persistent Threats: Can a defender win this game? [C]// 2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing, and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC). 2013: 396-403

[8] 蒋泽. 可信网络中用户行为可信评估的研究[D]. 重庆: 重庆大学, 2011
Jiang Ze. Study on Evaluation of User Behavior Trust in Trusted Network[D]. Chongqing: Chongqing University, 2011

[9] 林闯, 田立勤. 可信网络中用户行为可信评价的研究[J]. 计算机研究与发展, 2008, 45(12): 2033-2043
Lin Chuang, Tian Li-qin. Research on User Behavior Trust in Trustworthy Network[J]. Journal of Computer Research and Development, 2008, 45(12): 2033-2043

[10] 温博为. 可信计算平台技术应用研究[D]. 西安: 陕西师范大学, 2013
Wen Bo-wei. Application Research on Trusted Computing Platform Technology[D]. Xi'an: Shaanxi Normal University, 2013

[11] 王志勇. 基于行为声明的动态可信度量技术研究[D]. 北京: 北京工业大学, 2013
Wang Zhi-yong. Dynamic Trustworthiness Measurement Technology Based on Behavior Declaration[D]. Beijing: Beijing University of Technology, 2013

[12] 曾梦岐, 卿昱, 谭平璋, 等. 一种基于标识认证的信任链建立方法[C]// 第一届中国可信理论与实践学术会议论文集. 北京: 清华大学出版社, 2009
Zeng Meng-qi, Qing Yu, Tan Ping-zhang, et al. Establishment Method of Trust Chain Based on Identity Authentication[C]// Proceedings of the first China credible theory and practice conference. Beijing: Tsing Hua University Press, 2009

[13] 孙迪. 软件行为可信技术研究[D]. 北京: 北京工业大学, 2013
Sun Di. Research on Software Behavior Trustworthy Technology[D]. Beijing: Beijing University of Technology, 2013

[14] Ruiz J L, Duenas J C, Usro F, et al. Deployment in dynamic environments[C]// DECOR04 (2004). 2004: 85-98

[15] Hall R, Heimbigner D, Wolf A L. Specifying the Deployable Software Description Format in XML: CU-SERL-207-99[R]. Software Engineering Research Laboratory, University of Colorado, 1999

[16] Hall R S, Heimbigner D, Wolf A L. Evaluating Software Deployment Languages and Schema[C]// 14th IEEE International Conference on Software Maintenance (ICSM'98). 1998: 177

[17] Van Hoff A, Partovi H, Thai T. The Open Software DescriptionFormat (OSD) [OL]. http://www.w3.org /TR/NOTE-OSD

[18] Distributed Management Task Force. Common Information Model (CIM) Specification (Version 3.0) [OL]. http://www.dmtf.org/spec/cim_spec_v30

[19] Coupaye T, Estublier J. Foundations of enterprise software deployment[C]// Proc. of the Euromicro Conf. on Software Maintenance and Reengineering. Zurich: IEEE Computer Society, 2000

[20] 王怀民, 唐扬斌, 尹刚, 等. 互联网软件的可信机理[J]. 中国科学(E辑), 2006, 36(10): 1156-1169
Wang Huai-min, Tang Yang-bin, Yin Gang, et al. Trustworthiness Mechanism of Internet Software[J]. Science in China (Series E), 2006, 36(10): 1156-1169