郑州大学毕业设计（论文）

题 目： 基于动态分析的分布式恶意

代码检测系统的设计与实现

指导教师： 庞海波 职称： 助教

学生姓名： 赵晓宁 学号： 20117740467

专 业： 计算机科学与技术(软件测试)

院（系）： 软件与应用科技学院

完成时间： 2017.04.20

2017年 04月 20日

# 目录

[目录 2](#_Toc480129146)

[摘要 1](#_Toc480129147)

[ABSTRACT 2](#_Toc480129148)

[第1章 绪论 3](#_Toc480129149)

[1.1 系统背景 3](#_Toc480129150)

[1.2 系统开发目标 3](#_Toc480129151)

[1.3 论文结构安排 4](#_Toc480129152)

[第2章 系统设计技术及开发工具 6](#_Toc480129153)

[2.1 系统设计技术概述 6](#_Toc480129154)

[2.1.1 Bootstrap技术 6](#_Toc480129155)

[2.1.2 HighCharts技术 6](#_Toc480129156)

[2.1.3 ThinkPHP技术 8](#_Toc480129157)

[2.2 项目设计工具 9](#_Toc480129158)

[2.2.1 WampServer工具 9](#_Toc480129159)

[2.2.2 MySQL数据库 9](#_Toc480129160)

[2.3 本章小结 10](#_Toc480129161)

[第3章 家庭理财系统需求分析 11](#_Toc480129162)

[3.1 系统总体需求分析 11](#_Toc480129163)

[3.2 系统功能需求 11](#_Toc480129164)

[3.3 系统数据需求分析 13](#_Toc480129165)

[3.4 本章小结 14](#_Toc480129166)

[第4章 家庭理财系统模块设计 15](#_Toc480129167)

[4.1 家庭理财系统设计原则 15](#_Toc480129168)

[4.2软件技术架构设计 15](#_Toc480129169)

[4.3 项目总体框架体系 16](#_Toc480129170)

[4.3.1用户管理模块设计 16](#_Toc480129171)

[4.3.2记账模块设计 16](#_Toc480129172)

[4.3.3报表模块设计 16](#_Toc480129173)

[4.3.4理财日记模块设计 17](#_Toc480129174)

[4.3.5周期账本模块设计 17](#_Toc480129175)

[4.3.6 设置模块设计 17](#_Toc480129176)

[4.3.7 小功能模块设计 17](#_Toc480129177)

[4.4 数据库总体设计 17](#_Toc480129178)

[4.4.1数据库设计的基本原则 17](#_Toc480129179)

[4.4.2数据库概念设计 18](#_Toc480129180)

[4.4.3数据库逻辑设计 23](#_Toc480129181)

[4.4.4数据库物理设计 23](#_Toc480129182)

[4.5 本章小结 26](#_Toc480129183)

[第5章 家庭理财系统实现 27](#_Toc480129184)

[5.1数据库实现 27](#_Toc480129185)

[5.1.1数据表的创建 27](#_Toc480129186)

[5.1.2数据库数据的添加 28](#_Toc480129187)

[5.1.3数据库数据的更新 28](#_Toc480129188)

[5.1.4数据库数据的删除 28](#_Toc480129189)

[5.2 系统实现 28](#_Toc480129190)

[5.2.1 用户管理模块实现 28](#_Toc480129191)

[5.2.2 记账模块实现 30](#_Toc480129192)

[5.2.3 报表模块实现 32](#_Toc480129193)

[5.2.4 理财日记模块实现 33](#_Toc480129194)

[5.2.5 周期账本模块实现 33](#_Toc480129195)

[5.2.6 设置模块实现 34](#_Toc480129196)

[5.2.7 小功能模块实现 35](#_Toc480129197)

[5.9 本章小结 35](#_Toc480129198)

[第6章 家庭理财系统的测试 36](#_Toc480129199)

[6.1系统测试原则 36](#_Toc480129200)

[6.2系统功能测试 36](#_Toc480129201)

[6.3本章小结 38](#_Toc480129202)

[结论 39](#_Toc480129203)

[参考文献 41](#_Toc480129204)

[致谢 42](#_Toc480129205)

# 

# 摘要

随着电脑手机的普及和互联网的不断发展，我们在生活中和工作中越来越离不开它们，我们使用它们购物，支付，玩游戏，聊天。计算机病毒和木马也越来越多的影响我们的生活。由于接触互联网的人越来越多，病毒和木马的制作已经越来越容易被人们掌握。面对日益增多的海量恶意代码传统的检测恶意代码的方式已不能满足检测的需求，因此急需一种能短时间快速检测大量恶意代码的方式。

本文的主要工作体现在两个方面：

1.提出使用动态分析技术对恶意代码进行检测。分析系统会提前在virtualbox中准备好各种类型版本的客户机，在进行分析时将恶意代码上传至用户选择好的客户机中，恢复此虚拟机的快照，然后将恶意代码以及相关的python代码上传至客户机，python脚本会执行恶意代码并且启动各个分析模块将分析的结果实时上传至主节点的mongodb数据库。

2.提出使用分布式集群对恶意代码进行检测。集群中的分支节点提供各种类型的客户机，当有大量恶意软件到来时调度系统可以创建队列，为每个恶意代码检测任务分配分析节点和对应类型的客户机，动态开启客户机运行恶意软件，记录其行为。

# ABSTRACT

# 第1章 绪论

## 1.1 系统背景

现今，恶意代码的数量呈几何增长，之前靠单机就可以处理恶意代码的方法已经不能应对海量的分析需求。因此，急需一种能快速检测海量恶意代码的系统来自满足这种需求。

## 

## 1.2 系统开发目标

## 1.3 论文结构安排

论文共由6个章节组成，主要内容及结构安排如下：

第1章，绪论，介绍本课题的背景来源、研究意义和主要研究内容，并对相关技术现状做简要分析；

第2章，系统设计技术及开发工具，对整个系统的设计方案做总体的介绍以及现场控制单元方案的设计及选择。

第3章，需求分析，介绍了设计中所使用的相关技术，包括CrossWorks、ARM技术、无线通信技术、NTC热敏电阻测温原理技术等。

第4章,系统的设计，采用EDA技术对系统原理图进行设计,包括整体原理图的设计和各个部分的设计及PCB的设计开发与布线。

第5章,系统的实现，在CrossWorks开发环境中利用C语言对系统编程实现各个功能,包括测温子程序,控制主程序及串口发送程序的流程图,程序功能描述以及源程序的编写。

第6章,系统的测试 ,在CrossWorks开发环境中对系统进行功能调试,包括系统硬件调试和系统软件调试以及软硬件之间的联调,及在调试中遇到的问题及解决方案。

# 第2章 系统设计技术及开发工具

## 2.1 系统设计技术概述

分布式：

分布式是相对中心化而来，强调的是任务在多个物理隔离的节点上进行。中心化带来的主要问题是可靠性，若中心节点宕机则整个系统不可用，分布式除了解决部分中心化问题，也倾向于分散负载，但分布式会带来很多的其他问题，最主要的就是一致性。简单来说，分布式处理就是多台相连的计算机各自承担同一工作任务的不同部分，在人的控制下，同时运行，共同完成同一件工作任务。分布式处理系统包含硬件，控制系统，接口系统，数据，应用程序和人等六个要素。而控制系统中包含了分布式操作系统，分布式数据库以及通信协议等。

集群：

集群（cluster）技术是一种较新的技术，通过集群技术，可以在付出较低成本的情况下获得在性能、可靠性、灵活性方面的相对较高的收益，其任务调度则是集群系统中的核心技术。集群是一组相互独立的、通过高速网络互联的计算机，它们构成了一个组，并以单一系统的模式加以管理。一个客户与集群相互作用时，集群像是一个独立的服务器。集群配置是用于提高可用性和可缩放性。

负载均衡：

负载均衡，英文名称为LoadBalance，其意思就是分摊到多个操作单元上进行执行，例如Web服务器、FTP服务器、企业关键应用服务器和其它关键任务服务器等，从而共同完成工作任务。负载均衡建立在现有网络结构之上，它提供了一种廉价有效透明的方法扩展网络设备和服务器的带宽、增加吞吐量、加强网络数据处理能力、提高网络的灵活性和可用性。

动态分析：

动态分析（dynamicanalysis）是对变动的实际过程所进行的分析，其中包括分析有关变量在一定时间过程中的变动，这一些变量在变动过程里相互影响和彼此制约的关系，及在每个时点上变动的速率等。动态分析法一个重要特点为考虑时间因素的影响。并将变化当作一个连续的过程来看待。动态分析因为考虑各种变量随时间延伸而变化对整个体系的影响，因而难度较大。

## 2.2 项目设计工具

Python：Python是一种面向对象的解释计算机编程语言，由荷兰语Guido van Rossum于1989年发明，是1991年发布的第一个公开发行版。其语法简洁扼要，其中一个特点是被迫使用空白（空白）作为声明缩进。 Python有一个丰富而强大的库。它通常是昵称的胶水语言，可以用其他语言制作各种模块（特别是C / C ++）很容易链接在一起。常见的应用场景是使用Python来快速生成程序的原型（有时甚至程序的最终界面），然后用更合适的语言（如3D图形）重写它们，性能要求特别高，您可以使用C / C ++重写，然后封装Python可以调用扩展类库。应该注意的是，当您使用扩展类库时，可能需要考虑平台问题，有些可能不提供跨平台的实现。

Flask：flask是一个使用 Python 编写的轻量级 Web 应用框架。其 WSGI 工具箱采用 Werkzeug ，模板引擎则使用 Jinja2 。它使用 BSD 授权。Flask也被称为 “microframework” ，因为它使用简单的核心，用 extension 增加其他功能。Flask没有默认使用的数据库、窗体验证工具。然而，Flask保留了扩增的弹性，可以用Flask-extension加入这些功能：ORM、窗体验证工具、文件上传、各种开放式身份验证技术。最新版本为0.11。

Django：Django是一个开放源代码的Web应用框架，由Python写成。采用了MVC的框架模式，即模型M，视图V和控制器C。它最初是被开发来用于管理劳伦斯出版集团旗下的一些以新闻内容为主的网站的，即是CMS（内容管理系统）软件。并于2005年7月在BSD许可证下发布。这套框架是以比利时的吉普赛爵士吉他手Django Reinhardt来命名的。Django 项目是一个python定制框架，它源自一个在线新闻 Web 站点，于 2005 年以开源的形式被释放出来。Django 框架的核心组件有：用于创建模型的对象关系映射；为最终用户设计的完美管理界面；一流的 URL 设计；设计者友好的模板语言；缓存系统。由于Django在近年来的迅速发展，应用越来越广泛，被著名IT开发杂志SD Times评选为2013 SD Times 100，位列“API、库和框架”分类第6位，被认为是该领域的佼佼者

Mongodb：MongoDB是一个基于分布式文件存储的数据库。由C++语言编写。旨在为WEB应用提供可扩展的高性能数据存储解决方案。它是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。他支持的数据结构非常松散，是类似json的bson格式，因此可以存储比较复杂的数据类型。Mongo最大的特点是他支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。它的特点是高性能、易部署、易使用，存储数据非常方便。

PostgresQL：PostgresQL是以加州大学伯克利分校计算机系开发的POSTGRES，现在已经更名为PostgreSQL，版本4.2为基础的对象关系型数据库管理系统（ORDBMS）。PostgrSQL支持大部分SQL标准并且提供了许多其他现代特性：复杂查询、外键、触发器、视图、事务完整性、MVCC。同样，PostgreSQL可以用许多方法扩展，比如，通过增加新的数据类型、函数、操作符、聚集函数、索引。免费使用、修改、和分发 PostgreSQL，不管是私用、商用、还是学术研究使用。PostgreSQL是一个自由的对象-关系数据库服务器(数据库管理系统)，它在灵活的BSD-风格许可证下发行。它提供了相对其他开放源代码数据库系统(比如MySQL 和 Firebird)，和专有系统(比如 Oracle、Sybase、IBM 的 DB2 和 Microsoft SQL Server)之外的另一种选择。

Sublime: PyCharm是一种PythonIDE，带有一整套可以帮助用户在使用Python语言开发时提高其效率的工具，比如调试、语法高亮、Project管理、代码跳转、智能提示、自动完成、单元测试、版本控制。此外，该IDE提供了一些高级功能，以用于支持Django框架下的专业Web开发。PyCharm是由JetBrains打造的一款Python IDE，VS2010的重构插件Resharper就是出自JetBrains之手。同时支持Google App Engine，PyCharm支持IronPython。这些功能在先进代码分析程序的支持下，使 PyCharm 成为 Python 专业开发人员和刚起步人员使用的有力工具。首先，PyCharm用于一般IDE具备的功能，比如，调试、语法高亮、Project管理、代码跳转、智能提示、自动完成、单元测试、版本控制。另外，PyCharm还提供了一些很好的功能用于Django开发，同时支持Google App Engine，更酷的是，PyCharm支持IronPython。

## 2.3 本章小结

# 第3章 分布式恶意代码检测系统需求分析

## 

## 3.1 系统总体需求分析

早期发现电脑病毒，因为检测范围是主机，是完全隔离和可控的环境，在这种环境下不需要协作但是，网络恶意代码遍布互联网，是一种孤立而不是完全可控的环境，特别是在互联网的快速发展，今天网络规模的不断扩大，反过来，每个网络的隔离，然后去除恶意代码，这种做法是不切实际的。同时，恶意代码的传播速度非常快，导致网络危害性增加，对恶意代码的快速反应要求越来越高。可以在分布式环境中分布来解决这个问题。每个检测节点异常检测到捕获的数据，发现未知的恶意代码，获取其样本，然后进行特征分析以获得其特征。然后将功能更新到整个网络中的所有功能检测节点，从而完成恶意代码协作的应急响应，还可以包括异常分析，通知响应方式，特征广播等。

## 3.2 系统功能需求

该系统需要让用户可以通过restful api或者web界面提交恶意代码样本至主节点的数据库中，调度系统将会为每个任务分配其请求的资源（虚拟机的类型，版本以及网络条件等），当样本的数量足够多时可以将它们负载均衡到集群里面每个节点上进行分析，并将结果返回到主节点的

## 3.3 系统数据需求分析

## 

## 3.4 本章小结

本章主要介绍了分布式恶意代码监测系统的需求分析，包括系统总体需求分析，系统功能需求分析和系统数据需求分析

# 第4章 分布式调度系统模块设计

## 4.1 分布式调度系统设计原则

系统设计是新系统的物理设计阶段。根据系统分析阶段所确定的新系统的逻辑模型、功能要求，在用户提供的环境条件下，设计出一个能在计算机网络环境上实施的方案，即建立新系统的物理模型。这个阶段的任务是设计软件系统的模块层次结构，设计数据库的结构以及设计模块的控制流程，其目的是明确软件系统"如何做"。这个阶段又分两个步骤：概要设计和详细设计。概要设计解决软件系统的模块划分和模块的层次机构以及数据库设计；详细设计解决每个模块的控制流程，内部算法和数据结构的设计。这个阶段结束，要交付概要设计说明书和设计说明，也可以合并在一起，称为设计说明书。在系统分析的基础上，设计出能满足预定目标的系统的过程。系统设计内容主要包括：确定设计方针和方法,将系统分解为若干子系统,确定各子系统的目标、功能及其相互关系，决定对子系统的管理体制和控制方式，对各子系统进行技术设计和评价，对全系统进行技术设计和评价等。系统设计通常应用两种方法：一种是归纳法，另一种是演绎法。应用归纳法进行系统设计的程序是：首先尽可能地收集现有的和过去的同类系统的系统设计资料；在对这些系统的设计、制造和运行状况进行分析研究的基础上,根据所设计的系统的功能要求进行多次选择,然后对少数几个同类系统作出相应修正，最后得出一个理想的系统。演绎法是一种公理化方法，即先从普遍的规则和原理出发，根据设计人员的知识和经验，从具有一定功能的元素集合中选择能符合系统功能要求的多种元素，然后将这些元素按照一定形式进行组合（见系统结构），从而创造出具有所需功能的新系统。在系统设计的实践中，这两种方法往往是并用的。

## 4.2软件技术架构设计

## 

## 4.3 项目总体框架体系

分布式调度模块主要分为主节点和从节点两大部分：

主节点负责接收任务，调度任务和汇总任务的结果。从节点负责接受主节点的调度和运行主节点为他分配的任务，并将分析的结果传回主节点。

系统的运行也分为两部分：首先是从节点的启动，在系统开机后首先需要运行virtualbox虚拟机创建vboxnet0虚拟网卡。然后运行cuckoo.py文件，该文件启动之后会创建服务器，加载各个分析模块，获取该节点当前可用的客户机，并周期性的查询数据库的tasks表，取出所有的任务并筛选出status为pending的任务，如果有处于等待状态的任务则取出该任务，并准备好该任务所需要的条件（将样本文件，分析所需要的python脚本和dll动态链接库打包成一个zip文件）然后开启虚拟机，恢复该虚拟机之前保存的快照（该快照运行了一个agent.py脚本创建了一个服务器，调度程序会去连接该服务器并将打包好的文件上传至虚拟机，然后该脚本将收到的zip压缩文件解压到之前配置的目录。），虚拟机启动之后该进程会发送指令运行各个模块（包括模拟鼠标点击，截图等），并让虚拟机启动样本文件并注入之前准备好的动态链接库（该动态链接库会hook一些系统敏感的函数当测试样本调用这些函数时它会记录这些函数的调用情况并连接agent.py创建的命名管道将收集的的信息发送至agent进程），agent进程会连接cuckoo.py文件创建的

## 4.4分布式调度模块设计

分布式调度模块设计：

分布式调度模块主要有三个部分：

1. 分配任务：找出所有可用节点，查询节点的状态，找出处于等待状态的任务，将任务分配到查询出来的空闲节点（设置任务的节点id）。
2. 设置节点状态：每隔一段时间向所有存活节点发送心跳根据心跳回应获取该节点此时的状态，然后将节点的状态保存至数据库。
3. 监控每个可用的节点，从数据库取出该节点所有分配到的处于等待状态的任务，将任务使用restful api远程提交至该节点。并周期性从该节点查询出任务列表，如果有处于完成状态的任务则通过restful api从该节点获取该任务的报告（分析结果包括截图等信息的压缩包），完成之后再删除该任务。

## 4.4 主节点restful api设计

分布式框架restful api设计：

restful api使用python flask框架设计。

restful api的设计分为两部分:主节点的restful api和从节点的restful api

主节点restful api包括：

任务提交

节点注册

节点删除

获取节点的信息

提交任务

获取任务状态

列出所有任务

## 4.4 从节点restful api设计

从节点restful api包括：

创建任务

提供此节点任务列表

提供此节点某个任务的状态

提供任务报告

提供任务的pcap包

提供此节点状态

提供此节点的客户机信息

分布式模块Web界面设计：

Web界面采用python django框架

用户可通过web界面提交任务和查看结果：web界面包括三个标签：任务统计，查看历史任务和提交任务；

任务统计界面：

显示历史任务的总个数，样本的个数以及所有任务的状态统计。

历史任务界面：

显示分析过的所有任务的列表，点击对应的id可以查看该任务的详细信息（包括恶意代码运行时的截图，进程的调用关系，代码执行时的流量信息，静态分析结果，恶意代码运行时的行为）

提交任务界面：

用户可以选择提交任务的种类文件还是url，或者apk，选择任务的超时时间以及网络；

## 4.4 数据库总体设计

分布式模块数据库采用postgresql和mongodb

因为每次分析所使用的模块可能不同所以分析结果是不固定的因此使用mongodb存储每个任务的分析结果。

其他数据用postgresql数据库存储，主要有三张表task，node和machine。

Task表负责任务的添加分配以及结果的获取，其结构为：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名称 | 名称 | 属性类别 | 长度 | 空值 | 键类 |
| ID | id | int | 11 | Null | Yes |
| Path | Username | varchar | 255 | Null |  |
| Filename | Password | varchar | 255 | Null |  |
| Package | Email | varchar | 255 | Null |  |
| Timeout | Sex | varchar | 255 | Null |  |
| Priority | Phone | varchar | 255 | Null |  |
| Options | Icon | varchar | 255 | Null |  |
| Machine | nikename | varchar | 255 | Null |  |
| Platform | last\_login\_time | varchar | 255 | Null |  |
| Tags | login\_count | varchar | 255 | Null |  |
| Custom | registered\_time | varchar | 255 | Null |  |
| Owner | realname | varchar | 255 | Null |  |
| Memory | safe\_q\_1 | varchar | 255 | Null |  |
| Clock | safe\_q\_2 | varchar | 255 | Null |  |
| Enforce\_timeout | safe\_q\_3 | varchar | 255 | Null |  |
| Node\_id | safe\_a\_1 | varchar | 255 | Null |  |
| Task\_id | safe\_a\_2 | varchar | 255 | Null |  |
| status | safe\_a\_3 | varchar | 255 | Null |  |
| submitted | submitted | timestamp |  | Null |  |
| Deleted | Deleted | timestamp |  | Null |  |
| Started | Started | timestamp |  | Null |  |
| completed | completed | timestamp |  | Null |  |
| Md5 | Md5 | varchar |  | Null |  |
| category | category | varchar |  | Null |  |

表4-1 task表

Node表负责节点的创建删除；

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名称 | 名称 | 属性类别 | 长度 | 空值 | 键类 |
| ID | id | int | 11 | Null | Yes |
| Name | name | varchar | 255 | Null |  |
| url | url | varchar | 255 | Null |  |
| Mode | mode | varchar | 255 | Null |  |
| Enabled | enabled | varchar | 10 | Null |  |

表4-2 node表

Machine表负责展示每个节点拥有的客户机：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名称 | 名称 | 属性类别 | 长度 | 空值 | 键类 |
| ID | id | int | 11 | Null | Yes |
| Name | name | varchar | 255 | Null |  |
| Platform | Platform | varchar | 255 | Null |  |
| Tags | Tags | varchar | 255 | Null |  |
| Node\_id | Node\_id | Integer | 10 | Null |  |

## 4.5 本章小结

# 第5章 分布式调度系统实现

## 5.1主节点restful api实现

主节点的restful api采用Python flask框架来实现，在distribute/distribute/views/api.py文件用flask通过修饰器对路径和视图函数进行了映射主要有;

任务提交：通过@app.route("/task", methods=["POST"])建立映射，用户向http://url/api/task发送post请求，并提供任务提交时的参数，包括提交的样本文件，以及任务所需要的package，timeout，priority，options，machine，platform，tags，custom，owner，memory，clock，enforce\_timeout等信息，服务器收到该请求后调用建立任务方法，将传递过来的各种参数保存至数据库的task表中，并返回创建任务成功的json字符串。

节点注册：通过@blueprint.route("/node", methods=["POST"])建立映射，用户向http://url/api/node发送post请求，并且提供节点的name和url。服务器收到请求后先查询数据库的node表，判断节点的名称是否之前已经被注册过，如果未被注册过则向从节点发送POST请求，获取该节点拥有的客户机的种类，数量以及其他信息并将返回的集合中的每条信息与该node对应起来，最后将这些信息保存在node表和machine表中并返回创建节点成功的json字符串。

节点删除：通过@app.route("/node/<string:name>",methods=["DELETE"])建立映射,用户使用curl向http://url/api/node发送 -XDELETE 请求。服务器收到请求后先判断该节点名称是否在node表中存在，如果不存在则返回404及节点不存在错误，如果存在，并不会删除该节点，而是该节点的enabled状态设置为false最后返回删除节点成功的json字符串。

获取节点的信息：包括两个映射:通过@app.route("/node")建立映射，获取所有节点的信息，通过@app.route("/node/<string:name>")建立映射,获取指定名称节点的信息。用户向http://url/api/node或http://url/api/node/<string:name>发送get请求。服务器收到请求后先判断name是否为none并从 node表中查询出所有节点的信息，如果name不存在返回节点不存在，如果存在则继续从machine表中查询该节点所拥有的客户机的集合并将得到的信息保存在字典中，最后返回所有节点和其所拥有客户机的json字符串。

获取任务状态：通过@app.route("/task/<int:task\_id>")建立映射，用户向http://url/api/task/<int:task\_id>发送get请求。服务器收到请求之后先判断taskid是否存在于task表中，如果不存在则返回404和任务不存在，如果存在则从数据库查询出该id对应的任务的记录，并封装为json字符串返回。

获取任务状态：通过@app.route("/task")建立映射，用户向http://url/api/task发送get请求并传递offset，finished，status，owner等信息。服务器收到请求之后先判断finished和status是否有值，如果不存在则返回404和不存在，如果存在则从数据库查询并筛选出对应的任务的记录，并封装为json字符串返回。

## 5.2从节点restful api实现

主节点的restful api采用Python flask框架来实现，在utils/api.py文件用flask通过修饰器对路径和视图函数进行了映射主要有;

创建任务：通过@app.route("/tasks/create/file", methods=["POST"])建立映射，主节点向http://url/tasks/create/file发送post请求并传递file，package，timeout，priority, options, machine, platform, tags, custom, owner, tid等信息。服务器收到请求之后先将传过来的恶意代码文件存储到磁盘并返回存储的路径，然后将返回的路径以及其他信息存储至mysql数据库并返回创建任务的taskid。

提供此节点任务列表：

## 5.3web界面实现

@app.route("/tasks/create/file", methods=["POST"])

## 5.4数据库实现

## 5.4 本章小结

# 第6章 分布式恶意代码检测系统的测试

## 6.1系统测试原则

系统测试，英文是SystemTesting。是将已经确认的软件、计算机硬件、外设、网络等其他元素结合在一起，进行信息系统的各种组装测试和确认测试，系统测试是针对整个产品系统进行的测试，目的是验证系统是否满足了需求规格的定义，找出与需求规格不符或与之矛盾的地方，从而提出更加完善的方案。系统测试发现问题之后要经过调试找出错误原因和位置，然后进行改正。是基于系统整体需求说明书的黑盒类测试，应覆盖系统所有联合的部件。对象不仅仅包括需测试的软件，还要包含软件所依赖的硬件、外设甚至包括某些数据、某些支持软件及其接口等。

## 6.2系统功能测试

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 功能模块 | 测试用例 | 测试结果 |
| 1 | Restful api任务提交 | 用例： curl http://localhost:9003/api/task -F file=@calc.exe | 测试结果：成功添加任务 |
| 2 | Restful api节点注册 | 用例：curl http://localhost:9003/api/node -F name=cuckoo1 -F url=http://192.168.16.102:8090/ | 测试结果：成功注册节点 |
| 3 | Restful api节点删除 | 用例： curl -XDELETE http://localhost:9003/api/node/cuckoo1 | 测试结果：成功删除该节点 |
| 4 | Restful api获取节点信息 | 用例：curl http://localhost:9003/api/node/cuckoo1 | 测试结果：成功获取该节点信息 |
| 5 | Restful api获取任务状态 | 用例：curl http://localhost:9003/api/task/1 | 测试结果：成功获取id对应的任务状态 |
| 6 | 从节点Restful api创建任务（文件） | 用例：curl http://192.168.16.102:8090/ tasks/create/file  data = dict(  tid=task["id"],  package=task["package"],  timeout=task["timeout"],  priority=task["priority"],  options=task["options"],  machine=task["machine"],  platform=task["platform"],  tags=task["tags"],  custom=task["custom"],  owner=task["owner"],  memory=task["memory"],  clock=task["clock"],  enforce\_timeout=task["enforce\_timeout"],  ) | 测试结果：成功在此节点创建任务（文件） |
| 7 | 从节点Restful api提供此节点任务列表 | 用例：curl http://192.168.16.102:8090/ tasks/ tasks/ list  params=dict(status=status) | 测试结果：成功获取从节点任务列表 |
| 8 | 从节点Restful api提供此节点某任务物状态 | 用例：curl http://192.168.16.102:8090/tasks/view/ <int:task\_id> | 测试结果：成功获取从节点对应id的任务状态 |
| 9 | 从节点Restful api提供任务报告 | 用例：curl http://192.168.16.102:8090 / tasks/ report/<int:task\_id> | 测试结果：成功获取此节点上对应id的任务报告 |
| 10 | 从节点Restful api提供任务的pcap包 | 用例：curl http://192.168.16.102:8090/ pcap/ get/<int:task\_id> | 测试结果：成功获取此节点上对应id任务的pcap包 |
| 11 | 从节点Restful api提供此节点的状态 | 用例：curl  http://192.168.16.102:8090//cuckoo/status | 测试结果：成功获取此节点的当前状态 |
| 12 | 从节点Restful api提供此节点的客户机信息 | 用例：curl http://192.168.16.102:8090//machines/list | 测试结果：成功获取此节点上所有的客户机信息 |
| 13 | 从节点Restful api创建任务（url） | 用例：curl http://192.168.16.102:8090/ tasks/create/url  data = dict(  tid=task["id"],  package=task["package"],  timeout=task["timeout"],  priority=task["priority"],  options=task["options"],  machine=task["machine"],  platform=task["platform"],  tags=task["tags"],  custom=task["custom"],  owner=task["owner"],  memory=task["memory"],  clock=task["clock"],  enforce\_timeout=task["enforce\_timeout"],  url=task["path"],  ) | 测试结果：成功获取此在节点上创建url任务 |
| 15 | Web界面：任务统计 | 用例：点击链接  http://127.0.0.1:8765/dashboard/ | 测试结果：成功查看任务统计结果 |
| 16 | Web界面：历史任务 | 用例：点击链接  http://127.0.0.1:8765/analysis/ | 测试结果：成功查看所有任务的列表 |
| 17 | Web界面：提交任务（文件） | 用例：点击链接  <http://127.0.0.1:8765/submit/>  选择file标签，选择文件，提交 | 测试结果：成功创建任务（文件） |
| 18 | Web界面：提交任务（url） | 用例：点击链接  <http://127.0.0.1:8765/submit/>  选择url标签，选择文件，提交 | 测试结果：成功创建任务（url） |
| 19 | Web界面：查看任务 | 用例：点击链接  <http://127.0.0.1:8765/analysis/101/> | 测试结果：成功查看id为101的任务的详细报告 |

## 6.3本章小结

# 结论

# 参考文献

# 致谢