|  |  |
| --- | --- |
| 产品名称Product name | 密级Confidentiality level |
| 概要设计文档 | *内部* |
| 产品版本Product version | Total pages 共页 |
|  |

***Http\_gateway概要设计文档***

|  |  |  |  |
| --- | --- | --- | --- |
| Prepared by  拟制 | *杨寅* | Date  日期 | 2016-03-31 |
| Reviewed by  审核 |  | Date  日期 |  |
| Approved by  批准 |  | Date  日期 |  |



ZHICloud Technologies Co., Ltd.

致云科技有限公司

All rights reserved

版权所有 侵权必究

（REP01T01 V0.1 / for internal use only）

（REP01T01 V0.1 / 仅供内部使用）

# 文档介绍

## 1.1目的

明确说明各功能的实现方式，确定软件的全部需求和软件组成模块，确定各模块的功能和用户接口，以此作为编码设计的依据和基础。

## 1.2范围

仅限公司内部开发设计人员阅读。

## 1.3术语

# 总体设计

## 2.1 模块功能

本模块主要完成web管理平台同步异步消息请求到云管理平台CS模块的消息透传（解析与封装）功能，以及通过手动授权方式增强恶意攻击的防御能力。

## 2.2 工作原理

通过HttpServer接收http同步异步业务请求，在请求处理中的对http请求进行头部解析、授权认证及密钥有效性验证处理，将验证通过的每个业务报文通过解析管理模块分配处理事务。对于同步消息，由事务处理对绑定的业务报文映射到平台内部通信的AppMessage消息，发送至CS进行处理，并对CS响应的消息再映射到http净荷中，再通过HttpServer发送到WEB平台。

异步消息处理，http报文接收过程与同步消息一致，在消息映射过程中将会立即返回http确认报文。而实际业务继续通过事务处理发送给CS，待CS响应后事务处理将会把返回的消息映射放入异步响应消息队列中，由HttpClient (ResponseHandler类)直接发送到WEB平台。



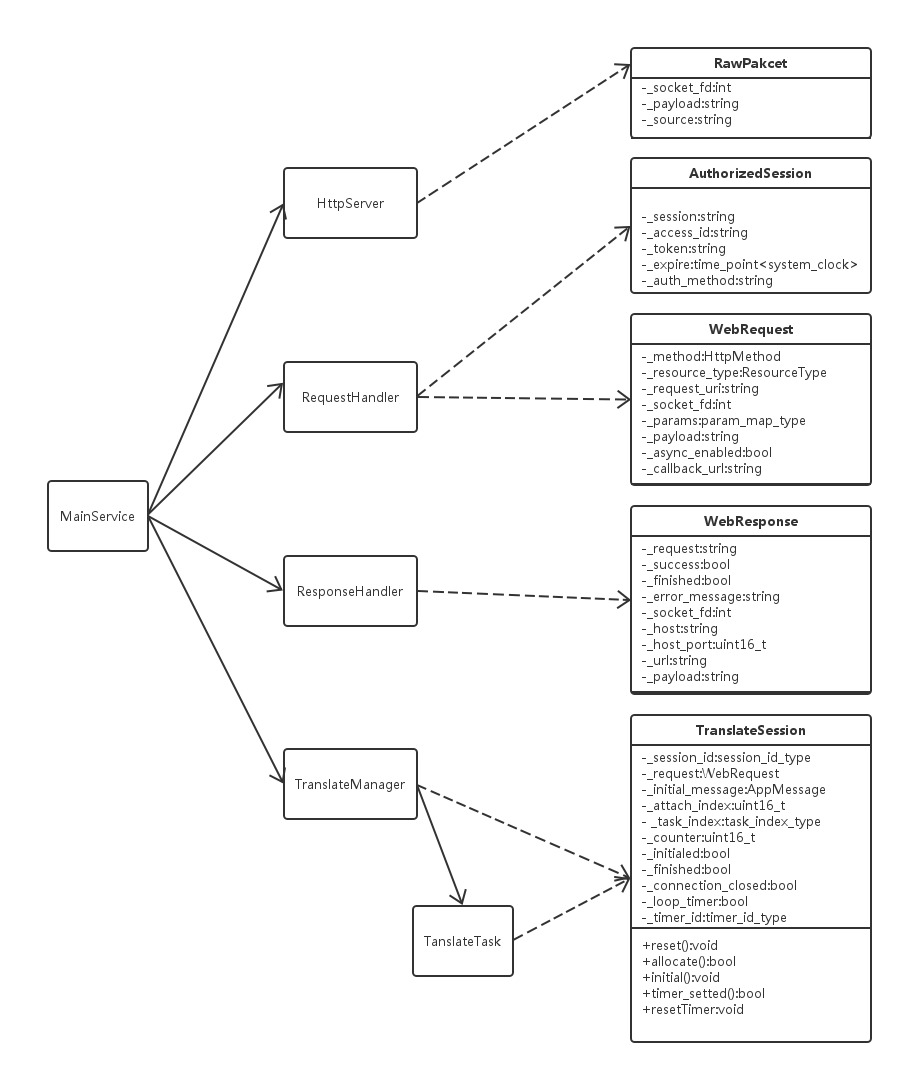
## 2.3 第三方库使用

Boost库。

# 软件架构

## 3.1 模块结构描述

## 3.1.1 类图



## 3.1.2 主服务模块（main\_service.cpp）

## 3.1.2.1 功能概述

本模块主要完成服务节点初始化、主服务启停、平台消息通道建立和通信以及消息处理接口绑定等功能。

## 3.1.2.2 流程图



## 3.1.3 httpServer模块（http\_server.cpp）

## 3.1.3.1 功能概述

本模块主要由TCPServer采用EPOLL边缘触发方式来完成web平台的高并发请求接收与发送工作，同时将获取到的TCP报文净荷放入http请求处理队列中，待请求处理模块进行解析认证。

## 3.1.3.2 流程图



## 3.1.4 请求处理模块（request\_handler.cpp）

## 3.1.4.1 功能概述

本模块为报文合法性、有效性的保障的核心，主要包括报文解析、授权、验证等功能。

工作原理：从TCPServer放入的消息队列中取出需要处理的请求报文，并对http报文的头部进行解析完成http报文格式验证，http报文格式验证成功后，开始对报文的合法性和有效性进行验证。如果不合法或已失效，则会返回未授权或已失效报文到WEB平台，要求WEB平台重新进行授权。对于有效且合法的报文，将会放入到下一级流水线处理队列（解析管理模块处理队列）。

## 3.1.4.2 报文解析处理

根据http协议对报文进行解析，同时验证是否满足本模块后续验证所需要的必要元素如:host、zc-date、zc-session、zc-request、zc-content-hash、zc-signed-header、zc-signature、zc-async以及异步时的回调键值zc-callback等其他待扩展元素。

## 3.1.4.3 授权处理

首先验证授权必需的http头部信息如:主机地址、访问时间、访问id、授权方式、挑战字段以及签名字段。然后根据访问发起时间与当前时间对比验证报文的时效性（报文的生存周期默认15分钟）。再验证报文的授权方式（目前支持一种授权方式：ZC-SHA1）,再获取报文中的访问id，查找本地中是否存在此访问id（access\_id），再对报文头部进行sha1签名验证，签名验证后，根据平台uuid生成模块生成session\_id及token并将授权信息保存至session.ini配置文件同时返回给web平台。

## 3.1.4.4 请求处理

在处理请求的过程中，首先验证http头部信息的完整性，再验证http请求的时效性，验证事务（session\_id）的生存周期以及事务的是否已授权，http净荷校验，http头部校验，签名校验，以及异步情况下获取回调url地址，然后将有效的请求报文映射到WebRequest交由解析管理模块处理。

## 3.1.4.5 请求处理流程图



## 3.1.5 解析管理模块（translate\_manager.cpp）

## 3.1.5.1 功能概述

本模块主要完成session和task资源的分配释放管理、WEB同步异步响应处理接口及平台消息处理接口绑定等功能，每个task单独对应一个请求的处理，根据资源类型、请求方式、task\_id、资源id以及映射消息进行预先注册并以此为依据对请求的消息属性来索引已注册的task来对消息进行处理。

## 3.1.5.2 WEB请求处理流程图



## 3.1.5.3 消息返回处理流程图



## 3.1.6 异步响应模块（response\_handler.cpp）

## 3.1.6.1 功能概述

本模块更加CS返回的异步消息对应的task获取到其对应请求的回调url地址，并对异步响应消息进行封装、发送处理。

## 3.1.6.2 异步响应流程图



## 3.2 功能流程图

参考3.1中的各模块流程图章节

# http请求校验及鉴权

## 4.1http请求校验

http请求校验基本原理就是http客户端和服务端通过随机数+请求参数+URL+时间戳等信息通过hash算法生成一个checksum，发送给接收端，接收端同样采用相同的算法，计算checksum是否一致来确定本次请求过程中数据有没有被篡改，从而保证了请求的合法性。

HttpGateway的请求校验主要分授权校验与业务校验两种方式。

1. 授权校验

就是HttpGateway对WEB平台的授权请求信息的校验，web平台通过Restful授权接口向HttpGateway发送访问授权请求，通过将请求方式+url+授权方式++access\_id+挑战字段+时间戳+access\_key信息进行sha1\_hex摘要算法生成checksum，而且HttpGateway接收到请求以后同样采用sha1\_hex算法生成checksum进行对比是否一致判定请求的完整性与合法性。

1. 业务校验

校验的原理同授权校验，不同在于参与sha1\_hex的字段不一样。HttpGateway与WEB平台两端通过获得授权的session的授权方式+时间戳+授权方式+resource\_path+canonical\_querys+canonical\_headers+signed\_header+payload的哈希值进行sha1\_hex算法生成checksum继续校验。

## 4.2http请求鉴权

HttpGateway采用目前最为主流的鉴权方式利用认证授权来验证数字签名的正确性。鉴权过程如下首先通过管理员注册access\_id及access\_key同时分派给web平台进行授权访问，授权访问请求通过授权校验合法后，同时判断请求是否超时以及access\_id是否已经注册都配置文件，再决定是否对WEB平台客户主机分配session\_id。

WEB平台客户主机获取到session\_id后，便可以开始进行业务访问了，在每次访问中都会通过session\_id作为数字签名形式发送到HttpGateway进行鉴权认证。

业务鉴权的过程中首先判断已授权session\_id签名的请求的生存时间判断其时效性，老化所有已经逾期的临时session\_id签名，判断session\_id是否为系统已分配，再重置缓存中当前session\_id的生存时间。

## 4.3鉴权流程图



# 内部实现

## 5.1 数据结构定义

## 5.1.1 事务授权类（AuthorizedSession）

|  |  |  |
| --- | --- | --- |
| 事务授权类 |  |  |
| 临时事务 | \_session | 为每个授权访问链接分配的临时事务id，同时将会保存在session.ini中生存时间48小时。  /var/zhicloud/config/http\_gateway/data/session.ini |
| 访问id | \_access\_id | 由管理员统一分配给用户并配置到access.ini文件中。  /var/zhicloud/config/http\_gateway/data/access.ini |
| 认证密钥 | \_token | 用于客户端与服务端双向认证的密钥。 |
| 授权方式 | \_auth\_method | 目前只支持ZC-SHA1授权方式。 |

## 5.1.2 原始包类（RawPacket）

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 报文收发套接字 | \_socket\_fd | 原始报文请求及响应收发套接字 |
| tcp报文净荷 | \_payload | http请求数据 |
| 源地址 | \_source | http请求源地址 |

## 5.1.3 WEB请求类（WebRequest）

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 请求方式 | \_method | 支持get/post/put/delete请求方式 |
| 请求资源类型 | \_resource\_type | 支持资源类型参考resource\_type.hpp |
| 请求资源uri | \_request\_uri | 请求平台资源的uri |
| 套接字 | \_socket\_fd | 等同于RawPacket中的套接字 |
| 净荷 | \_payload | 资源信息 |
| 异步使能 | \_async\_enable | 异步使能标志 |
| 回调url | \_callback\_url | 异步处理回调url |

## 5.1.4 事务解析类（TranslateSession）

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 内部事务处理id | \_session\_id | 用于绑定web请求与平台消息处理 |
| web请求 | \_request | Web请求消息 |
| 平台交互消息 | \_initial\_message | 平台内部交互消息 |
| 任务处理索引 | \_task\_index | 用于索引消息处理 |

## 5.2主要功能类定义

## 5.2.1 httpServer类

本类主要完成对tcp报文收发、http报文提取等工作，主要涉及成员函数及数据成员如下：

class HttpServer:public zhicloud::service::Runable

{

public:

typedef boost::signals2::signal< bool (RawPacket&) > packet\_event;

typedef packet\_event::slot\_type packet\_handler;

HttpServer(const string& listen\_ip, const uint16\_t& listen\_port, const uint16\_t& receive\_thread = 2, const uint16\_t& send\_thread = 2);

virtual ~HttpServer();

bool sendPacket(int socket\_fd, const string& content);

void bindHandler(const packet\_handler& handler);

protected:

virtual bool onStart() override;

virtual void onStopping() override;

private:

void listenProcess();

void acceptProcess();

void receiveProcess();

void sendProcess();

void monitorProcess();

void notifyProcess();

void disableMonitor(int socket\_fd);

private:

logger\_type logger;

string \_listen\_ip;

uint16\_t \_listen\_port;

int \_listen\_socket;

int \_epoll\_fd;

uint16\_t \_receive\_thread\_count;

vector< thread > \_receive\_thread;

uint16\_t \_send\_thread\_count;

vector< thread > \_send\_thread;

thread \_listen\_thread;

thread \_accept\_thread;

thread \_monitor\_thread;

thread \_notify\_thread;

const static size\_t queue\_size = 1024;

PassiveQueue< RawPacket, queue\_size > \_send\_queue;

PassiveQueue< RawPacket, queue\_size > \_receive\_queue;

PassiveQueue< int, queue\_size > \_connection\_queue;

PassiveQueue< int, queue\_size > \_receivable\_queue;

packet\_event \_onPacketReceived;

};

## 5.2.2 请求处理类

本模块主要完成http消息解析，授权认证信息的恢复与保存动作，同时将解析出的消息映射到平台内部资源ID，然后将请求压入队列由解析管理 模块进行下一步到AppMessage解析处理。

主要涉及成员函数及数据成员如下：

class RequestHandler: public zhicloud::service::Runable

{

public:

typedef boost::signals2::signal< bool (int, const string&) > packet\_event;

typedef boost::signals2::signal< bool (WebRequest&, uint32\_t&) > web\_request\_event;

typedef packet\_event::slot\_type packet\_handler;

typedef web\_request\_event::slot\_type request\_handler;

RequestHandler(const string& data\_root, const uint16\_t& worker\_count = 2);

virtual ~RequestHandler();

void initializer();

bool processRawPacket(RawPacket& packet);

void bindPacketHandler(const packet\_handler& handler);

void bindWebRequestHandler(const request\_handler& handler);

bool WebResponseHandler(const WebResponse& response);

protected:

virtual bool onStart() override;

virtual void onStopping() override;

virtual void onWaitFinish() override{

}

virtual void onStopped() override{

}

private:

void workProcess();

bool parsePacket(const string& content, HttpMethod& method, ResourceType& resource\_type, string& method\_name, string& resource\_id, string& resource\_path,

map< string, string>& query\_strings, unordered\_map< string, string>& headers, string& payload);

void handleAuth(int socket\_fd, const unordered\_map< string, string>& headers);

void handleRequest(int socket\_fd, const string& request\_ip,

const HttpMethod& method, const ResourceType& resource\_type, const string& method\_name, const string& resource\_id, const string& resource\_path,

const map< string, string>& query\_strings, const unordered\_map< string, string>& headers, const string& payload);

void replyError(int socket\_fd, const string& error = "", bool is\_auth\_fail = false);

void replyAuthSuccess(int socket\_fd, const string& session, const string& token, const uint32\_t& expire);

void parseMethodName(const string& name, HttpMethod& method);

void parseRequestURL(const string& url, ResourceType& resource\_type, string& resource\_id, string& resource\_path, map< string, string>& query\_strings);

void addResourceMap();

private:

typedef lock\_guard< recursive\_mutex > lock\_type;

logger\_type logger;

const string \_data\_root;

uint16\_t \_worker\_count;

const string \_access\_file;

const string \_session\_file;

vector< thread > \_worker\_thread;

const static size\_t queue\_size = 1024;

const static uint32\_t max\_request\_delay\_minutes = 15;

const static uint32\_t default\_session\_expire = 48\*3600;

const static string default\_auth\_method;

const static string default\_interface\_url;

PassiveQueue< RawPacket, queue\_size > \_packet\_queue;

recursive\_mutex \_authorization\_mutex;

//key = access id, value = key

unordered\_map< string, string > \_authorization\_map;

//key = access id, value = session id

unordered\_map< string, string > \_authorized\_id\_map;

recursive\_mutex \_session\_mutex;

//key = session id, value = authorized session

unordered\_map< string, AuthorizedSession > \_authorized\_session\_map;

//session id, expire

vector<pair<string, time\_point<system\_clock>>> \_authorized\_session\_aging\_pairs;

Generator \_generator;

packet\_event invoker\_send\_packet;

web\_request\_event invoker\_send\_request;

/\* BEGIN: Added by wangli, 2015/12/3 \*/

unordered\_map< string, ResourceType > \_resource\_type\_map;

/\* END: Added by wangli, 2015/12/3 PN: \*/

};

## 5.2.3 解析管理类

主要涉及成员函数及数据成员如下：

class TranslateManager: public zhicloud::service::Runable, TaskProxy

{

public:

typedef TranslateSession::session\_id\_type session\_id\_type;

typedef TranslateSession::task\_index\_type task\_index\_type;

typedef uint16\_t queue\_index\_type;

typedef boost::signals2::signal< bool (AppMessage&) > message\_event\_1;

typedef boost::signals2::signal< bool (WebResponse&) > web\_response\_event\_1;

typedef message\_event\_1::slot\_type message\_handler;

typedef web\_response\_event\_1::slot\_type web\_response\_handler;

TranslateManager(const session\_id\_type& session\_count, const queue\_index\_type& queue\_count);

virtual ~TranslateManager();

void initializer();

void bindMessageHandler(const message\_handler& handler);

void bindSyncResponseHandler(const web\_response\_handler& handler);

void bindAsyncResponseHandler(const web\_response\_handler& handler);

bool addTask(const ResourceType& resource\_type, const HttpMethod& method, TranslateTask&& task,

bool with\_resource\_id = true, const AppMessage::key\_type& id\_key = ParamEnum::domain, const ParamType& id\_type = ParamType::INT);

bool startTransaction(WebRequest& request, session\_id\_type& session\_id);

bool processMessage(const session\_id\_type& session\_id, AppMessage& msg);

bool containsTransaction(const session\_id\_type& session\_id);

bool sendToControlServer(AppMessage& msg);

bool sendSyncResponse(WebResponse& response);

bool sendAsyncMessage(WebResponse& msg);

timer\_id\_type setTimer(const uint32\_t& interval\_in\_second, const session\_id\_type& session\_id) ;

timer\_id\_type setLoopTimer(const uint32\_t& interval\_in\_second, const session\_id\_type& session\_id);

bool clearTimer(const timer\_id\_type& timer);

protected:

virtual bool onStart() override;

virtual void onStopping() override;

virtual void onWaitFinish() override{

}

virtual void onStopped() override{

}

private:

/\* BEGIN: Added by wangli, 2015/12/4 \*/

void initTasks();

/\* END: Added by wangli, 2015/12/4 PN: \*/

void onMessageReceived(AppMessage& msg, uint64\_t& pos, bool end\_of\_batch);

task\_index\_type getTask(const ResourceType& resource\_type, const HttpMethod& method, bool with\_resource\_id);

void deallocTransaction(const session\_id\_type& session\_id);

bool putMessage(AppMessage& msg, const queue\_index\_type& index);

bool isValid(const session\_id\_type& session\_id){

if((session\_id < \_min\_session)||(session\_id > \_max\_session))

return false;

return true;

}

void onTimeoutEvent(list< AppMessage >& event\_list);

private:

logger\_type logger;

session\_id\_type \_session\_count;

atomic< session\_id\_type > \_last\_seed;

session\_id\_type \_min\_session;

session\_id\_type \_max\_session;

uint16\_t \_queue\_count;

unordered\_map< session\_id\_type, TranslateSession > \_session\_map;

unordered\_map< session\_id\_type, CopyableAtomic< bool > > \_allocate\_map;

const static size\_t queue\_size = 1024;

vector< ActiveQueue < AppMessage, queue\_size > > \_queue\_vector;

std::atomic< queue\_index\_type > \_index\_seed;

//task map

vector< TranslateTask > \_task\_vector;

//ResourceType -> HttpMethod -> with resource id

map< ResourceType, map < HttpMethod , unordered\_map < bool, task\_index\_type > > > \_task\_index\_map;

TimerService \_timer\_service;

message\_event\_1 invoke\_send\_to\_control\_server;

web\_response\_event\_1 invoke\_send\_sync\_response;

web\_response\_event\_1 invoke\_send\_async\_message;

};

# 接口定义

请参见文档《Http\_gateway API.xmind》

# 参数配置

## 7.1 参数配置

访问配置文件部署路径:/var/zhicloud/config/http\_gateway/data/access.ini

文件实例:

[DEFAULT]

access\_id0 = zhicloud0 #access\_id由管理员自定义

access\_id1 = zhicloud1

access\_id2 = zhicloud2

access\_id3 = zhicloud3

临时事务文件部署路径:/var/zhicloud/config/http\_gateway/data/session.ini

此文件有RequestHandle类维护。

文件实例:

[DEFAULT]

session\_count=1

[SESSION\_0]

session=4dde5cc72ba2467fb7cd04d87fabc76d

access\_id=access\_id2

token=8922dd1e5e0e431fa7d8e374e765285d

expire=1453536351

auth\_method=ZC-SHA1

## 7.2 启停方式

应用程序部署路径:/home/zhicloud/ http\_gateway

启动方式: ./http\_gateway start

停止方式: ./http\_gateway\_stop

重启方式: ./http\_gateway restart