**TrustZone 机密计算资源池化 二期**

**系统部署说明**

**2022.8.19**

修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 修订版本 | 修改描述 | 作者 |
| 2022-8-19 | V1.0 | 部署步骤 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[TrustZone 机密计算资源池化 二期 1](#_Toc28261)

[系统部署说明 1](#_Toc2029)

[1 介绍 4](#_Toc26109)

[2 编译部分 5](#_Toc5569)

[2.1 依赖、工具安装 5](#_Toc2763)

[2.2 编译、安装gpworker（仅服务端编译） 5](#_Toc14359)

[2.3 编译、安装libdbusc\_gpw（仅服务端编译） 5](#_Toc32424)

[2.4 下载、patch、编译、安装c-spiffe 6](#_Toc9351)

[2.5 编译、安装libspiffejwt 6](#_Toc22371)

[2.6 编译、安装dbusjwt 7](#_Toc24021)

[2.7 编译、安装gpproxy、libteecc.so 7](#_Toc2075)

[2.8 下载、编译spire 8](#_Toc16488)

[3 gpproxy、libteecc.so配置 8](#_Toc25174)

[3.1 gpproxy配置 8](#_Toc2941)

[3.2 Libteecc.so配置 9](#_Toc21733)

[4 证书配置 10](#_Toc5379)

[4.1 TLS证书配置 10](#_Toc32472)

[4.2 服务端 10](#_Toc1343)

[4.3 客户端 10](#_Toc7942)

[5 运行部分 11](#_Toc8363)

[5.1 服务端 11](#_Toc27903)

[5.2 客户端 12](#_Toc6571)

[6 Docker配置 13](#_Toc3679)

[6.1 Docker安装与配置 13](#_Toc28941)

# 介绍

TrustZone是ARM架构下为解决可能遇到的软硬件安全问题提出的一种硬件解决方案。

本文档主要介绍了：

* 部署TrustZone 机密计算资源池化系统。
* 在已部署好的系统上的运行步骤。
* Docker的配置与部署
* 配置文件说明

# 编译部分

## 依赖、工具安装

安装vim、dbus-devel、wget、camke、patch、openssl、git、gcc、make、yaml-cpp-devel、golang、cjose-devel、subunit-devel、curl-devel、check-devel、uriparser-devel、jansson-devel

yum install dbus

若未找到则前往https://repo.openeuler.org/openEuler-21.09/OS/x86\_64/Packages/ 下载rpm包

//剩余工具进行同样操作。

安装并发工具parallel

wget <http://ftp.gnu.org/gnu/parallel/parallel-latest.tar,bz2>

tar xjf parallel-latest.tar.bz2

cd parallel-yyyymmdd

./configure

make -j 4

make install

parallel --citation

将链接库路径指定为/usr/local/lib64

vim /etc/profile //向其中添加一行 export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/usr/local/lib64

source /etc/profile //应用更改

## 编译、安装gpworker（仅服务端编译）

下载itrustee\_sdk

git clone   https://gitee.com/openeuler/itrustee\_sdk.git

将gpworker拷贝到itrustee\_sdk对应目录中

cp teeproxy/gpworker itrustee\_sdk/test/CA

进入gpworker的build目录中，执行cmake命令

cd build && make

## 编译、安装libdbusc\_gpw（仅服务端编译）

该操作生成gpproxy调用gpworker所需的库libdbusc\_gpw.so

进入build目录中，执行cmake.sh

cd build && bash cmake.sh

编译并安装libdbusc\_gpw.so

make -j && make install

----结束

## 下载、patch、编译、安装c-spiffe

该操作生成与spire-agent通信所需的基础库libclient.so

下载c-spiffe

git clone https://github.com/HewlettPackard/c-spiffe.git

将c-spiffe\_patch拷贝到c-spiffe同级目录，进入c-spiffe执行patch命令

cd c-spiffe && patch -p1 < ../c-spiffe\_patch //若提示Reversed (or previously applied) patch detected! 则按enter跳过

进入cmake目录执行cmake.sh

cd cmake && bash cmake.sh

注：此处与gpproxy、libteecc的编译相同，都可使用本地grpc，操作方法与可能遇见的问题见下方gpproxy\_libteecc的编译过程。

注：可使用本地安装grpc，在执行cmake.sh之前将common.cmake中的set (GRPC\_FETCHCONTENT 1) 1 改为 0，具体本地安装grpc见https://www.grpc.io/，若使用fetchcontent方法则可能需要网络代理进行下载。

cd cmake && bash cmake.sh

//////////////////////////////////

如果遇到下列问题，则表示第三方库没有安装好：

CMake Warning at cmake/abseil-cpp.cmake:30 (message):

gRPC\_ABSL\_PROVIDER is “module” but ABSL\_ROOT\_DIR is wrong:

cmake/\_deps/grpc-src/third\_party/abseil-cpp

Call Stack (most recent call first):

CMakeLists.txt:188 (include)

//////////////////////////////////////////

1. 进入cmake/\_deps+grpc-src/third\_patry
2. 执行 git submodule update --init
3. 在打印信息中找到报错文件的git地址（以abseil-cpp为例）

2、在build/\_deps/grpc-src/third\_party/中删除对应的文件夹 $ rm -rf abseil-cpp

3、git clone <https://github.com/abseil/abseil-cpp.git>

剩余报错也进行相应处理

编译并安装c-spiffe

make && make install

----结束

## 编译、安装libspiffejwt

该操作生成通过libclient.so获取、验证jwt的库libspiffejwt.so

进入build目录中，执行cmake.sh

cd build && bash cmake.sh

编译并安装libspiffejwt.so

make && make install

----结束

## 编译、安装dbusjwt

该操作生成dbuss\_fetchjwt与dbuss\_validatejwt两个可执行文件作为中间媒介通过dbus与gpproxy进行通信来获取、验证jwt，同时生成提供给gpproxy、libteec与这两个可执行文件通信的库dbusc\_jwt.so。

进入build目录中，执行cmake.sh

cd build && bash cmake.sh

编译并安装dbusjwt

make && make install

----结束

## 编译、安装gpproxy、libteecc.so

分别进入gpproxy、libteecc的build目录中，执行cmake.sh

注：此处与c-spiffe的编译相同，都可使用本地grpc，操作方法与可能遇见的问题见上方c-spiffe的编译过程。

cd build && bash cmake.sh

编译并安装gpproxy与libteecd.so

make && make install

----结束

## 下载、编译spire

下载spire

git clone --single-branch --branch v1.3.3 https://github.com/spiffe/spire.git

进入spire中

cd spire

编译spire-server与spire-agent

make -j

----结束

# gpproxy、libteecc.so配置

## gpproxy配置

gpproxy通过gpproxy\_config.yaml控制程序参数，安装时会将文件自动上传至/${HOME}/.gpp/gpproxy\_config.yaml，以下以root用户为例，通过更改该文件中参数进行对gpproxy的配置。

### 监听端口配置

参数：

GPPROXY\_ADDRESS: "[::]:50051"

### 证书路径配置

证书生成程序在安装时会自动上传至/${HOME}/.gpp/cert目录中，将证书上传至此路径，更改以下参数填入证书名：

NAME\_SERVER\_KEY: "server\_key.pem"

NAME\_SERVER\_CERT: "server\_self\_signed\_crt.pem"

NAME\_CLIENTCA\_CERT: "ca\_crt.pem"

### TLS配置

更改以下参数：

参数说明：

# 0, no tls

# 1, only server certificate

# 2, server and client certificate

# 注： libteec\_config.yaml配置文件中该参数应设置应与gpproxy中配置相同

GRPC\_TLS: 1

### 强制Jwt验证配置

更改以下参数：

参数说明：

# 0, do not validate jwt

# 1, foreced to validate jwt

FORCE\_VALIDATE\_JWT: 0

### 线程数、gpworker数设置

更改以下参数：

MAX\_NUM\_THREAD: 128 #线程数 范围为（1-128）

MAX\_NUM\_WORKER: 128 #gpworker数 范围为（1-128）

### Session、Context Timeout时间设置

更改以下参数：

TIMEDOUT\_SESSION: 60 #单位 秒（s）

TIMEDOUT\_CONTEXT: 90 #单位 秒（s）

## Libteecc.so配置

Libteecc.so通过teecc\_config.yaml控制程序参数，安装时会将文件自动上传至/${HOME}/.teecc/teecc\_config.yaml，以下以root用户为例，通过更改该文件中参数进行对libteecc.so的配置。

### 目的端口配置

更改以下参数：

GPPROXY\_ADDRESS: "gpserver.org:50051"

### 证书路径配置

证书生成程序在安装时会自动上传至/${HOME}/.teecc/cert目录中，将证书上传至此路径，更改以下参数填入证书名：

NAME\_SERVERCA\_CERT: "ca\_crt.pem"

NAME\_CLIENT\_KEY: "client\_key.pem"

NAME\_CLIENT\_CERT: "client\_self\_signed\_crt.pem"

### TLS配置

更改以下参数：

参数说明：

# 0, no tls

# 1, only server certificate

# 2, server and client certificate

# 注： libteec\_config.yaml配置文件中该参数应设置应与gpproxy中配置相同

GRPC\_TLS: 1

# 证书配置

## TLS证书配置

## 服务端

### gpproxy\_libteecc安装时会将证书生成脚本自动上传到//.gpp/cert目录下，以下以root用户为例，需进行如下操作，生成证书。

生成ca根证书与ca私钥

bash gen\_ca\_keycrt.sh //该操作需设置ca证书密码

生成服务端证书与私钥

bash gen\_server\_keycrt.sh //该操作需输入ca证书密码并设置服务端证书密码

等待到客户端发送来的私钥与生成请求后，生成客户端证书

bash gen\_client\_crt.sh //该操作需输入ca证书密码

## 客户端

### gpproxy\_libteecc安装时会将证书生成脚本自动上传到/${HOME}/.teecc/cert目录下，以下以root用户为例，需进行如下操作，生成证书。

生成客户端私钥与证书生成请求

bash gen\_client\_keycsr.sh //该操作需设置客户端证书密码

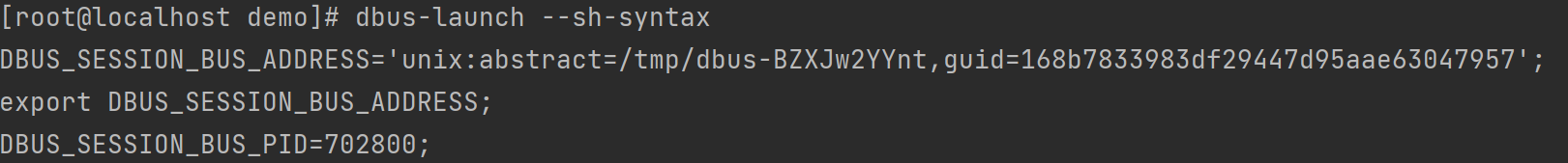
服务端根据证书生成请求，生成客户端证书

将密钥与证书生成请求发送给服务端，服务端生成证书传给客户端

# 运行部分

## 服务端

若运行时提示Connection Error (Unable to autolaunch a dbus-daemon without a $DISPLAY for X11) 将出现该提示的**终端**DBUS总线设置为**相同地址**

1. dbus-launch --sh-syntax 
2. 在各端口执行 export DBUS\_SESSION\_BUS\_ADDRESS= unix:abstract=/tmp/dbus-BZXJw2YYnt,guid=168b7833983df29447d95aae63047957
3. 查看dbus地址是否相同 $ env

设置bindv6only参数

sysctl net.ipv6.bindv6only=0

将IP地址映射为域名

vim /etc/hosts

在任意一行加入（以172.28.6.73为例）： 172.28.6.73 spireserver.org

更改gpproxy配置文件（参照配置说明）

vim /root/.gpp/gpproxy\_config.yaml

运行gpproxy

gpproxy //检查防火墙配置，防火墙打开gpproxy端口，默认为50051。同时也需要关闭代理，若开启TLS则需输入服务端证书密钥

运行gpworker

cd itrustee\_sdk/test/CA/gpworker/build/

bash r.sh

运行dbuss\_validatejwt

dbuss\_validatejwt

更改spire-agent配置文件

vim conf/agent/agent.conf //更改其中的server\_address = "spireserver.org" ，默认端口号为8081

删除spire-server中原有的数据文件

//若未删除则可能导致无法为spire-agent创建entry

rm -f .data/agent\_svid.der

rm -f .data/agent-data.json

rm -f .data/keys.json

运行spire-server

bin/spire-server run -config conf/server/server.conf &

注：spire中的运行所需命令已写成脚本，存放在script文件之中

命令对应脚本: runserver.sh

生成服务端spire-agent运行所需token

bin/spire-server token generate -spiffeID spiffe://example.org/myagent

命令对应脚本: gentoken\_agent.sh

为服务端spire-agent工作负载创建注册策略

bin/spire-server entry create -parentID spiffe://example.org/myagent \

-spiffeID spiffe://example.org/myservice -selector unix:uid:$(id -u)

命令对应脚本: createentry\_workload.sh 其中unix:uid:改为对应的用户id

//可执行

运行服务端spire-agent

将服务端spire-server生成的token填入下行命令

bin/spire-agent run -config conf/agent/agent.conf -joinToken <token\_string> &

命令对应脚本: runagent.sh

若无法连接上spire-server，检查防火墙配置，防火墙打开spire-server端口，默认为8081。

生成客户端spire-agent运行所需token

bin/spire-server token generate -spiffeID spiffe://example.org/myagent

命令对应脚本: gentoken\_agent.sh

为客户端spire-agent工作负载创建注册策略

bin/spire-server entry create -parentID spiffe://example.org/myagent \

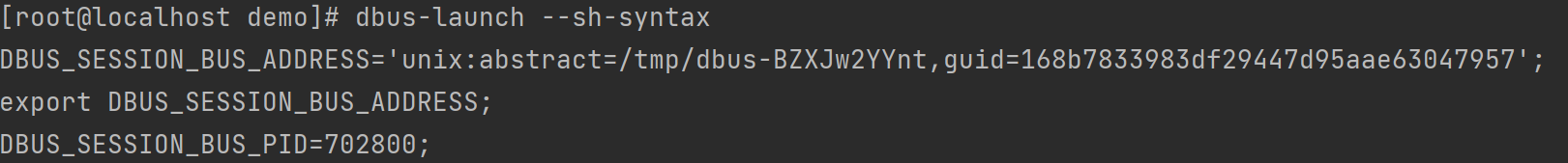
-spiffeID spiffe://example.org/myservice -selector unix:uid:$(id -u)

注：在客户端时，将$(id-u)改为实际用户id

命令对应脚本: createentry\_workload.sh

## 客户端

若运行时提示Connection Error (Unable to autolaunch a dbus-daemon without a $DISPLAY for X11) 将出现该提示的**终端**DBUS总线设置为**相同地址**

1. dbus-launch --sh-syntax 
2. 在各端口执行 export DBUS\_SESSION\_BUS\_ADDRESS= unix:abstract=/tmp/dbus-BZXJw2YYnt,guid=168b7833983df29447d95aae63047957
3. 查看dbus地址是否相同 $ env

将服务器IP地址映射为域名

vim /etc/hosts

在任意一行加入（以172.28.6.73为例）： 172.28.6.73 gpserver.org

更改libteecc.so配置文件

vim /root/.teecc/teecc\_config.yaml

运行dbuss\_fetchjwt

dbuss\_fetchjwt

更改spire-agent配置文件

vim conf/agent/agent.conf //更改其中的server\_address = "spireserver.org" 默认端口号为8081

运行spire-agent

将服务端spire-server生成的token填入下行命令

bin/spire-agent run -config conf/agent/agent.conf -joinToken <token\_string> &

命令对应脚本: runagent.sh

同步服务端与客户端时间

服务端与客户端若时间不同步则可能导致jwt获取与验证失败

编译、运行testcase

进入文件demo/testcase/testcase1/中，在main()、testcase10()中将指向TA文件的路径更改为本机TA文件路径

进入demo/testcase/testcase1/build目录中，执行cmake.sh

$ cd demo/testcase/testcase1/build && bash cmake.sh

编译testcase

$ make

运行testcase //若TLS模式为2 则需输入设置的客户端证书密钥

$ ./testcase 1 （可更改testcase后的数字，更换测试内容）

# Docker配置

## Docker安装与配置

安装docker

yum install -y docker

运行docker

systemctl start docker

下载openeuler镜像

docker pull openeuler/openeuler

运行镜像

docker run --rm -d --privileged -v /var/run:/var/run -v /etc/localtime:/etc/localtime --name openeuler openeuler/openeuler sleep infinity //挂载/var/run是为了挂载docker.sock使spire-agent可以与spire-server通信，挂载/localtime使时间统一。

安装vim、dbus、wget、camke、patch、openssl、git、gcc、g++、make、yaml-cpp- devel、golang、cjose-devel、subunit-devel、curl-devel、check-devel、uriparser-devel、 jansson-devel

yum install dbus

若未找到则前往https://repo.openeuler.org/openEuler-21.09/OS/x86\_64/Packages/ 下载rpm包

//剩余工具进行同样操作。

将链接库路径指定为/usr/local/lib64

vim /etc/profile //向其中添加一行 export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/usr/local/lib64

source /etc/profile //应用更改

保存镜像

docker commit Container\_ID name（自己设置镜像名称）