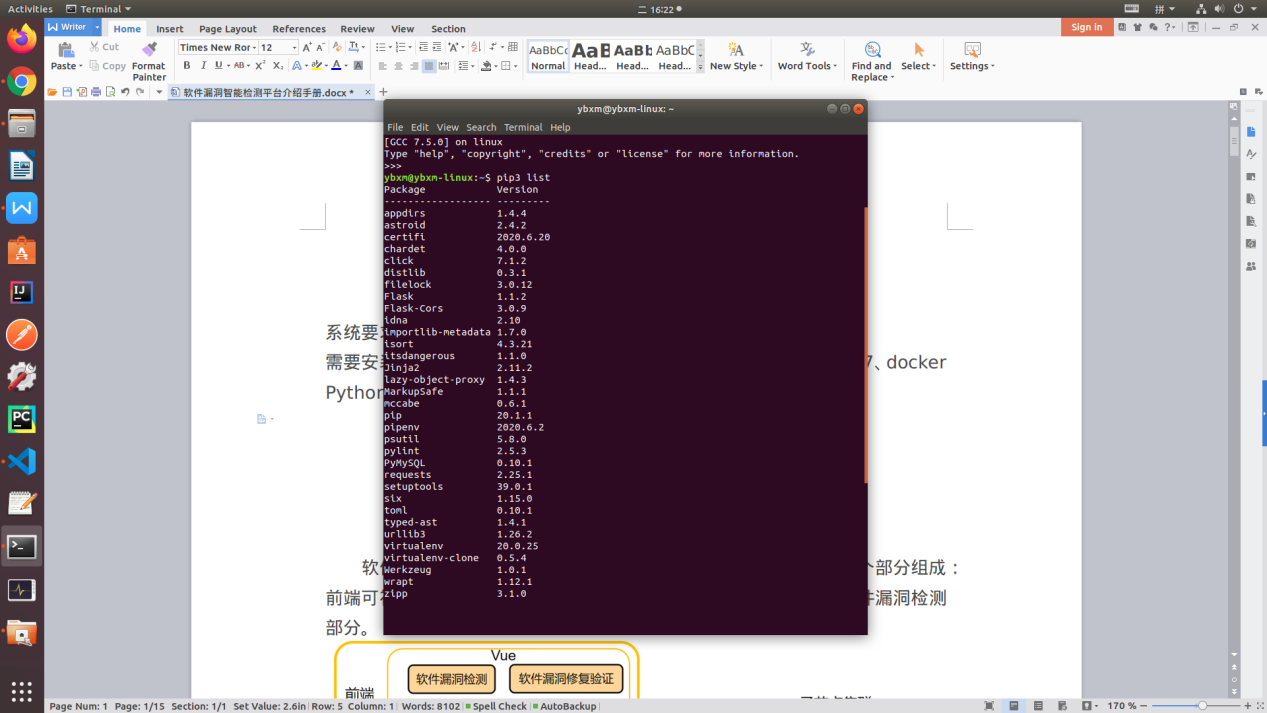
## 一、安装部署

1.系统要求：ubuntu18.04

2.需要安装的软件及其版本：nodejs:v8.10.0、npm:5.6.0、Python3.7、docker。

　　Python的所需的依赖包：



　　子节点上，需要安装afl、aflfast、fairfuzz、afl++(2.59c)、honggfuzz、qsym的docker镜像、aflgo(该框架对应的功能尚未实现)。

1. 启动命令:

　　前端：npm run dev

　　backend1: python3 app.py

　　backend2: python3 run\_backend2.py

子节点：python3 run.py

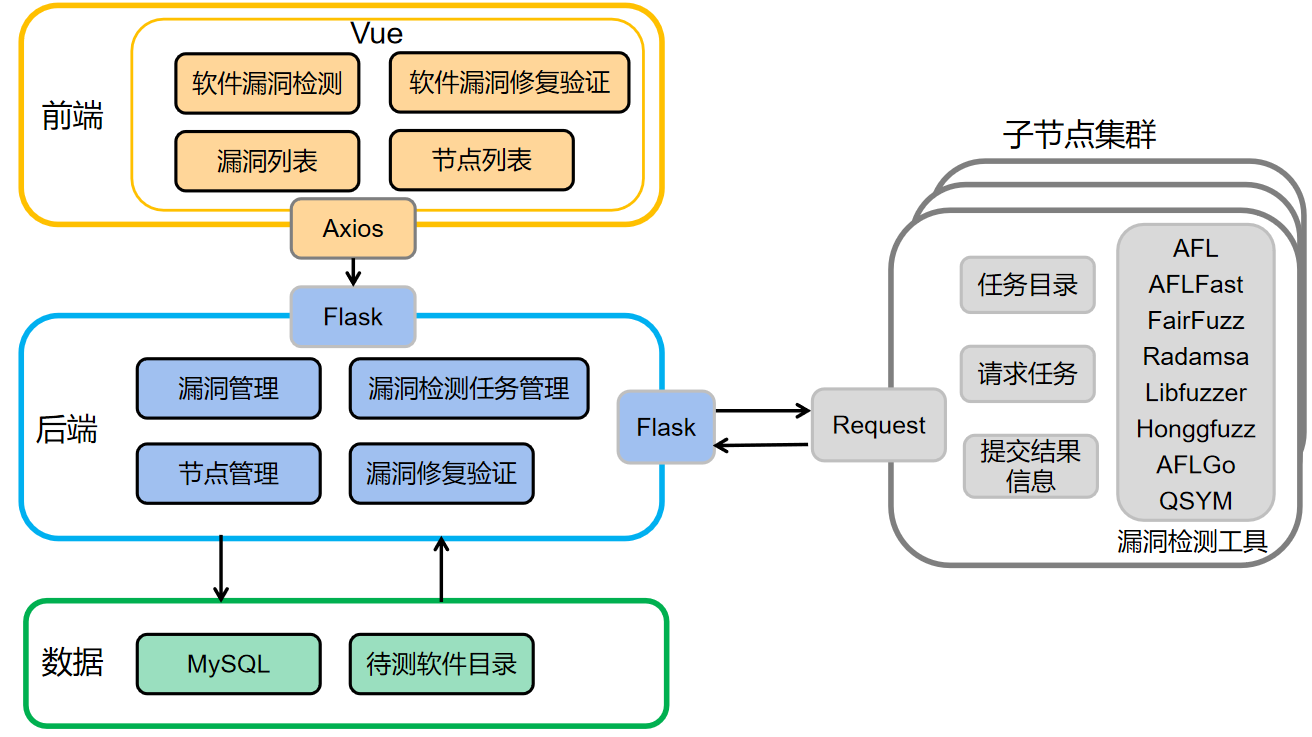
1. 系统目录

MyServer目录下主要分为三个子目录：frontend、backend1、backend2。其中frontend目录下存放的是前端代码；backend1目录下的代码用于处理来自前端的消息；backend2目录下的代码用于将任务分配给各个子节点，同时接收来自各子节点的消息。

　　MyClientFuzzers目录下存放的是子节点的代码，需要部署到子节点服务器上才可运行。

## 二、平台整体架构

软件漏洞智能检测平台的整体结构如下图所示，本平台主要由３个部分组成：前端可视化操作界面部分、后端任务管理和任务调度部分、子节点软件漏洞检测部分。



软件漏洞智能检测平台架构图

（1）前端可视化操作界面部分。前端的可视化操作界面部分主要包括4个界面：软件漏洞检测界面、软件漏洞修复验证界面、漏洞列表界面和节点列表界面。在软件漏洞检测界面，可以选择多种软件漏洞检测方式：普通模糊测试、集成模糊测试、混合模糊测试、目标点模糊测试。此外，可以设定测试的时间和测试节点的数量，通过测试任务的优先级来自主分配测试资源。在软件漏洞修复验证界面，需要选择已发现的漏洞并上传修复好的软件，由平台来检测漏洞是否已经修复。在漏洞列表界面，可以查看到所有经过软件漏洞检测后发现的漏洞信息及其对应的错误栈，通过错误栈可以便于开发人员进行软件的漏洞修复。在节点列表界面，可以看到平台拥有的所有子节点信息，包括子节点的名称、ip地址、是否空闲等，然后测试人员可以依据该信息合理规划和使用测试资源。

前端主要使用Vue框架，并结合Element库的组件展示（包括数据列表、按钮、图标、展开图等），由Axios库实现异步请求数据，通过Webpack实现模块化开发。

（2）后端任务管理和任务调度部分。后端包括图5.1中的服务端部分和数据部分两块。服务端主要包括４个功能：Fuzz任务管理、漏洞修复验证、漏洞管理、节点管理。服务端负责处理来自前端的任务需求，然后通过任务调度机制调度各个子节点去执行任务，当子节点完成任务后，再由服务端将结果信息反馈至前端进行展示。数据部分由MySQL数据库和待测软件目录两部分构成，MySQL数据库中存储相关的任务信息，主要包括4张数据表：Jobs、Reproduce、Crashes、Nodes，分别代表软件漏洞检测任务表、漏洞修复验证任务表、软件漏洞信息表和子节点信息表。待测软件目录中则会存放需要执行软件漏洞检测任务的待测开源软件，以及已经修复了漏洞等待验证漏洞修复的软件。

后端部分的代码由python编写实现，采用python的flask框架实现web服务，既可以接收来自前端的任务信息，也可以实现与各个子节点的信息交互。此外，通过python的pyMySQL库来实现与MySQL数据库的连接，使得任务信息和测试结果可以存储到MySQL数据中进行集中管理。

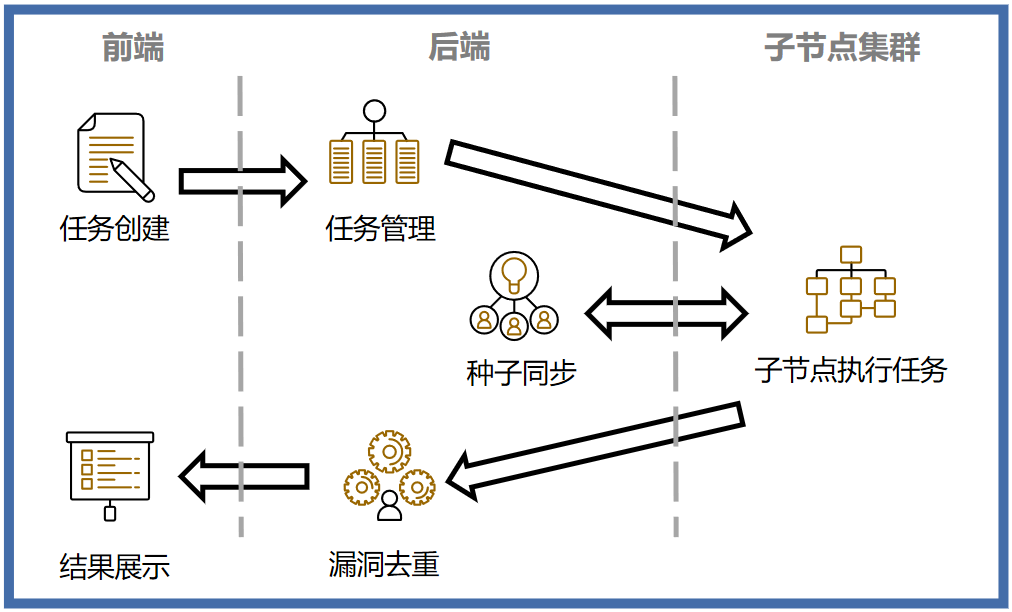
（3）子节点软件漏洞检测部分。子节点是指具有软件漏洞检测能力的主机，本平台可以管理多个子节点，实现分布式并行化的软件漏洞检测，以提升软件漏洞的检测效率。每个子节点均集成了7款模糊测试框架：AFL、AFLFast、FairFuzz、Radamsa（由AFLplusplus[59]中的-R模式来实现）、Libfuzzer、Honggfuzz、AFLGo，以及一款符号执行工具QSYM，通过这些工具可以实现软件漏洞检测、软件漏洞修复验证。子节点会依据任务信息中指定的模糊器，来选择特定的模糊测试框架执行漏洞检测任务和软件漏洞修复验证任务。同时，为了提升模糊测试的效率，执行相同任务的子节点之间会定时同步种子，共享最新的模糊测试信息，提升整体模糊测试的效率。具体的同步方法是：各子节点通过将种子上传至后端，由后端实现种子的中心化管理，再由后端定期将种子同步给各个子节点。

子节点部分的代码由python编写实现，为每一个模糊测试框架编写了调用接口，通过调用接口可以调用模糊器进行软件漏洞检测和漏洞修复验证。子节点中主要使用了python的request库和multiprocessing库，通过使用request库实现与后端的信息交互，通过multiprocessing库实现模糊测试子进程的创建。

## 三、功能模块

### 3.1软件漏洞检测模块

在本模块，可以对软件进行自动化的漏洞检测。软件漏洞检测的过程如下图所示，主要包括6个步骤：任务创建、任务管理、子节点执行任务、子节点执行时种子同步、漏洞去重、结果展示。首先，在前端进行软件漏洞检测任务的创建，有多种漏洞检测方法可供选择。之后任务信息会被传输到后端，由后端对任务进行管理、按照任务所需的资源调度空闲的子节点执行任务。在任务的执行过程中，各子节点会通过后端实现种子的同步，依靠种子同步，可以实现各子节点之间的测试信息共享，提升整体的测试效率。同时，各子节点在软件漏洞的检测过程中，会将发现的漏洞上传给后端，由后端进行统一的管理，当任务执行完毕后，后端会启动漏洞去重操作，将多个节点提交的相同的漏洞去除，最后将唯一的漏洞信息上传至前端显示。接下来，本节将会对部分关键步骤进行详细阐述。



软件漏洞检测过程图

3.1.1 任务创建（4种软件漏洞检测方式介绍）

在任务创建时，一共有4种软件漏洞检测方式可以选择，分别是普通模糊测试、集成模糊测试、混合模糊测试、目标点模糊测试。其中，普通模糊测试是从AFL、AFLFast、FairFuzz、Libfuzzer、Honggfuzz这5款模糊测试器中选择一款，在设定好测试节点数量后，各子节点均采用同一款模糊测试器对软件进行漏洞检测。5款模糊测试器之间的区别如下表所示：

各模糊测试器之间的差异表

|  |  |  |  |
| --- | --- | --- | --- |
| 模糊测试器 | 软件编译 | 覆盖率信息获取 | 测试过程 |
| AFL | 通过AFL框架自带的afl-gcc/afl-g++ 编译脚本进行编译，并对软件进行边缘级别插桩 | 待测软件执行测试用例时，会触发所覆盖的软件中的插桩点，该信息会记录在bitmap中，AFL通过获取bitmap得到测试用例的分支覆盖信息 | AFL启动forkserver来控制待测软件，为待测软件创建子进程，然后通过检测子进程的状态来判断是否触发漏洞。该模糊测试过程过程可以一直反复执行，即使发现漏洞也不会停止 |
| Libfuzzer | 借助clang++进行编译，生成.o文件，最后链接libFuzzer.a生成可执行文件。在编译时可通过设定参数对软件进行三种粒度的插装 | 代码覆盖率信息由LLVM的SanitizerCoverage 工具提供 | Libfuzzer与待测软件链接，形成一个测试整体，可通过特定的模糊入口点（也称为“目标函数”）将测试用例喂给待测软件。测试过程中一旦发现漏洞，变会终止测试 |
| Honggfuzz | 通过Honggfuzz自带的编译脚本对软件进行编译，并借助ASAN框架，可以实现函数级、基本块级、边缘级三种粒度的插桩 | 1. 以基于软件反馈的方式获取覆盖率信息，与AFL类似  2. 以基于硬件反馈（Intel BTS、Intel PT、CPU分支指令数）的方式获取覆盖率信息 | 与AFL类似，模糊器与待测软件分开，通过创建子进程并监控其运行状态。测试过程可一直持续，即使发现漏洞也不会停止 |
| AFLFast | 与AFL相同 | 与AFL相同 | 与AFL相同 |
| FairFuzz | 与AFL相同 | 与AFL相同 | 与AFL相同 |

集成模糊测试。由于现实中软件具有复杂性和多样性，单一的模糊测试器无法很好的应对所有的软件，导致测试过程缺乏鲁棒性，所以Chen等人提出了将多种模糊测试器集成在一起的方法，使得模糊策略多样化，提高系统的鲁棒性。但是，现有的集成模糊测试方法EnFuzz、CUPID均是使用固定的模糊测试器组合，在面对现实中不同的软件时，会存在一定的局限性。因此本文提出了一种自适应的选择方法，在面对不同的待测软件时，系统可以依据分配的资源数自动选择最适合待测软件的模糊测试器组合。如果用户选择创建集成模糊测试任务，平台会使用改进后的集成模糊测试方法对待测软件进行测试，先对待测软件进行初步测试，对各个模糊测试器的表现进行评分，然后，依据任务创建时分配的资源数来选取评分高的模糊测试器进行组合。

混合模糊测试。混合模糊测试是指将模糊测试与其他的软件漏洞检测方式结合在一起对软件进行漏洞检测的方法，多种方法之间可以形成互补，有效的提升漏洞检测的效率。在本文中，具体指模糊测试与符号执行的结合。模糊测试在执行过程中很难通过变异生成覆盖到复杂条件分支的测试用例，导致对软件测试的覆盖率低的问题，而符号执行却可以有效的求解出复杂条件分支的约束；符号执行则存在路径爆炸和部分约束无法求解的问题，而模糊测试可以有效的解决路径爆炸。因此，两者的结合可以形成互补。在本平台中，要创建混合模糊测试任务，至少需要启动两个子节点，一个子节点启用QSYM负责进行符号执行，其余的子节点启动AFLFast进行模糊测试。

目标点模糊测试。本方法可以定向的检测软件中可能存在漏洞的代码部分，减少对软件整体进行测试的开销，例如，检测软件中新引入的补丁是否会存在漏洞安全隐患。因为软件中不同的代码块存在漏洞的概率是不一样的，所以对软件不同部分进行相同力度的测试并不合适。本平台中的目标点模糊测试方法，使用AFLGo模糊测试器对软件进行测试。

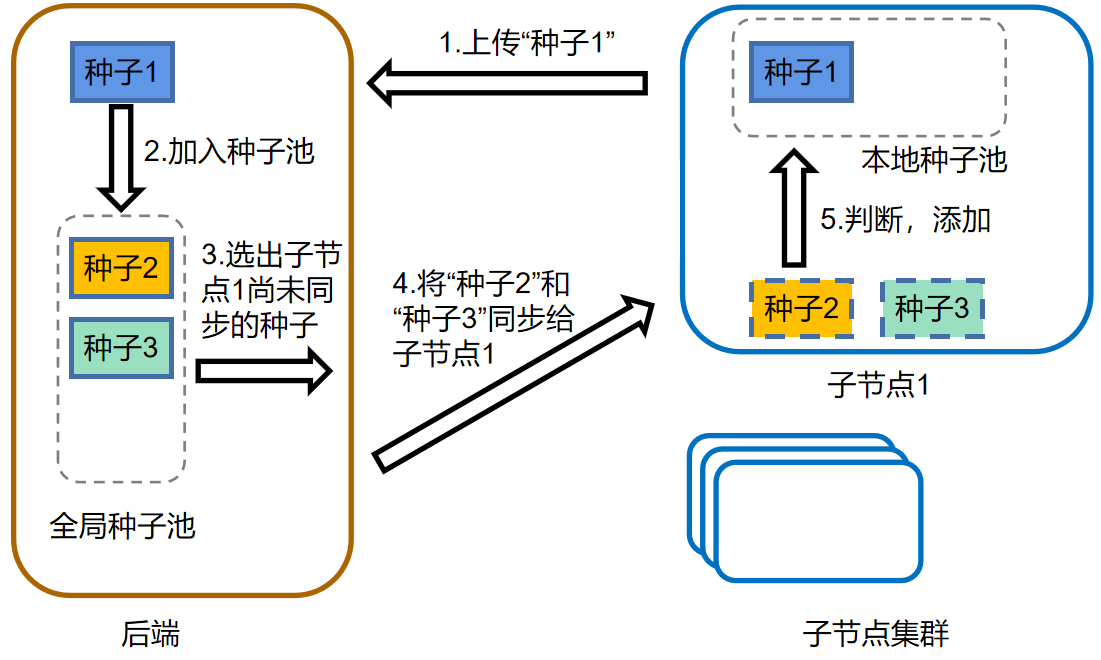
3.1.2 种子同步

针对执行同一个软件漏洞检测任务的各个子节点，进行种子同步有助于各个子节点之间的协同互助，例如，当某个子节点完成了对软件中部分代码的测试后，其余子节点可以通过同步种子获得该进度，然后再此基础上对软件中的其余的部分进行测试，节省了其余子节点的测试时间。各节点之间的种子同步过程如下图所示，为了实现跨主机节点之间的种子同步，所以采用了以后端为中心的集中式同步方式。这里以子节点1为例，具体步骤如下：

（1）子节点1将本地种子池中尚未上传的“种子1”上传给后端。

（2）后端接收来自子节点1上传的新种子，并将其添加入全局种子池中。然后，从全局种子池中选择其余子节点上传的且尚未同步给子节点1的种子，将这些种子同步给子节点1。

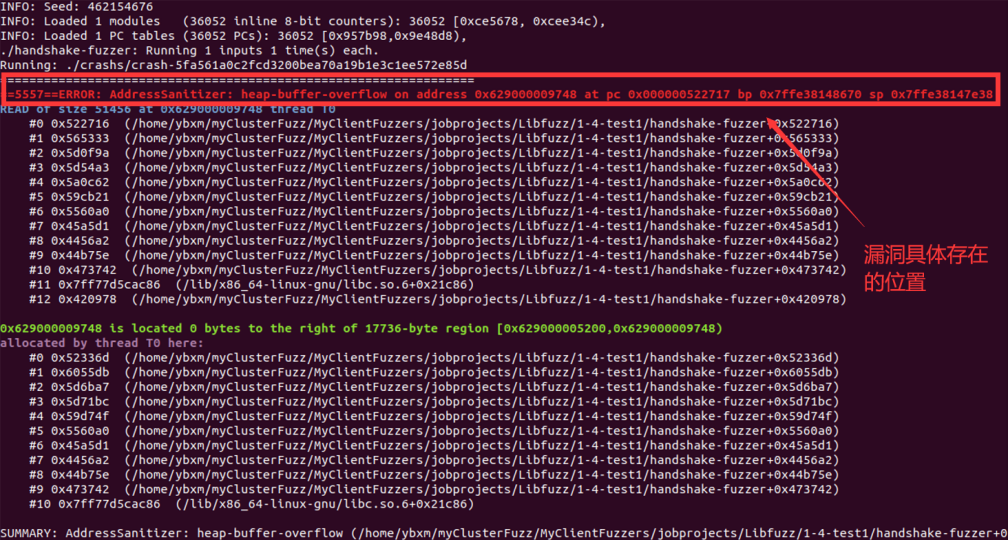
（3）子节点1收到来自后端的种子后，对其进行检验，判断是否是有趣的种子，如果是，则添加入本地种子池中，否则丢弃。



种子同步过程图

3.1.3 漏洞去重

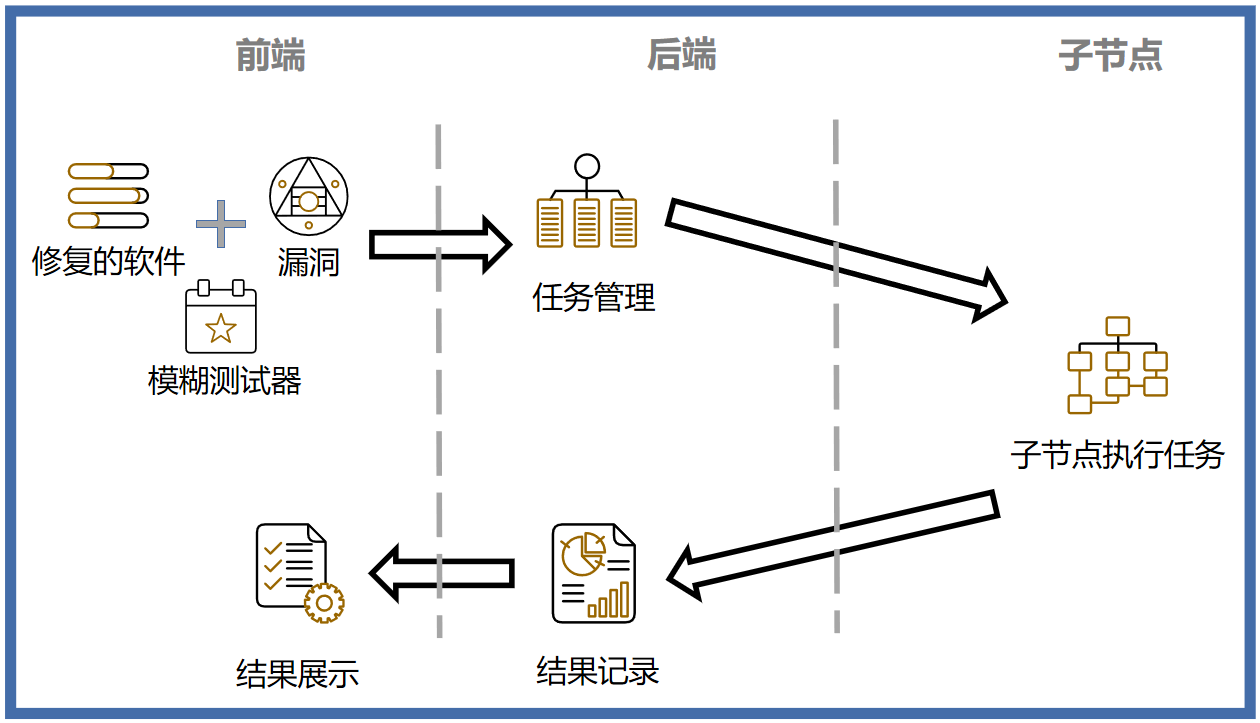
若软件漏洞检测任务启用多个子节点进行测试，则各个子节点在检测的过程中可能会发现相同的漏洞，这些漏洞最终都会汇集于后端处，需要对其进行去重后才可反馈给前端。为了实现漏洞去重，需要借助ASAN框架[57]，由其产生漏洞对应的错误栈信息，如下图所示。从图中我们可以看到，每一个漏洞的产生均有其特定的位置，可以通过比对“ERROR”记录中漏洞产生的位置来判断漏洞的唯一性，将多个产生于软件同一位置的漏洞进行去除，只保留一个。



漏洞错误栈信息

### 3.2 软件漏洞修复验证模块

本平台设有软件漏洞修复验证功能模块，可以用已发现的漏洞对开发人员修复的软件进行自动验证，以判断软件是否已正确修复，其过程如下图所示。测试人员在前端创建任务，选定漏洞和模糊测试器并上传修复的软件，经由后端将任务交给子节点执行。子节点在执行完成任务后，会将检测结果反馈至后端进行存储，最后在前端进行展示。若是最终漏洞已经修复，则后端会将对应漏洞标记为“已修复”，否则，由开发人员继续针对漏洞进行修改。



软件漏洞修复验证过程图

## 四、数据库介绍

为了对软件漏洞智能检测平台的数据信息进行存储和管理，我们使用了MySQL数据库。软件漏洞智能检测平台的数据库主要包括4张数据表：Jobs、Reproduce、Crashes、Nodes，分别代表软件漏洞检测任务表、漏洞修复验证任务表、软件漏洞信息表和子节点信息表。下面将详细介绍各表的详细字段信息：

（1）软件漏洞检测任务表（Jobs）

软件漏洞检测任务表用来存储软件漏洞检测过程中的相关数据信息，表中所包含的具体字段如下表所示，一共有12个字段，其中id字段作为整张表的主键，用于唯一标记一个软件漏洞检测任务。

name字段的类型为char(30)，用于描述软件漏洞检测任务的名称。

type字段的类型为int，用于描述软件漏洞检测任务的类型，可分别取值1、2、3、4。其中1表示普通的模糊测试，2表示集成模糊测试，3表示混合模糊测试，4表示目标点模糊测试。

fuzzer字段的类型为char(50)，描述软件漏洞检测任务中选择的测试器或测试器组合，一共有8种测试器可以选，分别是AFL、AFLFast、FairFuzz、Radamsa、Libfuzzer、Honggfuzz、AFLGo、QSYM。

botnum字段的类型为int，表示软件漏洞检测任务中的测试节点数量，针对一个待测软件，平台可同时启动多个节点进行并行化测试，以提升测试效率。

surplusnum字段的类型为int，表示剩余的、还未进行软件漏洞检测的节点数，平台的后端会依据该值来调度空闲的节点执行软件漏洞检测任务。

completenum字段的类型为int，表示已经完成检测任务的节点数量，当completenum的值等于botnum时，即表明软件漏洞检测任务已经完成。

time字段的类型为int，表示软件漏洞检测任务中单个节点的测试时间。

exec字段的类型为char(50)，表示待测软件编译后的可执行文件的名字，子节点在进行模糊测试的过程中需要使用该字段值。

jobpath字段的类型为char(100)，表示软件漏洞检测任务中待测软件的存储路径。当子节点向后端请求任务的时候，后端会依据该值从对应目录下找到待测软件的压缩包然后发送给子节点。

crashnum字段的类型为int，表示在软件漏洞检测任务中所发现的软件中所存在的漏洞数量。

createtime字段的类型为datetime，表示软件漏洞检测任务创建的时间。

Jobs表字段信息表

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 描述 |
| id | int | 软件漏洞检测任务的唯一标记id |
| name | char(30) | 软件漏洞检测任务名 |
| type | int | 软件漏洞检测任务的类型 |
| fuzzer | char(50) | 软件漏洞检测任务中选择的模糊器 |
| botnum | int | 软件漏洞检测任务中所需并行节点数量 |
| surplusnum | int | 软件漏洞检测任务中未执行的节点数 |
| completenum | int | 软件漏洞检测任务中已完成的节点数 |
| time | int | 软件漏洞检测任务各节点的检测时间 |
| exec | char(50) | 待测软件编译后的可执行文件名 |
| jobpath | char(100) | 待测软件的存储路径 |
| crashnum | int | 待测软件中所检测出的漏洞 |
| createtime | datetime | 软件漏洞检测任务的创建时间 |

（2）漏洞修复验证任务表（Reproduce）

漏洞修复验证任务表用来存储软件漏洞修复验证过程中的相关数据信息，表中所包含的具体字段如下表所示，一共有10个字段。其中，除了一些常见的字段外，completed字段表示任务的完成情况，其类型为int，值为0表示任务还未执行，为1表示任务已经执行完成。

isfetch字段的类型为int，值为1表示已经有节点获取了该任务，其余的节点不会再请求执行该任务，而值为0则表示该任务尚未有节点获取执行。

isfixed字段的类型为int，值为0表示表示软件中存在的名为crashname的漏洞尚未被修复，值为1则表示该漏洞已经修复了。

Reproduce表字段信息表

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 描述 |
| id | int | 主键，漏洞修复验证任务唯一标记id |
| name | char(30) | 任务名称 |
| fuzzer | char(50) | 模糊测试器名称 |
| crashname | char(60) | 漏洞名称 |
| exec | char(50) | 待检测软件编译后的可执行文件名 |
| jobpath | char(100) | 待检验软件的存储路径 |
| completed | int | 任务完成情况 |
| isfetch | int | 任务是否已经被子节点获取 |
| isfixed | int | 软件中的漏洞是否已经被修复 |
| createtime | char(100) | 任务的创建时间 |

（3）软件漏洞信息表（Crashes）

软件漏洞信息表用来存储软件漏洞检测过程中所发现的漏洞的相关信息，在漏洞列表界面，会将该表中的数据进行显示。表中所包含的具体字段如下表所示，一共有6个字段。

name字段的类型为char(60)，用于表示漏洞的名字。

jobid字段作为外键，对应Jobs表中的主键id值，其类型为int，一条漏洞记录只对应Jobs中的一条任务记录，依据两张表之间的关系，可以得到在一次软件漏洞检测任务中所发现的所有漏洞记录。

crashpath字段的类型为char(100)，表示漏洞文件存储的路径。

isfix字段的类型为int，表示漏洞的修复情况，其中0表示漏洞未修复，1表示漏洞已经修复，默认值是0。

findtime字段的类型为datetime，表示漏洞的发现时间。

Crashes表字段信息表

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 描述 |
| id | int | 主键，漏洞的唯一标记id |
| name | char(60) | 漏洞名称 |
| jobid | int | 外键，对应Jobs表中的主键id值 |
| crashpath | char(100) | 漏洞文件存储的路径 |
| isfix | int | 漏洞的修复情况 |
| findtime | datetime | 漏洞的发现时间 |

（4）子节点信息表（Nodes）

子节点信息表用来存储软件漏洞智能检测平台所拥有的测试节点的相关信息，在节点列表界面，会将该表中的数据进行显示。表中所包含的具体字段如下表所示，一共有7个字段。其中，除了一些常见的字段外，ipaddr表示节点服务器的的ip地址，字段类型为char(20)。

jobtype 字段的类型为int，代表当前节点正在执行的任务类型，0表示当前节点空闲，1表示当前节点正在执行软件漏洞检测任务，-1表示当前节点正在执行漏洞修复验证任务。

fetchtime字段的类型为datetime，表示当前节点最近一次获取任务的时间，依据该时间和任务的执行的时常，可以推测出任务执行的进度。

last\_modify\_time字段的类型为datetime，表示当前节点的最后一次活跃时间，当节点超过1个小时未活跃，则可判定该节点失联，无法再执行任务，节点相关信息即可从表中删除。

Nodes表字段信息表

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 描述 |
| id | int | 主键，唯一标记一条主节点记录 |
| name | char(20) | 主节点名称 |
| ipaddr | char(20) | 主节点的ip地址 |
| jobtype | int | 执行任务的类型 |
| fetchtime | datetime | 任务获取时间 |
| last\_modify\_time | datetime | 节点最后的活跃时间 |
| createtime | datetime | 节点的创建时间 |

## 五、界面介绍及其操作

软件漏洞智能检测平台的前端可视化操作界面部分，主要由软件漏洞检测界面、软件漏洞修复验证界面、漏洞列表界面和节点列表界面共4个界面组成。通过平台前端的可视化界面，可以简化测试的操作流程，将复杂的测试操作过程转化为面向前端测试界面的操作，可以降低测试的难度和极大的提升测试的效率，并有助于测试结果信息更方便的采集、管理和显示。

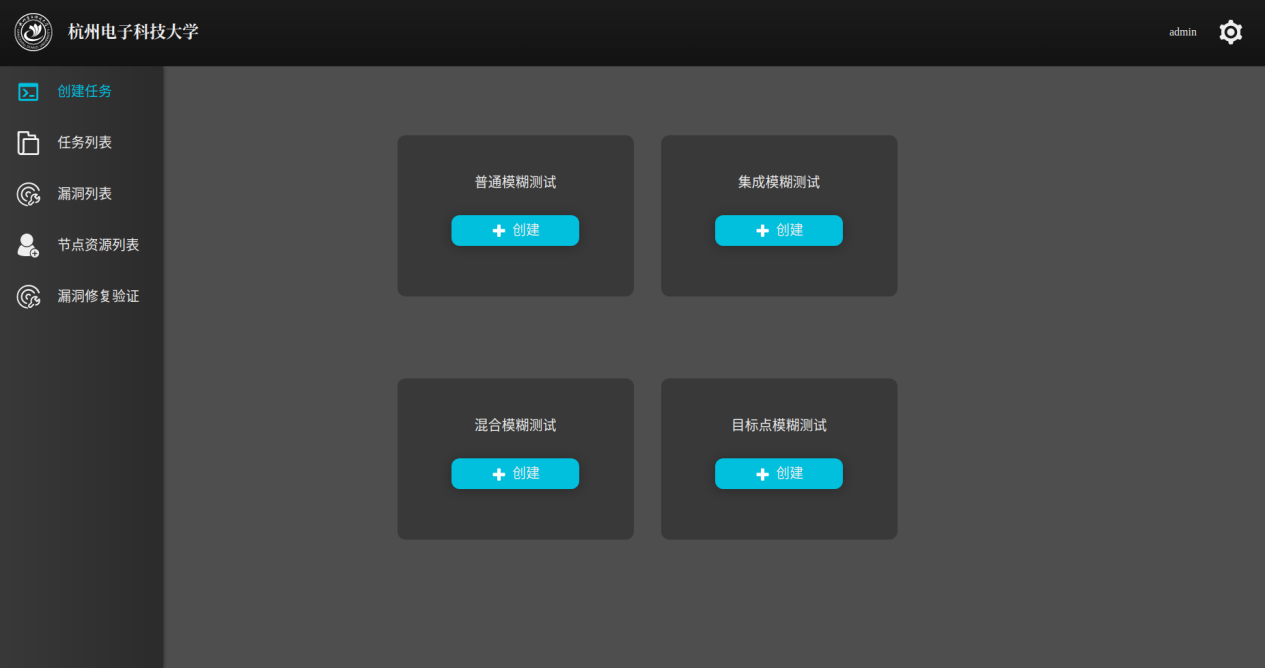
### 5.1软件漏洞检测界面及操作

在本界面，可以自由选择软件漏洞检测的方式，如下图所示，主要包括四种：普通模糊测试、集成模糊测试、混合模糊测试、目标点模糊测试（待实现）。其中，普通模糊测试是指从模糊测试器列表中选择一款模糊测试器进行软件漏洞检测，模糊测试器列表主要包括5款模糊测试器：AFL、AFLFast、FairFuzz、Libfuzzer、Honggfuzz，对于一般情况下的软件漏洞检测，推荐使用这种模式。

集成模糊测试依据待测软件的实际情况，从AFL、AFLFast、FairFuzz、Radamsa、Libfuzzer、QSYM共6款测试器中依据可用节点数来选择出最适合的模糊测试器组合，多模糊测试器协同测试可以充分发挥其各自的优点，形成优势互补，提升测试的鲁棒性，但同时也需要对待测软件进行不同的编译，生成符合各模糊测试器要求的可执行文件，适合对软件漏洞检测有更高要求的情况。

混合模糊测试，由一个节点启动QSYM，对待测软件进行符号执行，其余多个节点启动AFLFast，对待测软件进行模糊测试，符号执行器和模糊测试器之间的协同测试，可以解决符号执行过程中存在的路径爆炸问题、模糊测试过程中的无法突破复杂条件分支问题，使得两者之间形成互补，提升软件漏洞检测的效率。本模式适合对存在较多复杂条件分支且规模较大的软件的测试。

目标点模糊测试（待实现），子节点会启动AFLGo模糊测试器对待测软件的某些部分进行定向检测，而非对软件的整体进行细致的测试，因为软件中不同的代码块存在漏洞的概率是不一样的，所以对不同代码块进行相同力度的测试并不合适。本模式主要用于检测补丁更新后的软件，或者是已经明确软件特定部分可能存在漏洞的情况。



软件漏洞检测任务创建方式图

除了软件漏洞检测的方式可以自主选择外，还可以自由配置软件漏洞检测的测试资源。在任务创建时，可以依据任务的优先级，自主设定测试节点的数量和各测试节点的检测时间，提高测试的灵活性。

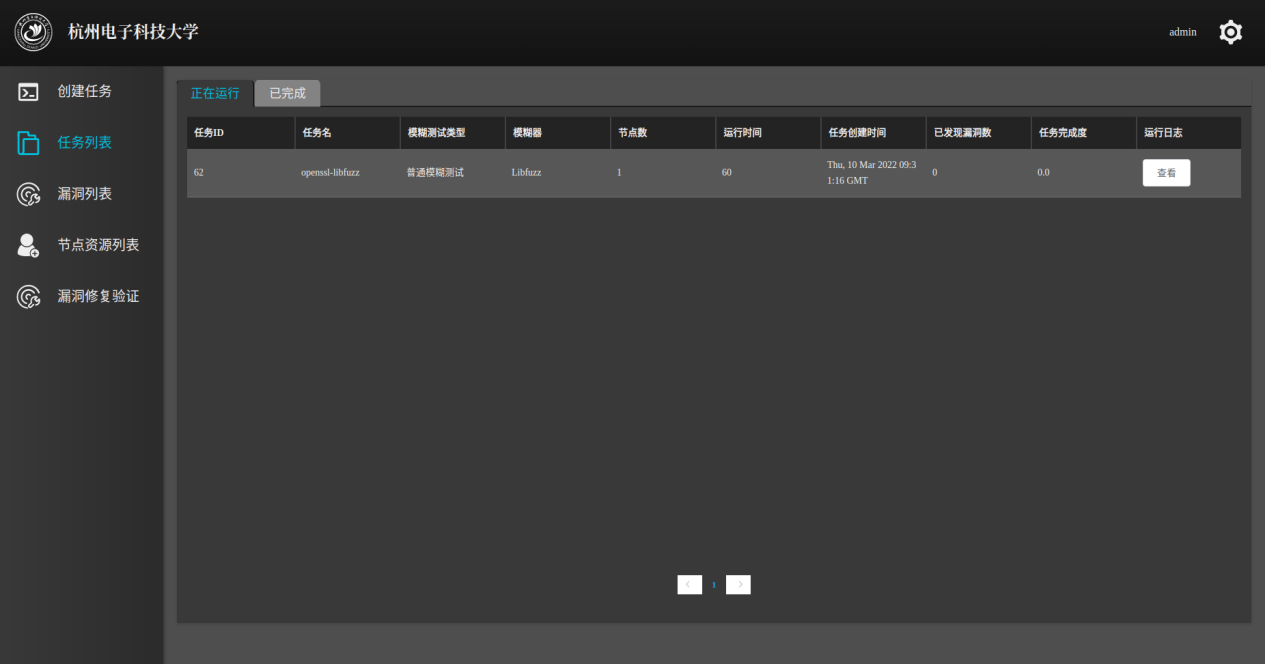
下面将以普通模糊测试为例详细介绍软件漏洞检测任务的操作流程：

1. 选择“普通模糊测试”，在任务创建框中依次填入任务名称、选择的模糊测试器、并行节点数量和测试时间、可执行文件名字以及待测软件编译后所生成的可执行文件，其中测试时间以秒为单位，完成之后点击提交，任务信息就会被传输到后端，存入数据库，由子节点申请任务并执行。如下图所示。



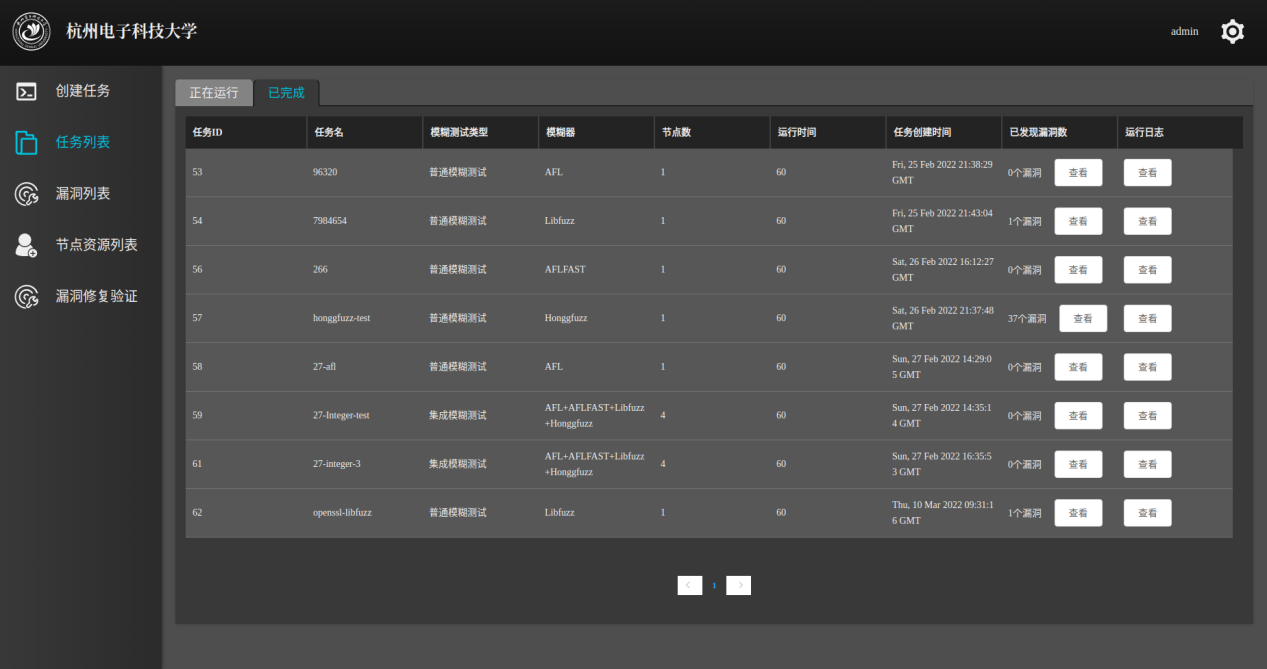
普通模糊测试图

1. 在任务列表的“正在运行”一栏，可以看到刚才所创建的任务的详细信息，并可以点击“运行日志”查看正在运行的任务日志。如下图所示。

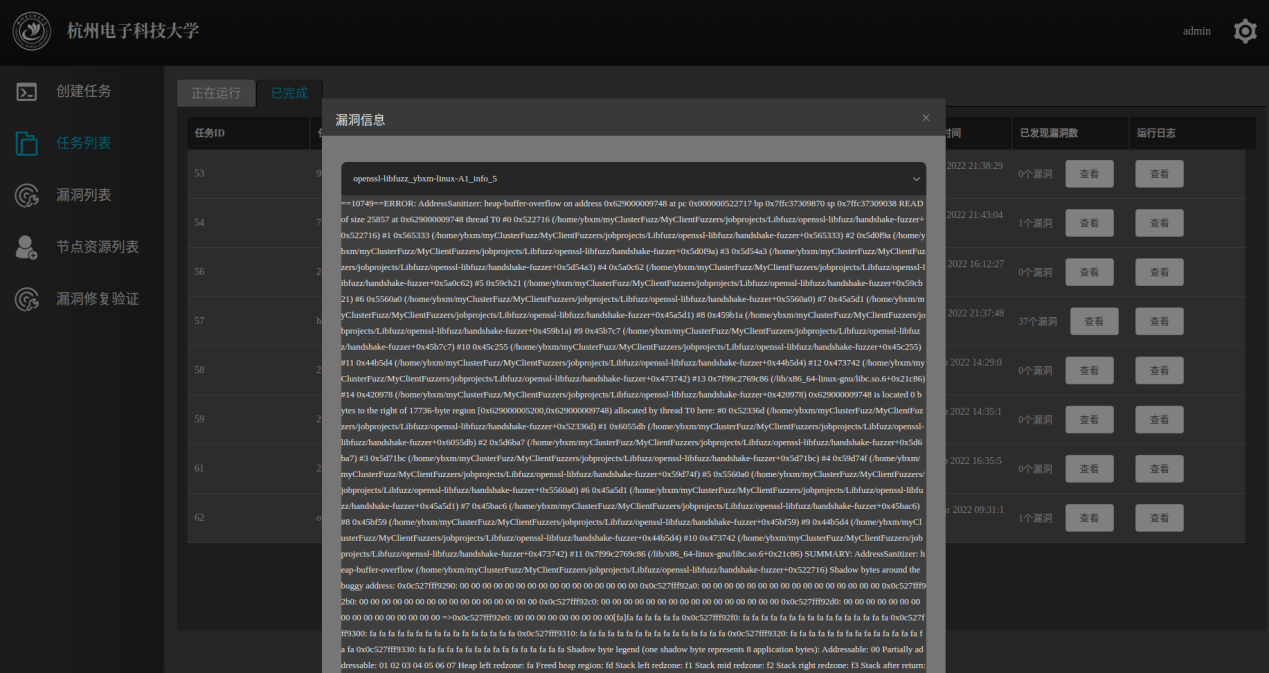


任务运行情况图

1. 当任务运行完成后，会出现在“已完成”一栏的任务列表中，如下图所示。从对应的任务记录中，我们可以查询到软件测试过程中发现的漏洞个数及漏洞的相关信息，如下图所示。通过该漏洞信息可以指导开发人员对软件进行修复。此外，任务记录中还包括了模糊器的运行日志，详细记录着漏洞检测过程中的测试进度、检测的软件路径数等信息。



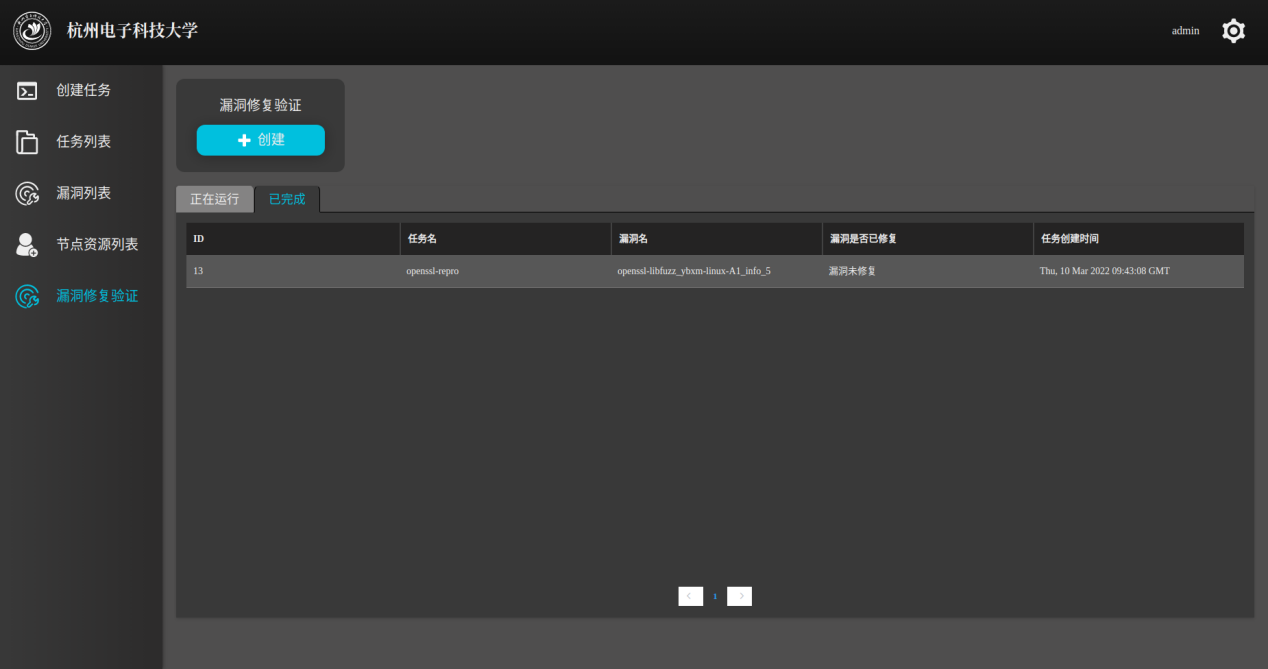
任务完成记录图



漏洞错误栈信息

### 5.2 漏洞修复验证界面

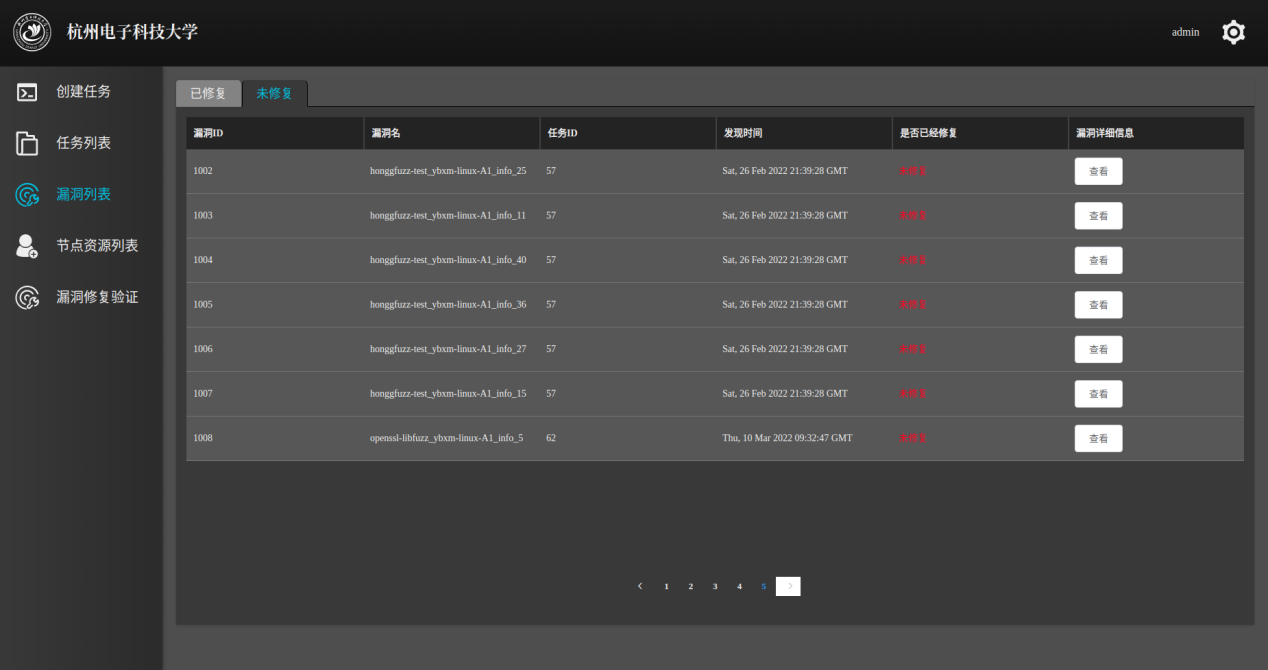
在本界面中，可以对开发人员修复的软件进行检验，以判断软件是否已正确修复。在漏洞修复验证任务框中，从漏洞列表中选取相应的漏洞用例，将其名称填入，并将已修复的软件编译后上传，平台即可利用该漏洞对软件进行漏洞修复验证，验证结果最后会显示在漏洞修复验证任务列表中，如下图所示。



漏洞修复验证界面图

### 5.3 漏洞列表界面

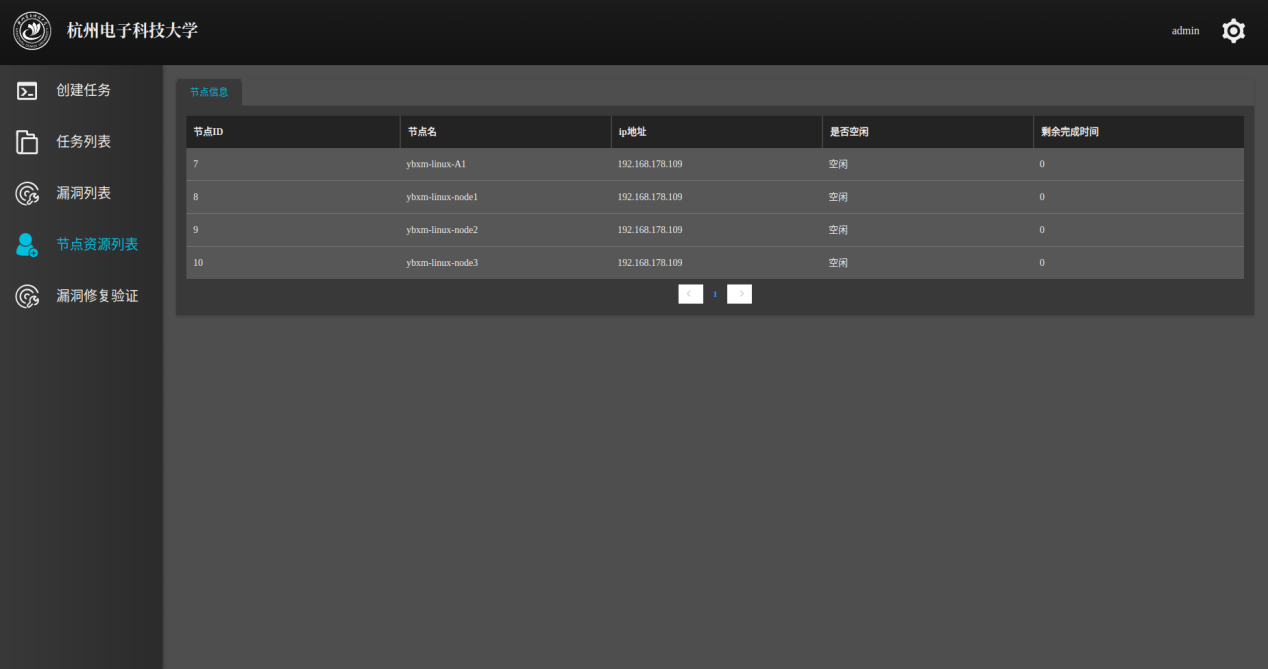
在漏洞列表界面，如下图所示，汇集了所有软件漏洞检测任务执行过程中发现的漏洞，漏洞集合被分成两栏：已修复和未修复。已修复一栏中出现的漏洞，均是经过漏洞修复验证被证明已经修复了的漏洞。而在未修复一栏中，可以查看到每个漏洞对应的错误栈信息，开发人员依据其可进行软件漏洞修复。



漏洞列表界面图

### 5.4 节点列表界面

在本界面，可以管理平台拥有的所有子节点，子节点是指服务器集群中具有软件漏洞检测能力的各台主机，平台可以调度子节点实现分布式并行化的软件漏洞检测，以提升软件漏洞的检测效率。节点信息列表中会包含每个子节点的名称、ip地址、是否空闲、剩余任务完成时间等信息，测试人员可以依据该信息合理规划和使用节点资源。如下图所示。



节点列表界面图

## 六、待改进

1. 软件漏洞检测方法之一的目标点模糊测试功能尚未集成。目标点模糊测试前期需要使用静态分析等方法获取软件中的可疑代码块，然后需要在测试前对软件进行插桩计算，实现过程复杂，固尚未实现。

2. 种子同步过程还需要进一步的修改，qsym在处理同步的种子的过程中，只能识别类似“id:000.......”开头的种子，故需要将同步过来的种子进行重命名。此外，对于类AFL的模糊测试器，同步得到的种子需要存入模糊器测试目录的同级目录下，类AFL的模糊测试器会自动同步，不能直接加入其种子队列中。