鸿渐源代码检测系统

产品白皮书

(RedRocket-SAST)

2020.12

目录

1 概述

1.1 背景

代码是软件的“基因”，在代码层次上进行静态分析，相对于传统的动态功能测试、单元测试、渗透更加系统和精确。首先，程序的静态分析可以打开程序的结构，从程序的逻辑结构进行分析，这种白盒的方式能够明显降低漏报。其次，代码分析往往在软件开发或测试阶段，研究表明，开发阶段发现缺陷的成本是测试1/4,是运行阶段的1/10,而开发阶段发现出现缺陷的数量是测试阶段3倍，是运行阶段的10倍。由此，越来越多的组织采用源代码检测工具进行质量或安全检测，代码审计也成了很多工业标准(IEC61508,DO-178B/C等)和安全标准(等保2.0等)必要的检测手段。

近年来，源代码静态分析工具种类众多，很多企业希望通过部署源代码检测产品来验证和解决软件开发和测试过程中的出现的代码问题。近年来,虽然国内大型企业机构的检测能力有所提升,但是在代码质量保障体系建设上,仍存在很多误区和困难：

- **研发人员更关注的误报，而不关注漏报数量**

由于全部缺陷的数量无法确定，工具厂商更关注的误报，相对漏报偏高。2017年软件工程一篇顶会论文论述了100个真实CVE缓冲区溢出漏洞，用5个国际著名的静态工具检测加在一起仅报出了其中20个，漏报率达80%。国际通用的OWASP Benchmark采用大部分商业工具检测，漏报率也接近4成。由此很多企业采用了多款工具共用求合集的方式来降低漏报，但该方式增大了企业的成本。

- **国外工具对于企业内或行业内规则和缺陷的定制，很难实现**

大部分企业采用国外商业工具，国内代理商很难具备二次开发的能力，国外工具也不暴露接口，尤其针对领域内缺陷模式，企业或行业内部往往有自己的编码标准，很难针对性开发。

- **代码缺陷普遍存在，采用开源工具的二次开发很难达到检测严重缺陷**

具备一定开发能力的组织往往采用在开源工具上进行二次开发，但是开源工具提供的分析接口往往比较基础与商业工具的分析能力差距较大，不能检测更为深层次的缺陷，尤其是针对上下文敏感、跨函数、路径敏感的缺陷类型基本无法检测，导致检测能力偏弱。

1.2 源代码检测系统介绍

“鸿渐”源代码检测系统(简称鸿渐-SAST)是基于北京大学多年的研究成果，应用多种国际先进代码分析、深度学习技术，研发的源代码检测系统，面向组织的源代码检测需求，在不改变组织现有开发、测试流程的前提下，与源代码管理系统(Git、SVN等)、缺陷管理系统(如Jira、Bugzilla、禅道等)、持续集成工具(如Jenkins、禅道)无缝对接，将源代码检测融入企业的研发流程，实现了源代码编码规则检测、运行时缺陷检测、安全漏洞检测、度量统计、克隆检测、逆向架构图自动生成，并提供了检测器自主研发接口等功能，帮助组织快速构建源代码安全自主检测体系和能力。

鸿渐-SAST系统为纯软件部署方式，包括了公有云、私有云、命令行三种部署方式。公有云检测是将被检测代码传入部署的公共云检测系统中，在公有云部署的系统中进行检测。私有云检测是将系统安装在客户的环境中，可进行断网检测。用户也可以采用命令行方式调用系统进行检测，检测结果将以Web Services的方式返回。系统运行流程图1所示，为了支持多并发检测，系统采用集群的方式，在缺陷模式挖掘上，采用了基于深度学习的自动挖掘算法每天自动挖掘新的模式以支撑更多的框架，并提供相应的检测器开发接口以帮助使用者自研检测器。

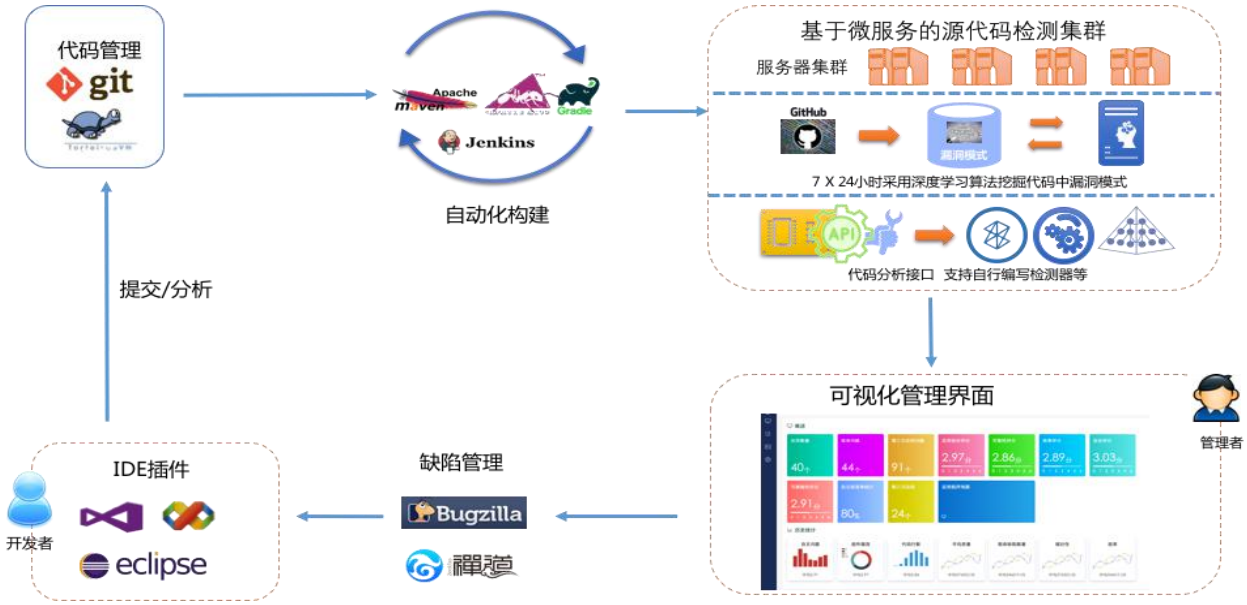


图1：系统基本运行流程

1.3 技术指标

1.3.1 主要技术及性能指标

项目	支持情况
兼容平台	支持 Windows、Linux、中标麒麟 等多种主流通用操作系统开发的源代码的检测。
支持语言	支持 C/C++、Java、Android、PHP、JSP、HTML、C#.net、VB.net、JavaScript、Python、Kotlin,Groovy,Scala 等约十余种主流开发语言
安全漏洞	OWASP TOP 10、CWE/SANS TOP 25、ISO 17961、CERT JAVA，包括：缓冲区溢出、SQL 注入、命令行注入、跨站脚本攻击等(见附录一)
缺陷	CWE(Common Weakness Enumeration)170余种缺陷类型，包括：内存泄漏、数据越界、空指针解引用、释放非堆内存、返回局部指针等(见附录一)
编码规则	MISRA 2004、MISRA 2008、MISRA2012、GJB5369、GJB8114、航天921
支持的编译器	ARM C/C++、Borland C++、GNU GCC C++、Intel C++、Keil compilers、SUN CC、SUN/Oracle JDK 、OPEN JDK 、Visual Studio、Wind River 等几十种编译器
Bug 管理系统	Bugzilla、Jira、TFS、禅道
扩展能力	自定义检测规则、检测报告
检测效率	平均 100 万行/小时

1.3.2 支持的框架

语言	框架
Java	Spring、Spring Boot、Spring JDBC、Spring LDAP、Spring MVC 、Spring Data JPA、Spring Web Flow、Spring Security、Struts1、 Struts2、Hibernate、Mybatis、Ibatis、Apache Axis1、Apache Axis2、Json-lib、Fastjson、Gson、log4j、SLF4J、Apache commons、HttpComponents 等。
C/C++	Qt4、Qt5
PHP	ThinkPHP、laravel、CodeIgniter、Zend Framework、CakePHP、Symfony
Python	Django、Diesel、Flask、Cubes
JavaScript	Jquery、Vue、Nodejs

1.1 部署方式

- **硬件配置要求**

序号	硬件或固件项名称	配置	用途

1	鸿渐-SAST分析服务器	CPU：酷睿i5-9400以上 内存：32GB 硬盘：1TB	提供代码检测服务以及检测结果的查看和确认
2	鸿渐-SAST命令行跟踪编译客户端	CPU： i5-9400及以上 内存：32GB 硬盘：1T	提供通过编译的方式检测源代码
3	鸿渐-SAST持续集成插件	CPU：i5-7400及以上 内存：8GB 硬盘：1T	持续集成插件辅助配置代码检测

- **支持的操作系统**

处理器	操作系统
Sun SPARC	Sun Solaris 8, 9 and 10
Intel and AMD	Linux、中标麒麟、SUSE、CentOS、Ubuntu
	MAC 10.1,10.6,10.8
	Windows 7 X32/X64 Windows Server 2008 Windows Server 2003 Windows Server 2012 Windows Server 2016 Windows Vista Windows 10

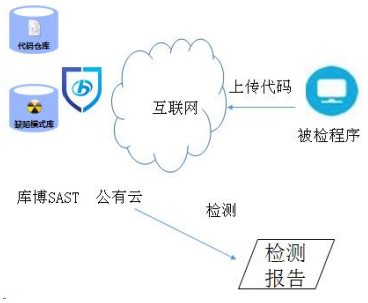
- **支持的浏览器**

Internet Explorer 9及以上、Edge浏览器， Google Chrome V28及以上， 火狐浏览器，

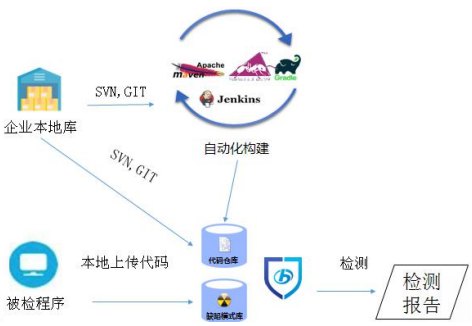
➢ 支持的部署方式

1. 可以通过命令行客户端检测通过跟踪编译的方式检测提交到鸿渐-SAST，该部署依赖当前的编译环境，检测结果误报和漏洞更低更加准确。
2. 公有云检测的部署方式用户可以通过文件压缩包，SVN，Git等方式上传到鸿渐-SAST代码仓库，代码的缺陷检测在互联网云端，缺陷模式库的更新自动进行，该方式的优点是没有部署成本易用，库更新及时，不需要编译通过即可检测，更加简单。
3. 私有云检测可以本地代码通过SVN，Git等方式同步到鸿渐-SAST检测服务端检测生成报告，相对公有云而言它所有的检测服务都在本地，可以断网检测，不需要编译通过即可检测，缺点是缺陷模式库需要手动更新。

指纹生成客户端检测



公有云部署检测



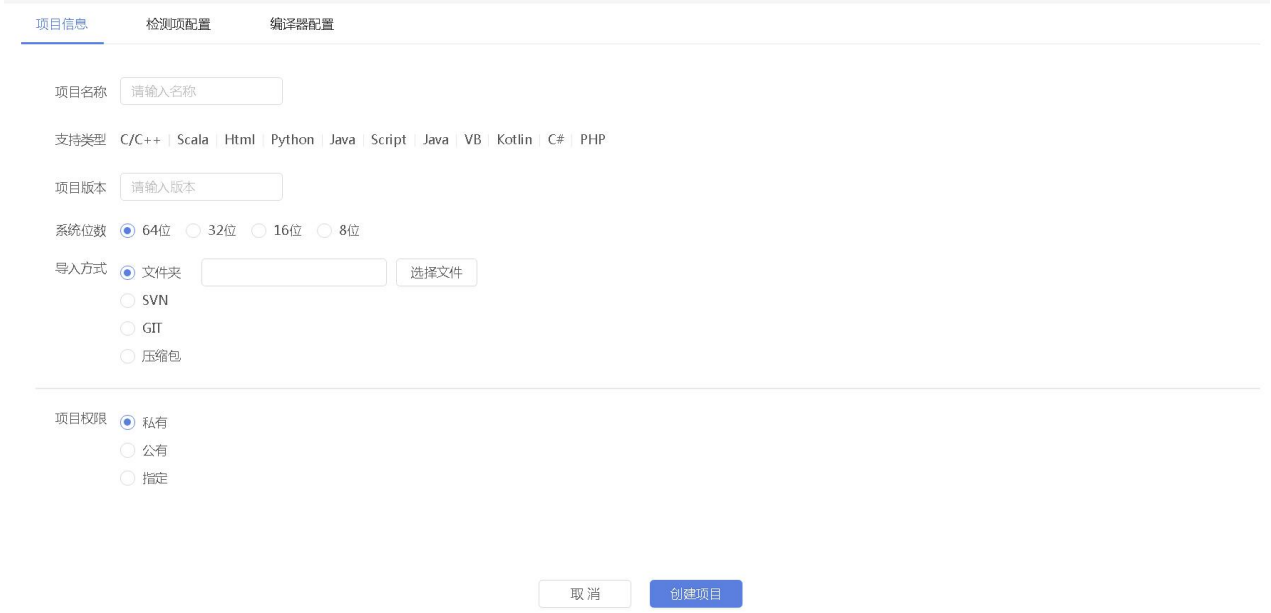
断网环境的私有云部署检测

2 系统功能

2.1 源代码缺陷检测

鸿渐-SAST可以在不运行程序的情况下全面扫描代码，快速报告软件中的漏洞，包括边界条件复杂的漏洞，是对动态测试的有效补充。在缺陷检测方面支持 C/C++、Java、Android、PHP、JSP、HTML、C#.net、VB.net、JavaScript、Python、Kotlin,Groovy,Scala等约10余种主流开发语言的软件源代码的缺陷检测。可检测的缺陷种类包括缓冲区溢出、SQL注入、跨站脚本等 74 个大类，2000 多个小类。兼容CWE(Common Weakness Enumeration)、OWASP TOP 10、CWE/SANS TOP 25、ISO 17961、CERT Java、MISRA系列、GJB系列等多种国际和国内标准

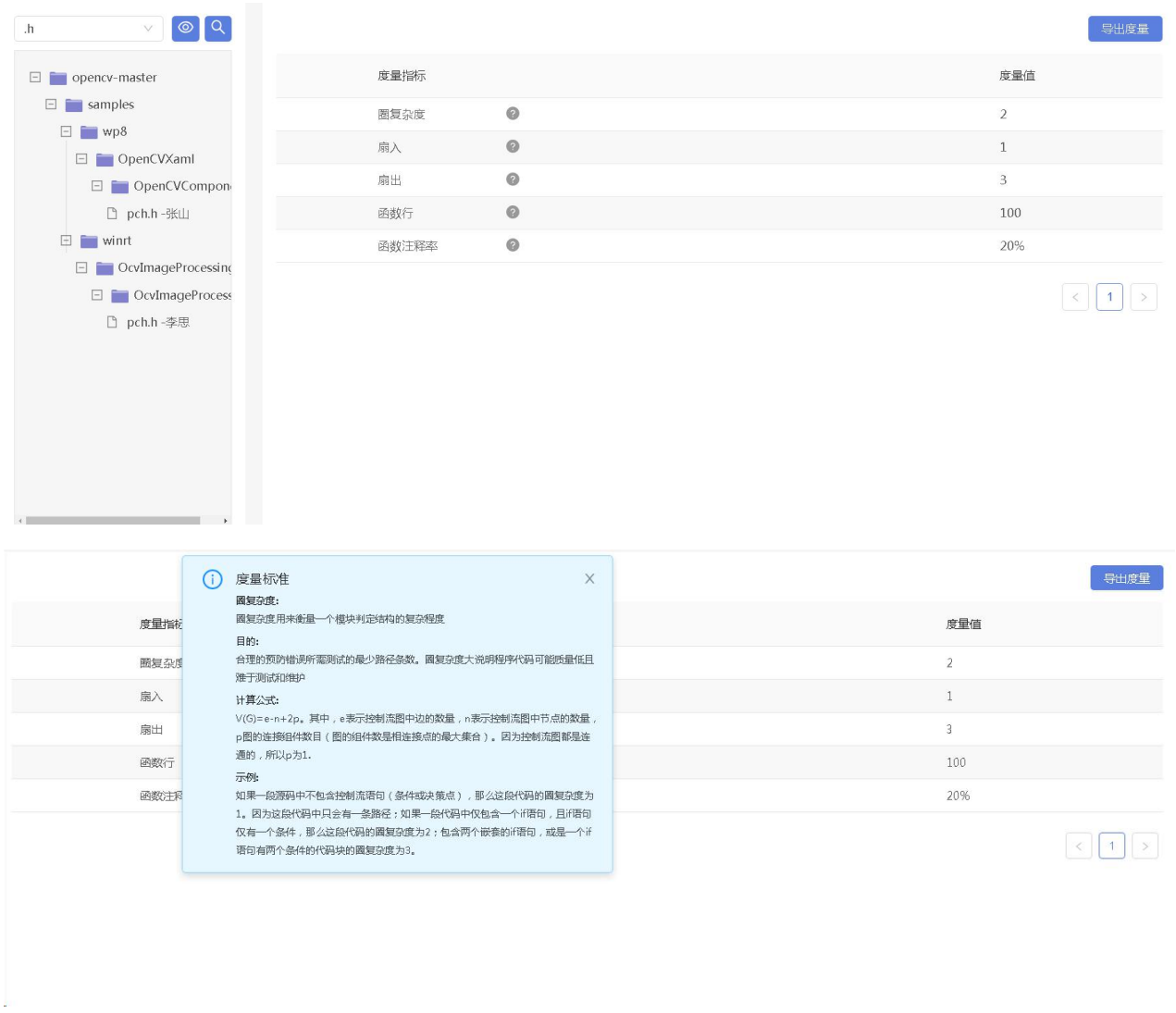
源代码缺陷检测如下图所示：



2.2 源代码度量

为了提高代码的可维护性，需要了解软件在代码行、圈复杂度、扇入扇出度等各个角度的度量指标。鸿渐-SAST支持从项目级、文件级、函数级等级别度量30余种，并将未达标的文件和函数分配给相应的开发人员进行修改。

源代码度量如下图所示：



2.3 源代码克隆分析

鸿渐-SAST支持代码克隆分析找出功能与原代码相同、但是进行了修改的代码。由于所使用的第三方软件或者企业内部代码在复用时可能会进行修改，需要使用克隆检测，发现该段代码的来源，并发现因代码克隆带来的软件不一致。鸿渐-SAST的克隆分析可以应用在代码抄袭检测、同源漏洞检测等方面。

克隆分析结果如下图所示：

共4个重复块

所选文件名	所选文件行	重复文件路径	重复文件行
auth_utils.c	15-22	3dstools/src/adobe/shit/pku/1727/auth_md5.c	222-231
mysql.c	112-331	3dstools/src/adobe/shit/pku/1727/mysql_back.c	115-336
auth_client.c	100-122	3dstools/src/adobe/shit/pku/1727/auth_client.c	200-222
auth_utils.c	150-233	3dstools/src/adobe/shit/pku/1727/auth_md5.c	330-413

< 1 >

/src/utils/auth_utils.c

```
Common_TestClass::Common_TestClass(void)
{
    p=new char[10];     //
}
Common_TestClass::~Common_TestClass(void)
{
    int i;
    i=path_fun(3);
    if(i < 0)
        delete []p;
}
int condition_path(char *j)
{
    if(j != NULL)
        return true;
    return false;
}
//MLK
int memory_leak(void)
{
    Common_TestClass tc;  //
    return 0;
}

//UFM-double free
void throw_mysqlx_error(int &error)
{
 if (!error)
    return;
    switch(error)
    {
      case 1:
        return;
      default:
        break;
    }
}
int double_free(int error)
{
    Common_TestClass *tc=new Common_TestClass
    if(error)
        {
            delete tc;          //
            throw_mysqlx_error(error);
        }
    delete tc;              //
    return 0;
}

//UFM-use after free

int use_after_free(char *j)
```

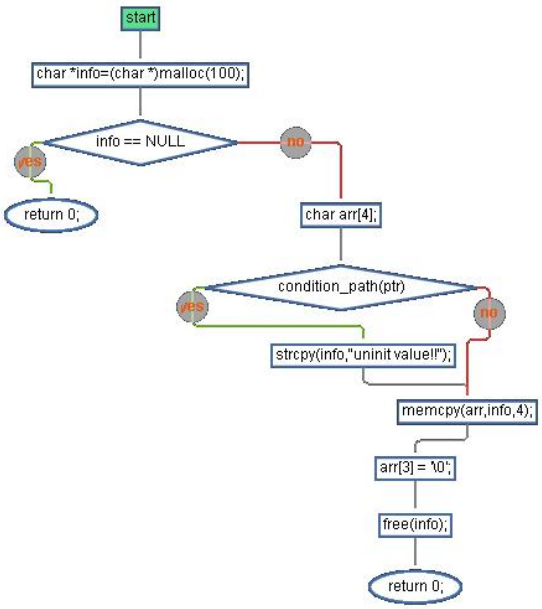
/src/utils/auth_md5.c

```
//string null Terminate
void foo(char * src)
{
    char buf[8];//
    char tgt[1024];

    strncpy(buf, src, 3);//
    strcpy(tgt, buf);    //
}

Common_TestClass::Common_TestClass(void)
{
    p=new char[10];     //
}
Common_TestClass::~Common_TestClass(void)
{
    int i;
    i=path_fun(3);
    if(i < 0)
        delete []p;
}
int condition_path(char *j)
{
    if(j != NULL)
        return true;
    return false;
}
//MLK
int memory_leak(void)
{
    Common_TestClass tc;  //
    return 0;
}

//UFM-double free
void throw_mysqlx_error(int &error)
{
 if (!error)
    return;
    switch(error)
    {
      case 1:
        return;
      default:
        break;
    }
}

//free stack
#define MAX_ALLOCATION 1000
int free_non_heap(int argc, char *argv[]) {
    char *c_str = NULL;
```

2.4 源代码分析图形化展示

鸿渐-SAST支持根据源代码反向生成程序架构图如：函数控制流图、文件函数调用图、项目函数调用图、继承关系图、UML 类图等。提供包括 PNG、SVG 等格式导出。

源代码分析图形如下图所示：



2.5 自定义检测器开发

企业中往往存在特殊的业务或者是应用场景，几乎所有源代码测试工具都无法覆盖此场景下的缺陷检测，为此、鸿渐-SAST提供的自定义检测开发模块，用户可以通过此模块开发检测器，鸿渐-SAST提供如:抽象语法树、调用分析、依赖分析、控制流分析、控制依赖分析、多态分析、区间分析、修改影响分析、类继承关系分析等数十种分析接口，协助用户形成自己的检测能力建设。

2.6 自动化检测流程及与环境集成

鸿渐-SAST支持SVN,Git等代码管理工具，并且检测结果可与Bugzilla，Jira，禅道等Bug管理工具进行整合，同时提供与Jenkins，Bamboo等持续集成环境的接口，用户可以将鸿渐-SAST，持续集成工具，代码库，Bug管理系统四者进行关联，持续集成工具通过任务设置定期获取代码并编译，再通过鸿渐-SAST持续集成工具进行检测，将结果推送至Bug管理系统，最大限度的减少人工干预的工作量。

2.7 绩效统计

鸿渐-SAST支持通过连接SVN、Git等代码管理系统中获取提交记录，结合源代码缺陷检测、源代码度量、源代码克隆分析，分析统计开发人员和项目的软件质量((问题的等级和个数)、工作难易度（可靠性、可维护性、复杂度)、工作效率/能力（代码提交量、问题的修复量和修复率、问题密度)等信息，为开发人员(Key Performance Index)考核提供重要的依据。

绩效统计如下图所示:

2.8 定时检测

鸿渐-SAST支持本地源代码检测的同时也可支持从 SVN、 Git 等代码管理系统中获取源代码进行检测， 支持定时自动检测， 用户可将鸿渐-SAST与代码库进行关联，配置好定时检测计划， 代码卫士将按照任务设置定期自动获取代码库中的源代码进行检测，检测后的结果可以根据根据代码库提交记录自动分配给相关负责人，并且可以通过邮箱发送提醒相关责任人。

2.9 外部接口

鸿渐-SAST支持RESTFUL风格的WEB接口，接口使用http或者https实现，提供的接口如：缺陷检测接口、设置检测项接口、代码度量接口、代码克隆分析接口、获取检测结果接口等数十种接口，方便用户与企业生成环境集成。

2.10 系统管理

- **用户管理**

系统提供基于部门、角色、用户的账号管理体系，管理员可以为不同的项目组创建拥有不同权限的角色和用户账号。

用户管理如下图所示：

- **日志管理**

系统会记录并提供本系统的操作日志、异常日志和系统运行日志，可以帮助系统运维人员更好的监控系统运行状况，及时修复平台问题，保障系统能够长久稳定的运行。

- **检测引擎管理**

提供检测引擎节点的管理功能和检测队列管理功能，可以查看系统当前的检测引擎节点信息和当前状态，可以对检测引擎和检测队列进行删除、修改等操作。

- **系统配置管理**

查看当前运行系统的基本状况，包括授权 key 信息、操作系统、CPU、硬盘、内存状态等信息。

3 产品优势

鸿渐-SAST运用了多种先进的静态分析技术保证了分析的效率、精度。结合了深度学习的模式挖掘技术，保证了代码分析引擎增加了缺陷模式，保证了分析的广度。包含了编码规则、缺陷以及漏洞三种类型，上千种类型的检测器，完成了多种类型的检测，集多个工具特点于一身，为企业降低了成本。提供了工具的分析引擎接口，用户可以根据自己缺陷类型进行定制研发。团队底层研发人员，全部为北京大学软件工程专业毕业，可以保证检测技术的先进性和可定制性。

3.1 国际顶尖技术——综合运用多种代码分析技术，检测更为快速、精准

鸿渐-SAST运用了多种国际先进静态分析技术，能够进行路径敏感、上下文敏感、对象敏感分析，能够做到发现更深层次的问题，检测效率达到了平均150万行每小时，能够做到精度和效率更优的平衡。

3.2 检测类型丰富——能够检测缺陷，漏洞，规则，节省企业采购成本

大多数同类工具只能支持漏洞、缺陷及编码规则中的一类，鸿渐SAST兼顾了全部类型的检测，全面发现代码中的各类问题。此外，基于深度学习的缺陷模式挖掘技术，在大数据的基础上持续进行漏洞模式挖掘和更新，保证了系统能够检测新框架中的问题。

3.3 专业标准符合度高——完全符合国际国内缺陷漏洞检测标准，完全符合GJB 8114等标准

很多工具往往根据工具中已存在的规则进行裁剪，尤其是针对国内编码标准，如GJB 8114,5369,与实际标准描述不一致，鸿渐-SAST完全按照规则描述进行研制与规范完全符合，并支持MISRA 系列，航天921，CERT系列等多个国内外著名质量及安全编码标准。

3.4 片段代码检测——无需搭建运行环境，一键式检测

多数代码分析工具需要完整的程序构建环境，测试部门很难根据不同的配置搭建构建环境，导致了很多工程检测困难。鸿渐-SAST可以在非编译通过即片段代码环境下，自动应用代码补全技术进行分析，保证了工具无须构建成功即可检测。

3.5 自主可控技术——底层研发在国内，为您量身定制各种检测服务

系统完全是自主可控，既不是国外工具改造，也不是开源工具改造，北大研发团队基于多年的科研成果，将高水平论文落地实践而成，保证了系统的延展性，可以为企业的特定需求进行改造。此外，提供了基于多种分析技术的检测器研发接口，使具有一定研发实力的用户可以自行开发检测器，为企业代码分析能力建设提供了基础设施和保障。

4 关键技术

4.1 基于多种分析技术融合的运行时缺陷检测技术

将多种分析技术融合用于海量代码，具体包括静态单赋值分析、控制依赖分析、域敏感分析、跨函数指针分析、变量修改影响分析、值依赖分析等技术，从不同角度对源代码进行准确建模。在检测运行时缺陷时，按需进行切片提取，并进行可达性分析等计算，避免了符号执行路径爆炸的问题，大幅度提高检测效率。

4.2 基于代码补全的片段代码分析技术

在程序解析、代码分析和缺陷检测阶段使用了多个启发式策略，对代码进行补全，尽可能地弥补缺失的编译信息，因此能够进行片段代码的缺陷检测。用户不需要设置编译环境来跟踪编译过程，即可直接对代码进行检测；即使在代码编译不通过的情况下，检测结果也拥有合理的精度。

4.3 基于持久化的大规模代码分析技术

代码检测完成后，进行持久化，将检测的中间数据、检测结果均存储于数据库中。支持增量分析，即当软件进行了修改时，只需要分析修改的代码影响的函数和文件，而复用未受影响部分的之前已存储的结果；若代码无修改，在下次进行检测时直接报告检测结果。因此，不仅能够进行超大规模代码的缺陷检测，而且可以迅速检测开发过程中迭代的代码。

4.4 基于深度学习的缺陷模式自动挖掘技术

基于工具积累的数万条缺陷及其对应代码，基于预先构建的依赖图自动进行缺陷模式的深度学习，发现新的缺陷模式；使用剪枝算法缩减图规模，解决大规模图匹配的效率问题。通过这种方法，能够准确识别缺陷的语义模式，丰富检测支持的缺陷模式种类。同时，随着检测数据的更新，不断扩充支持的缺陷模式。

5 客户价值

1.1 快速实现“软件基因”的修正

在研发和测试阶段采用白盒、一键式的方式，发现代码中问题，无论从缺陷发现的效率还是系统性上说都是最优的选择，在研发阶段快速检测可以让研发人员迅速消灭缺陷，防患于未然。

1.2 呈现软件研发人员的趋势、状态帮助管理者快速决策

企业对研发人员的质量量化评估一直是难题，系统可以通过代码量统计，度量以及缺陷检测评估等方式，给出项目的整体趋势、评分以及研发人员的量化评估，甚至进行研发人员“画像”以刻画出不同研发人员擅长的领域，帮助企业针对不同的项目选择不同能力的研发人员。

1.3 “一站式”自动检测服务，无缝接入基于Devops软件研发流程

通过配置定时检测任务，自动获取源代码进行检测，让开发、测试、安全人员集中精力关注检测结果，减少检测过程中繁琐的重复配置和等待时间，提高开发团队整体的工作效率。同时可以直接将检测结果提交至企业 Bug 管理系统中，方便开发人员查看和解决问题，企业无需改变原有流程，快速融入企业研发过程。

附录一：工具支持的典型缺陷和安全漏洞类型

1、支持运行时缺陷284种，其中严重程度较高86种，典型缺陷模式如下：

(1)C/C++运行时缺陷170种，其中严重程度较高的56种，比较典型的种类如下表：

使用释放后的内存 (cwe-416)	多次释放内存 (cwe-415)	变量未初始化使用 (cwe-457,cwe-665)
一般的数组越界(cwe-122,cwe-127, cwe-176,cwe-190,cwe-193,cwe-195)	由于污染数据导致的数组越界 (cwe-119,cwe-120,cwe-122, cwe-170,cwe-176,cwe-190)	结构体成员变量发生的数组越界 (cwe-121,cwe-122)
被赋值为NULL的指针发生空指针解引用(cwe-476)	函数返回局部变量的地址 (cwe-562)	与EOF比较变量类型不当 (cwe-253)
栈缓冲区溢出 (cwe-121,cwe-127,cwe-176, cwe-190,cwe-193,cwe-195)	堆缓冲区溢出 (cwe-121,cwe-127,cwe-176, cwe-190,cwe-193,cwe-195)	分配内存未释放 (cwe-401)
指针加法位数错误	指针加法位数错误 (cwe-468)	函数地址当做调用
可疑的内存泄漏 (cwe-401)	线程死锁 (cwe-833)	资源泄漏(cwe-404,cwe-770, cwe-772,cwe-773,cwe-775)
返回值类错误 (cwe-393,cwe-394)	释放未分配的内存 (cwe-590)	资源释放后继续使用 (cwe-672)
字符串差一错误 (cwe-170,cwe-193)	除零操作	

(2)Java运行时缺陷114种，其中严重程度较高的31种，比较典型的种类如下表：

在需要对象的情况下不要使用null (cwe-476)	比较类而不是类名称 (cwe-486)	不要忽略方法的返回值 (cwe-252)
不要对同一数据进行位运算和数学运算	不要返回对私有可变类成员的引用(cwe-375)	对可变输入和可变的内部组件创建防御性复制
不要使用公有静态的非final变量 (cwe-493,cwe-500)	不要调用垃圾回收函数 (cwe-586,cwe-583,cwe-568)	使用合适的访问权限创建文件 (cwe-279,cwe-276,cwe-732)
发现并处理与文件相关的错误 (cwe-22,cwe-426)	对读取一个字符或者字节的方法，使用int类型的返回值	在程序终止时执行正确的清理动作(cwe-705)
不要偏离序列化方法的正确签名 (cwe-502)	不要对实现定义的不可变因素使用默认的序列化格式(cwe-502)	不要基于高层并发对象的内置锁来实现同步
保证对于共享变量的组合操作是原子性的 (cwe-366,cwe-413,cwe-567,cwe-667)	使用线程池时，确保ThreadLocal变量可以重新初始化	

2、支持安全漏洞223种，其中严重程度较高的125种，典型漏洞模式如下：

(1)C/C++支持安全漏洞65种，其中严重程度较高的14种，比较典型的种类如下表所示：

SQL注入 (cwe-89)	命令行注入 (cwe-78)	LDAP注入 (cwe-90)
路径遍历 (cwe-36,cwe-23,cwe-22)	不可信的搜索路径 (cwe-426,cwe-20)	安全配置错误 (cwe-15)
明文传输敏感数据 (cwe-15)	在释放前未清除敏感信息 (cwe-226)	明文存储密码 (cwe-256,cwe-259)

缺少需要的加密步骤 (cwe-325)	不安全地绑定套接字	敏感信息泄露 (cwe-209,cwe-497,cwe-600)
不建议使用容易出现电子欺骗漏洞的系统调用	污染数据对象直接引用	使用RSA算法没有使用OAEP算法
使用含有已知的漏洞	进程控制 (cwe-114)	不安全加密 (cwe-326)

(2)Java支持安全漏洞158种，其中严重程度较高的111种，比较典型的种类如下表所示：

跨站脚本漏洞XSS (cwe-79,cwe-116)	跨站请求伪造CSRF (cwe-352)	LDAP注入 (cwe-90)
命令行注入 (cwe-78)	XPath注入 (cwe-643)	文件路径攻击 (cwe-22)
SQL注入 (cwe-89)	XML注入 (cwe-116)	XML外部实体攻击 (cwe-116,cwe-776)
代码注入 (cwe-94,cwe-95,cwe-116)	JSON注入	隐私违反 (cwe-226,cwe-359)
Apache Axis2系统信息泄露 (cwe-497)	资源泄露 (cwe-404)	缺少安全检查
Axis2配置错误(cwe-311,cwe-319)	struts不正确的验证方法 (cwe-103，cwe-101)	J2EE配置错误
未验证cookie的正确性和完整性 (cwe-565)	未加密的套接字 (cwe-319)	远程包含JSP动态输入文件攻击
不要向Runtime.exec()方法传递非受信、未净化的数据(cwe-78)	路径遍历 (cwe-36)	进程控制 (cwe-114)
HTTP响应截断	伪造日志 (cwe-117)	未经验证的重定向 (cwe-601)