

# 代码自动化扫描系统的建设(下)

2018-09-05 原创

上一篇文章《代码自动化扫描系统的建设(上)》主要介绍了自动扫描系统的背景和要实现的目标，这篇里我们将会详细介绍各个层与模块的设计。

## 一、系统设计

### 1.1 基础与准备

这里我们主要使用 Linux 来搭建我们的自动化扫描系统，按照设计的角色划分，我们这里需要三台 CentOS 7 的服务器，当然服务器可以是物理设备也可以是虚拟机，如果公司内部的扫描项目较多或为以后扩展考虑意见使用物理机。

服务器数量	操作系统	扫描引擎	数据库	开发语言	角色
3台	CentOS 7	SonarQube	MySQL	Python	<ul style="list-style-type: none"><li>• UI+MySQL+MQ</li><li>• 扫描节点</li><li>• 第三方引擎 (SonarQube)</li></ul>

服务器划分：

这里我们假定一个 codeaudit 域，三台服务器的主机名称分别为：

- **ui.codeaudit**: 负责后台管理系统的部署，包括数据库、MQ。
- **task.codeaudit**: 负责调度扫描引擎。
- **sonarqube.codeaudit**: SonarQube的后台服务端。

### 1.2 技术说明

这里会讨论到所需的具体技术点，有些技术或方法可能不是最佳的方案，但是已经过我们测试检验是可行的。

以下为实际开发中用到的一些技能：

- Python/Django/Jquery/Celery
- Gitlab API/Sonar API
- Git/Gitlab CI/Jenkins
- Centos 7/Shell
- NFS/Nginx/uWSGI
- MySQL/RabbitMQ/Redis
- 安全漏洞知识

#### CentOS 7

CentOS 7 与 6 的版本会有一些区别，我们需要具有 Linux 的基本操作基础，了解 `systemctl`、`firewall-cmd`、`crontab` 等命令；了解 SELinux，修改 SELinux 状态；并能编写 `systemd` 的自启动脚本。

#### Git

了解 Git 的基本操作命令, 使用 SSH 密钥的方式提交或拉取代码; 熟悉 git clone、git log、git pull、git branch、git remote、git fetch、git for-each-ref、git ls-files 等基本命令的操作。

例如:

- 使用 git for-each-ref 来得到当前分支的最后一次 commit id;
- 使用 git ls-files 来判断项目中是否存在 sonar-project.properties 配置文件;
- 使用 git log -n1 /path/file 来获取文件最后一次 commit 的作者。

## CI/CD

不论是集成到 Gitlab CI 或是 Jenkins, 我们都需要先了解项目上线的基本流程, 如: 开发的代码规范、测试环节 (单元测试/功能测试)、发布部署环节等。一般我们会将代码扫描环节放在在测试环节后, 发布部署前。

## Python

这里我们使用 Python 进行后台的服务端开发, 使用 Django 进行前台 UI 的开发, 使用 django-rest-swagger 来开发 API 接口, 使用 Celery 作为扫描任务的调度框架。

## 数据库与中间件

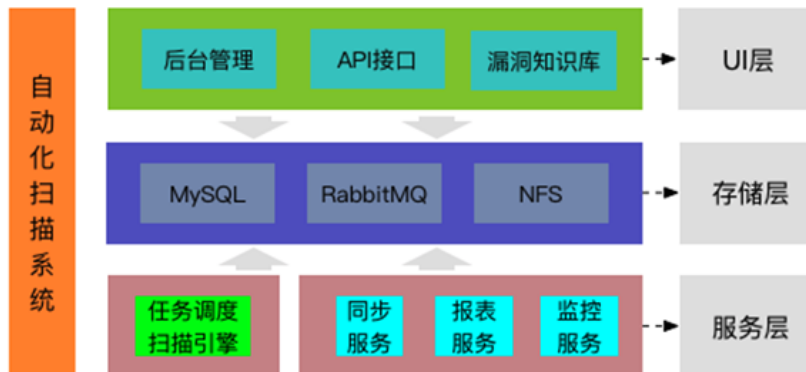
数据库我们选择使用 MySQL 5.7(或MariaDB, 他们在使用上没有太大的区别); Celery 的消息中间件可以使用 Redis 或是 RabbitMQ, 这里你可以在开发的时候使用 Redis, 正式部署时使用 RabbitMQ。

## 安全漏洞知识

- 了解 OWASP TOP 10 的漏洞类型原理与解决方案;
- 了解 CWE 的漏洞信息;
- 了解公司主流的开发语言。

## 1.3 模块设计

下图中我们自上而下按照逻辑大致划分了“四”层: UI层、存储层、服务层、任务调度扫描引擎层 (由于任务调度与服务同在后台运行, 所以又统称为服务层)。



### 1.3.1 UI 层

提供扫描系统的后台管理、API接口、漏洞知识库等一系列的交互功能入口, 不同的人员或系统可以根据各自的需求通过不同的交互接口来满足自己需求。如: CI/CD系统可通过 API 接口创建扫描任务并获取扫描结果; 安全审计人员可通过后台进行规则或插件的添加; 开发人员可通过漏洞知识库来获取相关语言或技术的漏洞信息。

### 1.3.2 存储层

主要包括关系型数据库、消息中间件(指MQ)、NFS(网络文件系统), 这里我们使用了 MySQL 5.7 的数据库; RabbitMQ 是作为 Celery 调度框架的消息中间件; NFS担当网络共享存储, 用于存储代码与扫描日志。

### 1.3.3 调度层

扫描任务的执行流程，主要可分为：

- **初始化**：扫描任务的环境初始化，如：日志目录、日志文件、加载插件、加载漏洞规则等；
- **分析项目**：项目代码统计、依赖组件统计、漏洞知识库关联等；
- **扫描漏洞**：调用第三方扫描引擎、统计扫描结果；
- **漏报处理**：使用黑名单规则和插件进行扫描；
- **误报处理**：使用白名单规则和插件进行误报处理；
- **闭环漏洞**：针对高危漏洞在 GitLab 或 Jira 系统中创建一个 Issue。

### 1.3.4 服务层

后台的服务，其主要包括：GitLab 系统中的项目同步、报表生成、调度进程监控。

## 二 系统功能

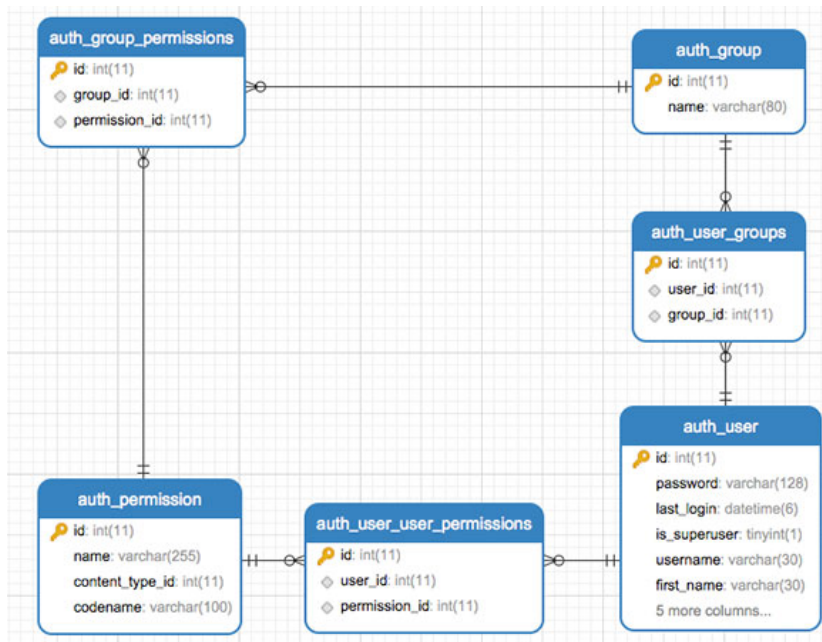
### 2.1 数据库设计

#### 2.1.1 权限相关

权限控制，这里使用 django 自带的权限表来进行权限控制，我们可以通过 `auth_group` 表来创建用户组，为不同的用户组赋予不同的角色权限 `auth_group_permissions`，你可以访问官方地址：<https://docs.djangoproject.com/en/2.1/topics/auth/default/#topic-authorization> 来获得更多关于权限的信息。

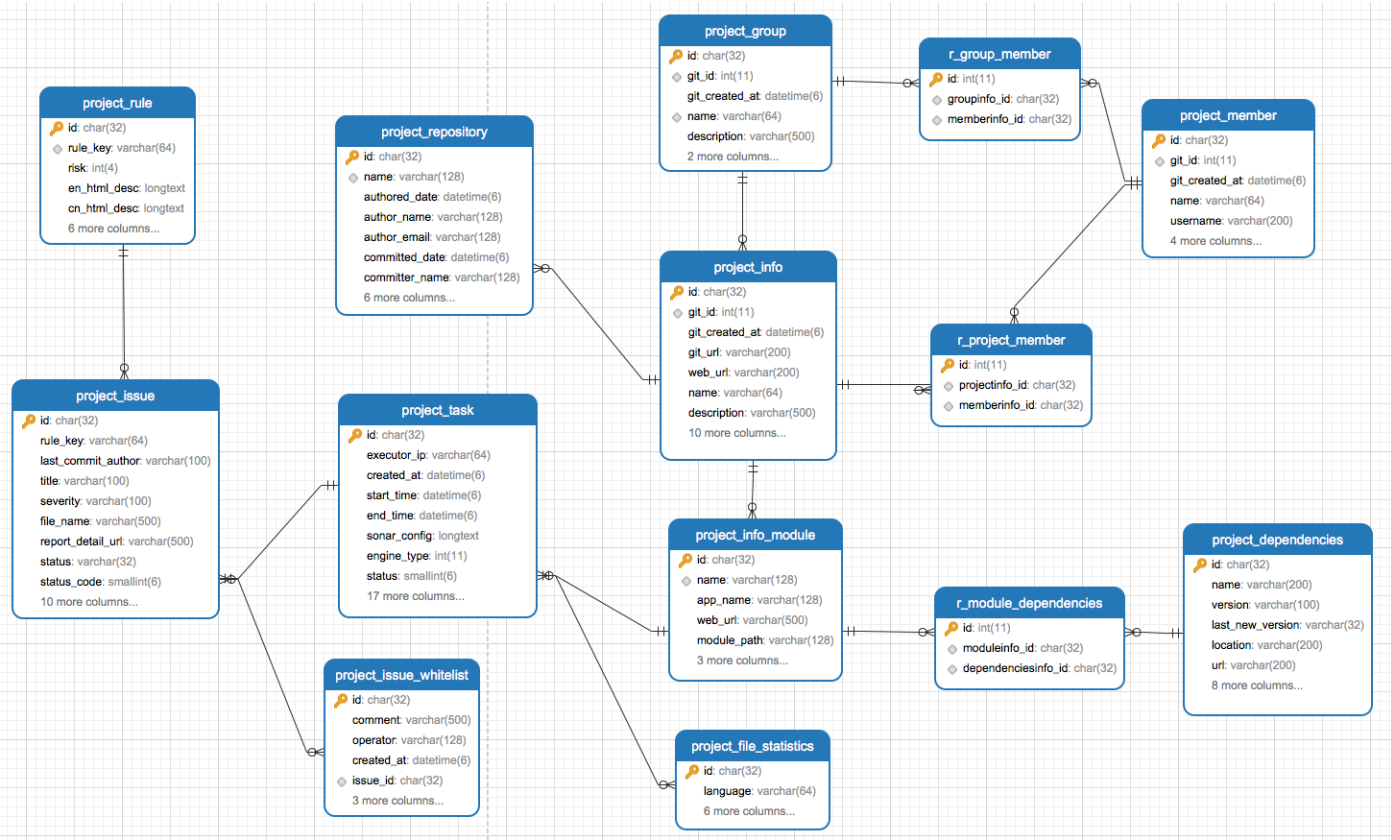
django 权限表如下：

- `auth_group`
- `auth_group_permissions`
- `auth_permission`
- `auth_user`
- `auth_user_groups`
- `auth_user_user_permissions`



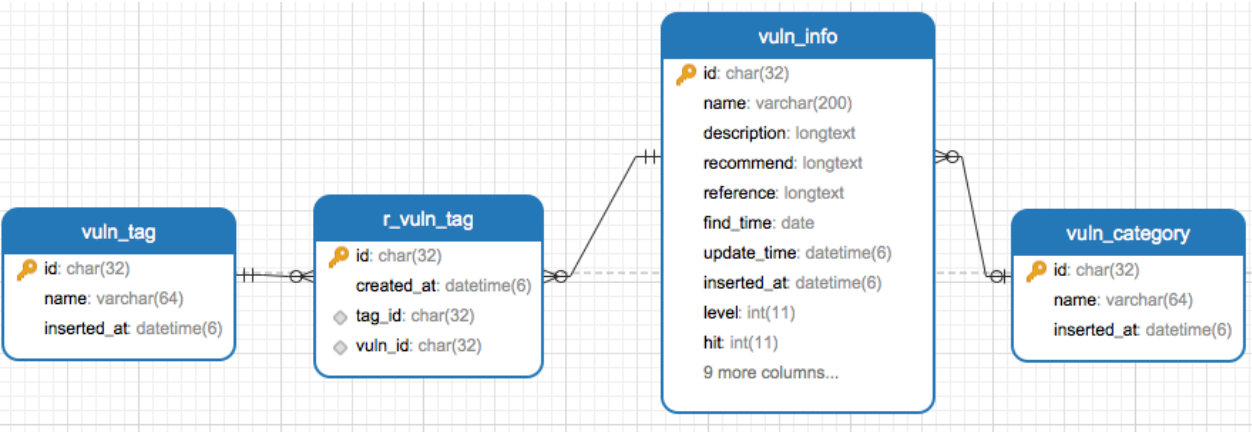
#### 2.1.2 项目相关

项目表主要包括：项目组、项目、分支与TAG、统计信息、依赖组件、插件规则、扫描任务等相关表。



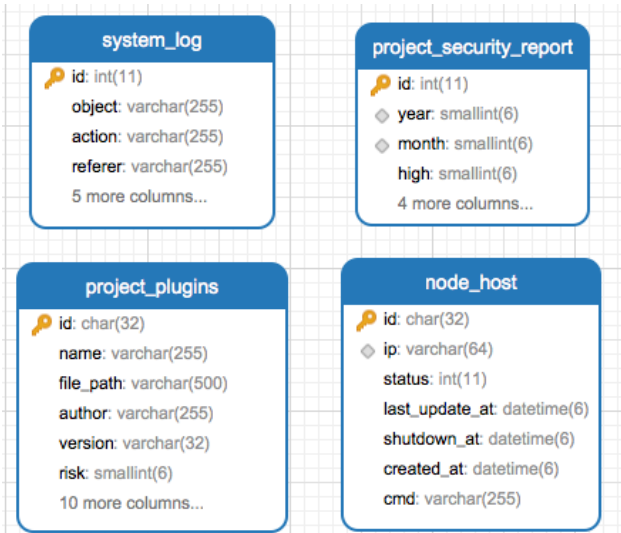
2.1.3 漏洞知识库

漏洞知识库，这里主要存储漏洞类型、漏洞知识等内容。



2.1.4 系统相关

系统表主要包括系统的安全周报、节点监控、系统日志等信息。



## 2.2 UI系统

扫描系统的后台，方便安全审计人员管理项目和系统。

### 2.2.1 项目管理

#### 2.2.1.1 项目组

项目组我们通过 GitLab 的 API 同步所有项目组信息到我们的扫描系统，项目组的信息包括：项目组名称、项目组描述、创建时间、URL地址、项目成员等。

#### 2.2.1.2 项目

项目是从分组中获取得到，需要注意的是可能会存在项目名相同但分组不同的情况。项目基本信息应包括：项目名称、项目描述、所属分组、默认分支、Git地址、项目成员、代码统计、依赖组件、分支管理、TAG管理等。

基本信息	扫描信息	依赖组件(33)	项目成员(14)	分支管理(16)	TAGS(0)
扫描时间: 2018-08-27 10:42:18					
项目路径: <span style="background-color: #f0f0f0; padding: 2px;">[redacted]</span> n-api					
扫描报告: <a href="http://[redacted]ean-api">http://[redacted]ean-api</a>					
漏洞统计 <span style="background-color: #f08080; padding: 2px;">高危漏洞: 7</span> <span style="background-color: #ffa500; padding: 2px;">中危漏洞: 0</span> <span style="background-color: #add8e6; padding: 2px;">低危漏洞: 0</span> <span style="background-color: #90ee90; padding: 2px;">信息数量: 73</span>					
语言	文件数	代码行数	空白行数	注释行数	
Java	653	53845	11961	16814	
XML	71	9216	448	152	
YAML	4	267	28	105	

#### 2.2.1.3 扫描任务

扫描任务会有四种状态：等待调度、正在扫描、扫描完成、扫描失败。每一次创建扫描任务时，都会查询是否存在等待调度或正在扫描的任务，如果存在则提示创建失败。

## 所有任务

请输入项目名称

所有状态

所有引擎

开始

<input type="checkbox"/>	项目名称	模块名称	执行地址	开始时间	创建时间	当前状态
<input type="checkbox"/>	wowfe	wowfe	157	2018-08-27 13:00:52	2018-08-27 13:00:52	正在扫描
<input type="checkbox"/>	statistics	statistics.j	158	2018-08-27 13:00:42	2018-08-27 13:00:42	正在扫描
<input type="checkbox"/>	app-cont-web	.cont.j	157	2018-08-27 13:00:24	2018-08-27 13:00:24	正在扫描

## 2.2.2 规则插件

## 2.2.2.1 规则

这里使用正则表达式来做特征匹配，并可通过限定文件的后缀来提高精准度。

正则表达式标志位：

- 忽略大小写
- 支持多行匹配

添加规则黑名单

正则名称

描述

表达式

包含的文件后缀

\*.html,\*.js,\*.cs,\*.java,\*.py

知识库

[component:com.alibaba.fastjson] fastjson 远程代码#

标志

忽略大小写

启用

☒

创建

## 2.2.2.2 插件

这里使用了 Python 的反射机制，任务初始化时会优先初始化插件，当扫描完成时，扫描引擎会使用插件批量进行检测。插件入口函数为 `run()`，漏洞详情对象会作为 `**kwargs` 参数的上下文传到该函数中。

pass\_file\_dir.py 1.45 KB

```
1  # coding: utf-8
2
3  import re
4  import os
5
6  from .data import logger
7  from .issue import process_false_positives
8
9
10 _NAME_ = '测试文件与测试目录跳过扫描'
11 _AUTHOR_ = '
12 _VERSION_ = '0.2'
13 _RISK_ = 1
14 _KBID_ = ''
15
16
17 REGEX_CHECK_LIST = [
18     re.compile('/test/', re.I), # case: modules/manager-base/src/test/java/
19     re.compile('^test/', re.I), # case: test/test_utils.py
20     re.compile('^test', re.I), # case: api/test_utils.py
21 ]
22
23
24 def run(**kwargs):
25     """
26     入口函数
27     :param kwargs:
```

2.2.2.3 规则知识库

规则知识库是区别与漏洞知识库的，往往规则知识库的内容要比漏洞知识库的内容简单，但是结构清晰。如：漏洞示例代码、漏洞说明、解决办法、参考链接等信息。

<input type="checkbox"/>	语言	类型	标题	标签	引擎	操作
<input type="checkbox"/>	组件	component:com.alibaba.fastjson	fastjson 远程代码执行漏洞		自定义	编辑 查看项目
<input type="checkbox"/>	组件	component:org.codehaus.plexus	codehaus/plexus-archiver zip slip漏洞(CVE-2018-1002200)	CVE-2018-1002200	自定义	编辑 查看项目
<input type="checkbox"/>	config	config:hardcode	Gitlab 中配置文件敏感信息泄漏		自定义	编辑 查看项目
<input type="checkbox"/>	组件	component:org.springframework	Spring Framework 远程代码执行(CVE-2018-1270)	CVE-2018-1270	自定义	编辑 查看项目
<input type="checkbox"/>	python	python:os.system	系统命令执行	命令执行	自定义	编辑 查看项目
<input type="checkbox"/>	js	javascript:S2819	限制跨文档消息传递域	html5,owasp-a3	SONAR	编辑 查看项目
<input type="checkbox"/>	java	squid:S3355	应该使用定义的过滤器	injection,owasp-a1	SONAR	编辑 查看项目

2.2.3 漏洞知识库

2.2.3.1 漏洞类型

这里建议使用 CWE 的漏洞标准，可参考这个文档：<https://www.hackerone.com/sites/default/files/2017-03/WeaknessAndLegacyVulnerabilityTypeRelationship.pdf>

Legacy	New	
Vulnerability Type	Weakness	Reference
Authentication	Improper Authentication - Generic	CWE-287
Command Injection	Command Injection - Generic	CWE-77
Cross-Site Request Forgery (CSRF)	Cross-Site Request Forgery (CSRF)	CWE-352
Cross-Site Scripting (XSS)	Cross-site Request Forgery (XSS) - Generic	CWE-79
Cryptographic Issue	Cryptographic Issues - Generic	CWE-310
Denial of Service	Denial of Service	CWE-400
Design Issue	Violation of Secure Design Principles	CWE-657
HTTP Response Splitting	HTTP Response Splitting	CWE-113
Information Disclosure	Information Disclosure	CWE-200
Memory Corruption	Memory Corruption - Generic	CWE-119
Missing Best Practice	Violation of Secure Design Principles	CWE-657
None Applicable	-	-
Privilege Escalation	Privilege Escalation	CAPEC-233
Remote Code Execution	Code Injection	CWE-94
SQL Injection	SQL Injection	CWE-89
Server-Side Request Forgery (SSRF)	Server-Side Request Forgery (SSRF)	CWE-918
UI Redressing (Clickjacking)	UI Redressing (Clickjacking)	CAPEC-103
Unvalidated / Open Redirect	Open Redirect	CWE-601
XML External Entities (XXE)	XML External Entities (XXE)	CWE-611

2.2.3.2 漏洞管理

主要包括添加漏洞和管理漏洞，漏洞的信息应该包括：CVE/CNVD/CNNVD编号、漏洞标题、风险等级、漏洞来源、发现时间、受影响范围、漏洞详情、漏洞类型、解决版本等基本信息。

基本信息

漏洞描述

规则匹配

漏洞名称:

漏洞名称

相关编号:

CVE 编号

CNVD 编号

CNNVD 编号

漏洞等级:

0 - 信息

来源:

seebug.org

漏洞类型:

代码执行

影响版本:

影响版本, 如: v1.2.0

添加

发布时间:

27/08/2018

TAG:

提交

这里我们要实现漏洞知识库与识别出的组件联动功能，主要通过两个属性：

- 组件标签



这里需要为每个漏洞添加一个 Tag 属性，其属性值如：org.springframework、com.alibaba.fastjson，建议标签一律使用小写字母。

• 版本规则

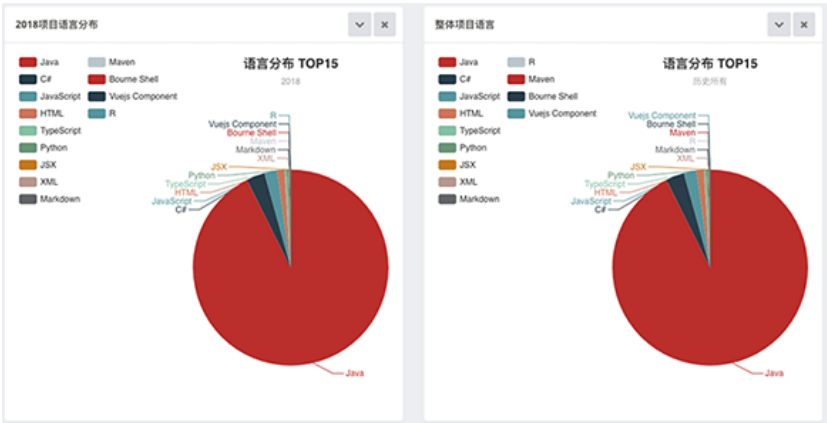
使用正则表达式来进行匹配，如：CVE-2018-1270 中受影响的 Spring Framework 版本为：5.0.x-5.0.5 和 4.3.x-4.3.16，那么我们的规则可以写成如下：

1	5\\.0 ### 5.0
2	(5\\.0\\.([0-4]){1}) ### 5.0.x -5.0.5
3	(4\\.3\\.1[0-5]{1}) ### 4.3.1x.release
4	(4\\.3\\.([0-9]){1}\\. ) ### 4.3.x.release

2.2.4 报表管理

2.2.4.1 语言与项目统计

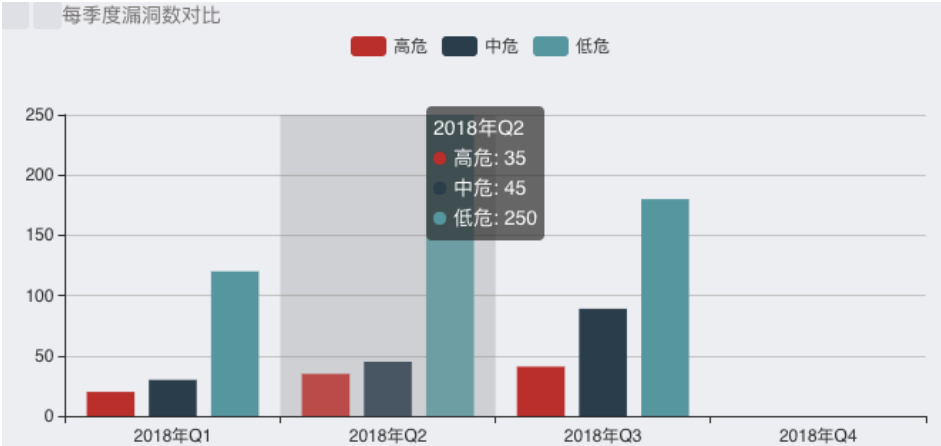
按照年份进行项目的语言统计。



2.2.4.2 周期性漏洞统计

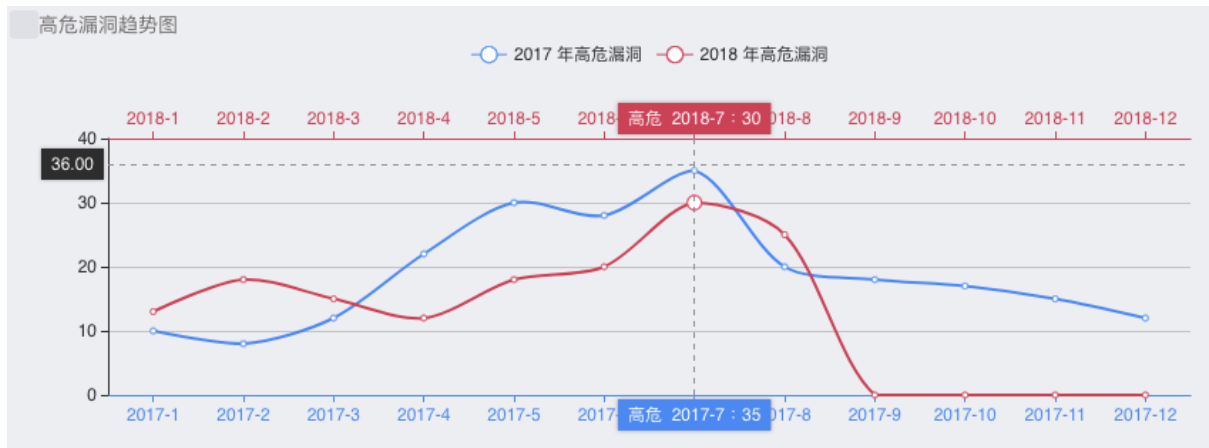
每季度漏洞数对比

季度统计是为了对比同一段时期的漏洞数。



高危漏洞趋势图

高危漏洞环比，今年实施的安全政策是否合乎预期，可以大概分析出来。



## 2.3 API接口

### 2.3.1 接口认证

使用 `rest_framework` 的 API 来做验证, 首先根据登陆的用户 id 生成一个 Token。

```

1  from rest_framework.auth_token.models import Token
2
3  def create_token(request, user_id=None):
4      if request.user.id != int(user_id):
5          return HttpResponseRedirect("/error/403")
6      try:
7          user = User.objects.get(id=user_id)
8      except Token.DoesNotExist:
9          token = Token.objects.create(user=user)
10
11     return HttpResponseRedirect("/users/{0}".format(user_id))

```

验证接口使用说明, 添加 Authorization 的认证 Token。

### Access Token

需要添加 **Authorization** 认证头, 格式如下:

```

{
  "Authorization": "Token e6f307b1a362b6b55e00098aa17d3733d9004498"
}

```

curl 测试如下

```

curl -X GET --header 'Accept: application/json' \
  --header 'Authorization: Token e6f307b1a362b6b55e00098aa17d3733d9004498' \
  'http://127.0.0.1:8000/api/v1/project/c3003c859e11403fa9c6371441298e13'

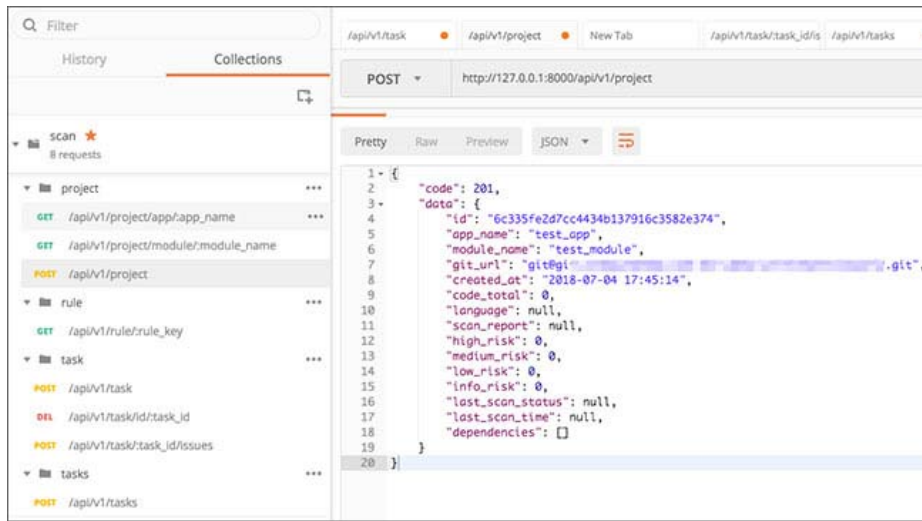
```

### 2.3.2 项目信息接口

#### 信息同步

为什么需要信息同步? 这是因为 GitLab 中的项目名称可能不是最终上线的 APP 名称(这里有些绕)。拿一个 Java 的项目举例, 该项目的 GitLab 地址为: `http://git.companyname.com/A/cloud`, 那么这个 Java 的包名有可能是 `com.companyname.cloud`。

我们使用项目的 git 地址来同步信息, 建议把 git 地址全部转换为小写。APP 的名称(包名)可以 CI/CD 系统获取或是通过配置文件的硬编码方式来定义。



## 创建扫描

信息同步完成后，我们就可以根据 APP 名称和 git 地址来创建一个扫描任务，请求参数参考如下：

- **app\_name**: APP名称(可选)
- **module\_name**: 模块名称(可选)
- **version**: 当前版本(可选)
- **git\_url**: git地址 (必选)
- **branch\_name**: 分支名称(必选)

## 2.3.3 任务信息接口

### 查询扫描任务

根据项目的 git 地址、分支来查询扫描任务，你也可以根据上一步创建扫描任务的 ID 来查询扫描结果。

### 查询任务漏洞列表

当扫描任务状态为 扫描完成/扫描失败 时，就可以根据任务 ID 来查询扫描出的安全漏洞信息。

## 2.3.4 漏洞规则接口

### 查询漏洞规则知识

通过漏洞信息中的漏洞规则 ID 或者 Key 来查询相关的规则知识库，该知识库应当包括：漏洞原因、漏洞示例代码、解决修复意见等。

## 2.4 后台服务

### 2.4.1 gitlab 的信息同步

使用 crontab 每两个小时遍历一遍 GitLab 上的所有项目，并同步项目信息到扫描系统中。

```
git:(develop) * python cli.py --sync-all --sync-threads 10

Autohome CodeAuditSystem/0.2.0-20180706

[10:24:24] [INFO] Start syncing gitlab project...
[10:24:25] [INFO] update project:
[10:24:25] [INFO] update project:
[10:24:25] [INFO] update project:
[10:24:26] [INFO] add repository:
[10:24:26] [INFO] update project:
[10:24:26] [INFO] add repository:
[10:24:26] [INFO] update project:
[10:24:26] [INFO] update project:
[10:24:26] [INFO] add repository:master, type:Branch
[10:24:26] [INFO] add repository:master, type:Branch
[10:24:26] [INFO] add repository:master, type:Branch
[10:24:26] [INFO] update project:net info, git id:3501, group name: CMD8, branch:ma
```

## 2.4.2 报表生成服务

使用 crontab 每日凌晨12点生成，季度对比和年度的安全统计数据。

## 2.4.3 扫描进程监控

使用 `ps aux | grep codescan` 来查看进程是否存活，当然这种暴力方式不能检测到进程的业务健康度的(比如：扫描任务卡死，状态一直为：正在扫描)。

# 2.5 SonarQube 搭建

## 2.5.1 服务搭建

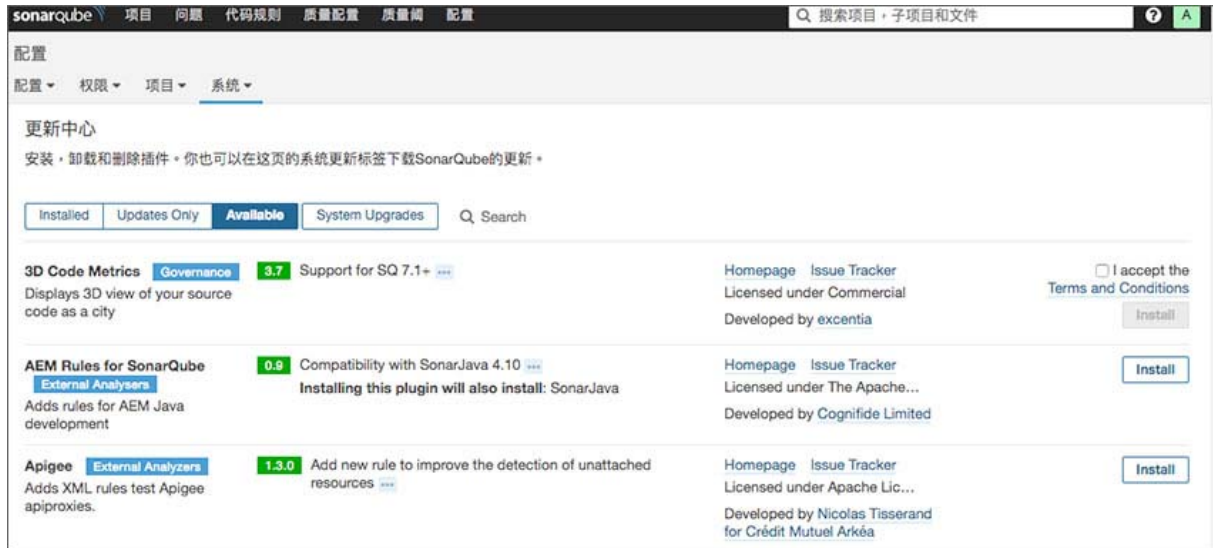
下载最新版本 <https://www.sonarqube.org/downloads/> 上传到 sonarqube.codeaudit 服务器上并解压。进入到 `bin/linux-x86-64/` 目录下，执行 `sh ./sonar.sh start`。SonarQube 启动成功后，使用浏览器打开 `http://192.168.10.3:9000`，输入 `admin/admin` 即可正常访问。

## 2.5.2 插件管理

SonarQube 6.4 版本登陆的后台管理系统，选择“配置”->“系统”->“更新中心”，选择对应插件点击“Install”进行安装。

SonarQube 7.3 版本，“Administration”->“Marketplace”，选择对应插件点击“Install”进行安装。

SonarQube 6.4 截图



## 2.6 引擎调度

程序部署在“task.codeaudit”服务器上，服务需要安装 cloc 与 sonar-scanner 工具。

### 2.6.1 代码同步

同步代码分为以下几个步骤：

#### 克隆项目

这里可能会遇到一些坑，比如项目历史比较久远，完整克隆下来可能会达到上百M或G，我们这里可以使用 --depth 1 参数进行克隆下载。有的项目可能会存在不规范的情况，比如拿 git 当 svn 使用，每个版本创建一个目录。

#### 切换分支

根据扫描任务中的分支名称 checkout 到指定分支。

#### 更新代码

针对已经克隆的项目进行 pull 操作，来同步 GitLab 上的项目更新代码。

### 2.6.2 sonar-scanner 扫描

ssh 登录到 task.codeaudit 服务器上，执行 `cd /opt && wget https://sonarsource.bintray.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-3.2.0.1227-linux.zip && unzip sonar-scanner-cli-3.2.0.1227-linux.zip` 来下载并解压，执行成功后使用 `ln -s /opt/sonar-scanner-3.2.0.1227-linux/bin/sonar-scanner /usr/bin/sonar-scanner` 命令创建一个 sonar-scanner 的软连接。这里我们会使用 sonar-scanner 命令来进行项目的代码扫描。

你也可以通过 <https://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube+Scanner> 来下载不同平台的 sonar-scanner-cli-3.2.0.1227-linux.zip。

[页面](#) / [Analyzing Source Code](#)

## Analyzing with SonarQube Scanner

由 OLD - Evgeny Mandrikov 创建, 最终由 Antoine Vigneau 修改于 七月 03, 2018

By [SonarSource](#) – GNU LGPL 3 – [Issue Tracker](#) – [Sources](#)

**Download SonarQube Scanner 3.2**  
Compatible with SonarQube 5.6+ (LTS)  
[Linux 64 bit](#)      [Windows 64 bit](#)      [Mac OS X 64 bit](#)      [Any\\*](#)

\*This package expects that a JVM is already installed on the system - with same Java requirements as the SonarQube server.

**Table of Contents**

- [Features](#)
- [Installation](#)
- [Use](#)
- [Troubleshooting](#)
- [Going Further](#)

### 2.6.3 代码统计

使用 `cloc` 工具进行文件与代码行数的统计, 这里你可能需要通过 `--exclude-ext`、`--exclude-dir` 参数来过滤一些无意义的文件, 比如: 字体、图片、声音、视频等。举个例子, 过滤所有图片后统计: `cloc ./目标路径 --exclude-ext=.jpg,.jpeg,.png,.bmp,.gif,.ico`。

### 2.6.4 项目组件分析

组件分析主要是针对如使用 Java 语言开发项目时使用 Maven 管理的 `pom.xml` 配置文件; Python 中的 `requirements.txt` 文件; JS 项目中的 `package.json` 文件做解析。这里我写了一个 `clocwalk` 工具可以分析项目的依赖组件, 这个项目目前已经开源, 你可以通过 <https://github.com/MyKings/clocwalk> 地址来获取这个工具。

### 2.6.5 漏报处理

关于漏报问题, 你可以根据自己企业 SRC 中的漏洞, 总结出一套适合自己企业的黑名单规则; 或者你可以添加一些 CWE 的漏洞规则, 关于 CWE 的信息你可以访问这个地址 <https://cwe.mitre.org/data/index.html>。

### 2.6.6 误报处理

关于误报问题可能会较多, 比如扫描出单元测试或功能测试的硬编码问题; 比如变量参数 `String PARAM_NAME_PASSWORD="passwd_txt";` 问题。

以上的问题我们可以通过白名单插件处理, 比如插件中对文件路径和方法判断是否存在 `test` 关键字, 如果存在我们就认为这个是误报。另外针对某些特殊类型的误报, 比如在 A 项目下才是误报, 其他项目就是漏洞的情况, 我们可以设置这个项目的白名单漏洞 Case, 其匹配规则条件为: 项目名称、漏洞文件、漏洞类型、漏洞所在行, 当所有条件都同时满足时, 那么这个漏洞就可以判断为误报。

### 2.6.7 漏洞闭环

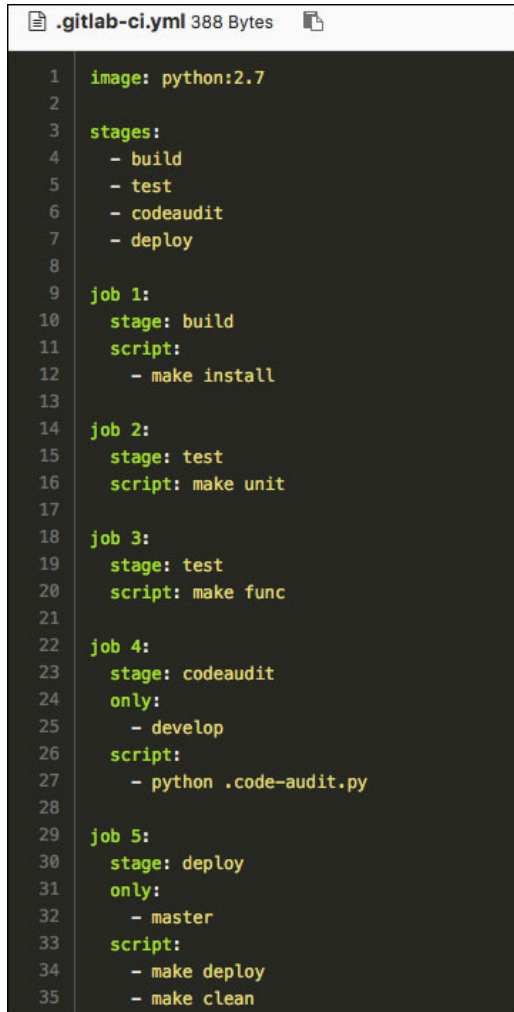
当一个高/中危漏洞被发现并确认时, 我们应该如何跟踪这个漏洞的生命周期? 往往安全人员会将一个漏洞提交到内部的 SOC 系统中, 由于 SOC 系统没有和项目开发的流程控制系统 (如: Jira) 没有直接联系, 开发人员可能会忽视或忘记修复这个高危漏洞, 如何避免这种情况? 我们这里以 GitLab 举例, 当扫描系统扫描出高危漏洞时, 系统会通过 GitLab 的 `POST /projects/:id/issues` API 接口来自动创建一条 issue 并指派给当前项目的 master, 项目负责人同时会收到一条邮件提醒, 那么项目负责人可根据漏洞严重程度来安排项目的迭代计划, 这样我们就把审计系统扫描出的漏洞与项目开发流程很好的结合起来了。

## 2.7 GitLab CI 触发

当然也可以使用 Jenkins 来做 CI/CD 系统。我们这里开发了一个 `.code-audit.py` 触发脚本, Jenkins 你也可以使用 Python 脚本或是开发 Jenkins 插件来达到触发目的。

### 2.7.1 配置项目

这里需要了解 `.gitlab-ci.yml` 文件格式的编写，下面是一个 Python 项目的配置。可以看出整个 CI 过程分为 4 个阶段：build、test、codeaudit、deploy。其中 codeaudit 是我们的代码扫描阶段，这里我们限制了只有 develop 的动作才会触发扫描。



```
1 image: python:2.7
2
3 stages:
4   - build
5   - test
6   - codeaudit
7   - deploy
8
9 job 1:
10  stage: build
11  script:
12    - make install
13
14 job 2:
15  stage: test
16  script: make unit
17
18 job 3:
19  stage: test
20  script: make func
21
22 job 4:
23  stage: codeaudit
24  only:
25    - develop
26  script:
27    - python .code-audit.py
28
29 job 5:
30  stage: deploy
31  only:
32    - master
33  script:
34    - make deploy
35    - make clean
```

## 2.7.2 扫描脚本

触发扫描脚本如下图，其大体的执行流程如下：

- 获取 GitLab CI 中关于项目的环境变量信息；
- 设定要拦截的漏洞级别，默认：中、高危漏洞不通过测试；
- 同步项目信息到扫描系统，如果失败扫描代码不通过；
- 创建扫描任务，如果失败扫描代码不通过；
- 异步查询扫描结果，超时时间10分钟，如果超时扫描代码不通过；
- 扫描结果完成，统计是否存在预定义级别的漏洞，如果存在扫描代码不通过。

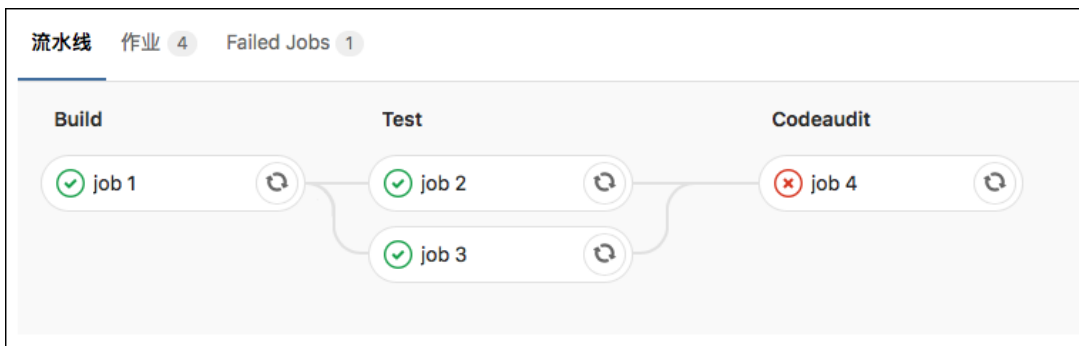


```

3
4 import json
5 import sys
6 import time
7 import urllib2
8 import os
9
10 APP_NAME = os.environ.get('CI_PROJECT_NAME')
11 MODULE_NAME = os.environ.get('CI_PROJECT_NAME')
12 CODEAUDIT_ENGINE_TOKEN = 'dc...'
13 BRANCH_NAME = os.environ.get('CI_BUILD_REF_SLUG')
14 GIT_URL = os.environ.get('CI_PROJECT_URL')
15 VERSION = os.environ.get('CI_COMMIT_SHA')
16
17 RISK_LISTS = [
18     "high_risk",
19     "medium_risk"
20 ]
21
22 sync_project_url = 'http://...com/api/v1/project'
23 make_scan_url = 'http://.../api/v1/task'
24 get_scan_url = 'http://...api/v1/task/{0}'
25 get_vuln_url = 'http://...api/v1/task/{0}/issues'
26
27
28 def _make_request(url, data=None, timeout=5, is_trace=True):
29     """..."""
30     try:
31         if is_trace:
32             sys.stdout.write('* * 80)
33             sys.stdout.write('\nStart visiting "{0}", post data: {1}\n'.format(url, data))
34             sys.stdout.write('-'*80)
35         req = urllib2.Request(url)
36         req.add_header('Content-Type', 'application/json')
37         req.add_header('Authorization', 'Token {0}'.format(CODEAUDIT_ENGINE_TOKEN))
38         if data:
39             resp = urllib2.urlopen(req, data=json.dumps(data), timeout=timeout)
40         else:
41             resp = urllib2.urlopen(req, timeout=timeout)
42             result = resp.read()
43         if is_trace:
44             sys.stdout.write('\nReturn content: {0}\n'.format(result))
45             sys.stdout.write('* * 80)
46         return result
47     except Exception as ex:
48         raise ex
49
50
51 def scan():...
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116 if __name__ == '__main__':
117     scan()

```

下图为整个 CI 过程的截图。



下图为代码扫描失败的反馈结果，图中可以看出发现了一个漏洞。

```

...
08:59:56", "app_name": "code-audit", "branch": "develop", "module_name": "code-audit", "version": ""}}
*****

[+]Vulnerabilities found: 1

=====
标题: 使用不安全的eval()方法, 漏洞文件: "acweb/static/js/json2.js", 最后一次提交作者: z...n>, 详情地址: http://...
tatic/js/json2.js&line=515
=====
ERROR: Job failed: exit code 1

```

### 三 总结



关于“自动代码审计系统的建设”文章这里就此完结了。下篇中有些章节可能说的比较笼统宽泛，但是要对每一个章节详详细细的说明，恐怕每一个章节都会写下不止一篇文章了，本篇文章只是为大家提供一种思路，具体实施效果还是要靠大家自己来实践总结，就这样吧：)

## 四 参考链接

- [https://linuxtools-rst.readthedocs.io/zh\\_CN/latest/tool/crontab.html](https://linuxtools-rst.readthedocs.io/zh_CN/latest/tool/crontab.html)
- <https://git-scm.com/docs>
- <https://docs.gitlab.com/ee/api/>
- <https://about.gitlab.com/features/gitlab-ci-cd/>
- <https://docs.gitlab.com/ce/ci/yaml/README.html>
- <https://docs.gitlab.com/ce/ci/examples/README.html>
- <https://docs.gitlab.com/ee/api/issues.html>
- <https://docs.gitlab.com/runner/install/docker.html>
- <https://docs.sonarqube.org/display/DEV/Web+API>
- <https://docs.djangoproject.com/en/2.1/topics/auth/default/#topic-authorization>
- <https://www.sonarqube.org/downloads/>
- <https://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube+Scanner>
- <https://www.hackerone.com/sites/default/files/2017-03/WeaknessAndLegacyVulnerabilityTypeRelationship.pdf>
- <https://github.com/MyKings/clocwalk>



### Post Directory

#### 文章目录

##### 一、系统设计

- 1.1 基础与准备
- 1.2 技术说明
- 1.3 模块设计
  - 1.3.1 UI 层
  - 1.3.2 存储层
  - 1.3.3 调度层
  - 1.3.4 服务层

##### 二 系统功能

- 2.1 数据库设计
  - 2.1.1 权限相关
  - 2.1.2 项目相关
  - 2.1.3 漏洞知识库
  - 2.1.4 系统相关
- 2.2 UI系统
  - 2.2.1 项目管理
    - 2.2.1.1 项目组
    - 2.2.1.2 项目
    - 2.2.1.3 扫描任务
  - 2.2.2 规则插件
    - 2.2.2.1 规则
    - 2.2.2.2 插件
    - 2.2.2.3 规则知识库
  - 2.2.3 漏洞知识库
    - 2.2.3.1 漏洞类型
    - 2.2.3.2 漏洞管理
  - 2.2.4 报表管理

2.2.4.1 语言与项目统计

2.2.4.2 周期性漏洞统计

## 2.3 API接口

2.3.1 接口认证

2.3.2 项目信息接口

2.3.3 任务信息接口

2.3.4 漏洞规则接口

## 2.4 后台服务

2.4.1 gitlab 的信息同步

2.4.2 报表生成服务

2.4.3 扫描进程监控

## 2.5 SonarQube 搭建

2.5.1 服务搭建

2.5.2 插件管理

## 2.6 引擎调度

2.6.1 代码同步

2.6.2 sonar-scanner 扫描

2.6.3 代码统计

2.6.4 项目组件分析

2.6.5 漏报处理

2.6.6 误报处理

2.6.7 漏洞闭环

## 2.7 GitLab CI 触发

2.7.1 配置项目

2.7.2 扫描脚本

## 三 总结

## 四 参考链接

