代码自动化审计系统的建设(上)

2018-08-07 原创

代码审计一直是企业白盒测试的重要一环，面对市场上众多商业与开源的审计工具，你是否想过集众家之所长来搭建一套自动化的扫描系统呢？也许这篇文章会给你一些思路：）

一、背景

1. 为什么需要自动化扫描?

互联网的快速发展，程序员是功不可没的。从软件开发的瀑布模型到现在的敏捷开发, 软件的开发周期从数年到数月、从数月到数天，时间不断变换缩减。传统的代码扫描方式已经不能跟进新时代的软件开发流程中，这就需要改变我们的代码扫描方式，它应该在有限的时间内尽量发现足够多的安全问题，并能够结合 CI (持续集成) 来触发代码扫描。

2. 自动化扫描时扫描引擎用什么?

你可以使用任何提供 API 接口的开源或商业的扫描引擎。这里我们把 扫描引擎 (指第三方的审计工具)与 自动化系统 剥离(解耦)开来， 扫描引擎 只负责安全漏洞扫描； 自动化系统 负责漏洞收集、分析、处理等动作。同时你也可以通过 自动化系统 的后台来添加一些安全规则。

3. 怎么触发自动化扫描?

两种方式：

- 管理后台手动触发
- CI(持续集成)/CD(持续交付) 系统中触发

4. 如何解决漏报和误报?

两种方式： 基于规则 和 基于插件 ，这里使用白名单表示误报、使用黑名单表示漏报。

基于规则

这里的规则是指基于文本的处理方式，如：使用正则表达式来匹配文件中的某些特征，使用字符串来判断是否存在敏感信息等。

PS: 这里有一些细节需要注意。

a. 正则表达式：

- 是否区分大小写
- 是否支持多行匹配

b. 字符串：

- 是否首部包含
- 是否尾部包含
- 是否在当前字符串中

基于插件

插件的自由度较大，其本质是把要扫描的文件信息作为插件执行入口的上下文，文件信息包括：路径、名称、内容等。你也可以在插件中写一些判断逻辑或调用第三方工具。

5. 漏报率和误报率怎么样?

漏报率暂时无法很好的测量， **漏报率 = 漏报数/(漏洞数+漏报数) × 100%** 而实际中很少有人(这里指非安全审计人员)会发现漏报，理论上你写的规则或插件越多越会减少漏报。

误报是可以通过白名单规则或插件处理的，理论上可以百分百消除误报，但是需要手动进行设置。

6. Web 通用型漏洞可以都覆盖么?

Web 通用型漏洞以 OWASP 2017 TOP 10 为例，其大多数都在黑盒测试的范围内。适用白盒测试仅限于： `A3:2017-Sensitive Data Exposure` 、 `A9:2017-Using Components with Known Vulnerabilities` ，而这两项也是我们自动化扫描系统的基本要求。

7. 扫描出来的漏洞如何形成闭合?

这需要结合公司自身的软件开发方式而定，大多数公司采用 `Gitlab + Confluence + Jira + Jenkins` 的工作方式，那么我们可以通过 Jira 或 Gitlab 的 API 接口来创建问题工单。

二、设计要求

1. 目标

系统功能:

- 尽量发现足够多的安全问题
    - 硬编码问题
    - 敏感信息泄露
    - 使用存在已知漏洞的组件
    - 危险函数识别
- 可集成第三方扫描引擎
- 可自动化处理误报
- 可通过 CI 方式触发扫描
- 可根据条件自动创建 Issue

扫描目标主要分为两种：一种为线上 Git 扫描；一种为离线扫描。 `线上 Git 扫描` 其主要应用场景为企业内部使用如 Gitlab 这种代码托管系统，我们定时同步 Gitlab 上的项目信息，通过 CI 来调用 API 接口进行扫描，自动化扫描就是这种模式，其执行流程为：“**后台服务定时同步项目**” -> “**API接口下发扫描任务**“ -> “**后台调度执行扫描**”。 `离线扫描` 其形式为审计人员手动上传一个 zip 或 rar 的源码包，扫描系统自动解压后进行代码扫描。

这两种模式的执行流程略有不同，所以后端实现也略有不同，第一种的执行流程要比第二种较为复杂，我们这里会以第一种方式来实现自动化扫描。

2. 要求

系统要求:

- 单个项目扫描控制在 20 分钟以内
- 支持调度节点监控
- 支持漏洞知识库管理
- 支持 API 接口
- 支持分布式部署, 方便扩展来提升扫描能力

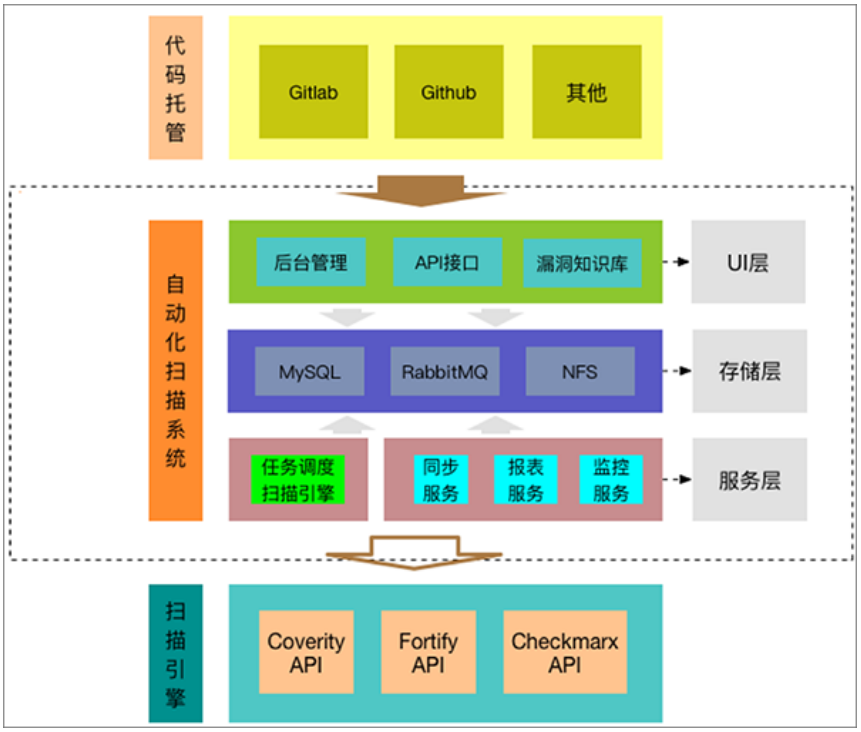
时间控制是为了保证 CI/CD 过程不会太长，避免影响项目发布。调度节点监控，这里存在两种情况：一种是调度进程的心跳监控；一种为扫描任务的超时监控。漏洞知识库主要为了与组件分析模块分析出的依赖组件进行漏洞匹配。API 接口是为了方便第三方(代指CI/CD)调用来下发扫描任务或查询结果。分布式部署方式，可以水平扩展来提高调度节点和API节点的处理速度与能力。

3. 模块设计

主要系统:

- 代码托管子系统
- 自动化扫描子系统
- 第三方扫描子引擎

这里从整体角度来观察，大致分为三个子系统， 自动化扫描子系统 依赖 代码托管子系统 ， 但不依赖 扫描引擎(泛指第三方商业或开源审计产品)子系统 ， 这是由于 自动化扫描子系统 内部集成了组件漏洞识别、黑白名单规则、黑白名单插件等功能，最后我们用一张图来说明。



三、总结

代码扫描固然重要，但是它不会解决所有项目的安全问题。项目安全应该从多个维度、多角度的来进行。如：项目立项时开始SDL；开发迭代过程中的代码扫描；项目上线前的黑盒测试。

参考链接

- CI (持续集成)
- CD (持续交付)
- OWASP 2017 TOP 10

Post Directory

文章目录