前言

本文章主要讲解一款漏洞验证框架的构思，并未详解内部的实现细节，本文篇幅稍长，请耐心看。

需求

做一款漏洞扫描器，首先要了解该扫描需要达到的效果。它的需求什么？需要支持什么？

可能需求如下：

- 跨平台运行，可以在Windows、Mac OS、Linux等操作系统运行或者使用 WEB 端控制
- 自定义POC、EXP多语言支持，POC支持多种开发语言实现，例如支持 Java、python 等。
- 多种运行方式，支持多线程、多进程、分布式运行等多种方式
- 可视化，漏洞扫描框架通过可视化操作
- 资产管理，通过探测结果的资产进行POC验证以及EXP利用
- 。。。。。。

需求分析

上面简单明确了一些一款漏洞扫描器的需求，需求的实现最好根据需求选取适合的技术来进行实现。

跨平台运行

跨平台的开发语言有很多，例如C、C++、Java、Python、Go 等等。

开发语言的选择，需要根据框架需要满足的运行效率、开发效率等多种角度来进行考虑。

如果需要很高的运行效率以及开发效率高的情况下，可以选择使用 Java 或 Python 等语言，如果需要追求极高的效率可以使用 Go 语言。

Java

Java 程序实际是在 Java 虚拟机(JRE 是软件实现)中运行，Java 虚拟机类似一个模拟执行环境，在不同的操作系统上拥有不同的 Java 虚拟机实现，但是这些 Java 虚拟机遵循统一的规范来解释 class 文件，并将 class 文件中的指令转换为本地操作系统对应的指令，这样就实现了相同的 class 文件，可以通过 Java 虚拟机转换为对应操作系统上的对应指令，实现 class 文件，也就是 Java 程序的跨平台性。

Python

Python 是一门跨平台的脚本语言，Python 规定了一个 Python 语法规则，实现了 Python 语法的解释程序就成为了 Python 的解释器。

自定义 POC

自定义POC、EXP本质就是基于模版进行编写的POC，能够让扫描框架识别并且运行。模版的组成一般分为漏洞基本信息、POC验证信息、Exp利用信息。

多种运行方式

多种运行方式，支持多进程、多线程、分布式运行等方式。如果扫描器需要大规模的的扫描、探测建议使用分布式节点的方式进行操作。一般情况下，多线程、多进程可以满足大部分的需求。

一般分布式适用于大型的企业内网或大范围的扫描探测。

可视化

可视化技术的选型，目前有三种比较流行的方案。

第一种为使用语言自带的可视化编程模块（QT）进行开发可视化程序。

第二种为使用 Electron 或 nw.js 进行开发可视化应用，其中比较有名的FOFA Pro 客户端、GoBy 、中国蚁剑等就是基于此方案进行开发的， Electron 的官网有很多基于此框架开发的程序。

第三种为采用 B/S 架构进行开发，后端控制使用 WEB进行控制，节点使用 Python、Java 等语言进行开发，其中 BugScan、w8scan、巡风等都是基于B/S架构实现。

Electron

> Electron 是由 Github 开发，用 HTML，CSS 和 JavaScript 来构建跨平台桌面应用程序的一个开源库。 Electron 通过将 Chromium 和 Node.js 合并到同一个运行时环境中，并将其打包为 Mac，Windows 和 Linux 系统下的应用来实现这一目的。
>
> Electron 于 2013 年作为构建 Github 上可编程的文本编辑器 Atom 的框架而被开发出来。这两个项目在 2014 春季开源。
>
> 目前它已成为开源开发者、初创企业和老牌公司常用的开发工具。

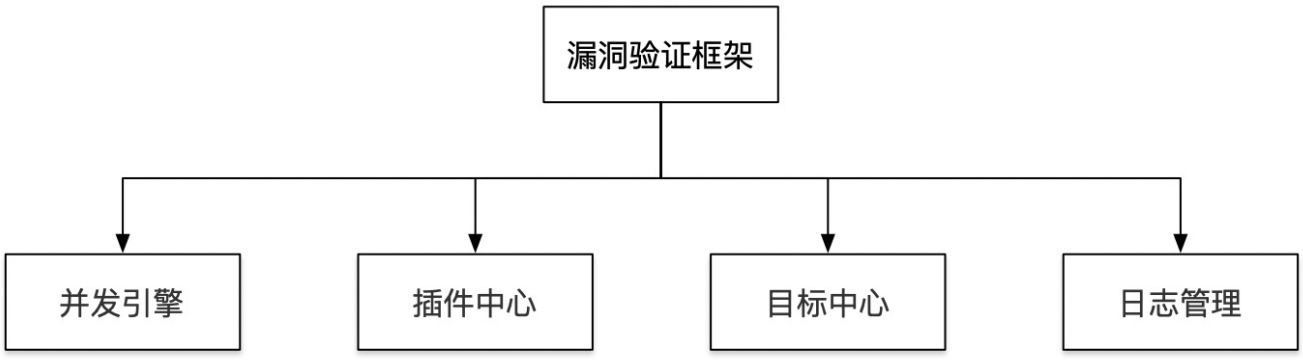
nw.js

> nw.js 基于 Chromium 和 Node.js。NW.js 利用 Web 技术结合 Node.js 及其模块进行桌面应用开发。

资产管理

资产管理，用于管理扫描探测结果的资产信息，用于资产与对应 POC 关联使用。其中资产管理中主要含有资产识别规则，用于识别资产的信息。例如通过资产的标题、body、证书、header、banner进行制定规则用于识别。

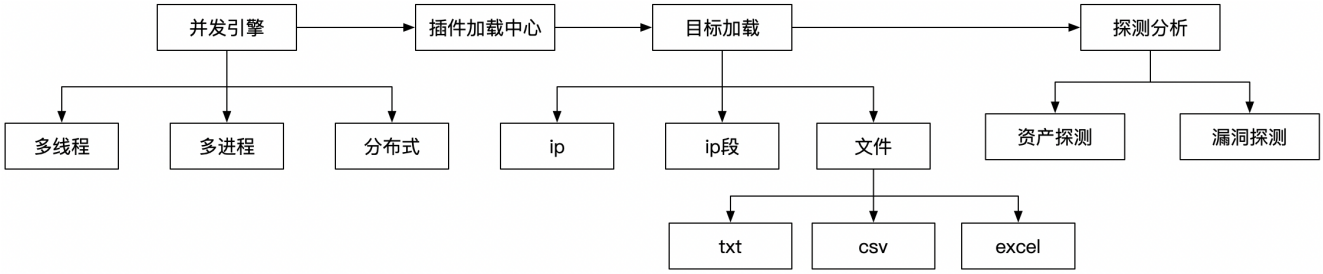
设计



整体框架分为并发引擎、插件中心、目标中心、漏洞验证四大模块。

- 并发引擎：主要提供漏洞验证框架运行方式，例如多线程、多进程、分布式等运行方式。
- 插件中心：主要提供漏洞 POC 的模版、POC 的加载以及 POC 的管理。
- 目标中心：目标加载、管理目标资产、漏洞管理。
- 日志管理：管理框架中所有网络请求以及响应信息，提供后续的回溯。

流程



框架运行流程

1. 选择合适的运行引擎，一般情况下使用多线程和多进程进行探测，如果探测大型企业内网以及大范围的扫描探测，分布式运行比较合适。
2. 插件加载中心，进行加载使用的 POC。
3. 目标加载，加载运行需要探测目标，可以导入单个ip、ip段以及文件形式进行加载。
4. 探测分析，探测分析主要分为资产探测和漏洞探测，资产探测通过资产识别规则进行识别资产信息，漏洞探测根据识别的资产信息进行漏洞探测分析。

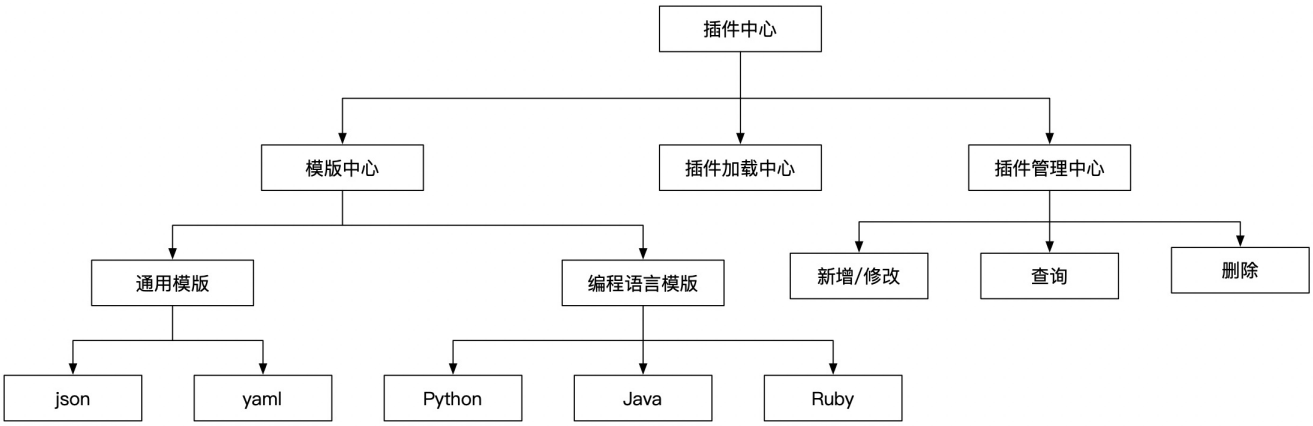
并发引擎

并发引擎主要为漏洞验证程序运行时选择的运行方式，主要有多线程、多进程、分布式 3 种运行方式，使用多线程、多进程、分布式运行方式来保证漏洞验证程序的效率。

正常情况下使用多线程、多进程适用于大部分使用的场景。

分布式运行用于大型企业内网探测或大规模的网络探测。

插件中心



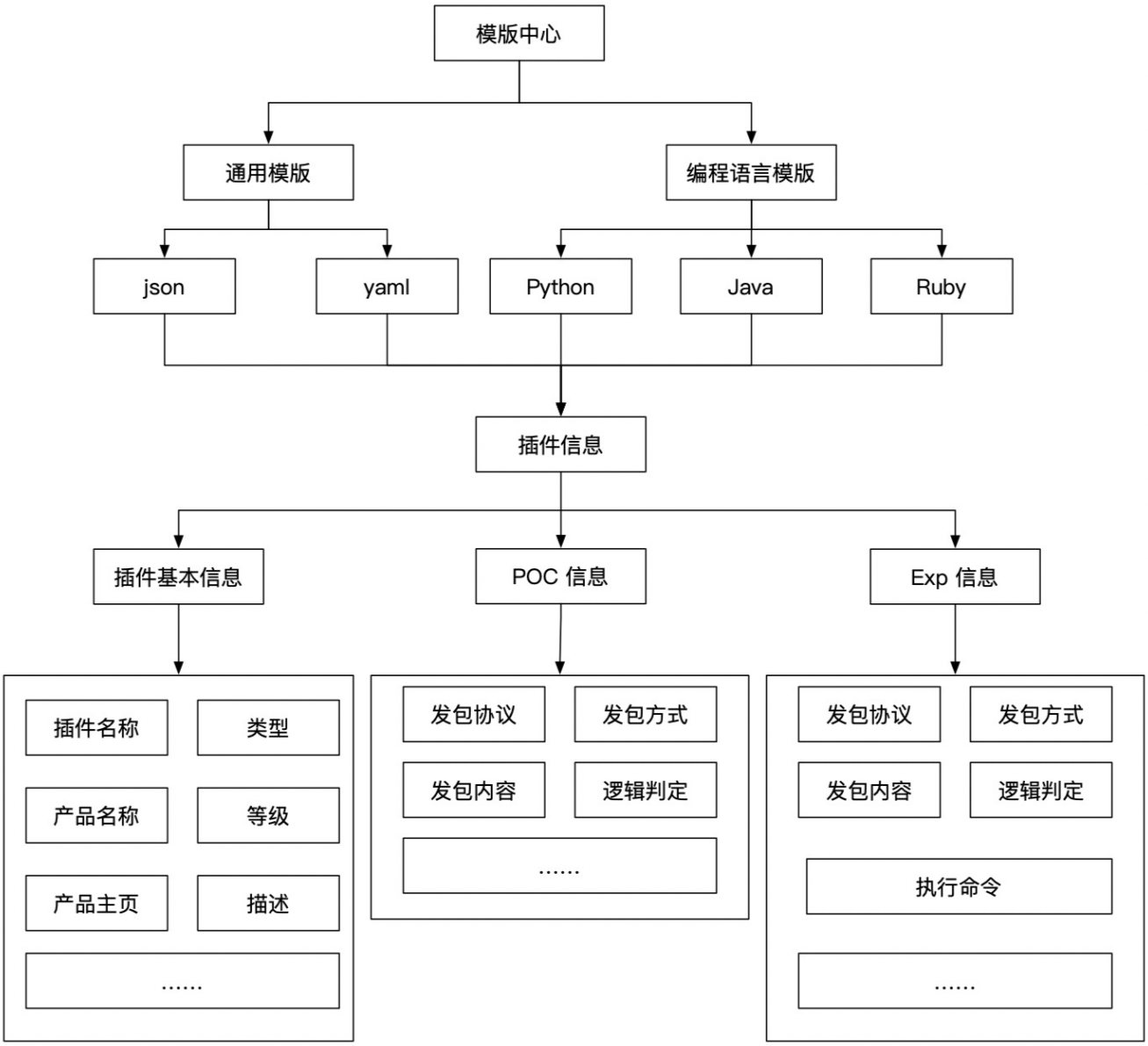
插件中心分为模版中心、插件加载中心以及插件管理中心组成。

模版中心提供 POC、Exp模版信息，主要分为通用模版和编程语言模版俩种。

插件中心，提供用户选择性需要测试的插件进行加载。

插件管理中心进行管理自定义编写的 POC。

模版中心



模版中心主要分为通用模版、编程语言模版，两大模块。其中各自组成都是由插件基本信息、POC 信息、Exp 信息三大部分组成。

插件基本信息由插件名称（漏洞名称）、类型、产品名称、等级、产品主页、描述等组成。

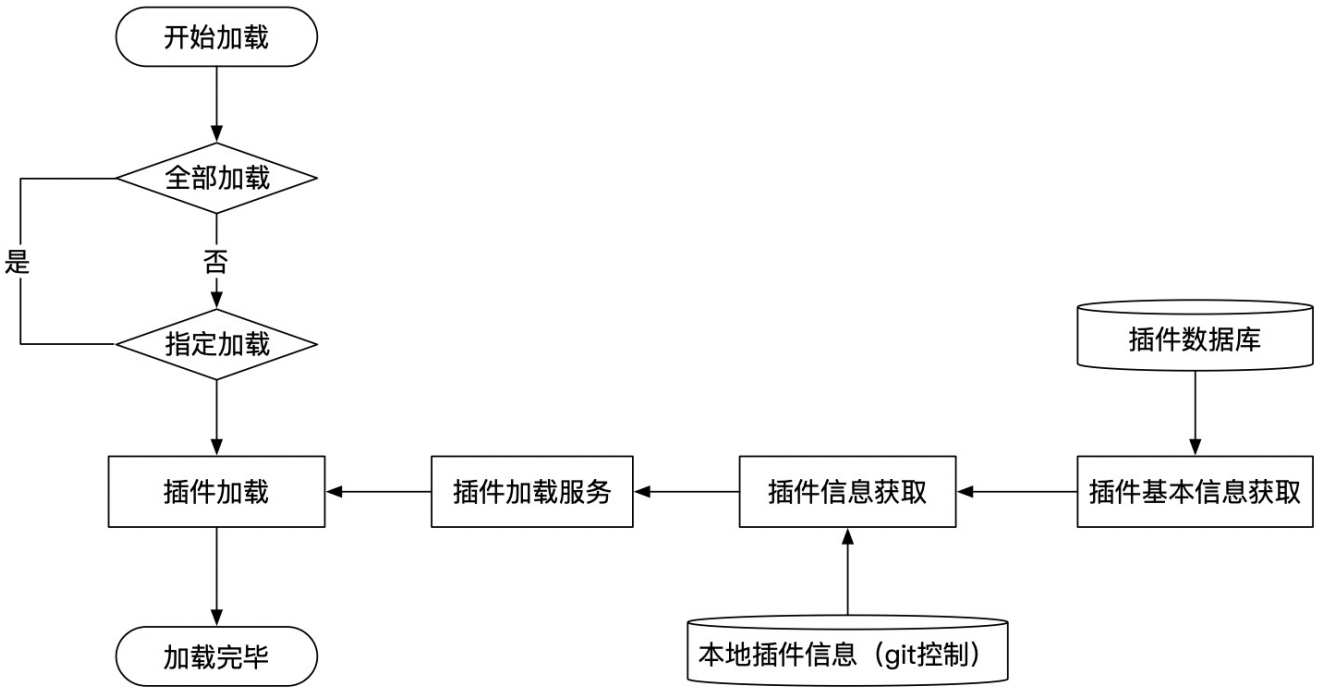
POC 信息由发包协议、发包方式、发包内容、逻辑判定等组成。

Exp 信息由POC 信息以及执行命令进行组成。

通常发包协议为 HTTP、HTTPS、TCP、UDP等，协议（部分）发包方式有 GET、POST、PUT、

HEAD、DELETE、OPTIONS、TRACE、CONNECT等，逻辑判定由“与或非”。

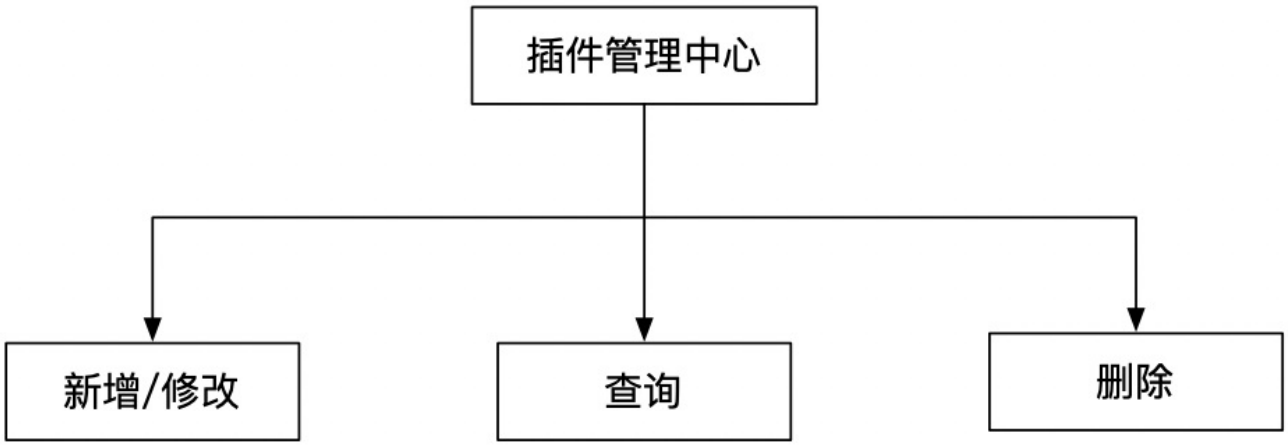
插件加载中心



插件加载通过用户选用方式（全部加载、指定加载）进行加载插件。

1. 从插件数据库中获取需要加载的插件**基本信息**。
2. 获取插件中的代码信息，根据信息从本地的 git 仓库中进行获取插件代码信息。
3. 然后通过对应的加载服务进行加载 POC。

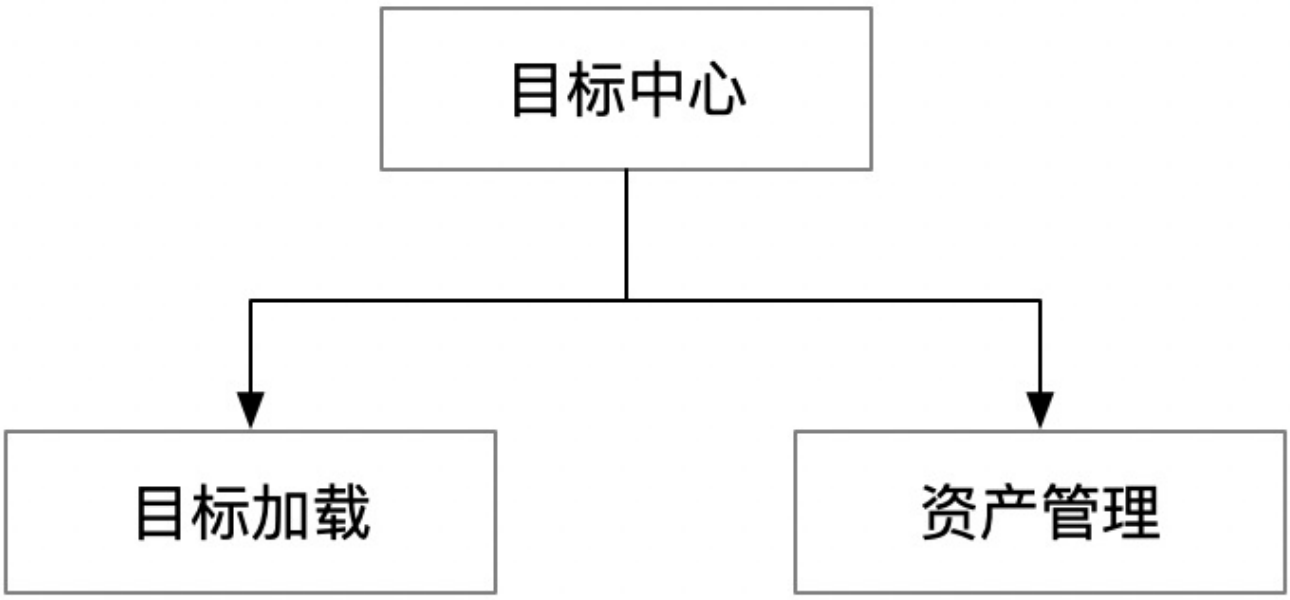
插件管理中心



插件管理中心进行管理编写的插件信息，针对插件进行修改、删除、查询等操作。

- 新增/修改：基于模版中心的通用模版和编程语言模版进行新增以及修改操作。
- 查询：通过模版中心中的字段作为依据进行多维度查询，获取插件信息提供后续的修改以及利用。
- 删除：通过模版中心中的字段作为依据进行多维度查询并且删除。

目标中心

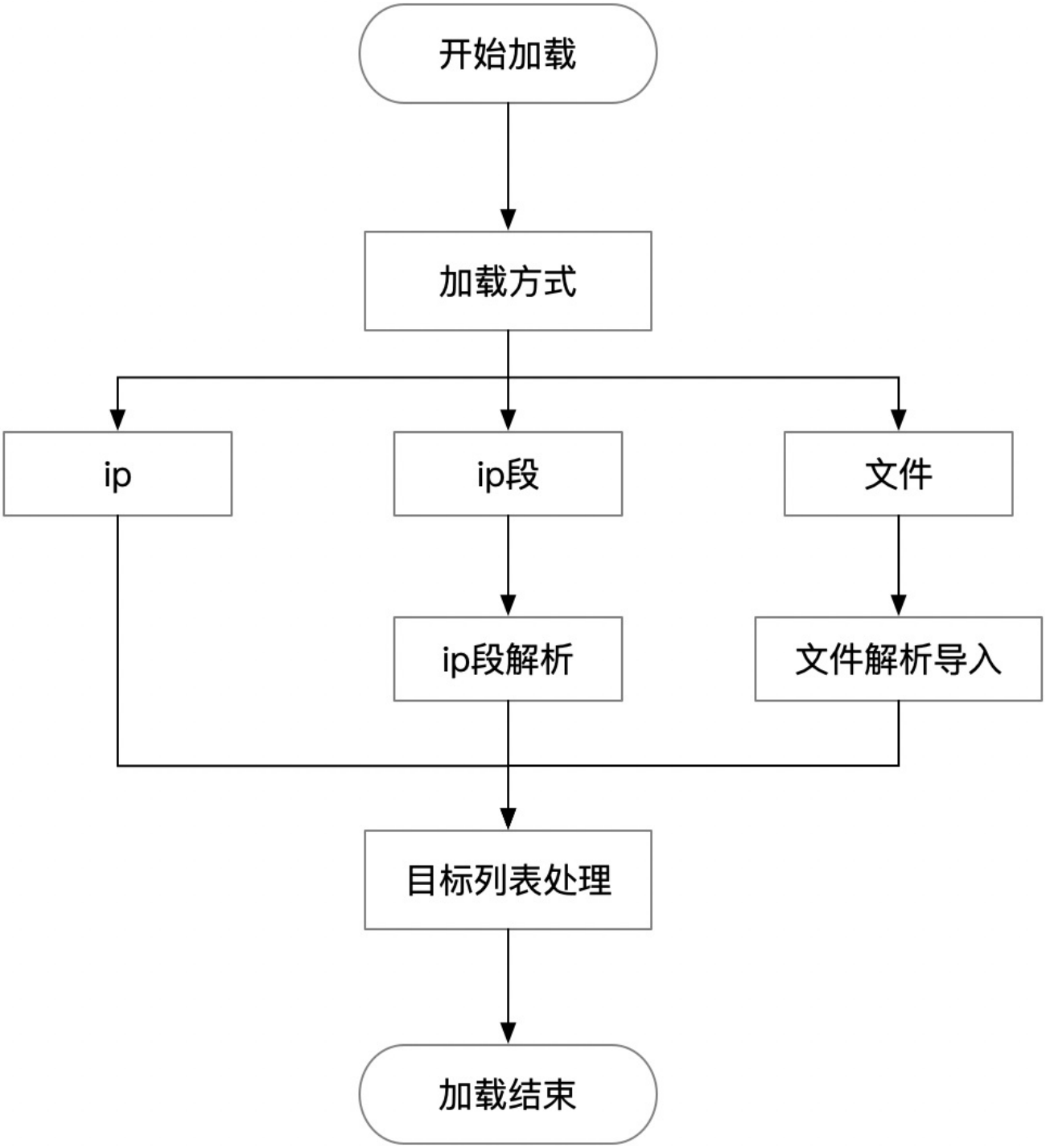


目标中心主要负责分析目标加载、资产管理以及漏洞管理三大模块。

- 目标加载模块：主要用来加载需要分析的目标。
- 资产管理：主要用来分析目标探测的资产信息、识别规则、漏洞信息管理。

漏洞插件信息可以通过使用数据库配合 Git 来进行存储，数据库中进行存储插件的基本，Git 存放插件的具体代码信息，使用 Git 可以进行控制插件的版本，可进行回退等操作。

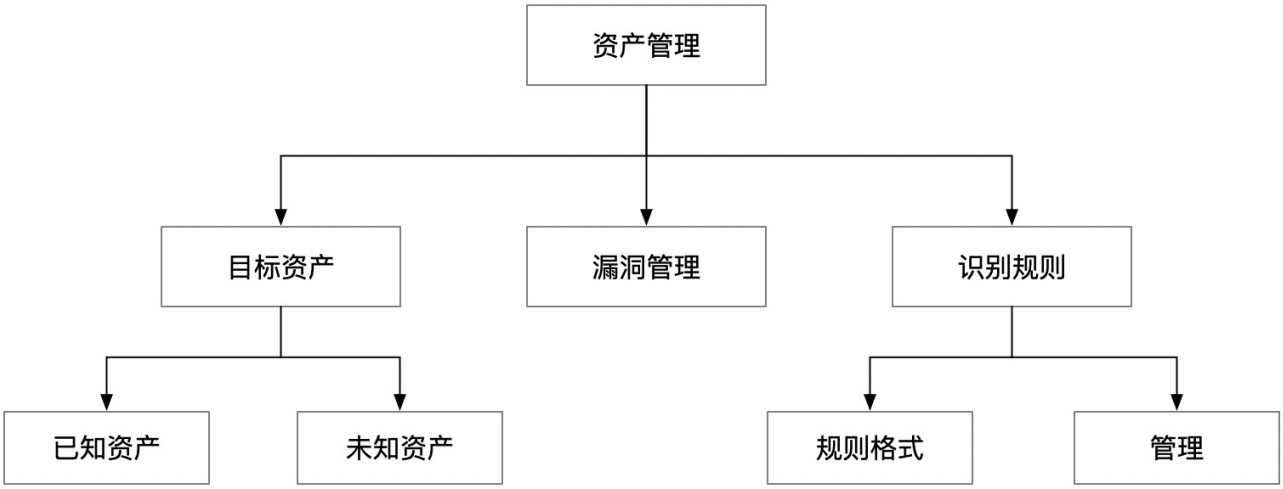
目标加载



目前加载可以通过加载单ip、ip段以及文件导入的方式进行导入。

- ip 段加载：进行生成 ip 段中的探测目标列表。
- 文件加载：解析导入的文件格式，通过对应的格式进行加载目标。

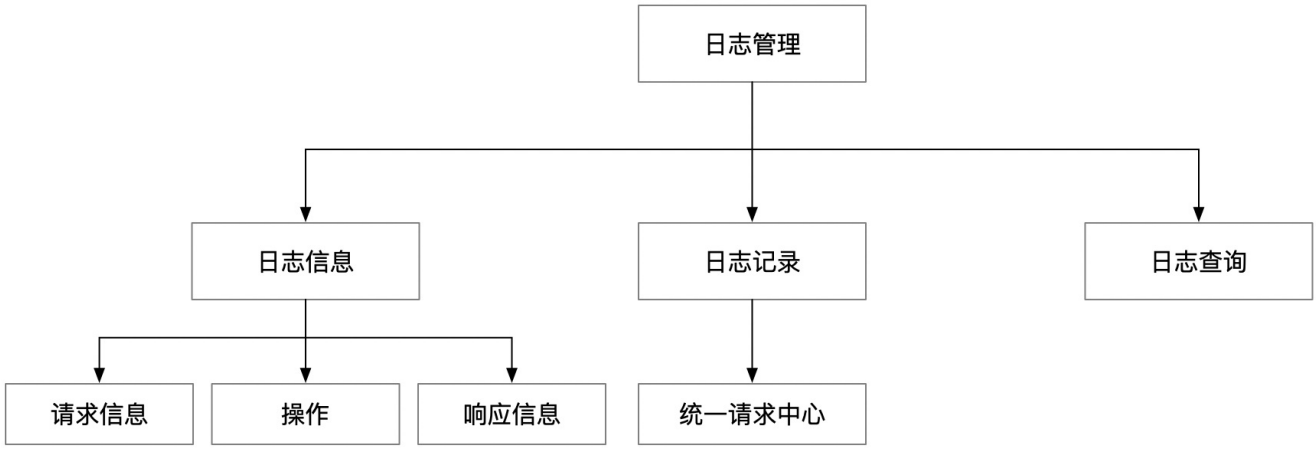
资产管理



资产管理模块主要进行管理分析目标识别成功的资产信息、漏洞管理、以及识别规则管理。

- 目标资产：通过探测响应的信息与识别规则进行匹配进行识别资产，如果识别成功为已知资产，否则为未知资产，未知资产便于后续的规则录入提高资产的识别率。
- 漏洞管理：通过与已识别资产与对应的漏洞检测插件进行检测的结果。
- 识别规则：识别规则构造格式，可以通过探测响应的协议、响应内容、Header、服务类型等信息进行构造规则作为识别规则。

日志管理



日志管理模块主要管理整体框架的日志信息包含日志记录、查询等操作。

- 日志信息包含，框架内部请求信息（请求URI、请求方式（POST、GET、DELETE等）、请求参数等信息）以及响应信息（响应头、响应内容等信息）以及框架执行操作（资产探测、插件检测等）。
- 日志记录，可以通过统一请求中心记录网络日志信息。
- 日志查询，通过查询ip、请求URI、请求方式、响应内容等多维度进行查询，便于后续的漏洞漏洞排查。

总结

文中的流程图：https://github.com/0nise/scripts

本文描述的为我所构思的一款漏洞探测框架，如果有文章内描述不符以及问题，请各位师傅不要吝啬，烦请各位师傅斧正。