**开源解析与漏洞精准定位技术合作项目**

**漏洞精准定位**

**技术方案设计文档**

修订记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **版本号** | **变更时间** | **变更范围** | **变更简要说明** | **变更**  **类别** | **变更人** |
| 1.0 | 2020-11-21 | 全文 | 初始化创建文档 | N | 陆宸昊、王中亮、谢隽丰、许聪颖 |
| 2.0 | 2020-11-26 | 全文 | 详细阐述技术部分 | M | 许聪颖、谢隽丰 |

注：N-New 新建；M-Modify 修改；A-Add 新增；D-Delete 删除；

**目录**

[1 DR007技术方案 4](#_Toc76508053)

[1.1 需求描述 4](#_Toc76508054)

[1.2 技术挑战 4](#_Toc76508055)

[1.3 技术方案设计 4](#_Toc76508056)

[1.3.1 方案概述 4](#_Toc76508057)

[1.3.2 命名规范化处理详细说明 5](#_Toc76508058)

[1.3.3 漏洞数据库检索详细说明 5](#_Toc76508059)

[2 DR008技术方案 6](#_Toc76508060)

[2.1 需求简介 6](#_Toc76508061)

[2.2 技术挑战 6](#_Toc76508062)

[2.3 技术方案设计 6](#_Toc76508063)

[2.3.1 方案概述 6](#_Toc76508064)

[2.3.2 漏洞数据库增量更新详细说明 6](#_Toc76508065)

[3 DR009技术方案 8](#_Toc76508066)

[3.1 需求简介 8](#_Toc76508067)

[3.2 技术挑战 8](#_Toc76508068)

[3.3 技术方案设计 8](#_Toc76508069)

[3.3.1 方案概述 8](#_Toc76508070)

[3.3.2 CVE组件特征抽取详细说明 9](#_Toc76508071)

[3.3.3 构建组件搜索空间 9](#_Toc76508072)

[3.3.4 组件数据库检索与筛选详细说明 9](#_Toc76508073)

[4 DR010技术方案 11](#_Toc76508074)

[4.1 需求简介 11](#_Toc76508075)

[4.2 技术挑战 11](#_Toc76508076)

[4.3 技术方案设计 11](#_Toc76508077)

[4.3.1 方案概述 11](#_Toc76508078)

[4.3.2 补丁识别详细说明 12](#_Toc76508079)

[4.3.3 基于投票的补丁筛选详细说明 13](#_Toc76508080)

[4.3.4 补丁信息解析详细说明 13](#_Toc76508081)

# DR007技术方案

## 需求描述

根据指定的软件清单作为输入，通过对软件名称的各种别名、变形名处理，在CVE、NVD等漏洞网找出对应的漏洞清单，最终解析结果包括：软件名称、漏洞ID清单。

## 技术挑战

漏洞库有海量的CVE，给定一个软件名，需要能够快速而准确地找到软件对应的所有CVE。同时软件名又存在别名、相似名等噪音干扰。因此，在大规模漏洞数据库中准确而全面地识别出与特定软件相关的漏洞是一个难点。

## 技术方案设计

### 方案概述

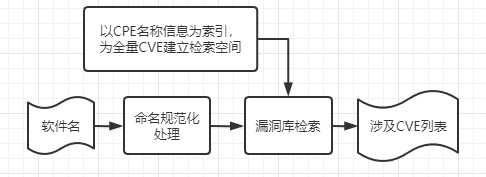


图1.1 DR007技术方案框架图

首先我们获取了NVD的全量漏洞数据库信息，并以CVE的CPE的Vendor与Product信息为索引，建立全量CVE ID的检索空间。值得一提的是，虽然NVD数据库不仅在CPE中包含软件信息，在Description、Reference中也有可能出现软件信息，但由于上述两个渠道的信息包含较多噪音，且未必一定包含软件信息，所以尝试使用这两者建立索引时的效果并不好。而CPE本就是漏洞影响软件的信息，故只对CPE建立的索引。

其次是社区中软件名称不统一不规范。因此为了达到更好的检索效果，我们对输入的软件名进行规范化处理，统一软件名格式；最后，将规范化处理后的软件名，输入漏洞数据库进行检索，返回候选漏洞列表。

### 命名规范化处理详细说明

该步骤主要是针对输入软件名命名形式不统一，存在连字符、下划线、驼峰命名、大小写不一致等情况进行处理。例如，Hadoop Common、HadoopCommon、hadoop-common及hadoop common实则都指向同一软件制品，对此，该步骤将其统一处理为“hadoop common”，便于后续漏洞数据库检索及软件名匹配步骤进行处理。

### 漏洞数据库检索详细说明

截至2021年7月，有效CVE条目总量已达到15万，为实现快速检索漏洞数据库的应用需要，我们本可以直接使用NVD官方提供的API。但出于离线化的需要，且官方API本就不稳定，我们从官网下载了所有年份的CVE数据，并使用Lucene检索引擎定制了一个通过软件关键词搜索漏洞ID的搜索引擎。向Lucene输入的用于建立检索空间的每条数据包括CVE ID与CPE两个信息，以CPE的单位为单位建立索引。同时，我们也自定义且优化了检索方法，使之适应本需求的需要。例如，CPE的索引方法以单词为单位，不区分大小写；输入的软件信息关键词只有当全部在CPE单词中出现时，该CVE才会被返回。于是最后的流程即为，将所有年份的CVE数据提取CVE-CPE信息并输入Lucene，Lucene以CPE的每个单词建立索引。在之后的查询中，输入软件名，Lucene将检索其漏洞库并返回CVE。

# DR008技术方案

## 需求简介

根据指定的漏洞清单（CVE ID清单）作为输入，通过分析漏洞披露网站的内容，输出漏洞描述和影响范围等基本信息。最终解析结果包括：漏洞的CVE ID、漏洞描述、官方发布日期、官方更新日期、影响的软件名称和版本、CVSS\_V3和对应的Vector、CVSS\_V2和对应的Vector、References信息。

## 技术挑战

漏洞的基本信息往往会经常变化（如更新描述和references信息），而漏洞数据库的规模较大。因此，确保漏洞数据基本信息的实时同步性是一个难点。

## 技术方案设计

### 方案概述

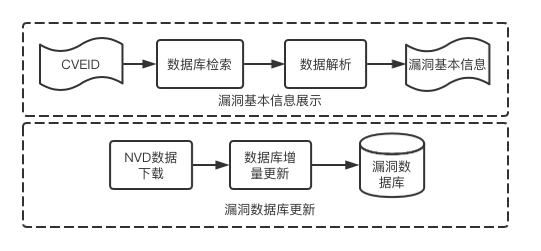


图2.1 DR008技术方案框架图

DR008的技术设计可分为两个模块，漏洞基本信息展示模块和数据库更新模块。漏洞基本信息展示模块，用以接收输入的CVEID，在漏洞数据库中检索该CVE条目的基本信息。

为确保漏洞数据保持实时更新，本技术设计了数据库自动更新模块，每半天自动下载NVD数据，并针对新增的或基本信息有改动的CVE进行增量更新。

### 漏洞数据库增量更新详细说明

初次建立漏洞数据库时，该模块全量下载并新增所有CVE条目数据，并从NVD平台逐个爬取CVE条目所对应的CPE列表信息（例如： CVE-2019-1003004[[1]](#footnote-1)）。为了便于后期漏洞数据库维护，增量更新时，判断该CVE条目基本信息是否变化，该步骤计算该CVE条目所有信息的哈希值（MD5）并存为该CVE的基本属性。

后期维护时，该模块每半天（12小时）自动下载最新NVD数据集，逐个解析CVE条目基本信息、计算哈希值（MD5）并与库中CVE条目数据进行对比，以判断该CVE条目为新增、修改或是删除。针对所有新增及修改的CVE条目，该模块启动CPE列表爬取程序，逐个爬取CVE条目所对应的CPE列表信息。通过以上步骤，确保漏洞数据库的实时性及完整性。

# DR009技术方案

## 需求简介

漏洞实体级的精准定位能力，从软件级到组件级定位。输入指定的CVE ID，返回当前漏洞影响的组件清单，包括组件的GroupId、ArtifactId与组件程序语言等信息。

## 技术挑战

CVE漏洞只与软件（CPE）建立了关联，而一个软件往往包含多个组件。因此，在漏洞的基本信息（如描述和补丁）中抽取能够表征相关组件的特征是一个难点。此外，不同编程语言的组件规模巨大且语料特征差异大，不同编程语言可能存在相似的组件名，而组件名在演化过程中可能发生更改。因此，建立各个编程语言的组件库并准确地识别与漏洞相关组件坐标是一个难点。

## 技术方案设计

### 方案概述

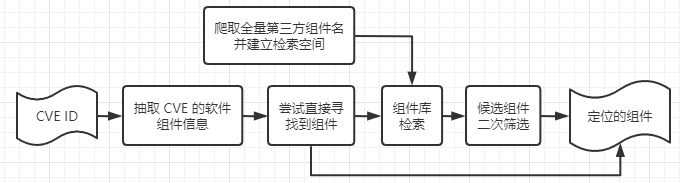


图3.1 DR009技术方案框架图

由于漏洞条目仅与软件级信息建立关联，为定位出具体组件坐标，该技术首先爬取了全量的第三方组件库的名称与发行版本，并为所有组件名建立索引方便后续检索。针对输入的指定CVE条目，首先尝试搜索其补丁以直接找到组件。若未能找到补丁、或补丁信息亦不直接给出组件，则从补丁描述、漏洞描述、参考文献等信息中自动抽取组件特征，并将组件特征信息投入组件数据库检索；由于社区中不同程序语言的组件规模及命名风格有较大差异，还需针对不同语言定制搜索引擎以实现快速准确检索的应用需求。此外，为避免组件列表中别名、类似名、演化变形名等信息的干扰，还需对数据库的检索结果进行筛选，以准确定位出漏洞相关的组件清单。

### CVE组件特征抽取详细说明

为提取CVE条目中的组件特征，该步骤从多个渠道获取组件级信息，包括：DR010部分识别的补丁信息、NVD平台提供的CPE信息、Description与Reference信息。

针对补丁信息中Git Commit类型的数据，该步骤首先尝试直接从仓库中的包管理器文件中抽取组件（例如Maven的pom.xml，NPM的package.json文件）。若能成功抽取组件，则流程直接结束。但并非所有仓库都可以抽取，例如由Gradle组织的项目则无法抽取。此时我们提取Commit修改文件的文件路径与文件后缀，尝试从路径中抽取出组件级别的单词信息与语言信息。此外，CPE数据不仅包含了Vendor与Product这一软件级别的信息，还包含了受影响版本这一重要信息。因此我们使用一定匹配规则，将补丁路径信息与CPE匹配起来，获得“软件-组件-版本-程序语言”这一条组件特征信息。而如果未能找到补丁信息，我们则会再尝试再从Reference中提取有用信息。首先我们使用TF-IDF算法从Reference中获取可能是软件信息的单词，与CPE的Product共同组成可能的软件名，然后回到Reference中找出与软件名相关的词组，即为组件特征。

### 构建组件搜索空间

若要根据CPE、Patch文件路径等一系列单词检索出相关组件，需要为所有第三方目标组件构建一个基于文本相似度的搜索空间。第三方组件数量庞大、不同语言的组件名特征也不尽相同，因此所实现的检索功能不仅对检索效率有要求，对如何评判“相似度”也提出了挑战。

为了解决检索效率问题，该技术使用了开源的检索引擎框架Lucene，该框架可以基于文本相似度快速地对搜索空间内的文档进行模糊匹配，并返回相似度最高的若干结果。同时该框架也提供了高自由度，可以替换修改相似度算法、匹配算法等。

针对各种语言（目前包括Java、Javascript与Python），从社区中该语言的主流组件管理平台获取所有第三方组件清单作为搜索空间。具体而言，针对Java，我们以Maven中央仓库中的组件清单构建搜索空间，共包括 29万个组件；针对Javascript，我们从NPM仓库中获取组件清单，共118万个组件；针对Python我们从Pypi仓库中获取组件清单，共27万个组件，。

不同语言的组件数量、组件名命名习惯差异较大，因此如果将不同语言的组件均导入同一个检索空间、使用同样的相似度计算规则，极易产生检索结果偏向某一语言导致准确率下降的问题。为此，该步骤为每种程序语言分别建立了搜索空间，并且考虑不同语言组件的命名特征定制检索时的相似度计算规则。

### 组件数据库检索与筛选详细说明

首先，将先前获取的每一条组件信息输入每个语言的搜索空间进行检索。这些信息可能来自补丁路径信息、Reference信息，也可能只是CPE信息。针对每个语言，都会返回TopN条相似度从高到低排序的搜索结果，每条搜索结果即为一个组件（如果指定了语言则仅针对该语言进行检索）。然后，进行组件筛选。其中最重要的步骤是通过漏洞组件的版本信息，与上一步检索出来的所有组件进行比较，若漏洞版本特征与组件版本差异巨大，则予以过滤。除了版本过滤，也设立了一些规则筛选其他不符合条件的组件，例如如个人开发者克隆官方的组件。最终，根据每一条“组件-版本-语言”的数据，经过“检索-筛选“后，返回Top1的匹配结果。细节如下：

1. 针对每个语言，从其流行的依赖管理工具（如Java的Maven、Javascript的NPM）收集其所有中央仓库的组件，包括组件名与组件版本。然后使用基于Lucene的检索引擎、针对每个语言分别对这些组件名称建立索引。建立索引时，也需要考虑到不同语言的命名风格差异，对不同的语言分别建立检索空间。
2. 如果能找到补丁且能直接定位到组件，则直接返回。
3. 否则提取CVE的各类数据如补丁的文件路径、Reference的词组信息、CPE的软件版本信息，并综合各类数据（详见3.3.2），最终获得的每条数据最多包含“软件关键词”、“组件关键词”、“语言”、“版本”信息，每条数据都将在后续步骤进行检索，最终每条数据都会对应一个或零个组件。
4. 将收集的组件关键词输入检索引擎，引擎根据词汇的相似度打分，返回相似度最高的若干候选组件。随后进行组件的二次筛选过滤，例如将CPE受影响版本与这些候选组件的版本根据一定策略进行匹配，过滤那些组件版本与CPE受影响版本严重不符的组件。经历相似度匹配与版本过滤后，对每一条CPE-Patch数据对，返回相似度最高的Top1作为结果并返回。

# DR010技术方案

## 需求简介

漏洞代码级的精准定位能力。以漏洞CVEID作为输入，通过分析 References 中各链接网站的内容，判断当前漏洞是否存在补丁、以及漏洞所影响的函数和代码行清单。最终解析结果包括: 漏洞ID、是否存在补丁、漏洞所影响的函数和代码行清单。

## 技术挑战

漏洞补丁信息藏匿在多源异构的网页信息（如NVD官方网站的References、各类缺陷追踪系统的讨论、mailing list的讨论、各软件产品的官方漏洞披露网站）中，同时在多源异构信息中存在的补丁信息还可能是不可靠的（如存在着与漏洞不相关的代码变更信息），导致难以有一种通用的方法去定位漏洞补丁。因此，利用数据驱动的方法去总结漏洞补丁定位模式并基于模式准确地定位漏洞补丁是一个难点。此外，漏洞补丁数据也往往是异构的（如git commit、svn commit、DIFF文件、嵌在文本中的 DIFF描述）。因此，准确地抽取补丁的细粒度信息（如文件、函数）是一个难点。

## 技术方案设计

### 方案概述

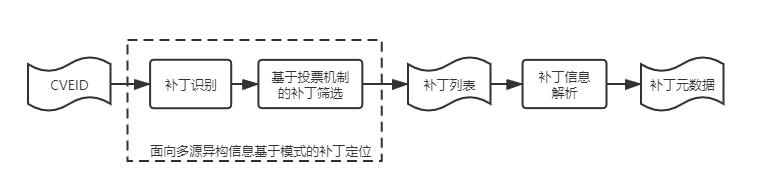


图4.1 DR009技术方案框架图

为了全面而准确地在多源异构网页信息中定位补丁信息，该技术利用数据驱动的方法设计补丁识别模块。针对输入的CVEID，该技术先从大量网页信息中识别补丁信息；但多源异构信息中常常存在着与漏洞不相关的代码变更信息，为了去除补丁信息中的噪声数据，该技术基于投票机制筛选出与输入CVE条目相关性较高的补丁作为候选；此外，针对候选补丁，该技术还将收集补丁的上下文语义信息，基于补丁与CVE条目间的语义相似度确认该补丁与该CVE条目的相关性。通过上述方式，定位出补丁列表。

为了有效地分析漏洞对项目安全威胁及影响情况，需解析出补丁的细粒度信息，如：文件、函数、代码行等信息；然而，补丁信息往往是异构的，如：git commit、svn commit、DIFF文件、嵌在文本中的 DIFF描述等形式；对此，该技术还需针对不同形式的补丁定制解析方法，以获取patch元数据信息。

### 补丁识别详细说明

1. 方案探索与研究

漏洞补丁信息往往藏匿于多源异构的网页信息中。为了实现全面且准确地补丁信息定位能力，复旦大学软件工程实验室团队进行了两批次人工数据分析，一方面，通过抽样调研漏洞数据，以评估补丁信息的可获取性及可获取比例；另一方面，探索、整理和归纳补丁定位的规律及模式。

通过人工分析至少100条CVE条目数据，发现漏洞可获取补丁信息的比例约占30%～40%，约60+%的CVE条目未对外界公开补丁信息；此外，对于可获取的补丁信息，多以git commit、svn commit、DIFF文件以及嵌在文本中的 DIFF描述形式出现，存在于漏洞Reference信息项、产品缺陷追踪系统（Issue Tracker）、产品官方漏洞披露报告（Vendor Advisory）、第三方漏洞披露报告（Third-party Advisory）以及GitHub repository中；综合以上发现，设计如下补丁识别技术。

1. 方案设计

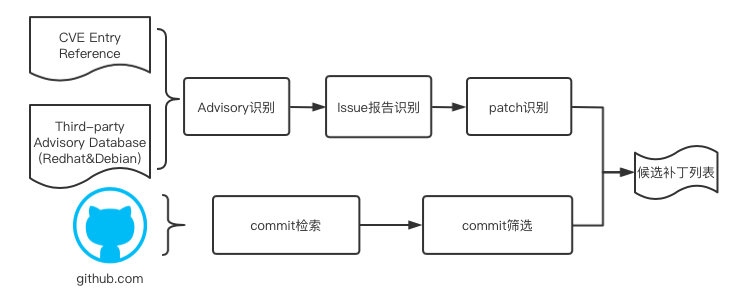


图4.2 DR009技术方案框架图

通过前期调研，该步骤将补丁（patch）的形式定义为以下四种：git commit、svn commit、DIFF文件以及嵌在文本中的 DIFF文本；此外，面向多源异构的网页信息，基于每种形式patch的网址（URL）及网页（文本）特征，进行补丁信息识别，具体实现如下。

1. 针对NVD平台中CVE条目Reference信息项，该步骤先进行advisory、issue网址的识别，再进行patch信息识别，具体策略为：
   1. “NVD-ref -> patch”，是指访问并识别NVD References中的含Patch标签或特征的网址。
   2. “NVD-ref -> Patch -> patch”，是指访问并解析NVD References中的Patch标签网页，以识别patch相关网址。
   3. “NVD-ref -> issue -> patch”，是指访问并解析NVD References中的含Issue标签或特征的网页，以识别patch相关网址。
   4. “NVD-ref -> vendor advisory -> patch”，是指访问并解析NVD References中的含Vendor Advisory标签的网页，以识别patch相关网址。
   5. “NVD-ref -> vendor advisory -> issue -> patch”，是指访问并解析NVD References中的含Vendor Advisory标签的网页，以识别issue相关网址；并解析issue相关网页，以识别patch相关网址。
2. 针对第三方漏洞披露平台的报告信息，该步骤同样先进行issue网址的识别，再进行patch信息识别，具体策略为：
   1. “Redhat bugreport comment -> patch”，是指访问并解析Redhat报告的评论中含Patch特征的网址。
   2. “Redhat bugreport comment -> issue -> patch”，是指访问并解析Redhat报告的评论中含Issue特征的网页，以识别patch相关网址。
   3. “Debian bugreport note -> patch”，是指访问并解析Debian报告的评论中含Patch特征的网址。
   4. “Debian bugreport comment -> issue -> patch”，是指访问并解析Debian报告的评论中含Issue特征的网页，以识别patch相关网址。
3. 针对Github repository，该步骤以CVEID及前序步骤中识别的IssueID作为关键词，于Github.com全站检索message中出现CVEID或IssueID关键词的git commits；由于存在很多与漏洞非直接相关的代码提交（commit）信息，该步骤通过计算commit所属的Owner-Repository信息与CVE条目的Vendor-product信息间匹配度，过滤非直接相关的代码提交信息。该策略标记为“GitHub -> commit”。

### 基于投票的补丁筛选详细说明

基于**4.3.2补丁识别模块**的结果，针对输入的漏洞条目，可获得通过多途径收集的补丁信息。由于多源异构的网页信息较杂乱且噪声较多，为避免非指定漏洞相关的补丁信息干扰，需对识别的补丁信息进行汇总、排序及筛选。

对补丁识别结果进行汇总后，该步骤采用“加权投票”机制，分为：加权、投票和选定结果三个部分。

1. 加权，**4.3.2补丁识别模块**中共有10条漏洞识别策略，而每个策略识别出的补丁可信度并不一致；该步骤通过对单个策略的补丁结果在precision、recall及F1-score方面的评估以确定各策略的权重；因为指标F1-score计算自precision及recall，所以该步骤将F1-score的结果作为每个策略patch结果的权重。
2. 投票，该步骤旨在汇总单个候选补丁信息在多个策略中的权重结果。例如：“NVD-ref -> patch”权重为1，“NVD-ref -> issue -> patch”权重为0.8，若补丁patch1同时出现在这两个策略中，则投票结果为1.8；其他情况，依此类推。
3. 选定结果，该步骤旨在依据投票结果，选定漏洞的补丁列表。具体为：
   1. 若patch总量为1时，直接作为结果返回
   2. 若patch总量大于1，取投票分数大于“NVD-ref -> patch”权重阈值的所有patch作为结果返回
   3. 汇总后，若patch总量大于1，且无加权分数大于“NVD-ref -> patch”阈值的patch，则返回分数最高的patch作为结果

通过上述加权、投票和选定结果三个步骤，筛选出与输入CVE条目对应的补丁列表。

### 补丁信息解析详细说明

漏洞补丁数据往往是异构的，在**4.3.2补丁识别模块**中所定义的补丁就包含：git commit、svn commit、DIFF文件以及嵌在文本中的 DIFF描述四种形式。由此可见，补丁数据的异构性为补丁信息解析带来较大困难。对此，该模块旨在通过定制化方法，准确地抽取补丁的细粒度信息（如：文件、函数、代码等）。

1. git commit元数据信息解析，是指通过Github获取git commit详细信息，并从中解析出：文件、函数和代码行信息，具体过程如下：
   1. 调用Github提供的API接口，获取指定commit的数据内容（形式为json文件）。
   2. 解析json文件，可获取该commit涉及的所有文件信息。
   3. 解析json文件中*“patch”*项，获取该commit涉及的函数及代码行信息，具体方法为：
      1. 针对不同语言设置不同的正则表达式以匹配各个语言的函数定义语句，程序设计语言大致可以分为两类：一类以C/C++为代表，函数定义时有返回值，访问限制符等元素，这时使用*“）(.\*?){”*作为匹配的正则表达式，去除可能的噪声*“if/for/catch”*，匹配前者且不匹配后者的即为一个函数定义式；第二类以Python/Go为代表，在定义函数时有指定关键字（如*def,func*等），这种情况下匹配上该关键字即为一个函数定义式。
      2. 对于上一步而言，已经得到了所有可能被修改的函数名，在该步骤中去除噪声即可（噪声：即出现在patch项中但实际上没有被修改的函数），可能的情况有：在省略行之后的三行未修改行中有新的函数定义的产生，这时省略行中的函数名（如果有的话）就实际上未被修改，为噪声；在一个函数被定义后，很快就结束了，而导致其整体未进入修改部分，此时改函数也未被修改，为噪声。去除这两种噪声后，即可得到正确的function列表。
      3. 对于function列表中的每个函数名进行分类，如果其行开头为“+”则为added，如果为”-”则为deleted，其余的即为modified。

1. https://nvd.nist.gov/vuln/detail/CVE-2019-1003004/cpes?expandCpeRanges=true [↑](#footnote-ref-1)