

北京郵電大學

本科 毕业 设计 (论文)



题目：基于漏洞知识图谱的可视化系统的设计与实现

姓 名 马嘉骥
学 院 计算机学院（国家示范性软件学院）
专 业 计算机科学与技术
班 级 2018211303
学 号 2018211149
班内序号 17
指导教师 方维

2022 年 5 月

北京邮电大学

本科毕业设计（论文）任务书

学院	计算机学院	专业	计算机科学与技术	班级	2018211303
学生姓名	马嘉骥	学号	2018211149	班内序号	17
指导教师姓名	方维	所在单位	北邮计算机学院	职称	副教授
设计(论文)题目	(中文) 基于漏洞知识图谱的可视化系统的设计与实现				
	(英文) Design and Implementation of Visualization System Based on Vulnerability Knowledge Graph				
题目分类	工程实践类 <input checked="" type="checkbox"/>	研究设计类 <input type="checkbox"/>	理论分析类 <input type="checkbox"/>		
题目来源	题目是否来源于科研项目	是 <input checked="" type="checkbox"/>	否 <input type="checkbox"/>		
	科研项目名称:	视频监控网络安全检测平台技术研究			
	科研项目负责人:	方维			
主要任务及目标: 1、构建漏洞知识图谱。 2、实现基于图数据的漏洞知识图谱存储。 3、实现漏洞知识图谱的可视化系统					
主要内容: 1、收集漏洞知识，包括受影响产品、可利用代码及补丁等信息 2、将多个数据源抽取的知识进行融合，抽取漏洞的实体及关系，建立漏洞图数据库 3、前端提供可视化交互接口进行展示、知识筛选等操作					
毕设所需知识: 1、熟悉 HTML、CSS、JavaScript 等前端开发语言及 vue、react 等前端页面开源框架； 2、熟悉 Django、Flask 或 SpringBoot 一种后台开发框架 3、熟悉 Scrapy 框架的使用 4、熟悉 Neo4j 图数据库和 D3.js 库的使用 5、熟悉机器学习相关知识以及 sklearn/PyTorch 的使用					
主要参考文献: [1] 张吉祥, 张祥森, 武长旭, 赵增顺. 知识图谱构建技术综述 [J/OL]. 计算机工程:1-16 [2021-11-06]. https://doi.org/10.19678/j.issn.1000-3428.0061803 . [2] 陶耀东, 贾新桐, 吴云坤. 一种基于知识图谱的工业互联网安全漏洞研究方法 [J]. 信息技术与网络安全, 2020, 39(01):6-13+18.					

- [3] Han Z , Li X , Liu H , et al. DeepWeak: Reasoning common software weaknesses via knowledge graph embedding[C]// 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2018.
- [4] Yan Jia, Yulu Qi, Huaijun Shang, Rong Jiang, Aiping Li.A Practical Approach to Constructing a Knowledge Graph for Cybersecurity[J].Engineering,2018,4(1):53-60.
- [5] <https://d3js.org/>
- [6] <https://neo4j.com/docs/>
- [7] <https://www.cve.org/>

进度安排:

- 2021/12/12 – 2021/12/24 (两周) 明确任务,了解课题背景,制定计划,查找相关论文资料,对课题的研究方法形成大体框架并提交开题报告,
- 2021/12/27 – 2022/01/14 (三周) 学习基本 Web 知识和 Scrapy 爬虫框架的使用,爬取公开漏洞、CPE、CWE、POC 等数据
- 2022/03/1 – 2022/03/13 (两周) 学习 SpringBoot 和 vue 框架,分析漏洞知识间的关系;分析系统功能,完成系统的需求分析。
- 2022/03/14 – 2022/03/26 (两周) 使用基于规则或机器学习的方法抽取漏洞实体及关系,构建漏洞数据图模型,建立图数据库。完成系统的概要设计。
- 2022/03/27 – 2022/04/17 (三周) 完成系统详细设计以及部分代码实现,完成系统前端漏洞知识图的可视化、知识筛选等基本功能。接受中期检查。
- 2022/04/19 – 2022/05/02 (两周) 完善前端功能,完成前后端全部代码。
- 2022/05/03 – 2022/05/16 (两周) 进行系统测试、排错,及功能扩展、优化,编写文档记录。
- 2020/05/17 – 2022/05/30 (两周) 撰写毕业论文,完成毕业设计

指导教师签字	方维	日期	2022 年 3 月 2 日
--------	----	----	----------------

编号: _____

北京邮电大学本科生毕业设计（论文）成绩评定表

学生姓名	马嘉骥		所在学院	计算机学院（国家示范性软件学院）					
学号	2018211149	专业	计算机科学与技术		班级	2018211303			
论文题目	(中文) 基于漏洞知识图谱的可视化系统的设计与实现								
	(英文) Design and Implementation of Visualization System Based on Vulnerability Knowledge Graph								
指导教师姓名	方维	指导教师职称	副教授		指导教师单位	计算机学院（国家示范性软件学院）			
中期检查小组评分	(满分 10 分): 中期检查小组组长签字: 检查日期:								
指导教师评分	评价内容	具体要求			分值			评分	
	调研论证	能独立查阅文献和从事相关调研；能正确翻译外文资料；有收集、加工各种信息及获取新知识的能力和自学能力。			5	4	3.5	3	2
	方案设计	能独立提出符合选题的可行性研究方案、实验方案、设计方案，独立进行实验（如安装、调试、操作）和研究方案论证。			5	4	3.5	3	2
	能力水平	能综合运用所学知识和技能去分析与解决毕业设计（论文）过程中遇到的实际问题；能正确处理实验数据；能对课题进行理论分析，得出有价值的结论。			5	4	3.5	3	2
	学习态度	认真、勤奋、努力、诚实、严格遵守纪律，按期饱满完成规定的任务。			5	4	3.5	3	2
	设计（论文）水平	文题相符、综述简练完整，有见解；立论正确，论述充分，结论严谨合理；实验正确，分析处理科学；文字通顺，技术用语准确，设计（论文）有理论价值和应用价值。			5	4	3.5	3	2
	文本规范	装订顺序正确，字体字号等与基本规范相符，符号统一，编号齐全，图表完备、整洁、正确。			5	4	3.5	3	2
	指导教师评分合计 (满分 30 分): 评语:								
指导教师签字:					日期: 年 月 日				

复议	<input type="checkbox"/> 是 <input type="checkbox"/> 否 复议评分合计: _____ 复议人签字: _____ 复议日期: _____ 复议有权限修改指导教师评分, 选择复议后指导教师评分将由复议评分替换							
本科生毕业设计（论文）答辩成绩评定标准								
答辩小组成绩评定	评价内容	具体要求		分值			评分	
	选题	符合专业培养目标, 符合社会实际、结合工程实际, 难易适度, 体现新颖性、综合性。		5	4	3.5	3	2
	设计(论文)质量水平	全面完成任务书中规定的各项要求, 文题相符, 工作量饱满, 写作规范, 达到综合训练的要求, 有理论成果和应用价值。		20	16	14	12	8
	答辩准备	准备充分; 有简洁、清晰、美观的演示文稿; 准时到场。		5	4	3.5	3	2
	内容陈述	语言表达简洁、流利、清楚、准确, 思路清晰, 重点突出, 逻辑性强, 概念清楚, 论点正确; 实验方法科学, 分析归纳合理; 结论严谨; 表现出对毕业设计(论文)内容掌握透彻。		20	18	14	12	8
	回答问题	回答问题准确、有深度、有理论根据、基本概念清晰。		10	8	7	6	4
		答辩小组评分合计 (满分 60 分)						
	意见:							
	答辩小组组长签字: _____ 年 月 日							
	答辩小组成员:							
学院意见	最终成绩: 百分制 _____; 五分制 _____ 院长签章: _____ 学院盖章: _____ 年 月 日							
备注								

注: 1. 毕业设计(论文)成绩由中期检查评分(满分 10 分)、指导教师评分/复议评分(满分 30 分)和答辩小组评分(满分 60 分)相加, 得出百分制成绩, 再按 100-90 分为“优”、89-80 分为“良”、79-70 分为

“中”、69-60 分为“及格”、60 分以下为“不及格”的标准折合成五级分制成绩；

2. 此表原件一式三份，一份存入学生档案，一份装订到毕业论文中，一份交教务处存入档案馆。

北 京 邮 电 大 学

本科毕业设计（论文）诚信声明

本人声明所呈交的毕业设计（论文），题目《社交网络多媒体信息可信度评估》是本人在指导教师的指导下，独立进行研究工作所取得的成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：_____ 日期：_____

基于知识图谱的可视化系统的设计与实现

摘要

这是中文摘要的部分。

它可以拥有多段。这是中文摘要的部分。

它可以拥有多段。

如果你写的太长，甚至可以到第二页。

关键词 北京邮电大学 本科生 毕业设计 模板 示例

HAVE A TRY TO GUESS WHAT THE TITLE IS

ABSTRACT

This is ABSTRACT.

You can write more than one paragraph here.

If your abstract is too long, it will take up more pages.

KEY WORDS BUPT undergraduate thesis template example

目 录

第一章 絮论	1
1.1 项目背景及意义	1
1.1.1 选题背景	1
1.1.2 项目意义	1
1.2 开发目标	1
1.3 软件工程方法	2
第二章 技术选型分析	3
2.1 数据采集子系统	3
2.1.1 Scrapy	3
2.1.2 Pandas	3
2.2 知识图谱构建子系统	4
2.2.1 Ray	4
2.3 持久化子系统	4
2.3.1 MongoDB	4
2.3.2 Neo4j	5
2.4 后端服务子系统	5
2.4.1 Flask	5
2.4.2 Advanced Python Scheduler	6
2.5 前端子系统	6
2.5.1 Vue.js	6
2.5.2 Vuetify	7
2.5.3 Apache ECharts	7
第三章 系统需求分析	8
3.1 外部数据流分析	8
3.2 功能性需求分析	8
3.2.1 漏洞数据采集	8
3.2.2 漏洞数据处理	8
3.2.3 知识图谱构建	9
3.2.3.1 实体生成	9
3.2.3.2 关系生成	9
3.2.3.3 关系融合	9

3.2.4 知识图谱持久化存储	9
3.2.5 可视化展示知识图谱	10
3.2.6 日志功能	10
3.3 非功能性需求分析	10
3.3.1 性能需求	10
3.3.2 兼容性需求	10
3.4 用户视角的需求分析	10
第四章 系统设计	12
4.1 系统概要设计	12
4.1.1 系统总体设计	12
4.1.2 系统层次结构设计	13
4.1.2.1 表示层	13
4.1.2.2 逻辑层	15
4.1.2.3 持久化层	15
4.2 系统功能模块设计	15
4.2.1 数据采集子系统	15
4.2.2 知识图谱构建子系统	17
4.2.3 后端服务子系统	17
4.2.4 前端子系统	18
4.2.5 数据库设计	18
4.2.6 系统接口设计	18
4.3 系统详细设计	21
4.3.1 开发环境配置	21
4.3.2 各核心功能模块详细设计	21
4.3.2.1 类图	21
4.3.2.2 功能时序图	21
第五章 系统实现	25
5.1 工具模块	25
5.1.1 根目录模块实现 bot_root_dir.py	25
5.1.2 日志模块实现 logger_factory.py	25
5.1.3 环境配置文件 secret.py	25
5.2 数据采集子系统	26
5.2.1 Spider 模块实现	26
5.2.1.1 __init__() 实现	26
5.2.1.2 start_requests() 实现	26

5.2.1.3 parse() 实现	26
5.2.2 Item Pipeline 模块实现	27
5.2.3 其他模块实现	27
5.3 知识图谱构建子系统	27
5.3.1 实体生成	27
5.3.1.1 init_nodes() 实现	28
5.3.1.2 init_vul_ray() 实现	28
5.3.1.3 init_asset_ray() 实现	28
5.3.1.4 init_exploit_ray() 实现	28
5.3.1.5 init_asset_family_ray() 实现	29
5.3.2 关系生成	29
5.3.2.1 init_rels() 实现	29
5.3.2.2 create_rel_vaf_ray() 实现	29
5.3.2.3 create_rel_afa_ray() 实现	30
5.3.2.4 create_rel_evaf_ray() 实现	30
5.3.3 关系融合实现	31
5.4 持久化子系统实现	31
5.4.1 MyMongo 类实现	31
5.4.2 MyNeo 类实现	31
5.5 后端服务子系统实现	32
5.5.1 create_app.py 实现	32
5.5.2 graph.py 实现	32
5.5.2.1 /api/graph/ 实现	32
5.5.2.2 /api/graph/<limit> 实现	33
5.5.2.3 /api/graph/search/<keyword> 实现	33
5.6 前端子系统	33
5.6.1 项目结构	33
5.6.2 控制台视图实现	34
5.6.3 可视化视图实现	34
5.6.4 搜索视图实现	35
5.6.5 关于视图实现	35
第六章 系统测试	36
6.1 测试环境	36
6.2 数据采集子系统测试	36

6.3 知识图谱构建子系统测试	36
6.3.1 结点生成	36
6.3.2 关系生成	41
6.3.3 关系融合	41
6.4 数据库子系统与后端服务子系统测试	42
6.4.1 /api/graph/	42
6.4.2 /api/graph/<limit>	42
6.5 前端子系统测试	44
6.5.1 控制台视图测试	44
6.5.2 可视化视图测试	44
6.5.3 搜索视图测试	47
6.5.4 关于视图测试	48
第七章 结束语	49
参考文献	
致 谢	
附 录	

第一章 绪论

1.1 项目背景及意义

1.1.1 选题背景

近年来，随着互联网产业迅速发展，互联网安全漏洞问题的显著性也急剧增加。公共漏洞和暴露 (Common Vulnerabilities and Exposures, CVE) 等权威漏洞数据档案库的数据显示，自 1999 年该漏洞库首次披露安全漏洞以来，互联网安全漏洞年新增数量呈增长趋势。2019 年全年，新增漏洞两万余个；2020 年全年新增漏洞三万五千余个。随着现代软件系统复杂度提升、互联网加速漏洞信息传播，对攻击者而言，不仅漏洞攻击的学习成本和实施难度下降，其可以利用的漏洞数量也明显增多；对企业与开发者而言，随着开源化逐渐成为一种潮流趋势，各类计算机软硬件与互联网产品对开源项目的依赖性随之提高。正如近期 Java Log4j 日志组件漏洞造成全球互联网范围的大规模信息安全问题，计算机与互联网产业蓬勃发展的同时，也面临与日俱增的信息安全挑战。

1.1.2 项目意义

本课题针对上述问题，提出一种漏洞知识图谱可视化系统。基于爬虫抽取、知识图谱、图数据库、可视化前端等技术，对多种数据源的互联网公开漏洞数据，包括漏洞描述及风险评估、受影响资产、可利用代码及补丁等信息进行收集与分析，通过对异构数据源抽取的数据进行实体构建、关系构建、关系融合等处理，在图数据库中建立漏洞本体信息及其之间的关联信息，从而形成具有一定知识结构的知识图谱。基于该漏洞知识图谱，搭建基于 B/S 架构的可视化系统，提供易于使用的接口、用户友好的 UI 界面呈现漏洞知识图谱信息、进行创建统计图表、知识筛选等操作。

本系统采用自动化的方式，实现对漏洞信息的持续收集与整理，极大节省了人力资源的消耗。结合抽取关键信息建立互联网信息安全本体、对漏洞间关联性进行分析、构建漏洞知识图谱，将分散的漏洞信息转化为相互联系的图结构，本系统将为开发者提供项目依赖安全性参考、为计算机信息安全研究人员提供逻辑清晰、直观易于理解的统计数据与服务支撑，促进构建更高效安全的互联网环境。

1.2 开发目标

- 设计并实现对异构公开互联网信息漏洞数据源的信息采集系统。
- 设计并实现基于规则的漏洞知识图谱构建系统。
- 设计并实现基于图数据库的漏洞知识图谱持久化系统。
- 设计并实现基于上述漏洞知识图谱的后端服务，提供 RESTful API 访问点。

- 设计并实现基于 Web 服务的漏洞知识图谱可视化系统，具备独立前端。
- 对上述系统进行系统测试、排错、功能扩展、性能优化，编写文档记录。

1.3 软件工程方法

由于本项目为单人完成，且系统整体结构较为复杂，故采用原型开发与敏捷开发^[1]相结合的方式，先开发最简可行产品^[2]（Minimum Viable Product, MVP）验证核心概念，在此基础上不断细化细节、完善代码、扩展功能。同时，因为单人项目难以在前期进行完善的需求分析与系统设计，采用敏捷开发的方式可以增强灵活性。

例如，本项目立项初期技术选型，计划使用较为成熟的 Java Springboot 框架搭建后端服务。在之后的开发中发现，由于该项目对数据库操作需求较高，其他子系统使用 Python 而后端使用 Springboot 意味着需要使用 Python 和 Java 分别编写两套代码用于控制数据库驱动，增加无意义的项目复杂度、拖慢开发进度。得益于敏捷开发思想“响应变化高于遵循计划”等方针，本项目在知识图谱构建子系统的原型开发完成时即更改后端使用 Python Flask 进行开发，从而保证软件质量的同时提升开发效率。

第二章 技术选型分析

根据系统开发目标，可将系统层次结构从信息处理逻辑上分为三个层次：用于持久化存储信息的持久化层，用于对信息进行采集、处理、传输的逻辑层，用于面向用户展示信息的表示层；从功能上分为五个子系统：数据采集子系统、知识图谱构建子系统、持久化子系统、后端服务子系统、前端服务子系统。

本章从该五个子系统的角度分别介绍本项目使用的相关技术。

2.1 数据采集子系统

采用的技术或工具：Scrapy、Pandas

2.1.1 Scrapy

Scrapy¹ 是一个基于 Twisted 的自由且开源的协作式网络爬虫 Python 框架，用于从网站或 API 中提取需要的数据。具有快速、易用、可扩展等特性，主要优点是架构清晰、模块间的耦合程度低、通过 Middleware 钩子框架能灵活完成各种需求。Scrapy 项目围绕 Spider 类构建，并具有 Item Pipelines、Downloader、Scheduler 等多个模块，这些模块通过内置的 Scrapy Engine 进行管理与调度。Scrapy 设计理念遵循“一次且仅一次”(Once and only once, OAOO) 原则，主张为每个爬虫任务只需要编写一个自包含 (self-contained) 的 Spider 类，没有更多，没有更少。

相较于 urllib 或 Python Requests 等基础的库，Scrapy 提供任务日志、错误重试、并发控制，具备更灵活完善的机制以应对大型爬虫任务。而相较于 Selenium、Puppeteer、Playwright 等基于无头浏览器 (Headless Browser) 实现的爬虫功能，Scrapy 基于事件驱动的网络编程框架 Twisted，仅爬取 HTML 文档，默认不提供执行网页中的 JavaScript 代码功能，因此 CPU 与内存开销相较无头浏览器显著减小，是最适合本系统大量数据爬取需求的框架。

2.1.2 Pandas

Pandas² 是适用于 Python 语言的数据操纵与分析库，是基于 BSD 许可证发行的自由且开源的软件，尤其擅长数值表格与时间序列的数据结构和运算操作。

本数据采集子系统将从 cve.mitre.org 爬取其提供的包含全部 cve id 信息的 csv 文件，其中包含数十万条 cve 漏洞 id 条目。由于该 csv 文件包含表头信息及一些无效或重复的 cve id，在建立用于从异构数据源爬取项目的 cve id 索引时需要将无关信息删去。该 csv 文件体积较大（约数百兆字节），因此使用 Pandas 提供的流式读取 csv 文件、迭代操作的方式处理此文件。

¹<https://scrapy.org/>

²<https://pandas.pydata.org/>

2.2 知识图谱构建子系统

采用的技术或工具：基于规则的实体构建、基于规则的关系生成、Ray。实体构建与关系生成技术将在章节 4.2.2 知识图谱构建子系统展开。

2.2.1 Ray

Ray¹ 是一个开源的通用分布式计算框架，由加州大学伯克利分校的 RISE 实验室开发，为构建分布式应用提供了一个简单、通用的 API。Ray 提供了运行机器学习工作流的 Ray ML 工具箱、运行分布式应用的 Ray Core 核心框架、部署大规模工作负载的 Ray Cluster 集群等，具有通用的分布式计算抽象以及优秀的性能。

相较传统的分布式框架（如 Hadoop、Spark 等），Ray 可直接通过 pip 进行安装，具备轻量级的特性。同时它通过共享内存实现了高效的数据存储和传输，通过全局状态存储服务实现了全局的状态维护、去中心化的高效调度、远程调用。相较基于线程的并行库 threading，Ray 通过进程级并行绕过了 Python 全局解释器锁的限制。相较轻量级的 Python 进程级并行库 multiprocessing，Ray 具备同样简单易用的接口以及更优秀的性能与机器学习框架整合能力。

本项目通过使用 Ray 框架提供的通用分布式计算能力，对上层封装具体实现；将知识图谱构建过程抽象成创建结点、创建关系、关系融合三个操作，重点优化这三个步骤的代码实现，使“数据采集子系统得到的原始数据通过知识图谱构建子系统形成知识图谱”过程得以实现并行化计算，充分利用多核心处理器的计算能力。同时，得益于 Ray 框架对本地多进程与分布式多进程的统一抽象、Ray ML 框架与深度学习模型良好的整合能力，未来可以引入深度学习模型、将系统迁移扩展至分布式计算平台。

2.3 持久化子系统

采用的技术或工具：MongoDB、Neo4j

2.3.1 MongoDB

MongoDB² 是一个 C++ 语言编写的基于分布式文件存储的开源文档型数据库系统，是最受欢迎的 NoSQL 数据库。MongoDB 基于 JSON/BSON 格式的文档存储，相较传统 SQL 数据库可以表示灵活的数据结构；具备动态 DDL 能力、没有强 Schema 约束的特性使其支持开发者进行快速迭代；提供基于内存的快速数据查询，实现高并发计算性能；提供数据分片能力，分布式扩展性强。

本项目需求从异构数据源采集互联网信息安全漏洞数据，导致数据格式繁多，在项目初期规定数据锁采用的 Schema 格式将大幅增加数据转换工作量，并不现实。因此，传

¹<https://docs.ray.io/en/latest/>

²<https://www.mongodb.com/>

统 SQL 关系型数据库并不适合本项目的开发需求。而 MongoDB 提供面向文档的 JSON 存储，适配互联网爬虫数据采集需求。同时，JSON 文件具备良好的序列化能力，且与本项目后端服务子系统 RESTful API 所需的 Application/JSON Response 相匹配。更进一步地，MongoDB 具有优秀的分布式能力。综上原因，采用 MongoDB 作为存储知识图谱所用原始数据的数据库。

2.3.2 Neo4j

Neo4j¹ 是一个高性能的图形数据库，是目前全球范围内最受欢迎、使用人数最多的图数据库，社区版本采用 GPLv3 许可证授权。其特点是将结构化数据存储在网状结构上，使得长程、广范围关系查询变得更加容易实现，Neo4j 是目前使用最多的图数据库。本项目知识图谱构建子系统使用 Neo4j 存储图谱数据，将漏洞、资产、利用代码等实体作为结点，为结点之间添加“影响”、“具有”、“攻击”、“被攻击”、“父级”、“子级”等关系，形成知识网络图谱。

2.4 后端服务子系统

采用的技术或工具：Flask、Advanced Python Scheduler

2.4.1 Flask

Flask² 是一个使用 Python 编写的轻量级 Web 应用框架，具备微核心与高扩展性，使用 BSD 许可证授权。Flask 是在 Jinja2 模板引擎和 Werkzeug WSGI 工具箱的基础上构建的。Werkzeug 是一个语言网络服务器网关接口（Web Server Gateway Interface, WSGI），为请求、响应等功能实现软件对象；Jinja 是一个适用于 Python 的网页模板引擎。

本项目主要使用 Flask 框架的 WSGI 功能，构建一个 RESTful 风格的 Web API 服务，为 Vue.js 编写的前端提供与数据库的交互功能。本着“高内聚，低耦合”的设计原则，同时兼具部署简洁、外部依赖较少、用户控制灵活的优点，本基于漏洞知识图谱的可视化系统将前述数据采集子系统、知识图谱构建子系统抽象成自包含的服务，仅通过持久化层的 MongoDB 及 Neo4j 数据库进行纯数据交换，最大程度上减少模块间逻辑耦合。

由于数据采集子系统、知识图谱构建子系统、持久化子系统、后端子系统均采用 Python 语言编写，此四个子系统得以共用一个高扇入的数据库驱动模块。这种设计将会导致单个数据库控制类非常庞大，在本项目开发后期该数据库控制类相关代码已接近千行，造成运行时实例化开销昂贵。并且，基于知识图谱数据量庞大的事实，本系统采用由加州大学伯克利分校牵头开发的开源分布式高性能计算库 Ray，实现知识图谱数据构建的并行计算并提供可能的分布式扩展性。为此，数据库控制类需要实现线程安全。

¹<https://neo4j.com/>

²<https://flask.palletsprojects.com/en/2.1.x/>

为解决此问题，针对本系统将频繁进行数据库操作的预期，利用 Python 语言引入模块为单例的特性实现了一个“饿汉模式”的数据库控制类。在其他上级模块中引入数据库控制器类的实例化对象，使每个模块运行时在内存中只有一个实例，减少运行时频繁地创建和销毁类实例的开销。相较使用 Java Springboot 等其他语言框架编写后端服务，采用 Python Flask 并共用单例数据库控制模块的设计在减少重复编码工作的同时，极大增强了代码可维护性。

2.4.2 Advanced Python Scheduler

Advanced Python Scheduler¹(下称 APScheduler)是一个基于 Python 的定时任务库，提供 Cron 风格、间隔执行、单次延时执行三种任务规划模式；内存、SQLAlchemy、MongoDB、Redis 数据库等任务存储方式；asyncio、Twisted、Qt 等 Python 框架集成。APScheduler 并非命令行工具，而旨在向运行中的应用程序增加定时任务功能，且具备 Python 跨平台的优势，因此非常适合作为 Python 后端的定时任务模块。

后端服务作为一个高扇出模块，是整个系统的运行控制中心，同时控制后端 Web API 服务、数据采集服务、知识图谱构建服务。根据本项目需求，为实现知识图谱的动态更新，在 Flask 后端服务中集成 APScheduler 库用于控制和管理数据采集服务、知识图谱构建服务，实现周期性定时运行数据采集与知识图谱构建的功能。相较使用 crontab、Windows Task Scheduler 等依赖操作系统运行环境的外部工具，采用 Flask + APScheduler 进行服务运行管理的设计更加优雅简洁，在保持优秀可迁移性之外，还允许管理员用户透过 HTTPS 协议在前端子系统提供的 Web 应用中对系统运行状态进行远程管理，无需接触物理机或使用 SSH 等协议连接至终端。该设计增强系统安全性并提高了系统易用度。

2.5 前端子系统

采用的技术或工具：Vue.js、Vuetify、Apache ECharts

2.5.1 Vue.js

Vue.js² 是一套用于构建用户界面的开源 MVVM 渐进式前端 JavaScript 框架，使用 MIT 协议授权。通过 Vue.js 提供的模板语法、计算属性和侦听器、条件渲染、组件等功能，可以构建现代化的响应式前端页面。Vue 既可以驱动一套完整的单页应用，也易于与第三方库或既有项目进行整合。

本项目使用 Vue.js 作为前端逻辑的基础框架，结合 Vuex³、Vue Axios⁴、Vue Router⁵、

¹<https://apscheduler.readthedocs.io/en/latest/>

²<https://v3.cn.vuejs.org/>

³<https://vuex.vuejs.org/zh/>

⁴<https://www.npmjs.com/package/vue-axios>

⁵<https://router.vuejs.org/>

register-service-worker¹ 等实用库，构建一套现代化的前端页面，具备响应式、单页应用 (Single Page Application, SPA)、渐进式网页应用 (Progressive Web Application, PWA) 等特性。

2.5.2 Vuetify

Vuetify²是一个 Material 样式的 Vue UI 组件库，使用 MIT 协议授权。它包含一系列预定义样式的 UI 组件，可用于构建前端应用程序的用户界面。

本项目前端子系统采用 Vuetify 作为 UI 库，致力于搭建一个用户友好、信息密度大、界面简洁易用的前端应用。借助 Vuex 状态管理库及 Vue.js 计算属性，实现尽可能减少跨组件回调函数使用的“Single Source of Truth”模式单向数据流的前端应用。降低组件间逻辑耦合，实现快速开发、轻松扩展。

2.5.3 Apache ECharts

Apache ECharts³（下称 ECharts）是一个自由且开源的 JavaScript 可视化库，采用 Apache License 2.0 协议授权。ECharts 基于轻量级绘图库 zrender，为浏览器提供直观的、强大的、可交互的、高度自定义的数据可视化功能。

本项目使用 ECharts 中力引导图绘制可视化的知识图谱并提供一定交互功能；使用各类数据统计图，直观地呈现知识图谱的相关统计数据。借助 Vue.js 的模板组件与监听属性能力，实现可复用、响应式的可视化数据呈现能力。

¹<https://www.npmjs.com/package/register-service-worker>

²<https://vuetifyjs.com/zh-Hans/>

³<https://echarts.apache.org/en/index.html>

第三章 系统需求分析

产品需求分析是针对即将开发的软件施加的一种要求或限制。软件产品需求可分为功能性需求、非功能性需求等。^[3]

3.1 外部数据流分析

- 系统输入：来自多种数据源的漏洞相关异构信息数据，包括 html 文件、csv 文件、json 文件、xml 文件等。
- 系统输出：以文本、统计表、统计图、交互式可视化图等形式呈现的知识图谱信息。

3.2 功能性需求分析

3.2.1 漏洞数据采集

本系统需要构建互联网信息安全漏洞相关的知识图谱，为此需要一定的数据源提供整个系统的输入。由于安全感知态势千变万化、CVE 漏洞数据时刻更新，本系统需要实现自动化数据采集功能，向数据库中动态添加数据，而非将现有数据一次性导入。调研筛选出待爬取的网站有：

- cvedetails.com，一个以表格形式简单聚合 cve 漏洞信息的网站。
- cve.mitre.org，一个官方发布 cve 漏洞命名的网站。
- cpe.mitre.org，一个官方发布 cpe 资产信息的网站。
- nvd.nist.gov，一个发布 cve 详细风险评估及受影响 cpe 资产统计的数据库。
- exploit-db.com，一个提供漏洞利用代码的数据库。

3.2.2 漏洞数据处理

数据采集得到的输入数据不满足构建漏洞知识图谱所需数据结构或模式。例如：cvedetails.com 采集到的数据为 HTML 文档，而其中有效信息分散在 HTML DOM 树中；nvd.nist.gov 提供官方 JSON API 接口，采集得到的数据为 JSON 格式，但包含版本控制数据等与漏洞本身无关的信息；在 cve.mitre.org 得到的数据为独立 csv 文件，等。因此需要实现从采集到的数据转化为知识图谱数据的处理过程。

3.2.3 知识图谱构建

知识图谱是一种用图表表示的数据结构，旨在积累并表达现实世界的知识。知识图谱的图结点代表我们所研究的实体，结点之间的边代表实体之间的关系。用于存储知识图谱数据的图遵从基于图结构的数据模型，如有向的边标记图或属性图。^[4]

知识图谱的一种通用表示形式是三元组形式，即

$$G = (Entity_{head}, Relation, Entity_{tail})$$

$Entity_{head}$ 为三元组 G 中的头实体， $Entity_{tail}$ 为三元组 G 的尾实体， $Relation$ 则表示 $Entity_{head}$ 到 $Entity_{tail}$ 的关系。

3.2.3.1 实体生成

为了构建漏洞知识图谱，需要从爬虫获取的互联网信息安全漏洞数据中获得所需要的实体与关系。首先需要进行实体生成。如前所述，图谱中一个结点代表现实世界中一个实体，适用于此处即为一个与漏洞关联的实体，如漏洞条目、软件资产、硬件资产、资产家族、利用代码等。

由于实体分属于不同种类，因此需要标签（label）进行区分。每个实体还应该具有不同的属性，这些属性应该被存储到图数据库结点中。

3.2.3.2 关系生成

在有向边标记知识图谱中的关系通常只具有一些简洁的标签（label），代表 $head$ 结点是 $tail$ 结点的 $label$ 。^[4] 通过这种方式，可以将复杂的关系拆分成若干串简单关系链组成的网络。

对于已有的实体结点，需要为结点之间添加关系，以反映它们在现实世界中的联系。这一步骤既可以在实体生成时执行一个多次迭代的算法计算节点之间的关系，也可以在所有实体生成完毕后按照某一索引（在此情况下，是 cve id）对结点进行遍历生成关系。

3.2.3.3 关系融合

关系融合模块作为适配使用深度学习模型进行命名实体识别结点生成与关系抽取方案的模块，当前仅在系统数据加工管线中留有接口，以备未来扩展需要。当前系统的结构化数据实体与关系生成已经实现实体结点和关系的消除二义性。

3.2.4 知识图谱持久化存储

经前期调研，cve.mitre.org 现有 cve 条目约 16 万条，涉及漏洞条目、资产、漏洞利用等数据为百万数量级，且这些数据每日不断更新。因此，本系统需要实现一个高性能

的数据持久化子系统，用于存取更新图谱数据。

3.2.5 可视化展示知识图谱

作为一个可视化系统，面向用户的最主要需求是以文本、统计表、统计图、交互式可视化图等形式呈现的知识图谱信息。可视化系统需要实现：

- 知识图谱的相关信息展示，如漏洞数量、资产数量、利用代码数量；
- 各种种类漏洞百分比统计；
- 各种受影响资产类型百分比统计；
- 利用代码执行类型百分比统计；
- 按时间排序的最近漏洞信息、趋势统计；

等等。

3.2.6 日志功能

缺乏日志功能将无法跟踪系统运行轨迹，给运维带来困难。记录清晰定位准确的日志系统有效提升开发效率，对于较复杂系统而言是不可或缺的需求。

3.3 非功能性需求分析

3.3.1 性能需求

本系统应具备处理大量图数据的能力，并且具备良好的架构以灵活应对不断增长的数据量。这要求本系统能在可以接受的时间内完整建立知识图谱，并且能不断采集信息以更新自身知识图谱数据，因此数据采集及处理周期不应大于源数据更新周期。

3.3.2 兼容性需求

好的系统具备优秀的可移植性。本系统期望达到良好的可移植性，因此应尽可能采用平台无关的语言及框架如 Python、Java 等。可以使用 Advanced Python Scheduler 来避免使用 crontab 之类平台特定命令工具。且本系统使用网页前端作数据展示功能，可视化功能应保证在多种现代浏览器上正常运行。

3.4 用户视角的需求分析

- 查看控制台
 - 查看图谱统计数据

- 查看图谱统计图表
- 查看知识图谱
 - 拖拽浏览图谱
 - 点击结点或关系查看详情
- 搜索实体
 - 搜索漏洞
 - 搜索资产
 - 搜索代码利用
 - 搜索关系
- 利用代码执行类型百分比统计；
- 按时间排序的最近漏洞信息、趋势统计；

根据上述需求，图 3-1 即为本系统用户视角需求分析的用例图。

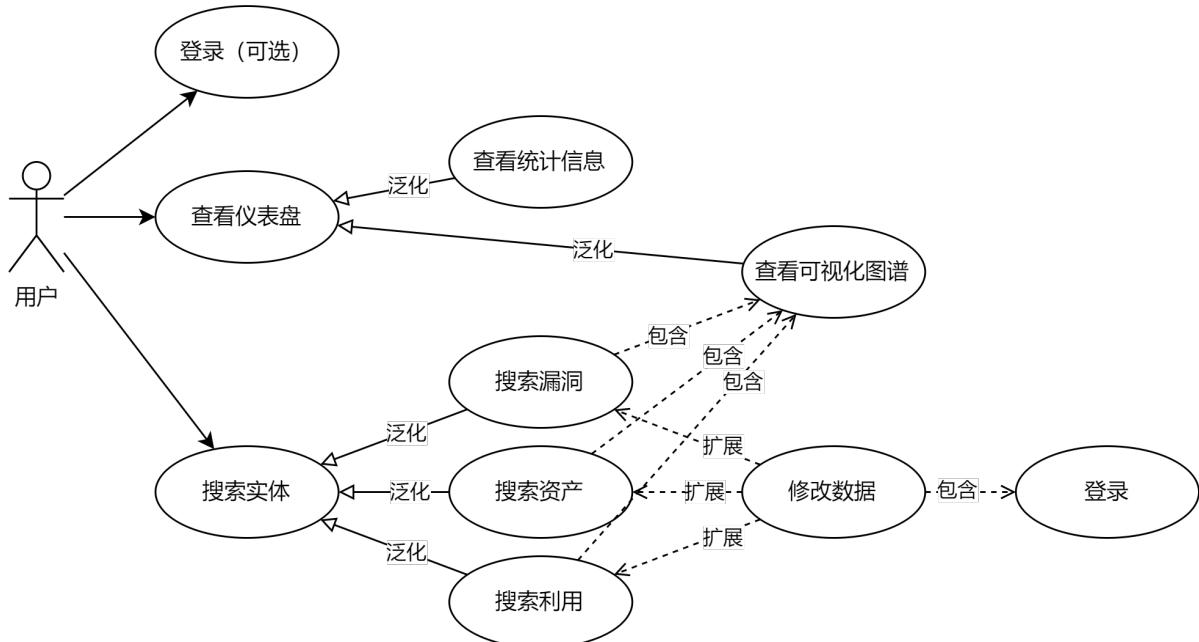


图 3-1 系统用例图

第四章 系统设计

4.1 系统概要设计

4.1.1 系统总体设计

系统数据流图如图 4-1 所示。

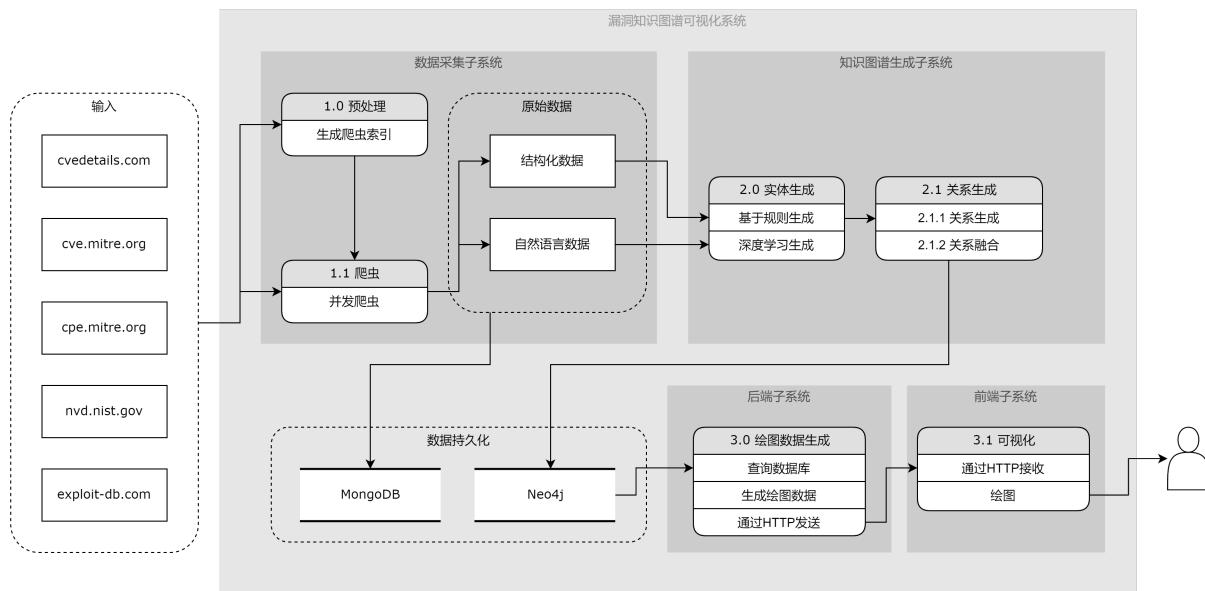


图 4-1 系统数据流图

系统输入为来自多个数据源¹的异构信息安全漏洞数据，分别经过数据采集子系统、知识图谱构建子系统、后端服务子系统、前端子系统的加工，最终流出系统，呈现给用户。其中：

1. 数据采集子系统的加工细化为：

- (a) 预处理：对获取到的 cve 目录、cpe 目录等文件，使用 Pandas 进行筛选、过滤、重新格式化、存储至 MongoDB 数据库，得到爬虫模块所需的索引文件。
- (b) 爬虫：使用 Scrapy 进行并发爬虫，从前述多个数据源获取数据。
- (c) 数据解析：对爬虫得到的网络数据，编写 Scrapy 数据管线（Item Pipeline）进行解析与格式化，得到与互联网信息安全漏洞相关的结构化数据与自然语言数据，作为原始数据。

2. 知识图谱构建子系统的加工细化为：

- (a) 实体生成：通过基于规则的或基于深度学习模型的方法，对前述爬虫得到的原始数据进行命名实体识别，提取所需实体，及其相关属性。从 nvd.nist.gov

¹cvedetails.com、cve.mitre.org、cpe.mitre.org、nvd.nist.gov、exploit-db.com 等。

获取的漏洞数据区块提取 cve 条目的 cve id 与危险等级等属性、从资产区块提取受影响资产的 cpe23uri 匹配属性、从 exploit-db.com 爬取的页面中提取漏洞利用代码、代码类型、影响平台等属性。

(b) 关系生成：为实体结点之间添加关系，从若干孤立结点形成网状结构。本加工步骤利用前述不同实体类别隐含的逻辑关系进行关系生成。如：

- 资产 -[具有]-> 漏洞，漏洞 -[影响]-> 资产
- 资产 -[是子级]-> 资产家族，资产家族 -[是父级]-> 资产
- 利用代码 -[攻击]-> 资产，资产 -[被攻击]-> 利用代码
- 利用代码 -[利用]-> 漏洞，漏洞 -[被利用]-> 漏洞代码

(c) 关系融合：消除具有二义性的实体结点与关系边，仅需留有接口。

3. 后端服务子系统的加工细化为：

- (a) : 查询数据库，将数据转换成 Python 数据结构。
- (b) : 将数据重新组织、添加前端绘图库所需的属性，生成绘图数据。
- (c) : 将绘图数据序列化成 JSON 字符串，通过 HTTPS 协议发送至前端。

4. 前端子系统的加工细化为：

- (a) : 通过 HTTPS 协议从后端接收绘图数据，反序列化成 JSON 对象。
- (b) : 调用绘图库使用接收到的绘图数据，在 canvas 上绘制图形。

4.1.2 系统层次结构设计

系统层次结构图如图 4-2 所示。系统可以分为三层：表示层、逻辑层、持久化层。表示层包含前端用户界面；持久化层包含 Neo4j 与 MongoDB 两个数据库；逻辑层包含后端服务、知识图谱构建服务、数据采集服务。借鉴微服务思想，逻辑层之间各服务通过持久化层传递数据，减少服务间耦合度。

4.1.2.1 表示层

表示层是用户与本系统交互的介面。用户与前端用户界面交互，从而获得信息。表示层在前端具备简单的数据处理能力，用于对从后端接收到的数据进行处理与转换。例如，将后端提供的适用于扇形图的数据转换为适用于柱状图的相同数据。这种设计可以充分利用前端设备的计算能力，减轻后端服务器负载，并且有助于减少 API 数量，降低复杂度。

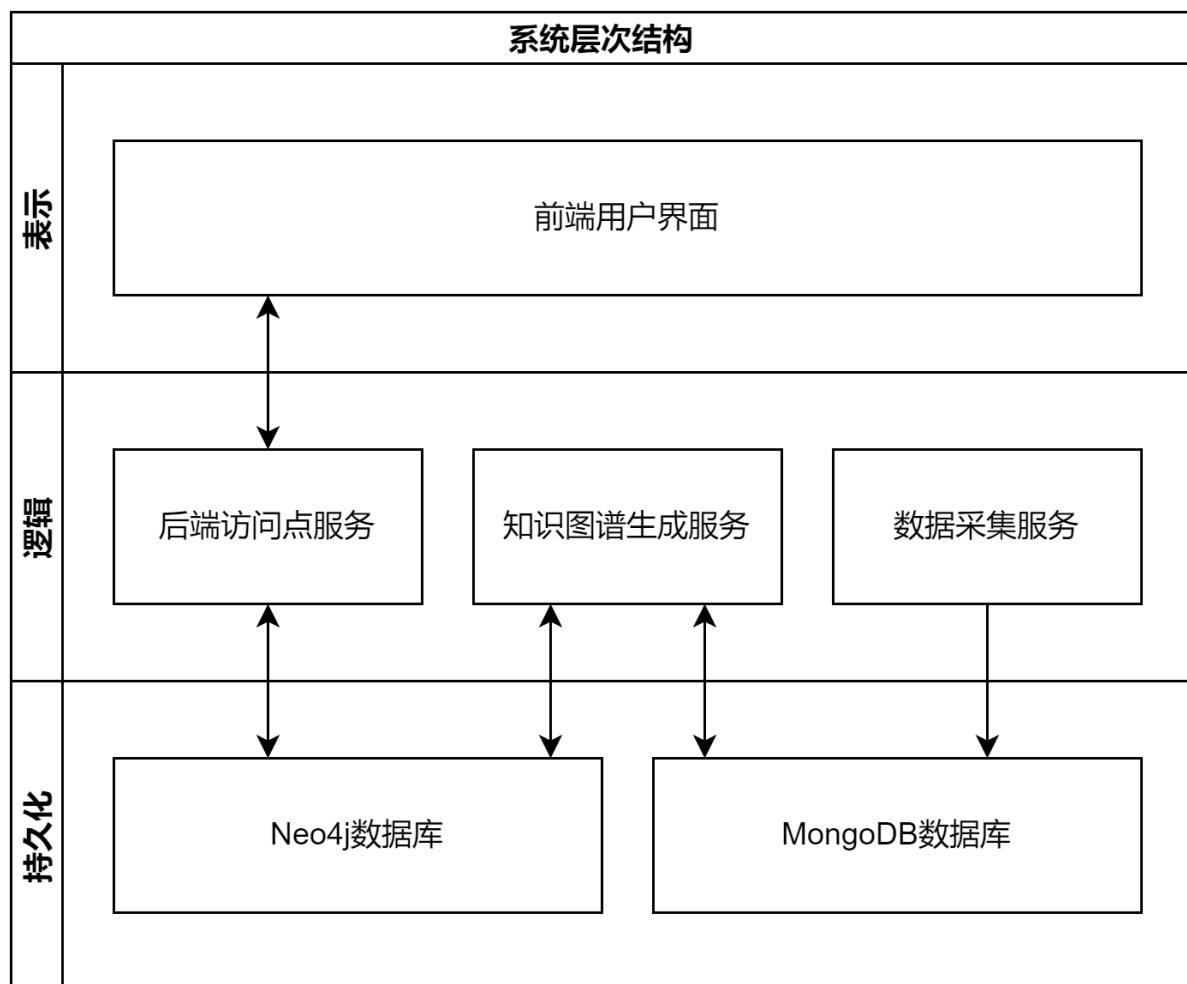


图 4-2 系统层次结构设计图

4.1.2.2 逻辑层

逻辑层实现系统核心功能，响应表示层传来的用户指令并且回传所需数据，通过与持久化层交互使数据从输入源不断被加工，最终输出给用户。同时，负责整个系统调度工作。

数据采集服务：通过生成索引、爬虫、数据清洗，向持久化层输入数据。数据采集服务在持久化层存储的数据包括用于指引爬虫生成新链接的索引、爬虫从各数据源爬取得到的 HTML 文档、JSON 等原始数据用于存档、经过数据管线处理得到的可用于下一步构建知识图谱的 JSON 格式化数据。数据采集服务与持久化层 MongoDB 进行交互，是系统数据输入界面。

知识图谱构建服务：从 MongoDB 数据库读取格式化数据，经过实体生成、关系生成等步骤，构建互联网信息安全漏洞的知识图谱。知识图谱构建服务与持久化层 MongoDB 和 Neo4j 进行交互，实现原始数据的加工，将图谱相关结点、关系、属性数据存储至 Neo4j 数据库中。//TODO

后端服务：与表示层通过 RESTful API 交互，获取存取数据、服务调度等用户指令。与持久化层 Neo4j 数据库进行加护，读取知识图谱相关结点、关系、属性数据，将其通过 HTTPS 协议传递给表示层。此外，后端服务还具有调度功能，通过 APScheduler 调度逻辑层其他服务的运行，从而保障整个系统正常运转。

4.1.2.3 持久化层

持久化层主要包含 MongoDB 数据库与 Neo4j 数据库。

MongoDB 数据库在系统中具备两种功能：存档数据采集服务获取的数据以供知识图谱构建服务进行加工；利用 MongoDB 内存技术特性缓存后端服务向 Neo4j 数据库进行大规模查询得到的结果，前端服务频繁查询直接读取缓存数据，减轻 Neo4j 数据库负载，缓存定时更新。

Neo4j 数据库在系统中主要发挥互联网安全信息漏洞知识图谱的承载功能。

4.2 系统功能模块设计

图 4-3 展示了除持久化子系统外，本系统各个子系统的功能模块设计。系统分为五个子系统：数据采集子系统、知识图谱构建子系统、后端服务子系统、前端子系统。

4.2.1 数据采集子系统

主要实现 Web 爬虫服务，包括请求生成器、异构数据解析器、异构数据管线三个模块。其中异构数据解析器包含包含 HTML 解析器、JSON 解析器、csv 解析器、gzip 解析器等。异构数据管线包含与异构数据解析器对应的各类数据管线。

请求生成器根据 csv 索引生成爬虫请求链接，交由 Scrapy Spider 进行并发爬虫。

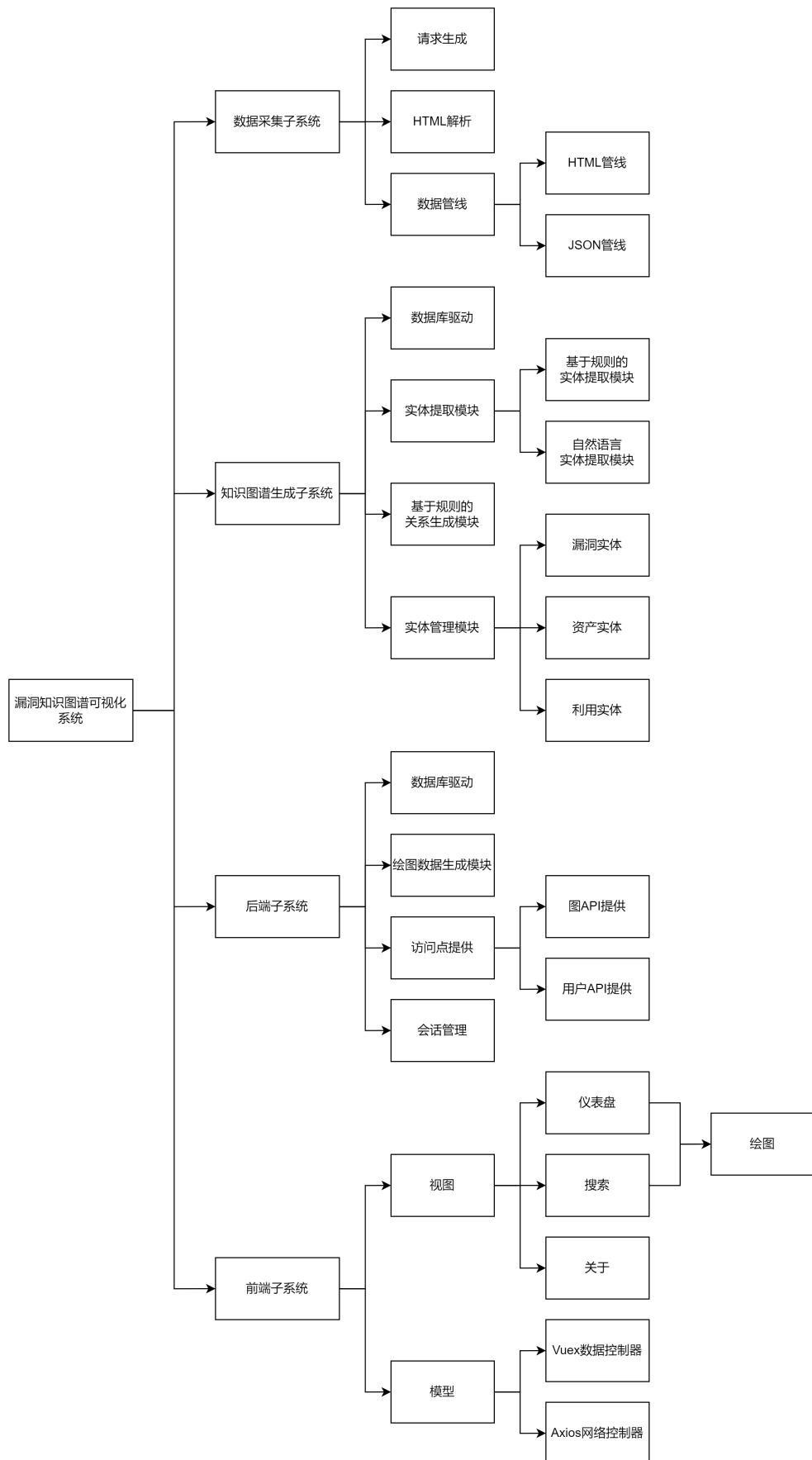


图 4-3 系统功能模块设计图

异构数据解析器负责使用对应解析器解析 Spider 爬虫返回的网络数据，传递给异构数据管线。此外，异构数据解析器中的 HTML 解析器应用在一些网站的爬虫任务时，还会解析页面上包含“下一页”等元素的 `<a>` 标签，通过 `yield` 将其中 `href` URL 交给请求生成器，生成新的爬虫请求。

异构数据管线负责接收其对应的数据解析器传来的数据，提取其中需要的信息进行格式转换等处理，填充固定格式的 Python Dictionary，再调用持久化子系统中的数据库控制器对象，将请求原始数据与 Dictionary 分别存储至不同的 MongoDB 数据库集合（Collection）中持久化存储，形成原始数据，完成数据采集工作。

4.2.2 知识图谱构建子系统

主要实现知识图谱构建服务，包含实体生成模块、基于规则的关系生成模块、关系融合模块。

实体生成模块目前包含基于规则的实体生成子模块，未来可引入基于深度学习模型的自然语言实体生成子模块。实体生成模块从原始数据中提取资产类型、属性等信息，调用持久化子系统的数据库控制对象将结点存入 Neo4j 数据库。具体而言，分为漏洞结点生成、资产结点生成、利用代码结点生成三部分，分别生成漏洞结点、资产结点与资产家族结点、利用代码结点。

基于规则的关系生成模块利用不同类型实体间隐含的联系，在 Neo4j 数据库中匹配对应结点，在两个结点间生成对应的边，并为其添加属性。因知识图谱中预计将有一百万个结点，使用最细粒度将每个漏洞结点与每个资产结点连接将产生超过一亿条边，这对于本项目的性能要求而言是不现实的。因此，关系生成模块会使用前述步骤生成的资产家族结点，在漏洞结点与漏洞结点所包含的受影响资产列表中的每条正则匹配项所匹配到的资产对应的资产家族之间添加关系。为了维持该关系指向资产的语义准确，将匹配到的资产列表作为属性添加到该关系边之上。

关系融合模块作为适配使用深度学习模型进行命名实体识别结点生成与关系抽取方案的模块，当前仅在系统数据加工管线中留有接口，以备未来扩展需要。当前系统的结构化数据实体与关系生成已经实现实体结点和关系的消除二义性，因此无需实现关系融合模块。

4.2.3 后端服务子系统

主要负责提供后端 Web API、对知识图谱的数据进行转换处理、调度各服务运行。包括绘图数据生成模块、Web API 访问点、APScheduler 调度模块。

Web API 访问点由 Flask 支持，使用 Flask 的 Blueprint 功能对注册在 RESTful API 根路径下的第一级 API 进行分组管理，实现模块化设计。在某个 Route 收到请求时，根据用户请求内容，将请求参数作为 CQL 参数，调用持久化层数据库控制对象对数据库进行操作，增删查改知识图谱中的结点和关系。若是可视化相关的 Route，则会对查询到的数据调用绘图数据生成模块进行转换。

绘图数据生成模块将存储在 Neo4j 数据库中的知识图谱数据转化为 ECharts 绘图库可以使用的数据格式。

APScheduler 调度模块: //TODO

4.2.4 前端子系统

前端子系统负责管理用户运行时相关数据、通过 UI 向用户展示信息。使用 Vue.js 的 MVVM 架构，主要包含视图与模型两个组成部分。

视图包含控制台、可视化、搜索、关于四部分。控制台视图为用户提供知识图谱信息概览，包含列表形式的数据展现，以及统计图表形式的数据可视化。可视化视图是一个全屏视图，通过该视图用户可以一览知识图谱全貌。搜索视图只是用户在图谱中搜索感兴趣的内容，并将搜索结果以力引导图可视化的形式展示。关于视图展示本系统相关信息及用户指南。

模型主要包含 Vuex 数据控制器、Axios 网络控制器两部分。Vuex 数据控制器统一管理前端子系统的数据与状态，是前端子系统的“Single Source of Truth”，为视图的响应式显示提供数据源，还包含在前端对数据进行简单处理功能。Axios 网络控制器封装一个 httpClinet 对象，以及若干与后端通信的方法。Vuex 数据控制器可以调用 Axios 网络控制器，从后端获取数据，更新 Vuex Store 中的状态。

4.2.5 数据库设计

图 4-4 所示为 MongoDB 数据模式图。//TODO

图 4-5 所示为 Neo4j 数据模式图，同时也是知识图谱实体与关系模式图。

4.2.6 系统接口设计

图 4-1 展示了系统接口设计。

表 4-1 系统接口设计

类别	路径	描述
	/<path>	前端路由路径
	/api/graph/	获取知识图谱统计信息
前端	/api/graph/<limit>	获取与 <limit> 个漏洞结点相关联的结点或关系组成的子图谱
	/api/search/<keyword>	搜索与 <keyword> 关键字相关的结点或关系

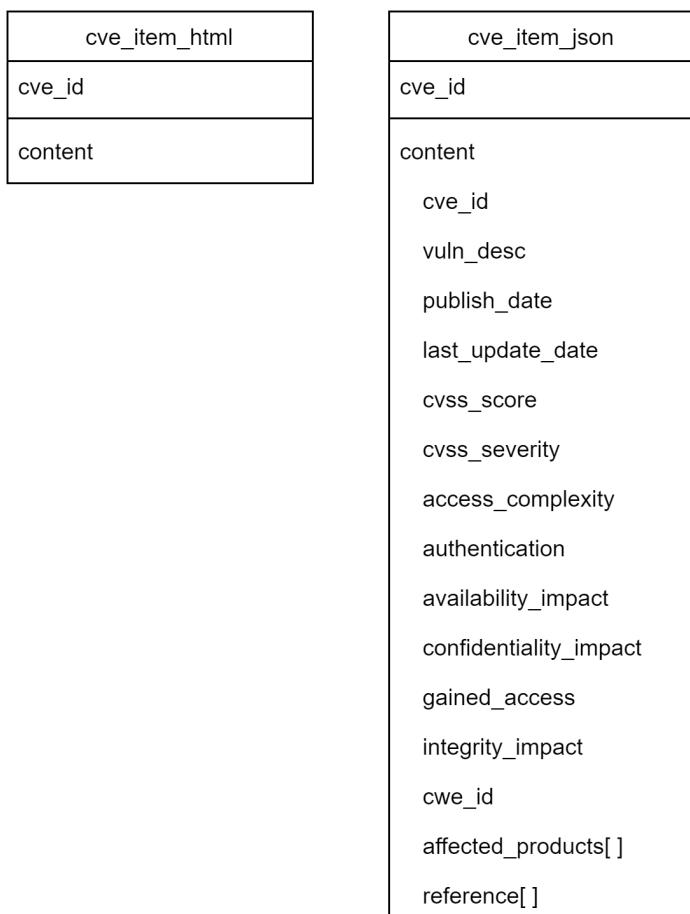


图 4-4 MongoDB 数据模式图

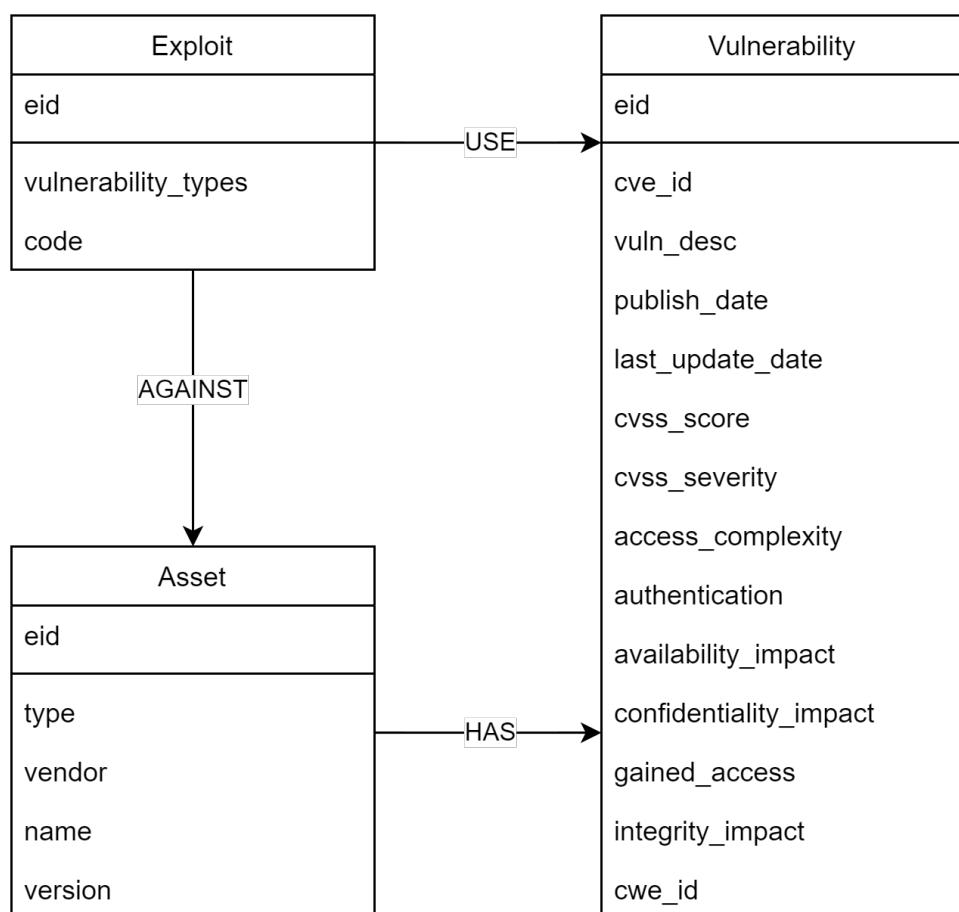


图 4-5 Neo4j 数据模式图

4.3 系统详细设计

4.3.1 开发环境配置

- 操作系统: Ubuntu 20.04, Windows 11 Pro
- 数据采集子系统: Python 3.8, Scrapy 2.6
- 知识图谱构建子系统: Python 3.8, Ray 1.11.1, uuid 1.3
- 持久化子系统: Python 3.8, Py2neo 2021.2.3, Pymongo 3.12, Neo4j 4.4
- 后端服务子系统: Python 3.8, Flask 2.0, flask-cors 3.0
- 前端子系统: JavaScript ES6, Vue 2.6, Vuetify 2.6, ECharts 5.3.2, vue-axios 3.4, Vuex 3.4, Moment 2.29.3, Lodash 4.17.21

4.3.2 各核心功能模块详细设计

4.3.2.1 类图

4-6

4-7

4-8

4.3.2.2 功能时序图

4-9

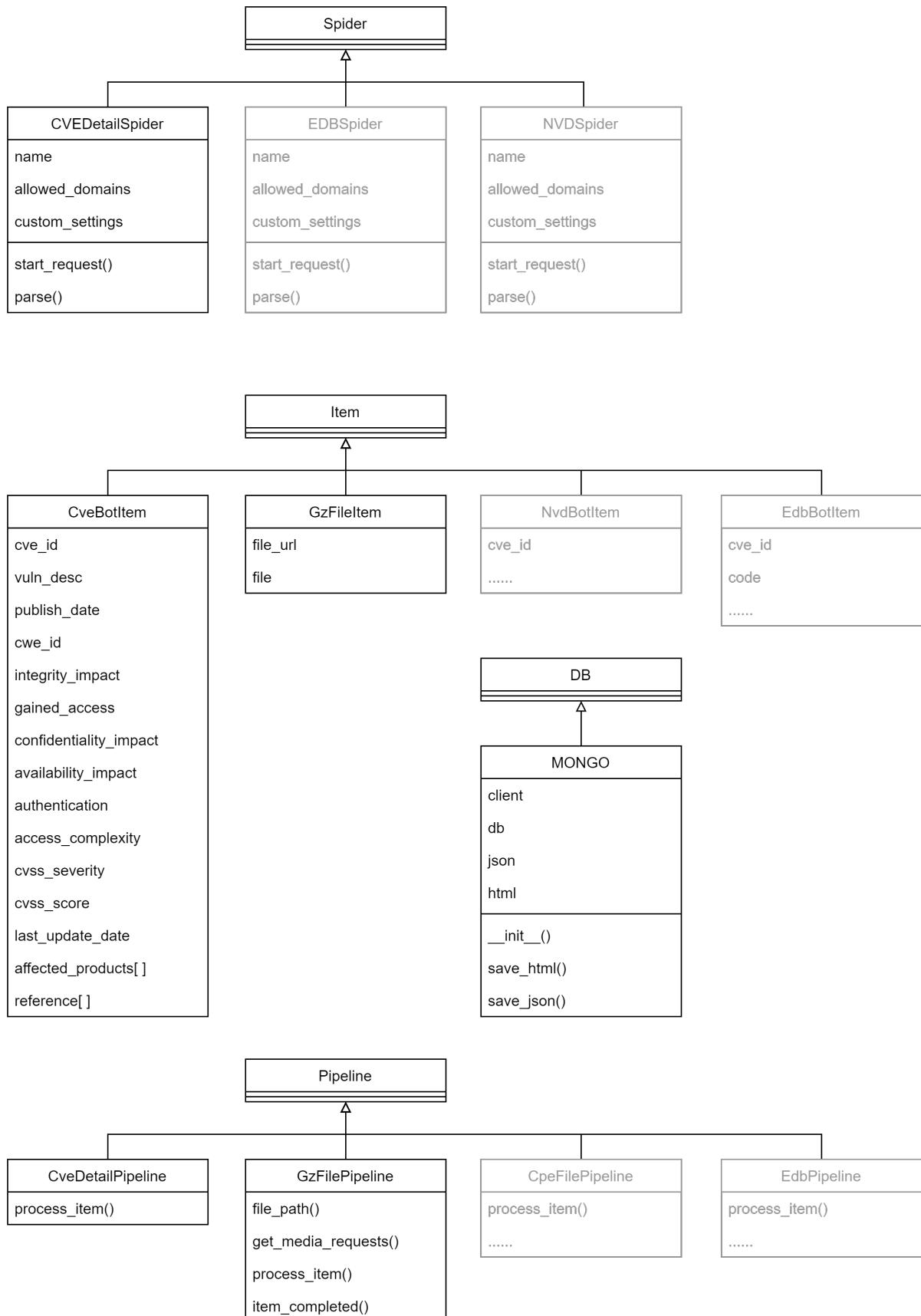


图 4-6 数据采集子系统类图

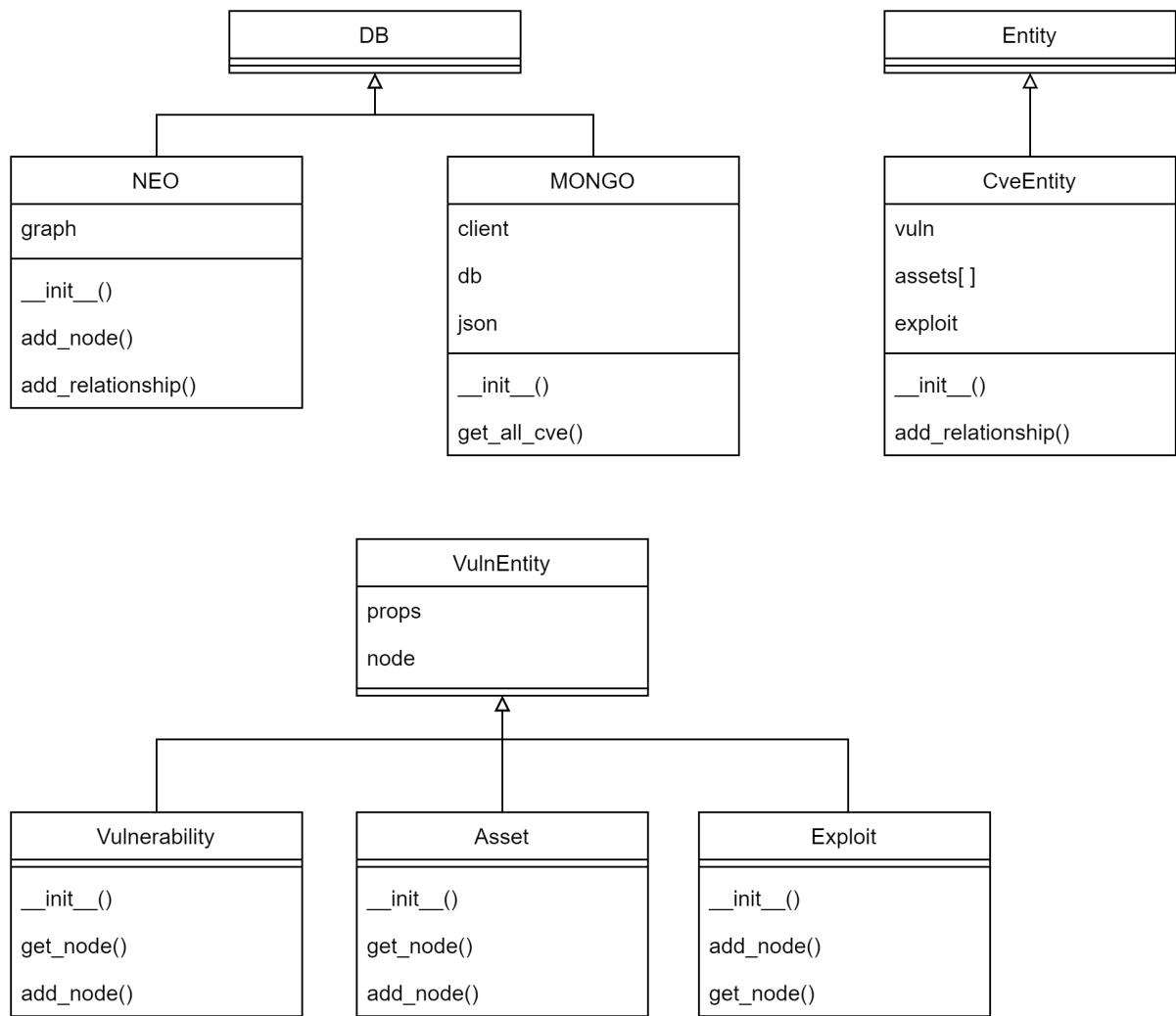


图 4-7 知识图谱构建子系统类图

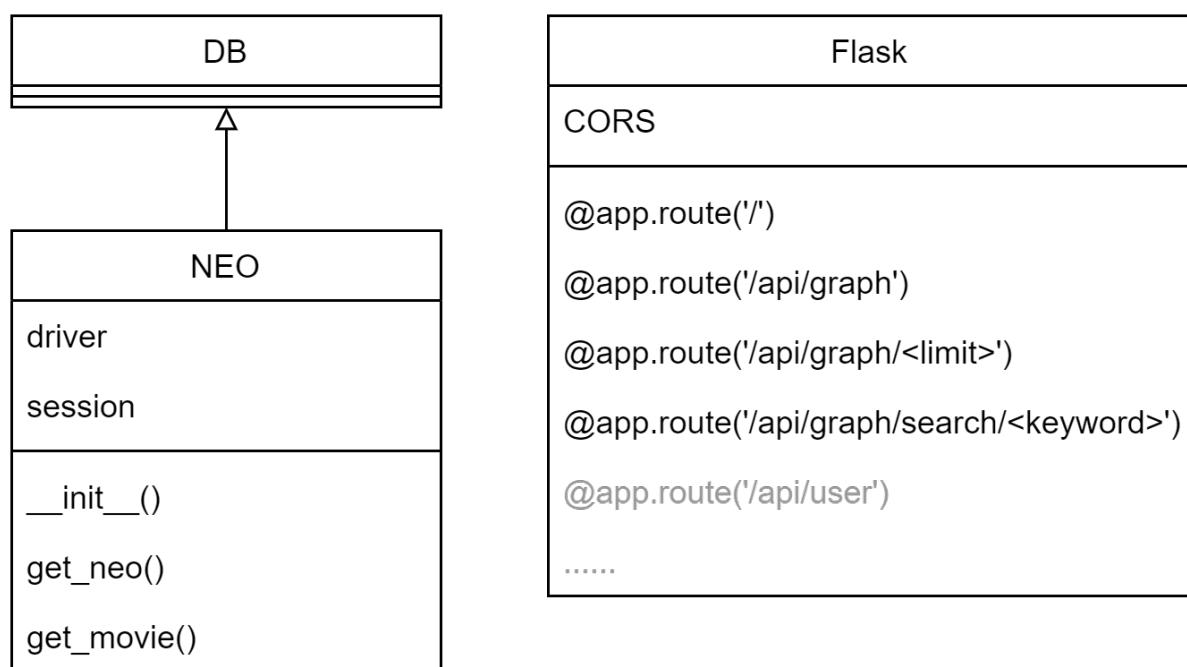


图 4-8 后端服务子系统类图

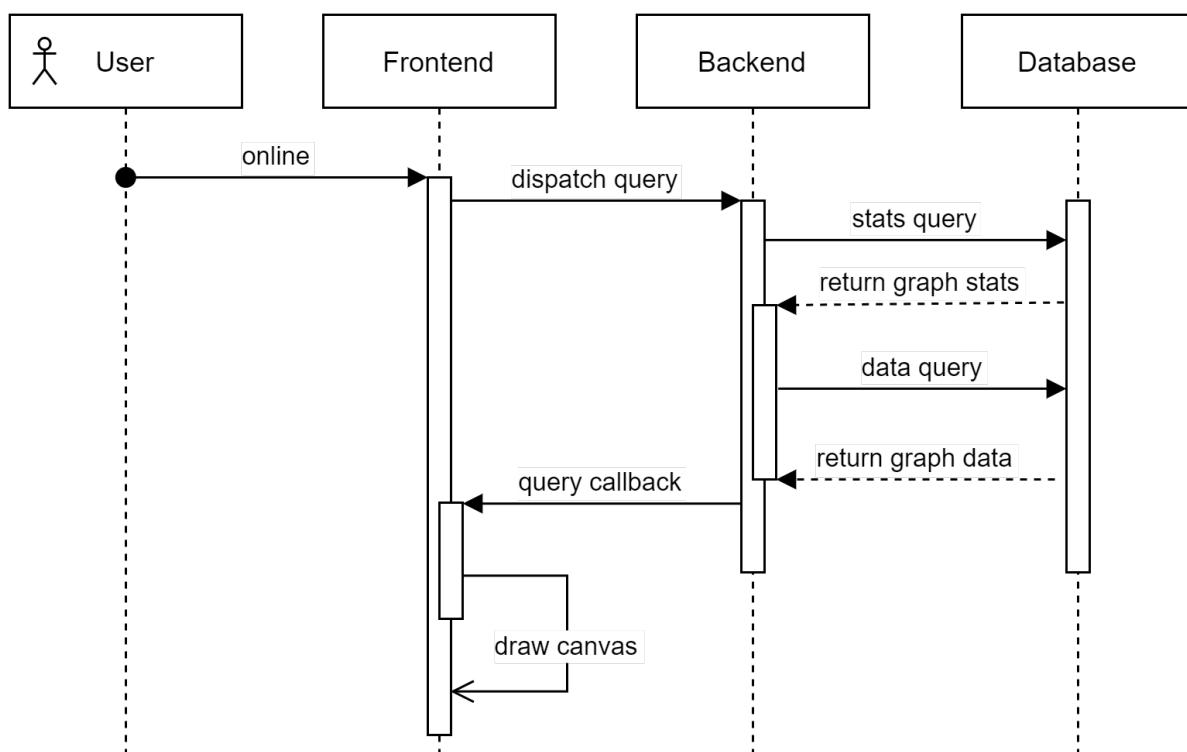


图 4-9 用户请求功能时序图

第五章 系统实现

5.1 工具模块

5.1.1 根目录模块实现 `bot_root_dir.py`

本模块位于项目根目录下。`get_bot_root_dir()` 方法使用 `pathlib` 的 `Path(__file__).parent` 方法获得项目根目录的路径，使得本项目内代码全部使用平台无关的相对路径，从而能在 Linux、macOS、Windows 等多种系统平台运行，保证可移植性。

本模块内包含许多路径相关的方法，如 `get_source_data_dir()`, `get_cve_data_dir()`, `get_log_dir()` 等。这些方法接受字符串形式的路径参数，返回 `Path` 对象。通过传入参数，控制系统生成的文件存放的位置，若目录不存在，则会自动创建所需目录。

本系统所有涉及文件系统的模块都依赖于此模块。

5.1.2 日志模块实现 `logger_factory.py`

在 Python Logging 模块基础上重新实现了自定义的日志功能。具有 `init_log_dir()`, `setup_logger()`, `mylogger()`, `loggers={}` 等方法和属性。

`init_log_dir()` 接受字符串路径可选参数，初始化日志记录路径。若参数为空，则使用默认路径。

`setup_logger(name, log_folder=None, lvl_file=None, lvl_stdout=None)` 是一个工厂函数，创建一个新的 `Logger` 对象，根据参数设置 `FileHandler()`, `StreamHandler()`，支持自定义日志路径、文件名称和不同 Handler 使用不同记录等级。

`mylogger()` 实现 `logger` 对象的单例模式^[5]。利用 Python 模块引入为单例的特性，使得在同一模块内多次调用相同 `<name>` 的 `mylogger()` 会返回相同的 `Logger` 对象。当第一次调用 `mylogger(<name>)` 时，会调用 `setup_logger()` 创建一个新的 `Logger` 对象，并且将其保存在全局变量 `loggers` 字典内。之后调用同样的 `mylogger(<name>)` 时，会根据 `<name>` 从 `loggers` 字典查找对应的 `Logger` 对象并返回。

在一个模块内首次调用 `mylogger(<name>)` 会初始化日志，添加日志的文件控制器与标准输出控制器，并且设定输出日志文件位置，以“本模块名 + 当前时分秒”命名。后续再次调用 `mylogger(<name>).log()` 会使用首次创建的 `logger` 对象进行日志记录。

5.1.3 环境配置文件 `secret.py`

以 Python 变量的形式存储 MongoDB 与 Neo4j 数据库的用户名、密码、数据库名、连接 url 等信息，和一些私有 API 的密钥等信息。此文件被 `.gitignore`，不会同步到远程仓库。

5.2 数据采集子系统

5.2.1 Spider 模块实现

每个 Spider 继承自 `scrapy.Spider` 类，重写了父类的 `__init__()`, `start_requests()`, `parse()` 方法。

5.2.1.1 `__init__()` 实现

`__init__()` 方法完成 Spider 类的初始化工作。在调用父类的 `super().__init__()` 方法之后，调用本系统基于 Python Logging 模块重新自定义封装的 `mylogger(<name>)` 函数初始化 logger。

5.2.1.2 `start_requests()` 实现

`start_requests()` 方法描述发起请求的步骤。

对于需要 cve id 索引的 Spider 如 `CveDetailSpider(scrapy.Spider)`，使用 Pandas 库的 `read_csv()` 方法，设置 `chunksize` 数值和 `iterator=True`，分块迭代读取 csv 文件。对每一行 cve id，生成所需 url 并使用 `yield scrapy.Request(url=url, callback=self.parse)` 发出请求。

对于需要从爬取页面中获得下一页 url 的 Spider 如 `EdbSpider(scrapy.Spider)`，`start_request()` 方法中仅指定初始页面发出请求，在解析收到的 response 时处理下一页的链接，并发出请求。

对于采用 JavaScript 动态加载页面的站点如 `exploit-db.com`，使用将 playwright 无头浏览器与 Scrapy 框架集成的 `scrapy-playwright` 库，对爬取到的 HTML 页面及其 JavaScript 代码进行渲染，将渲染生成的 DOM 返回给 Spider 的 `parse()` 方法。

5.2.1.3 `parse()` 实现

`parse()` 方法实现对请求返回数据的解析。

对于返回数据为 HTML 文档类型，使用 BeautifulSoup 库加载 DOM 进行解析。使用 `soup.select_one()` 方法配合 CSS 选择器提取所需标签的内容。对于需要在页面上查找下一请求 url 的站点，使用 `yield scrapy.Request()` 方法发出请求。将解析出的数据封装在 `scrapy.Item` 类中，使用 `yield item` 方法将数据传递给对应数据管线。

对于返回数据为 JSON 类型，使用 `response.json()` 获取返回数据，将其作为参数实例化 Item 对象，传递给 JSON 数据管线。

对于单个请求返回数据为 Gz 压缩的 xml 类型，使用工具模块 `gz.py` 的 `un_gz()` 方法解压文件得到 xml，并使用 `lxml` 模块的 `etree.iterparse(filepath, tag='')` 方法迭代遍历该 xml 文件，填充 item，最终调用持久化子系统的数据库控制对象提供的 `save_cpe()` 方法存储。

将以上代码段包裹在 Python 的 `try...except...finally...` 语句内，使日志模块能正常 `catch` 抛出的错误，并使程序不中断继续运行。

5.2.2 Item Pipeline 模块实现

每个 Pipeline 类实现一个 `process_item()` 方法，同时在 Spider 的 `custom_settings` 内设置对应的 Pipeline 类。Scrapy Engine 将 Spider 内 `yield item` 发送给 Pipeline 对象进行处理。

`GzPipeline` 负责处理压缩成 Gzip 格式的 cve 与 cpe xml 文件。将其解压并返回解压后文件的路径。

`NvdPipeline` 负责处理 NVD API 提供的 JSON，提取数据并调用持久化子系统的数据库控制对象提供的 `save_nvd_src()` 方法存储原始数据，使用 `save_nvd()` 方法存储处理过的数据。

其余 Pipeline 功能类似，根据爬取站点不同，实现有所不同。

5.2.3 其他模块实现

- `scrapy.cfg`: 存放 Scrapy 项目设置选项，如默认设置文件、项目名称等。
- `settings.py`: Python 变量形式存放 Scrapy 爬虫设置选项，如并发度、请求延时、User-Agent 等。
- `crawl_runner` 目录: 包含 `crawl_cpe.py`, `crawl_nvd.py`, `crawl_edb.py` 等。使用 `scrapy.cmdline.execute()` 方法运行爬虫程序。可以被调度模块调用，实现全自动爬虫。

5.3 知识图谱构建子系统

5.3.1 实体生成

实体生成模块主要包含

```

1  init_nodes(vul_num = 0, asset_num = 0, exploit_num = 0)
2  init_vul_ray(skip, _limit)
3  init_asset_ray(skip, _limit)
4  init_exploit_ray(skip, _limit)
5  init_asset_family_ray(skip, _limit)
6  get_step(num)

```

几个方法。

5.3.1.1 init_nodes() 实现

`init_nodes(vul_num, asset_num, exploit_num)` 方法接受可选参数漏洞数量、资产数量、利用代码数量，若参数不为零，则将数据根据并行度参数分割并行处理，使用 `ray.get()` 方法等待并行处理结束。示例代码如下：

```

1 arr = [init_vuln_ray.remote(skip=i, _limit=get_step(vuln_num) + 1) for i in
2     range(0, vuln_num, get_step(vuln_num))]
3 ray.get(arr)

```

若不传入漏洞数量等参数，默认使用单线程串行处理。

5.3.1.2 init_vul_ray() 实现

`init_vul_ray(skip, _limit)` 方法用于生成漏洞实体结点。`skip` 参数指定跳过数据库查询结果中的前多少个记录，`_limit` 参数指定最多处理多少个数据库查询结果。两个参数配合使用，可以实现从第 `<skip>` 个记录开始，处理 `_limit` 个记录。从而实现并行化处理。

创建一个持久化子系统数据库控制器单例对象 `_mg = MyMongo()`，调用方法 `get_nvd(cve_id = None)` 不传入参数，获取 MongoDB 数据库中全部 NVD 数据的迭代指针对象。根据传入参数从指定位置迭代处理查询记录。

对于每个记录，调用 `split_properties(doc, api_ver = ApiVersion.NVDv1)` 方法，`ApiVersion` 是一个自定义枚举类型，用于枚举不同的数据 API 类型。这个方法根据不同的 `api_ver` 使用不同的处理方式，将原始数据中的信息分割为不同语义，并将可用于索引的属性域提到外层。随后调用循环体外创建的持久化子系统数据库控制器单例对象 `neo = MyNeo()` 将实体结点不重复地加入知识图谱中。

将以上循环体内代码包裹在 `try...except...finally...` 语句内保证正确处理异常并且程序不会意外中断。此外，在处理开始、循环体中间、处理结束均使用前述日志模块的 `mylogger()` 方法进行日志输出。

5.3.1.3 init_asset_ray() 实现

`init_asset_ray(skip, _limit)` 方法用于生成资产实体结点。参数作用同 `init_vul_ray()` 实现章节，代码实现与 `init_vul_ray()` 实现章节所述类似。

5.3.1.4 init_exploit_ray() 实现

`init_exploit_ray(skip, _limit)` 方法用于生成漏洞利用实体结点。参数作用同 `init_vul_ray()` 实现章节，代码实现与 `init_vul_ray()` 实现章节所述类似。

5.3.1.5 init_asset_family_ray() 实现

`init_exploit_ray(skip, _limit)` 方法用于生成资产家族实体结点。参数作用同 `init_vul_ray()` 实现章节。

该方法与 `init_asset_ray()` 方法为串行依赖关系，需要在其之后执行。调用 `NodeMatcher().match()` 方法，从知识图谱中匹配全部资产结点，返回一个迭代指针对象。迭代遍历全部结点，对 `cpe23uri` 资产类型、制造商、产品名称都相同的节点，认为它们同属一个家族。为每个家族不重复地创建一个资产家族实体结点，存入知识图谱。

5.3.2 关系生成

实体生成模块主要包含

```

1  init_rels(vul_num = 0, exploit_num = 0)
2  create_rel_vaf_ray(skip, _limit)
3  create_rel_afa_ray(skip, _limit)
4  create_rel_evaf_ray(skip, _limit)
5  get_step(num)

```

几个方法。

5.3.2.1 init_rels() 实现

`init_rels(vul_num, exploit_num)` 方法接受可选参数漏洞数量、利用代码数量，若参数不为零，则将数据根据并行度参数分割并行处理，使用 `ray.get()` 方法等待并行处理结束。示例代码如下：

```

1  arr = []
2  arr.extend([create_rel_vaf_ray.remote(skip=i, _limit=get_step(vuln_num) + 1)
             for i in range(0, vuln_num, get_step(vuln_num))])
3  ray.get(arr)

```

若不传入漏洞数量等参数，默认使用单线程串行处理。

5.3.2.2 create_rel_vaf_ray() 实现

`vaf` 表示 Vulnerability 和 AssetFamily。`create_rel_vaf_ray(skip, _limit)` 方法用于生成漏洞实体结点与资产家族实体结点之间的关系边。`skip` 参数指定跳过数据库查询结果中的前多少个记录，`_limit` 参数指定最多处理多少个数据库查询结果。两个参数配合使用，可以实现从第 `<skip>` 个记录开始，处理 `_limit` 个记录。从而实现并行化处理。

根据漏洞实体结点属性中存储的受影响资产正则匹配信息，若按照“对于每个正则表达式，匹配知识图谱中符合规则的资产实体结点，为漏洞实体结点与资产实体结点添

加两条关系边”的最细粒度做法，知识图谱中的 120 万个结点将生成超过 1 亿条边，且全图正则匹配无法有效利用建立在属性域上的索引，本项目的性能需求是无法接受的。

因此，采用“对于每个正则表达式，匹配知识图谱中符合规则的资产家族实体结点，为漏洞实体结点与资产家族实体结点添加两条关系边，并把真实受影响资产列表存储在关系边属性上”的方法，可以在不改变使用效果的前提下，有效降低构建图谱所需的边数。

为了进一步加速匹配，把正则表达式中 `.*` 之前的固定模式串切割成子串，使用该固定模式串向数据库进行 `STARTS WITH` 查询，将返回的查询结果在 Python 中进行正则匹配进一步精确筛选。由于 Neo4j 数据库查询时支持 `STARTS WITH` 子句使用属性域索引，采用此种方法查询，使得查询速度提高了 100 倍以上。

5.3.2.3 `create_rel_afa_ray()` 实现

`afa` 表示 `AssetFamily` 和 `Asset`。`create_rel_afa_ray(skip, _limit)` 方法用于生成资产实体结点与资产家族实体结点之间的关系边。`skip` 参数指定跳过数据库查询结果中的前多少个记录，`_limit` 参数指定最多处理多少个数据库查询结果。两个参数配合使用，可以实现从第 `<skip>` 个记录开始，处理 `_limit` 个记录。从而实现并行化处理。

对于每个漏洞实体结点，遍历其受影响资产正则匹配表达式列表，对每个正则匹配表达式，通过寻找其前缀获得其属于的资产家族，调用持久化子系统数据库控制器对象的 `add_rel_cql_afa()` 方法，为该资产结点和资产家族结点之间添加“资产家族结点是资产结点父级”、“资产结点是资产家族结点子级”两个关系。

5.3.2.4 `create_rel_evaf_ray()` 实现

`evaf` 表示 `Exploit` 和 `Vulnerability` 和 `AssetFamily`。`create_rel_evaf_ray()` 方法用于生成漏洞利用代码结点与漏洞实体结点之间的关系边，并且从这条关系与漏洞实体结点指向的受影响资产家族结点，推理出漏洞利用代码结点与资产家族结点之间的关系。`skip` 参数指定跳过数据库查询结果中的前多少个记录，`_limit` 参数指定最多处理多少个数据库查询结果。两个参数配合使用，可以实现从第 `<skip>` 个记录开始，处理 `_limit` 个记录。从而实现并行化处理。

遍历漏洞利用代码结点，对于带有 `cve id` 属性的结点，从图谱中查询对应 `cve id` 的漏洞实体结点。为漏洞利用结点与漏洞实体结点之间添加“漏洞利用利用漏洞”、“漏洞被漏洞利用利用”两个关系。再从漏洞实体结点出发，寻找所有受影响的资产家族结点。根据“漏洞影响资产家族”关系边上的属性，为漏洞利用结点与资产家族结点添加“漏洞利用针对资产家族”、“资产家族被漏洞利用针对”两个关系，属性继承自前述关系边属性。

5.3.3 关系融合实现

关系融合模块作为适配使用深度学习模型进行命名实体识别结点生成与关系抽取方案的模块，当前仅在系统数据加工管线中留有接口，以备未来扩展需要。当前系统的结构化数据实体与关系生成已经实现实体结点和关系的消除二义性，因此关系融合模块暂未实现。

5.4 持久化子系统实现

持久化子系统利用 Python 默认引入模块默认为单例的特性，使用饿汉模式¹⁰ 实现灵活的单例数据库控制器类，分别为 `class MyMongo` 与 `class MyNeo`。

饿汉模式灵活之处在于，在单线程运行场景下，可直接引入在 `db.py` 内实例化的数据库控制器对象，如 `mg = MyMongo()`, `neo = MyNeo()`，实现单例模式。而在多线程运行场景下，可以引入 `class MyMongo` 与 `class MyNeo` 类，并在每个线程内分别实例化。

5.4.1 MyMongo 类实现

实现的方法如下：

```

1  check_index() # 不重复地创建数据库所有集合的索引
2  save_cvedetails_json(cve_id, doc) # 保存 CVEDETAILS JSON 数据
3  get_cvedetails_json(cve_id = None) # 查询 CVEDETAILS JSON 数据
4  save_nvd(cve_id, doc) # 保存 NVD JSON 数据
5  get_nvd(cve_id = None) # 查询 NVD JSON 数据
6  save_cpe(cpe23uri, doc) # 保存 CPE JSON 数据
7  get_cpe(cpe23uri = None) # 查询 CPE JSON 数据
8  save_edb_html(edb_id, doc) # 保存 EXPLOIT HTML 数据
9  save_edb_json(edb_id, doc) # 保存 EXPLOIT-DB JSON 数据
10 get_edb_json(edb_id = None) # 查询 EXPLOIT-DB JSON 数据
11 get_exploit_stats() # 查询漏洞利用统计信息

```

其中所有 `get` 方法的可选参数为空时，返回全部数据的一个迭代器指针。

5.4.2 MyNeo 类实现

实现的方法如下：

```

1  get_session() # 获取一个数据库连接
2  check_node_index() # 不重复地创建结点相关属性域的索引
3  check_rel_index() # 不重复地创建关系相关属性域的索引
4  get_node(*args, **kwargs) # 封装的通用查询结点方法
5  add_node(labels, props) # 封装的通用添加结点方法
6  add_relationship(start, _type, end, props = None) # 封装的通用添加关系方法

```

```

7  delete_relationship(cve_id = '') # 删除与某漏洞实体结点相关的所有关系
8  match_asset(pattern) # 根据正则表达式，匹配资产
9  match_asset_family(pattern) # 根据正则表达式，匹配资产家族
10 add_asset_family_node(cpe23uri) # 添加资产家族结点
11 add_rel_cql_vaf(cve_id, cpe23uri) #
    在漏洞结点与资产家族结点之间添加两条关系，使用 CQL 实现
12 add_rel_cql_afa(asset_uri) # 在资产结点与资产家族结点之间添加两条关系，使用
    CQL 实现
13 close_db() # 关闭数据库连接，释放资源

```

5.5 后端服务子系统实现

5.5.1 create_app.py 实现

创建 Flask App 实例的工厂函数^[6]。主要包含创建 Flask 实例、挂载中间件、注册蓝图等步骤。

5.5.2 graph.py 实现

知识图谱相关 API 的蓝图。向 Flask 注册 /api/graph/ 路径下的 API，并且实现这些 API 对应的功能。主要实现了以下方法：

```

1 assemble_graph_data(assets=(), vuls=(), exploits=(), rels=()) # 生成绘图数据
2 retrieve_graph_stats() # 获取知识图谱统计数据
3 label_to_id_field(label) # 通过字典映射转换，将标签字符串转换为索引属性域字符串
4 get_symbol_size(_type, cnt) # 计算结点在可视化力引导图中显示的大小
5 get_node_category(type_list) # 根据结点的标签列表，选出主要标签
6 retrieve_graph(limit) # 获取<LIMIT>个漏洞结点及与之相关的结点与关系绘图数据
7 search_graph(keyword) # 根据关键字搜索知识图谱

```

5.5.2.1 /api/graph/ 实现

获取知识图谱统计数据。调用持久化子系统的数据库控制器对象的 `get_session().read_transaction()` 方法，向数据库提交一个读事务。读事务 `work(tx)` 中定义查询的 CQL 语句，并调用 `transaction` 对象的 `tx.run()` 方法运行查询。查询获得漏洞数量、受影响资产类型及数量、漏洞利用代码数量及类型等信息。调用绘图数据生成方法，生成结果数据，返回给前端。

5.5.2.2 /api/graph/<limit> 实现

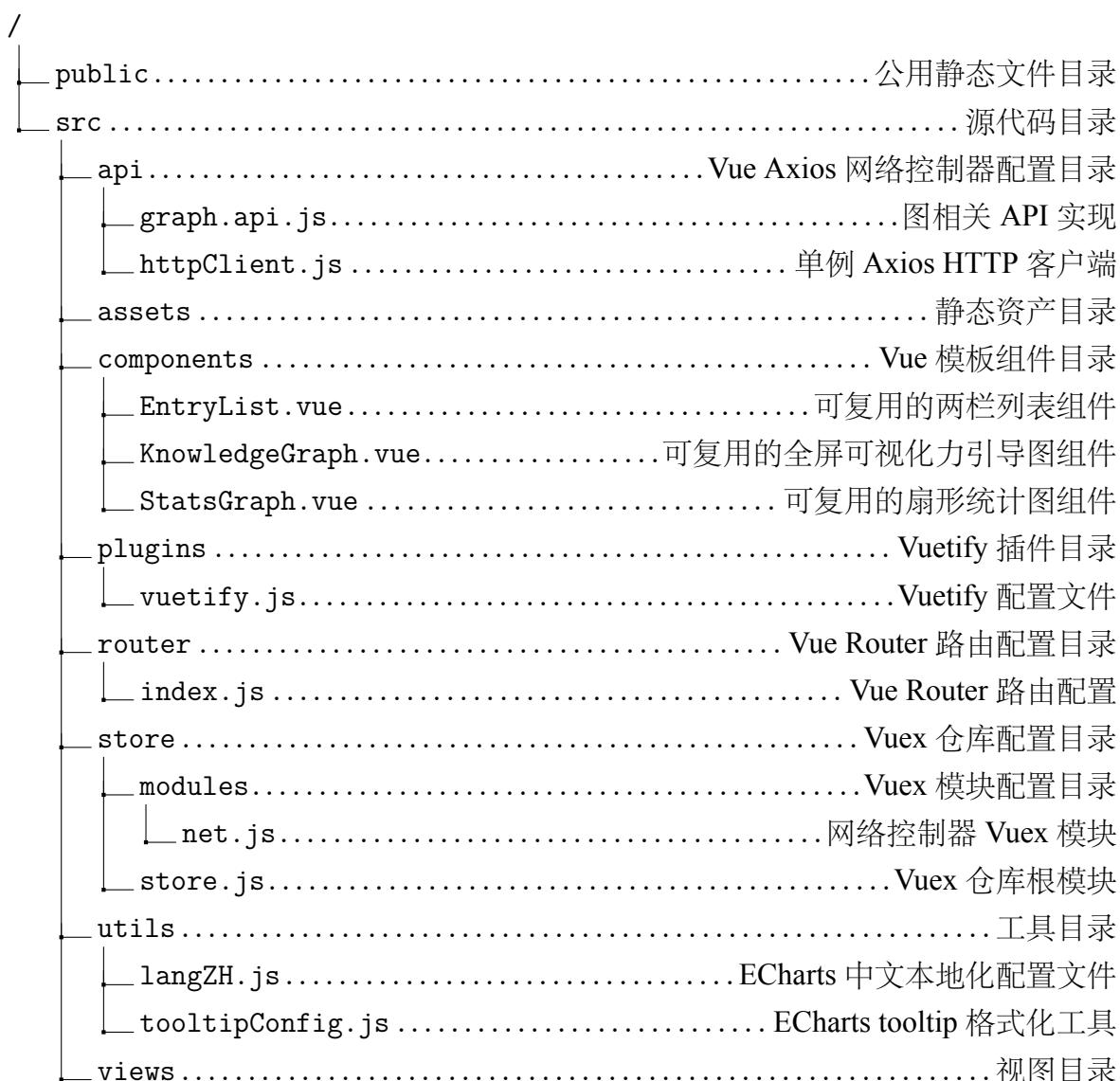
获取 <limit> 个漏洞结点及与之相关的结点与关系绘图数据。为防止数据量过大，实际查询将取 <limit> 与 200 间的较小值。根据经验数据，返回结点和关系数量不超过一万个。调用绘图数据生成方法，获得各类节点及关系绘图数据，返回给前端。

5.5.2.3 /api/graph/search/<keyword> 实现

根据关键字搜索知识图谱。根据关键字类型，判断搜索内容属于漏洞、资产、利用代码或是关系。将搜索到的结点/关系及其相关联（关系路径长度为 1）的结点或关系返回，调用绘图数据生成方法获得绘图所需数据，返回给前端。

5.6 前端子系统

5.6.1 项目结构



└ Welcome.vue	登录首屏欢迎界面视图
└ Dashboard.vue	控制台视图
└ Visualization.vue	可视化视图
└ Search.vue	搜索视图
└ About.vue	关于视图
└ App.vue	根组件，描述页面框架结构及基础功能
└ main.js	向页面引入并注册 Vue 组件
└ registerServiceWorkder.js	注册 ServiceWorkder，实现 PWA
└ .env	环境变量
└ package.json	npm 配置文件
└ babel.config.js	babel 配置文件
└ vue.config.js	vue 配置文件
└ eslintrc.js	eslint 配置文件
└ browserslistrc	支持的浏览器配置文件

5.6.2 控制台视图实现

控制台视图采用 Flex 流式卡片布局，通过设置 CSS 样式断点，每个卡片根据当前设备屏幕分辨率自动调节宽度。例如，宽度为 `lg` 时，界面布局为每行 3 列卡片。宽度为 `md` 时，界面布局为每行 2 列卡片。宽度为 `sm` 时，为每行 1 列卡片。

卡片中使用 `EntryList` 组件显示知识图谱统计信息，辅以 `StatsGraph` 组件显示可视化扇形图。图示见章节 6.5.1 控制台视图测试。

`EntryList` 与 `StatsGraph` 使用 `Vuex` 的 `mapState()` 计算属性作为组件属性，组件内部为 `props` 添加 `watch` 倾听器，实现数据响应式动态更新。

`Dashboard.vue` 在 `mounted()` 时会通过 `Vuex` 仓库的 `this.$store.dispatch()` 发出一个异步 I/O Action 请求，向后端请求知识图谱的统计数据。此时前端界面会优先加载，由默认空值或零值填充列表和统计图。当 Axios 请求回调时，`Vuex Action` 会向 `Vuex Store` 提交数据产生的更改。这引起各个视图或组件内计算属性值变化，倾听器触发，执行回调函数，调用 `ECharts` 实例对象的 `setOption(option)` 方法，从而动态更新数据。

5.6.3 可视化视图实现

可视化视图使用 `ECharts` 呈现一个全屏的可视化力引导图，鼠标按住或手指拖拽可以移动视图，鼠标按住或手指拖拽图中结点可以移动结点位置。图示见章节 6.5.2 可视化视图测试。

可视化视图同样采用来自 `Vuex Store` 的计算属性作为数据源，实现动态数据更新。在切换视图时，为节省内存 `ECharts` 实例会被销毁，但绘图数据保留在 `Vuex Store`。再次切换回可视化视图时，会优先加载缓存的绘图数据，节省网络带宽、减轻后端压力、加快加载速度。

5.6.4 搜索视图实现

搜索视图包含一个悬浮搜索框，显示在容器的左上角。在搜索框输入关键词并按下回车，发出异步 I/O 请求并等待返回绘图数据。可视化显示流程与上述章节 5.6.3 可视化视图实现相似。

5.6.5 关于视图实现

关于视图包含一些基本文字说明，使用 Flex 进行排版。

第六章 系统测试

6.1 测试环境

- CPU: AMD Ryzen ZEN3 架构 8C16T@4.8GHz
- RAM: 16GB*2 DDR4@3600MT/s
- 存储: PCIe Gen 3 固态硬盘
- 操作系统: Ubuntu 20.04

6.2 数据采集子系统测试

图 6-1 展示 MongoDB 数据库中的 Collections 概览。

通过 Scrapy 爬虫 cve.mitre.org, 获得约 23 万条 cve id 数据, 经筛选清洗获得约 16 万条有效 cve id 信息, 如图 6-2 所示。

爬取 cpe.mitre.org 定期发布的官方 cpe 信息文件, 使用 lxml 库解析, 经筛选清洗获得约 90 万条有效 cpe 资产信息 JSON 数据。其中包含用于识别的 cpe23uri、cpe 资产名称、参考资料链接等属性, 如图 6-3 所示。

爬取 cvedetails.com, 获得约 16 万个 HTML 页面, 清洗整理获得超 400 万个 cve 漏洞与资产相关属性 JSON 数据, 如图 6-4 所示。

通过 nvd.nist.gov JSON API 数据接口, 获得约 16 万条 JSON 数据, 包含漏洞名称与简介、漏洞条目发布日期、漏洞条目更新日期、CVSS 风险评级、参考资料、受影响资产的 cpe23uri 匹配正则表达式等信息, 如图 6-5 所示。

爬取 exploit-db.com, 获得约 5 万个 HTML 页面, 包含漏洞利用代码、作者、发布时间、适用平台、利用类型、利用 cve id 等信息, 清洗处理为 JSON 格式数据, 如图 6-6 所示。

6.3 知识图谱构建子系统测试

6.3.1 结点生成

使用 Ray 库提供的 @ray.remote 装饰器与 ray.get() 方法, 将全部漏洞数据根据并行度参数进行分割, 开辟多进程对分块漏洞数据进行生成结点操作。

开辟 32 个进程生成漏洞结点、资产结点、资产家族结点、漏洞利用结点, 总计约 120 万个结点。系统耗时约 5 分钟, 如图 6-7 与图 6-8 所示。

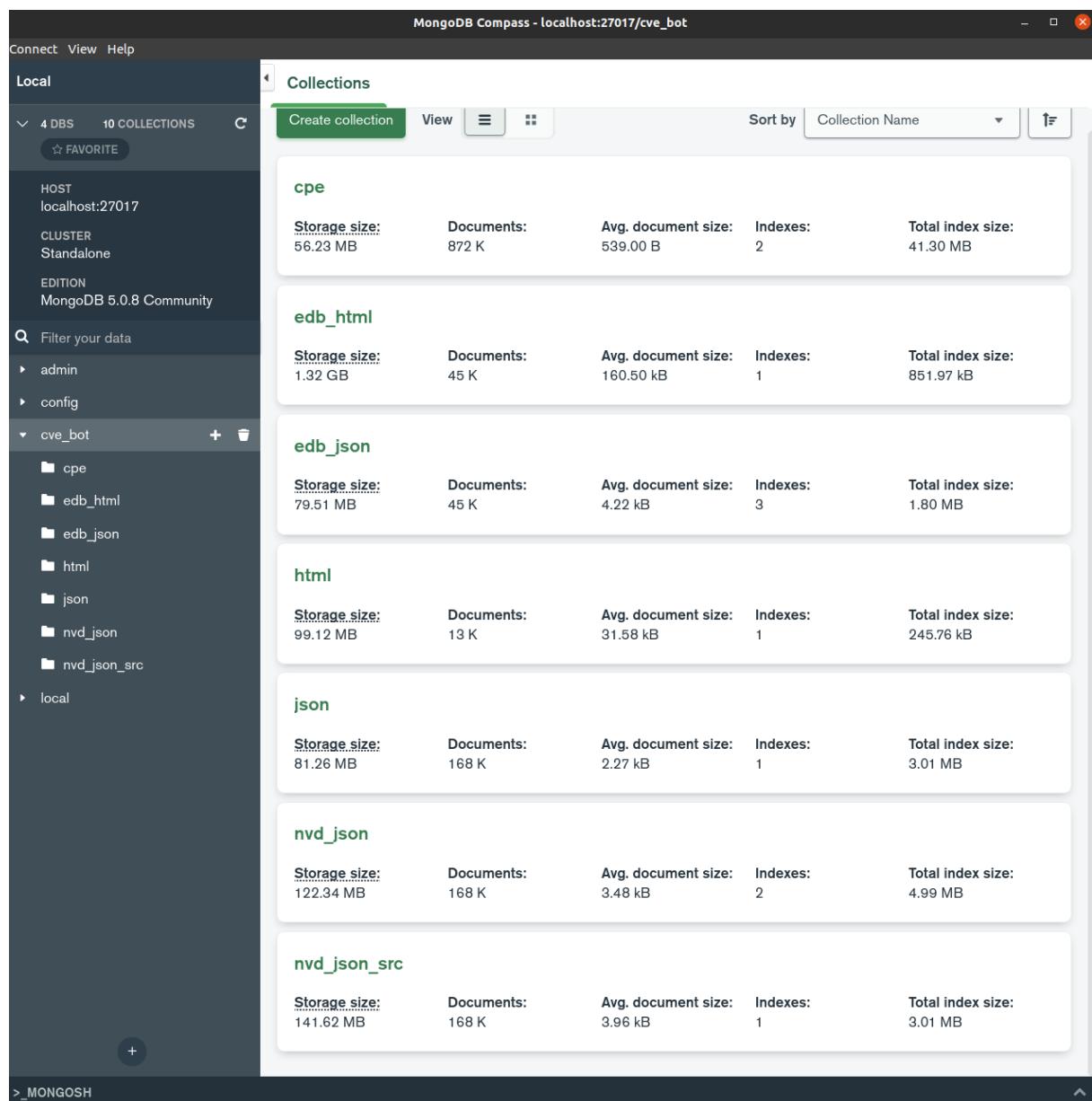


图 6-1 MongoDB Collections 概览

168339	CVE-2022-24196	Candidate	iText v7.1.17 was discovered	MISC: https://github.com/itext/itext7/pull/78	Assigned (20220131)
168340	CVE-2022-24197	Candidate	iText v7.1.17 was discovered	MISC: https://github.com/itext/itext7/pull/78	Assigned (20220131)
168341	CVE-2022-24198	Candidate	iText v7.1.17 was discovered	MISC: https://github.com/itext/itext7/pull/78	Assigned (20220131)
168342	CVE-2022-24218	Candidate	An issue in /admin/delete_imap	MISC: https://github.com/jsjcyber/bug_repo	Assigned (20220131)
168343	CVE-2022-24219	Candidate	eliteCMS v1.0 was discovered	MISC: https://github.com/jsjcyber/bug_repo	Assigned (20220131)
168344	CVE-2022-24220	Candidate	eliteCMS v1.0 was discovered	MISC: https://github.com/jsjcyber/bug_repo	Assigned (20220131)
168345	CVE-2022-24221	Candidate	eliteCMS v1.0 was discovered	MISC: https://github.com/jsjcyber/bug_repo	Assigned (20220131)
168346	CVE-2022-24222	Candidate	eliteCMS v1.0 was discovered	MISC: https://github.com/jsjcyber/bug_repo	Assigned (20220131)
168347	CVE-2022-24223	Candidate	AtomCMS v2.0 was discovered	MISC: https://github.com/thedigicraft/Atom	Assigned (20220131)
168348	CVE-2022-24249	Candidate	A Null Pointer Dereference vulnerability	MISC: https://github.com/gpac/gpac/issues/141	Assigned (20220131)
168349	CVE-2022-24259	Candidate	An incorrect check in the compare function	MISC: https://www.voipmonitor.org/changelog	Assigned (20220131)
168350	CVE-2022-24260	Candidate	A SQL injection vulnerability in	MISC: https://www.voipmonitor.org/changelog	Assigned (20220131)
168351	CVE-2022-24262	Candidate	The config restore function of	MISC: https://www.voipmonitor.org/changelog	Assigned (20220131)
168352	CVE-2022-24263	Candidate	Hospital Management System	MISC: http://packetstormsecurity.com/files/15447/Hospital%20Management%20System	Assigned (20220131)
168353	CVE-2022-24264	Candidate	Cuppa CMS v1.0 was discovered	MISC: https://github.com/CuppaCMS/Cuppa	Assigned (20220131)
168354	CVE-2022-24265	Candidate	Cuppa CMS v1.0 was discovered	MISC: https://github.com/CuppaCMS/Cuppa	Assigned (20220131)
168355	CVE-2022-24266	Candidate	Cuppa CMS v1.0 was discovered	MISC: https://github.com/CuppaCMS/Cuppa	Assigned (20220131)
168356	CVE-2022-24300	Candidate	Minetest before 5.4.0 allows at	MISC: https://bugs.debian.org/1004223	Assigned (20220202)
168357	CVE-2022-24301	Candidate	In Minetest before 5.4.0, player	MISC: https://github.com/minetest/minetest	Assigned (20220202)
168358	CVE-2022-24307	Candidate	Mastodon before 3.3.2 and 3.4.0	CONFIRM: https://github.com/mastodon/mastodon	Assigned (20220202)
168359	CVE-2022-24348	Candidate	Argo CD before 2.1.9 and 2.2.0	CONFIRM: https://github.com/argoproj/argo	Assigned (20220202)
168360	CVE-2022-24448	Candidate	An issue was discovered in fs	MISC: https://cdn.kernel.org/pub/linux/kernel	Assigned (20220204)
168361	CVE-2022-24450	Candidate	NATS nats-server before 2.7.2	CONFIRM: https://advisories.nats.io/CVE-2022-24450	Assigned (20220204)
168362	CVE-2022-24551	Candidate	StarWind SAN and NAS before	MISC: https://www.starwindsoftware.com/security-advisory	Assigned (20220206)
168363	CVE-2022-24552	Candidate	StarWind SAN and NAS before	MISC: https://www.starwindsoftware.com/security-advisory	Assigned (20220206)
168364					

图 6-2 cve id 索引 csv 数据

cve_bot.cpe

DOCUMENTS 872.4k STORAGE SIZE 56.2MB AVG. SIZE 539B INDEXES 2 TOTAL SIZE 41.3MB AVG. SIZE 20.7MB

Documents Aggregations Schema Explain Plan Indexes Validation

FILTER { field: 'value' } **OPTIONS** **FIND** **RESET**

ADD DATA **VIEW**

Displaying documents 1 - 20 of 872351

`_id: ObjectId("6270e673cb179fcb30d7bf88")
cpe23uri: "cpe:2.3:a:$0.99_kindle_books_project:$0.99_kindle_books:6:***:an..."
field: Object
 part: "a"
 vendor: "\$0.99_kindle_books_project"
 product: "\$0.99_kindle_books"
 version: "6"
 update: "*"
 edition: "*"
 language: "*"
 sw_edition: "*"
 target_sw: "android"
 target_hw: "*"
 other: "*"
references: Array
 0: Object
 tag: "Product information"
 href: "https://play.google.com/store/apps/details?id=com.kindle.books.for99"
 1: Object
 tag: "Government Advisory"
 href: "https://docs.google.com/spreadsheets/d/1t5GXwjw82SyunALVJb2w0z13FoLRIK..."
title: "$0.99 Kindle Books project $0.99 Kindle Books (aka com.kindle.books fo...)"
timestamp: "2022-05-05T03:51:07.759Z"`

`_id: ObjectId("6270e673cb179fcb30d7bf8b")
cpe23uri: "cpe:2.3:a:@thi.ng/egf_project:@thi.ng/egf:-***:node.js:***"
field: Object
references: Array
title: "@thi.ng/egf Project @thi.ng/egf for Node.js"
timestamp: "2022-05-05T03:51:07.759Z"`

图 6-3 cpe 资产 JSON 数据

cve_bot.json

DOCUMENTS	168.3k	STORAGE SIZE	81.3MB	AVG. SIZE	2.3KB
INDEXES	1	TOTAL SIZE	3.0MB	Avg. Size	3.0MB

Documents Aggregations Schema Explain Plan Indexes Validation

FILTER { field: 'value' } OPTIONS FIND RESET ...

ADD DATA VIEW { } Displaying documents 1 - 20 of 168290 < > C REFRESH

```

{
  "_id": ObjectId("620d45286e07ac4435f26c8f"),
  "cve_id": "CVE-1999-0001",
  "content": {
    "cve_id": "CVE-1999-0001",
    "vuln_desc": "ip_input.c in BSD-derived TCP/IP implementations allows remote attack...",
    "publish_date": "1999-12-30",
    "last_update_date": "2010-12-16",
    "cvss_score": 5,
    "cvss_severity": "Medium",
    "confidentiality": {
      "text": "None",
      "desc": "There is no impact to the confidentiality of the system."
    },
    "integrity_impact": {
      "text": "None",
      "desc": "There is no impact to the integrity of the system."
    },
    "availability_im": {
      "text": "Partial",
      "desc": "There is reduced performance or interruptions in resource availability..."
    },
    "access_complexi...": {
      "text": "Low",
      "desc": "Specialized access conditions or extenuating circumstances do not exist..."
    },
    "authentication": {
      "text": "Not required",
      "desc": "(Authentication is not required to exploit the vulnerability.)"
    },
    "gained_access": "None",
    "vulnerability_t": "Denial Of Service",
    "cve_id": 20,
    "affected_produc...": [
      {
        "0": {
          "type": "OS",
          "vendor": "Bsd",
          "name": "Bsd Os",
          "version": "3.1"
        }
      },
      {
        "1": {
          "type": "OS"
        }
      }
    ]
  }
}

```

图 6-4 cvedetails.com JSON 数据

cve_bot.nvd_json

DOCUMENTS	168.3k	STORAGE SIZE	122.3MB	Avg. Size	3.5KB
INDEXES	2	TOTAL SIZE	5.0MB	Avg. Size	2.5MB

Documents Aggregations Schema Explain Plan Indexes Validation

FILTER { field: 'value' } OPTIONS FIND RESET ...

ADD DATA VIEW { } Displaying documents 1 - 20 of 168273 < > C REFRESH

```

{
  "_id": ObjectId("625e9e0d1cd6563de4602607"),
  "cve_id": "CVE-1999-0001",
  "content": {
    "version": "nvd_v1",
    "timestamp": "2022-04-19T11:30Z",
    "vuln": {
      "assets": []
    },
    "exploit": null
  }
}

{
  "_id": ObjectId("625e9e0d1cd6563de4602609"),
  "cve_id": "CVE-1999-0002",
  "content": {
    "version": "nvd_v1",
    "timestamp": "2022-04-19T11:30Z",
    "vuln": {
      "cve_id": "CVE-1999-0002",
      "cwe_id": "CWE-119",
      "desc": "Buffer overflow in NFS mounted gives root access to remote attackers, m...",
      "impact": {
        "baseMetricv3": null,
        "baseMetricv2": {
          "cvssv2": {
            "version": "2.0",
            "vectorString": "AV:N/AC:L/Au:N/C:C/I:C/A:C",
            "accessVector": "NETWORK",
            "accessComplexity": "LOW",
            "authentication": "NONE",
            "confidentiality": "COMPLETE",
            "integrityImpact": "COMPLETE",
            "availabilityImp": "COMPLETE",
            "baseScore": 10
          },
          "severity": "HIGH",
          "exploitability": 10
        }
      }
    }
  }
}

```

图 6-5 nvd.nist.gov JSON 数据

cve_bot.edb_json

DOCUMENTS 45.0k STORAGE SIZE 79.5MB AVG. SIZE 4.2KB | INDEXES 3 TOTAL SIZE 1.8MB AVG. SIZE 600.7KB

Documents Aggregations Schema Explain Plan Indexes Validation

Displaying documents 1 - 20 of 44954 < > C REFRESH

```

_id: ObjectId("62632e0ca05963f1a3e87cf9d")
edb_id: "1"
> content: Object

_id: ObjectId("62632e2fa05963f1a3e87cf2")
edb_id: "2"
> content: Object

_id: ObjectId("62632e30a05963f1a3e87cf5")
edb_id: "3"
< content: Object
  edb_id: "3"
  title: "Linux Kernel 2.2.x/2.4.x (RedHat) - 'ptrace/kmod' Local Privilege Escalation"
  author: "Wojciech Purczynski"
  type: "local"
  platform: "Linux"
  date: "2003-03-30"
  code: "/*\r\n * Linux kernel ptrace/kmod local root exploit\r\n */\r\n * This...\r\n"
  cve_ids: Array
    0: "2003-0127"

_id: ObjectId("62632e31a05963f1a3e87cf8")
edb_id: "4"
> content: Object

_id: ObjectId("62632e32a05963f1a3e87cfb")
edb_id: "5"

```

图 6-6 exploit-db.com JSON 数据

```

107 P1076673-T140451038263104 init_asset_ray - 2022-05-09 11:32:50.838 - timer - INFO: init_asset.skip(218080) with limit 54521 runtime = 0:04:14.356317
108 P1076670-T140719109216064 init_asset_ray - 2022-05-09 11:32:50.976 - timer - INFO: init_asset.skip(872320) with limit 54521 runtime = 0:00:00.382658
109 P1076667-T139979017795392 init_asset_ray - 2022-05-09 11:32:51.223 - timer - INFO: init_asset.skip(0) with limit 54521 runtime = 0:04:15.134648
110 P1076678-T139793847834432 init_asset_ray - 2022-05-09 11:32:51.372 - timer - INFO: init_asset.skip(163560) with limit 54521 runtime = 0:04:14.903336
111 P1076675-T140375827486528 init_asset_ray - 2022-05-09 11:32:51.591 - timer - INFO: init_asset.skip(327120) with limit 54521 runtime = 0:04:14.792446
112 P1076678-T140424310404928 init_asset_ray - 2022-05-09 11:32:51.595 - timer - INFO: init_asset.skip(545200) with limit 54521 runtime = 0:04:14.516161
113 P1076679-T140299165006656 init_asset_ray - 2022-05-09 11:32:51.722 - timer - INFO: init_asset.skip(381640) with limit 54521 runtime = 0:04:14.923320
114 P1076671-T140540818261824 init_asset_ray - 2022-05-09 11:32:51.834 - timer - INFO: init_asset.skip(654240) with limit 54521 runtime = 0:04:14.047892
115 P1076666-T139698529421120 init_asset_ray - 2022-05-09 11:32:51.875 - timer - INFO: init_asset.skip(490680) with limit 54521 runtime = 0:04:15.010879
116 P1076674-T139786773575488 init_asset_ray - 2022-05-09 11:32:51.937 - timer - INFO: init_asset.skip(54520) with limit 54521 runtime = 0:04:15.632990
117 P1076668-T1404645220273984 init_asset_ray - 2022-05-09 11:32:51.956 - timer - INFO: init_asset.skip(708760) with limit 54521 runtime = 0:04:14.114999
118 P1076664-T140230025688896 init_asset_ray - 2022-05-09 11:32:51.965 - timer - INFO: init_asset.skip(436160) with limit 54521 runtime = 0:04:15.132061
119 P1076665-T140037714892668 init_asset_ray - 2022-05-09 11:32:52.127 - timer - INFO: init_asset.skip(763280) with limit 54521 runtime = 0:04:13.992295
120 P1076669-T139794027345728 init_asset_ray - 2022-05-09 11:32:52.999 - timer - INFO: init_asset.skip(817800) with limit 54521 runtime = 0:04:14.507559
121 P1076530-T139940318795584 <module> - 2022-05-09 11:32:53.047 - timer - INFO: Runtime = 0:05:22.802448
122

```

图 6-7 结点生成模块运行日志

vul-kg\$ match (a) return count(distinct(a))

	count(distinct(a))
Table	1
A	1175978
Code	

Started streaming 1 records after 4 ms and completed after 168 ms.

图 6-8 数据库查询结点数量

6.3.2 关系生成

使用 Ray 库提供的 @ray.remote 装饰器与 ray.get() 方法，将全部漏洞数据根据并行度参数进行分割，开辟多进程对分块漏洞数据进行生成操作。

开辟 32 个进程生成“资产 -[具有]-> 漏洞，漏洞 -[影响]-> 资产；资产 -[是子级]-> 资产家族，资产家族 -[是父级]-> 资产；利用代码 -[攻击]-> 资产，资产 -[被攻击]-> 利用代码；利用代码 -[利用]-> 漏洞，漏洞 -[被利用]-> 漏洞代码”等关系，由于资产关系生成涉及正则匹配运算，无法有效利用数据库索引，且 CPU 性能消耗较大，因此生成时间较长。对总计约 120 万个结点共生成约 45 万条边（更多详细信息存储在边属性中），耗时约 1 小时 50 分钟，如图 6-9 与图 6-10 所示。

```

29 P2746955-T140522747782976- setup_logger - 2022-05-09 17:49:33,530 - timer - INFO: Logger "timer" logging to 0-root.log and 1-timer.log with log_file level 20 and log_stdout level 20,
30 P2746955-T140522747782976- create_rel_VAF_ray - 2022-05-09 17:49:33,530 - timer - INFO: create rel VAF ray().skip(21032) with limit 10517 start
31 P2746955-T140522747782976- setup_logger - 2022-05-09 17:49:33,530 - timer - INFO: Logger "timer" logging to 0-root.log and 1-timer.log with log_file level 20 and log_stdout level 20,
32 P2746953-T14004353711936- create_rel_VAF_ray - 2022-05-09 17:49:33,534 - timer - INFO: create rel VAF ray().skip(157740) with limit 10517 start
33 P2746945-T140183429986000- setup_logger - 2022-05-09 17:49:33,558 - timer - INFO: "timer" logging to 0-root.log and 1-timer.log with log_file level 20 and log_stdout level 20,
34 P2746945-T140183429986000- create_rel_VAF_ray - 2022-05-09 17:49:33,558 - timer - INFO: create rel VAF ray().skip(0) with limit 10517 start
35 P2746950-T139966837622592- create_rel_VAF_ray - 2022-05-09 19:11:33,543 - timer - INFO: create rel VAF ray().skip(147224) with limit 10517 runtime = 1:22:00.218574
36 P2746950-T139966837622592- create_rel_VAF_ray - 2022-05-09 19:11:33,566 - timer - INFO: create rel VAF ray().skip(168256) with limit 10517 start
37 P2746950-T139966837622592- create_rel_VAF_ray - 2022-05-09 19:11:35,454 - timer - INFO: create rel VAF ray().skip(168256) with limit 10517 runtime = 0:00:01.908214
38 P2746956-T140922863601472- create_rel_VAF_ray - 2022-05-09 19:19:09,416 - timer - INFO: create rel VAF ray().skip(31548) with limit 10517 runtime = 1:29:36.0996204
39 P2746956-T140922863601472- create_rel_VAF_ray - 2022-05-09 19:19:09,416 - timer - INFO: create rel VAF ray().skip(31548) with limit 10517 runtime = 1:29:36.0996204
40 P2746954-T14087559360084- create_rel_VAF_ray - 2022-05-09 19:22:40,720 - timer - INFO: create rel VAF ray().skip(10517) with limit 10517 runtime = 1:33:07.117757
41 P2746949-T140675886377698- create_rel_VAF_ray - 2022-05-09 19:25:01,459 - timer - INFO: create rel VAF ray().skip(10517) with limit 10517 runtime = 1:35:28.115858
42 P2746957-T140879881623872- create_rel_VAF_ray - 2022-05-09 19:26:19,775 - timer - INFO: create rel VAF ray().skip(42664) with limit 10517 runtime = 1:36:46.457588
43 P2746948-T139934798317376- create_rel_VAF_ray - 2022-05-09 19:27:59,893 - timer - INFO: create rel VAF ray().skip(52580) with limit 10517 runtime = 1:38:26.572940
44 P2746947-T140478029363088- create_rel_VAF_ray - 2022-05-09 19:28:11,718 - timer - INFO: create rel VAF ray().skip(136708) with limit 10517 runtime = 1:38:38.391414
45 P2746951-T139704795939264- create_rel_VAF_ray - 2022-05-09 19:28:25,939 - timer - INFO: create rel VAF ray().skip(126192) with limit 10517 runtime = 1:38:52.597466
46 P2746944-T14055234877160- create_rel_VAF_ray - 2022-05-09 19:28:29,856 - timer - INFO: create rel VAF ray().skip(84126) with limit 10517 runtime = 1:38:56.510728
47 P2746943-T140522747782976- create_rel_VAF_ray - 2022-05-09 19:29:01,856 - timer - INFO: create rel VAF ray().skip(91624) with limit 10517 runtime = 1:39:00.810119
48 P2746943-T140522747782976- create_rel_VAF_ray - 2022-05-09 19:29:14,269 - timer - INFO: create rel VAF ray().skip(93612) with limit 10517 runtime = 1:39:48.866059
49 P2746946-T139919657174848- create_rel_VAF_ray - 2022-05-09 19:30:07,878 - timer - INFO: create rel VAF ray().skip(115676) with limit 10517 runtime = 1:40:34.514543
50 P2746955-T140522747782976- create_rel_VAF_ray - 2022-05-09 19:30:39,935 - timer - INFO: create rel VAF ray().skip(21032) with limit 10517 runtime = 1:41:06.437943
51 P2746945-T140183429986000- create_rel_VAF_ray - 2022-05-09 19:31:39,317 - timer - INFO: create rel VAF ray().skip(0) with limit 10517 runtime = 1:42:05.792520
52 P2746958-T14048085344832- create_rel_VAF_ray - 2022-05-09 19:36:03,566 - timer - INFO: create rel VAF ray().skip(105160) with limit 10517 runtime = 1:46:30.069641
53 P2746816-T140287563937600- <module> - 2022-05-09 19:36:03,580 - timer - INFO: Runtime = 1:46:33.444462

```

图 6-9 关系生成模块运行日志

vul-kg\$ match (a)-[r]-(b) return count(distinct(a)),count(distinct(r))		
	count(distinct(a))	count(distinct(r))
A Text	227071	441872
Code		

Started streaming 1 records in less than 1 ms and completed after 347 ms.

图 6-10 数据库查询关系数量

6.3.3 关系融合

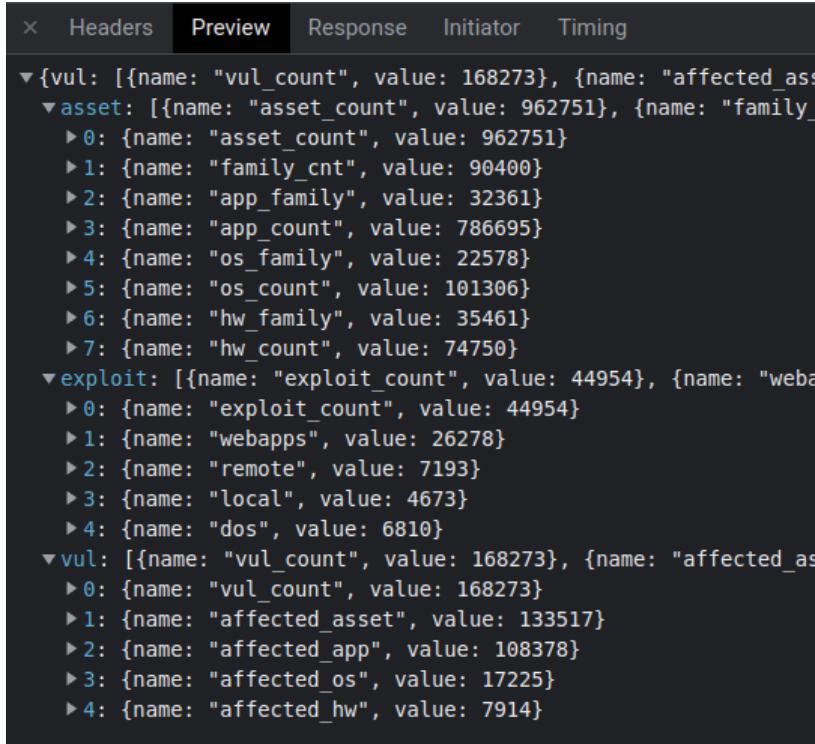
本系统设计上，结构化数据生成的漏洞实体具有 cve id 属性作为唯一标识；资产实体具有 cpe23uri 属性作为唯一标识；漏洞利用代码实体具有edb-id 属性作为唯一标识。且在数据库控制器对象的方法中，使用 MERGE 等 CQL 子句保证不会向数据库插入重复的实体或属性。因此当前知识图谱内并不会出现语义相同的不同实体或关系。

关系融合模块作为适配使用深度学习模型进行命名实体识别结点生成与关系抽取方案的模块，当前仅在系统数据加工管线中留有接口，以备未来扩展需要。因此暂不作测试。

6.4 数据库子系统与后端服务子系统测试

6.4.1 /api/graph/

向运行在 localhost 上的后端服务子系统发送 /api/graph/ 请求，获取当前知识图谱基础统计信息。后端返回各类资产数量统计、各类漏洞数量统计、各类代码利用数量统计等信息，如图 6-11 所示。



```

  {
    "asset": [
      {"name": "asset_count", "value": 962751},
      {"name": "family_cnt", "value": 90400},
      {"name": "app_family", "value": 32361},
      {"name": "app_count", "value": 786695},
      {"name": "os_family", "value": 22578},
      {"name": "os_count", "value": 101306},
      {"name": "hw_family", "value": 35461},
      {"name": "hw_count", "value": 74750}
    ],
    "exploit": [
      {"name": "exploit_count", "value": 44954},
      {"name": "webapps", "value": 26278},
      {"name": "remote", "value": 7193},
      {"name": "local", "value": 4673},
      {"name": "dos", "value": 6810}
    ],
    "vul": [
      {"name": "vul_count", "value": 168273},
      {"name": "affected_asset", "value": 133517},
      {"name": "affected_app", "value": 108378},
      {"name": "affected_os", "value": 17225},
      {"name": "affected_hw", "value": 7914}
    ]
  }

```

图 6-11 /api/graph/ 请求响应数据

耗时约 200 毫秒，如图 6-12 所示。

6.4.2 /api/graph/<limit>

向运行在 localhost 上的后端服务子系统发送 /api/graph/100 请求，获取当前知识图谱的 100 个漏洞结点以及与其相关的资产、利用代码结点、关系等信息，用于可视化。后端返回经绘图数据生成模块处理的绘图数据，包含分类、结点、关系三个数组，如图 6-13 所示。

耗时约 640 毫秒，如图 6-14 所示。

```
http://localhost:5000/api/graph/

HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 740
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.3 Python/3.8.10
Date: Sun, 15 May 2022 08:33:28 GMT

{"vul": [{"name": "vul_count", "value": 168273}, {"name": "affected"}]
Response file saved.

> 2022-05-15T163328.200.html

Response code: 200 (OK); Time: 200ms; Content length: 740 bytes
```

图 6-12 /api/graph/ 请求响应摘要

```
▼ {,...}
  ▼ categories: [{name: "漏洞"}, {name: "家族"}, {name: "资产"}],
    ► 0: {name: "漏洞"}
    ► 1: {name: "家族"}
    ► 2: {name: "资产"}
    ► 3: {name: "应用程序"}
    ► 4: {name: "操作系统"}
    ► 5: {name: "硬件"}
    ► 6: {name: "利用"}
  ▼ links: [,...]
  ▼ [0 .. 99]
    ► 0: {assets: ["cpe:2.3:a:numpy:numpy:*:*:*:*:*:*"], category: "ChildOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ChildOf", targetLineStyle: {curveness: 0.125}}
    ► 1: {category: "ParentOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ChildOf", targetLineStyle: {curveness: 0.125}}
    ► 2: {category: "ParentOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ChildOf", targetLineStyle: {curveness: 0.125}}
    ► 3: {assets: ["cpe:2.3:a:rsa:web_threat_detection:*:*:*"], category: "ChildOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ChildOf", targetLineStyle: {curveness: 0.125}}
    ► 4: {category: "ParentOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ChildOf", targetLineStyle: {curveness: 0.125}}
    ► 5: {category: "ChildOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ParentOf", targetLineStyle: {curveness: 0.125}}
    ► 6: {category: "ParentOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ChildOf", targetLineStyle: {curveness: 0.125}}
    ► 7: {category: "ChildOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ParentOf", targetLineStyle: {curveness: 0.125}}
    ► 8: {category: "ChildOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ChildOf", targetLineStyle: {curveness: 0.125}}
    ► 9: {category: "ParentOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ChildOf", targetLineStyle: {curveness: 0.125}}
    ► 10: {category: "ChildOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ParentOf", targetLineStyle: {curveness: 0.125}}
    ► 11: {category: "ParentOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ChildOf", targetLineStyle: {curveness: 0.125}}
    ► 12: {category: "ParentOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ChildOf", targetLineStyle: {curveness: 0.125}}
    ► 13: {category: "ChildOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ParentOf", targetLineStyle: {curveness: 0.125}}
    ► 14: {category: "ParentOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ChildOf", targetLineStyle: {curveness: 0.125}}
    ► 15: {category: "ChildOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ParentOf", targetLineStyle: {curveness: 0.125}}
    ► 16: {assets: ["cpe:2.3:a:ibm:tivoli_directory_server:*"], category: "ParentOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ChildOf", targetLineStyle: {curveness: 0.125}}
    ► 17: {category: "ParentOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ChildOf", targetLineStyle: {curveness: 0.125}}
    ► 18: {category: "ParentOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ChildOf", targetLineStyle: {curveness: 0.125}}
    ► 19: {category: "ParentOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ChildOf", targetLineStyle: {curveness: 0.125}}
    ► 20: {category: "ParentOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ChildOf", targetLineStyle: {curveness: 0.125}}
    ► 21: {category: "ParentOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ChildOf", targetLineStyle: {curveness: 0.125}}
    ► 22: {category: "ParentOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ChildOf", targetLineStyle: {curveness: 0.125}}
    ► 23: {category: "ParentOf", lineStyle: {curveness: 0.125}, parentCategory: "ParentOf", targetCategory: "ChildOf", targetLineStyle: {curveness: 0.125}}
```

图 6-13 /api/graph/100 请求响应数据

```

http://127.0.0.1:5000/api/graph/100

HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 1630803
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.3 Python/3.8.10
Date: Sun, 15 May 2022 08:33:13 GMT

Response file saved.
> 2022-05-15T163313.200.json

Response code: 200 (OK); Time: 637ms; Content length: 1630713 bytes

```

图 6-14 /api/graph/100 请求响应摘要

6.5 前端子系统测试

6.5.1 控制台视图测试

图 6-15 所示为控制台概览。从 localhost 后端 /api/graph/ 端口加载数据，实际返回，根据瀑布流图可看出，通过刷新页面冷加载控制台界面及各种网络请求耗时约 850ms，且加载过程中可视化图表有加载指示器提示正在加载、数据列表有默认数据“0”填充而非显示空白，用户体验较好。如图 6-16 所示。

6.5.2 可视化视图测试

图 6-17 所示为知识图谱概览。从 localhost 后端 /api/graph/100 端口加载数据，实际返回 1070 个结点、2035 条关系边，根据瀑布流图可看出，通过刷新页面冷加载可视化力引导图耗时约 1300ms，且加载过程中有加载指示器提示正在加载，用户体验较好。如图 6-18 所示。

图 6-19 所示为知识图谱细节图。力引导图为有向图，因此结点间的无向边使用两条有向边表示。图中蓝色结点为漏洞结点，绿色结点为资产家族结点，红色结点为受影响资产结点。可以看到，通过将漏洞所影响的大量资产关系替换成漏洞到资产家族的一个关系，大幅简化了图的结构，在使视觉观感清晰的同时，将鼠标移至漏洞至资产家族的边上可显示受该漏洞影响的所有资产，保证图谱数据的详细精确。

图 6-20 所示为当鼠标移动至或触摸图中元素，包括各类结点和各类关系，会显示



图 6-15 控制台视图预览

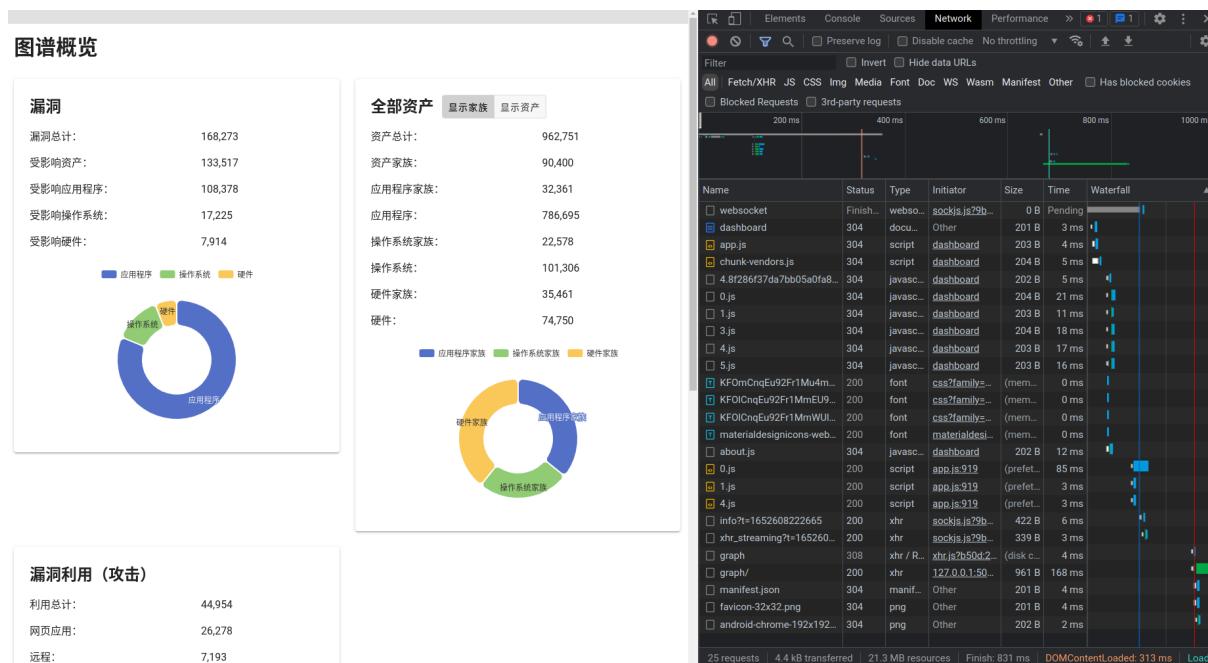


图 6-16 控制台视图加载请求瀑布流

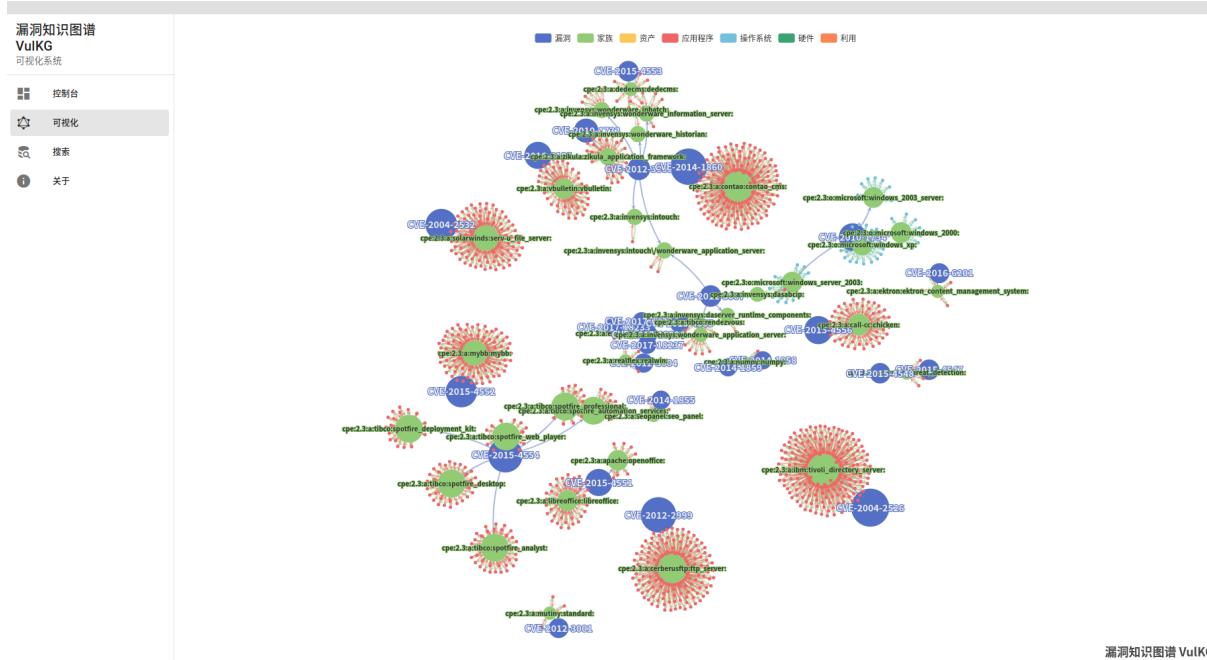


图 6-17 可视化视图概览

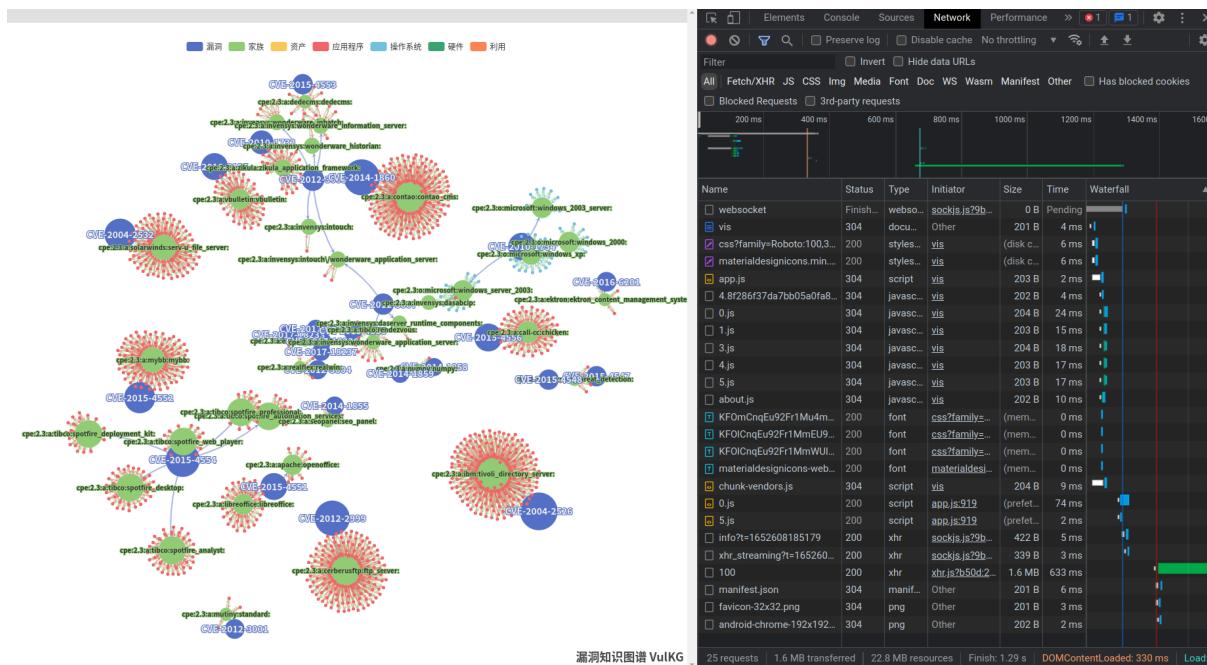


图 6-18 可视化视图加载请求瀑布流

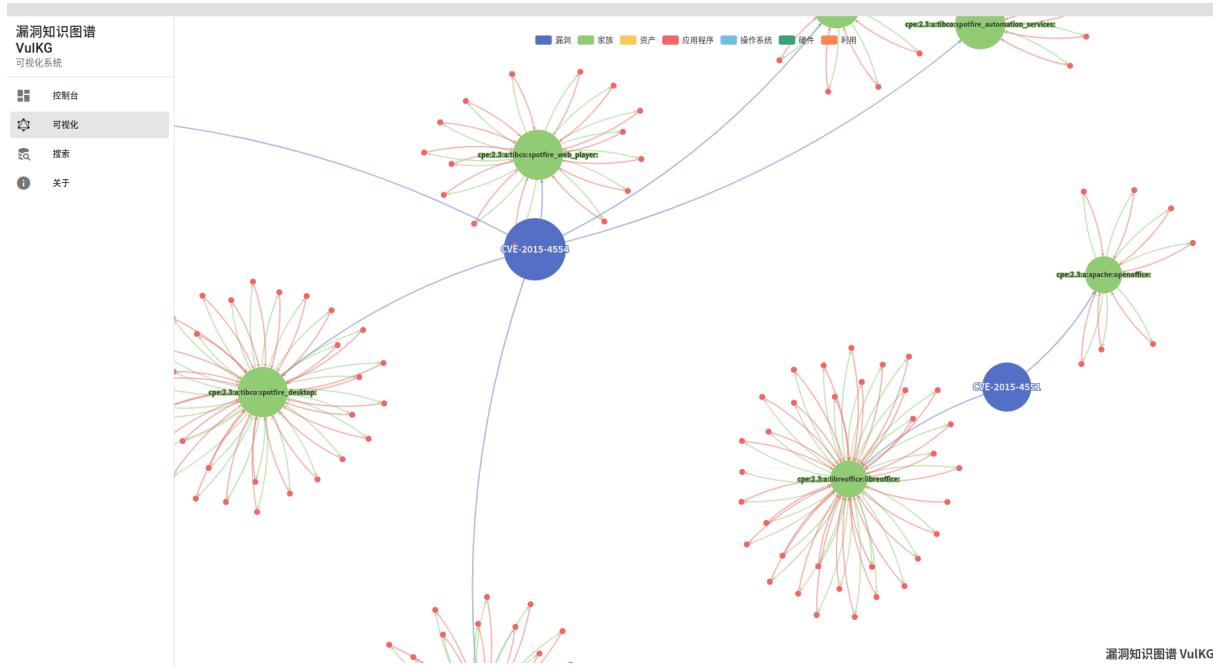


图 6-19 可视化视图细节

tooltip 指示，展现当前高亮元素的详细信息。

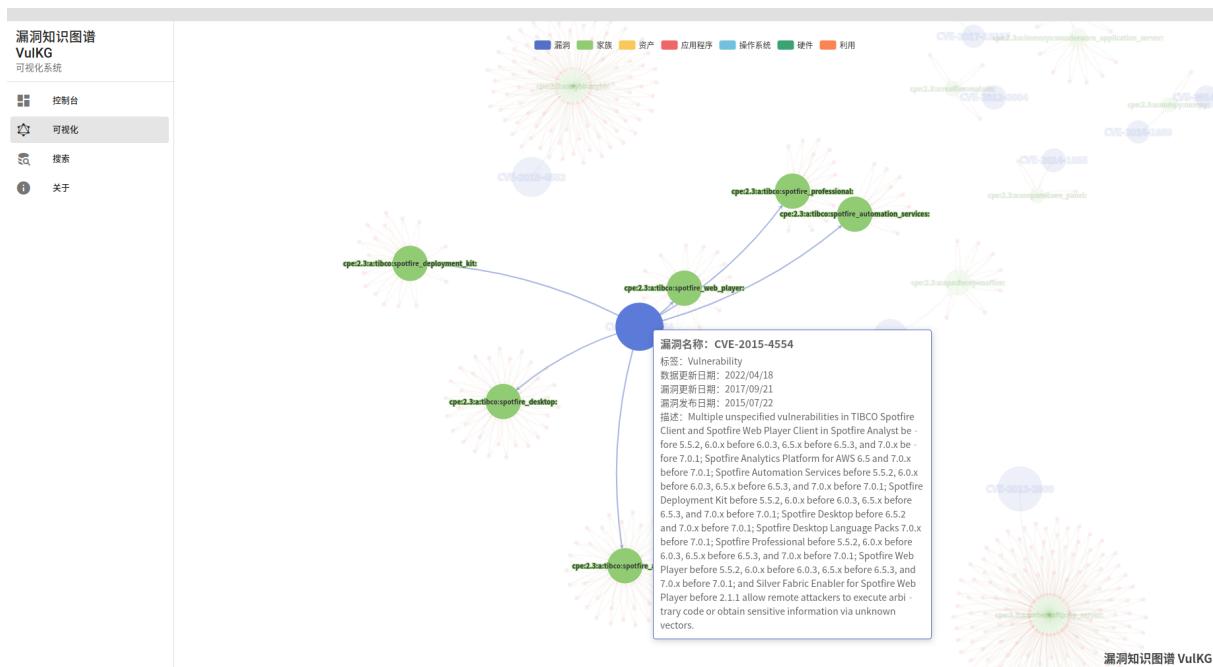


图 6-20 可视化视图 tooltip 展示详细信息

6.5.3 搜索视图测试

图??所示为搜索视图。搜索视图提供一个悬浮搜索框，允许用户对知识图谱进行精确查询。输入查询关键字并按下回车将发送异步 I/O 请求，后端服务将搜索整个知识图谱查找关键字相关信息，并将绘图数据返回前端。前端回调使用 ECharts 可视化库绘制

力引导图展示数据。

6.5.4 关于视图测试

图??所示为关于视图。关于视图清晰地展示了本系统名称、功能、作者、使用指南等信息。

第七章 结束语

参考文献

- [1] Manifesto for Agile Software Development [EB/OL]. <http://agilemanifesto.org/iso/zhchs/manifesto.htm>.
- [2] Minimum Viable Product: a guide [EB/OL]. <http://www.startuplessonslearned.com/2009/08/minimum-viable-product-guide.html>.
- [3] Bourque Pierre , Fairley R E, IEEE Computer Society. Guide to the software engineering body of knowledge [M]. 2014. OCLC: 973217192.
- [4] Hogan Aidan, Blomqvist Eva, Cochez Michael 等. Knowledge Graphs [J/OL]. ACM Computing Surveys. 54 (4). 2022, May: 1–37. <http://arxiv.org/abs/2003.02320>arXiv: 2003.02320.
- [5] Singleton Design Pattern [EB/OL]. https://sourcemaking.com/design_patterns/singleton.
- [6] Factory Method Design Pattern [EB/OL]. https://sourcemaking.com/design_patterns/factory_method.

致 谢

此处请写 dsffsd 致谢的内容。

它可以有多段。d

附录

附录 1 缩略语表

表 附-1 基于浏览者行为的特征

特征	描述	形式与理论范围
点赞量	微博的点赞数量	数值, \mathbb{N}
评论量	微博的评论数量	数值, \mathbb{N}
转发量	微博的转发数量	数值, \mathbb{N}

表 附-2 基于浏览者行为的复杂特征

类别	特征	不知道叫什么的表头	
		描述	形式与理论范围
正常互动	点赞量	微博的点赞数量	数值, \mathbb{N}
	评论量	微博的评论数量	数值, \mathbb{N}
	转发量	微博的转发数量	数值, \mathbb{N}
非正常互动	羡慕量	微博的羡慕数量	数值, \mathbb{N}

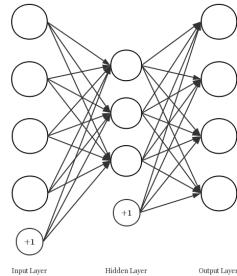


图 附-1 自编码器结构

代码 附-1 减法

```

1 def minusFunc(a, b):
2     return a - b

```

$$\max_{\mathbf{W}} \operatorname{tr}(\mathbf{W}^T \mathbf{X} \mathbf{X}^T \mathbf{W}) \quad \text{式 (附-1)}$$

附录 2 数学符号

数和数组

a	标量 (整数或实数)
\mathbf{a}	向量
$dim()$	向量的维数
A	矩阵
A^T	矩阵 A 的转置
I	单位矩阵 (维度依据上下文而定)
$diag(\mathbf{a})$	对角方阵, 其中对角元素由向量 \mathbf{a} 确定

SOCIAL SCIENCE

The spread of true and false news online

Soroush Vosoughi,¹ Deb Roy,¹ Sinan Aral^{2*}

We investigated the differential diffusion of all of the verified true and false news stories distributed on Twitter from 2006 to 2017. The data comprise ~126,000 stories tweeted by ~3 million people more than 4.5 million times. We classified news as true or false using information from six independent fact-checking organizations that exhibited 95 to 98% agreement on the classifications. Falsehood diffused significantly farther, faster, deeper, and more broadly than the truth in all categories of information, and the effects were more pronounced for false political news than for false news about terrorism, natural disasters, science, urban legends, or financial information. We found that false news was more novel than true news, which suggests that people were more likely to share novel information. Whereas false stories inspired fear, disgust, and surprise in replies, true stories inspired anticipation, sadness, joy, and trust. Contrary to conventional wisdom, robots accelerated the spread of true and false news at the same rate, implying that false news spreads more than the truth because humans, not robots, are more likely to spread it.

Foundational theories of decision-making (1–3), cooperation (4), communication (5), and markets (6) all view some conceptualization of truth or accuracy as central to the functioning of nearly every human endeavor. Yet, both true and false information spreads rapidly through online media. Defining what is true and false has become a common political strategy, replacing debates based on a mutually agreed on set of facts. Our economies are not immune to the spread of falsity either. False rumors have affected stock prices and the motivation for large-scale investments, for example, wiping out \$130 billion in stock value after a false tweet claimed that Barack Obama was injured in an explosion (7). Indeed, our responses to everything from natural disasters (8, 9) to terrorist attacks (10) have been disrupted by the spread of false news online.

New social technologies, which facilitate rapid information sharing and large-scale information cascades, can enable the spread of misinformation (i.e., information that is inaccurate or misleading). But although more and more of our access to information and news is guided by these new technologies (11), we know little about their contribution to the spread of falsity online. Though considerable attention has been paid to anecdotal analyses of the spread of false news by the media (12), there are few large-scale empirical investigations of the diffusion of misinformation or its social origins. Studies of the spread of misinformation are currently limited to analyses of small, ad hoc samples that ignore two of the most important scientific questions: How do truth and falsity diffuse differently, and what factors of human judgment explain these differences?

Current work analyzes the spread of single rumors, like the discovery of the Higgs boson (13) or the Haitian earthquake of 2010 (14), and multiple rumors from a single disaster event, like the Boston Marathon bombing of 2013 (10), or it develops theoretical models of rumor diffusion (15), methods for rumor detection (16), credibility evaluation (17, 18), or interventions to curtail the spread of rumors (19). But almost no studies comprehensively evaluate differences in the spread of truth and falsity across topics or examine why false news may spread differently than the truth. For example, although Del Vicario *et al.* (20) and Bessi *et al.* (21) studied the spread of scientific and conspiracy-theory stories, they did not evaluate their veracity. Scientific and conspiracy-theory stories can both be either true or false, and they differ on stylistic dimensions that are important to their spread but orthogonal to their veracity. To understand the spread of false news, it is necessary to examine diffusion after differentiating true and false scientific stories and true and false conspiracy-theory stories and controlling for the topical and stylistic differences between the categories themselves. The only study to date that segments rumors by veracity is that of Friggeri *et al.* (19), who analyzed ~4000 rumors spreading on Facebook and focused more on how fact checking affects rumor propagation than on how falsity diffuses differently than the truth (22).

In our current political climate and in the academic literature, a fluid terminology has arisen around “fake news,” foreign interventions in U.S. politics through social media, and our understanding of what constitutes news, fake news, false news, rumors, rumor cascades, and other related terms. Although, at one time, it may have been appropriate to think of fake news as referring to the veracity of a news story, we now believe that this phrase has been irredeemably polarized in our current political and media climate. As politicians have implemented a political strategy of labeling news sources that do not

support their positions as unreliable or fake news, whereas sources that support their positions are labeled reliable or not fake, the term has lost all connection to the actual veracity of the information presented, rendering it meaningless for use in academic classification. We have therefore explicitly avoided the term fake news throughout this paper and instead use the more objectively verifiable terms “true” or “false” news. Although the terms fake news and misinformation also imply a willful distortion of the truth, we do not make any claims about the intent of the purveyors of the information in our analyses. We instead focus our attention on veracity and stories that have been verified as true or false.

We also purposefully adopt a broad definition of the term news. Rather than defining what constitutes news on the basis of the institutional source of the assertions in a story, we refer to any asserted claim made on Twitter as news (we defend this decision in the supplementary materials section on “reliable sources,” section S1.2). We define news as any story or claim with an assertion in it and a rumor as the social phenomena of a news story or claim spreading or diffusing through the Twitter network. That is, rumors are inherently social and involve the sharing of claims between people. News, on the other hand, is an assertion with claims, whether it is shared or not.

A rumor cascade begins on Twitter when a user makes an assertion about a topic in a tweet, which could include written text, photos, or links to articles online. Others then propagate the rumor by retweeting it. A rumor’s diffusion process can be characterized as having one or more cascades, which we define as instances of a rumor-spreading pattern that exhibit an unbroken retweet chain with a common, singular origin. For example, an individual could start a rumor cascade by tweeting a story or claim with an assertion in it, and another individual could independently start a second cascade of the same rumor (pertaining to the same story or claim) that is completely independent of the first cascade, except that it pertains to the same story or claim. If they remain independent, they represent two cascades of the same rumor. Cascades can be as small as size one (meaning no one retweeted the original tweet). The number of cascades that make up a rumor is equal to the number of times the story or claim was independently tweeted by a user (not retweeted). So, if a rumor “A” is tweeted by 10 people separately, but not retweeted, it would have 10 cascades, each of size one. Conversely, if a second rumor “B” is independently tweeted by two people and each of those two tweets is retweeted 100 times, the rumor would consist of two cascades, each of size 100.

Here we investigate the differential diffusion of true, false, and mixed (partially true, partially false) news stories using a comprehensive data set of all of the fact-checked rumor cascades that spread on Twitter from its inception in 2006 to 2017. The data include ~126,000 rumor cascades spread by ~3 million people more than 4.5 million times. We sampled all rumor cascades investigated by six independent fact-checking organizations

¹Massachusetts Institute of Technology (MIT), the Media Lab, E14-526, 75 Amherst Street, Cambridge, MA 02142, USA. ²MIT, E62-364, 100 Main Street, Cambridge, MA 02142, USA.

*Corresponding author. Email: sinan@mit.edu

(snopes.com, politifact.com, factcheck.org, truthfiction.com, hoax-slayer.com, and urbanlegends.about.com) by parsing the title, body, and verdict (true, false, or mixed) of each rumor investigation reported on their websites and automatically collecting the cascades corresponding to those rumors on Twitter. The result was a sample of rumor cascades whose veracity had been agreed on by these organizations between 95 and 98% of the time. We cataloged the diffusion of the rumor cascades by collecting all English-language replies to tweets that contained a link to any of the aforementioned websites from 2006 to 2017 and used optical character recognition to extract text from images where needed. For each reply tweet, we extracted the original tweet being replied to and all the retweets of the original tweet. Each retweet cascade represents a rumor propagating on Twitter that has been verified as true or false by the fact-checking organizations (see the supplementary materials for more details on cascade construction). We then quantified the cascades'

depth (the number of retweet hops from the origin tweet over time, where a hop is a retweet by a new unique user), size (the number of users involved in the cascade over time), maximum breadth (the maximum number of users involved in the cascade at any depth), and structural virality (23) (a measure that interpolates between content spread through a single, large broadcast and that which spreads through multiple generations, with any one individual directly responsible for only a fraction of the total spread) (see the supplementary materials for more detail on the measurement of rumor diffusion).

As a rumor is retweeted, the depth, size, maximum breadth, and structural virality of the cascade increase (Fig. 1A). A greater fraction of false rumors experienced between 1 and 1000 cascades, whereas a greater fraction of true rumors experienced more than 1000 cascades (Fig. 1B); this was also true for rumors based on political news (Fig. 1D). The total number of false rumors peaked at the end of both 2013 and 2015 and again at the

end of 2016, corresponding to the last U.S. presidential election (Fig. 1E). The data also show clear increases in the total number of false political rumors during the 2012 and 2016 U.S. presidential elections (Fig. 1E) and a spike in rumors that contained partially true and partially false information during the Russian annexation of Crimea in 2014 (Fig. 1E). Politics was the largest rumor category in our data, with ~45,000 cascades, followed by urban legends, business, terrorism, science, entertainment, and natural disasters (Fig. 1F).

When we analyzed the diffusion dynamics of true and false rumors, we found that falsehood diffused significantly farther, faster, deeper, and more broadly than the truth in all categories of information [Kolmogorov-Smirnov (K-S) tests are reported in tables S3 to S10]. A significantly greater fraction of false cascades than true cascades exceeded a depth of 10, and the top 0.01% of false cascades diffused eight hops deeper into the Twittersphere than the truth, diffusing to depths

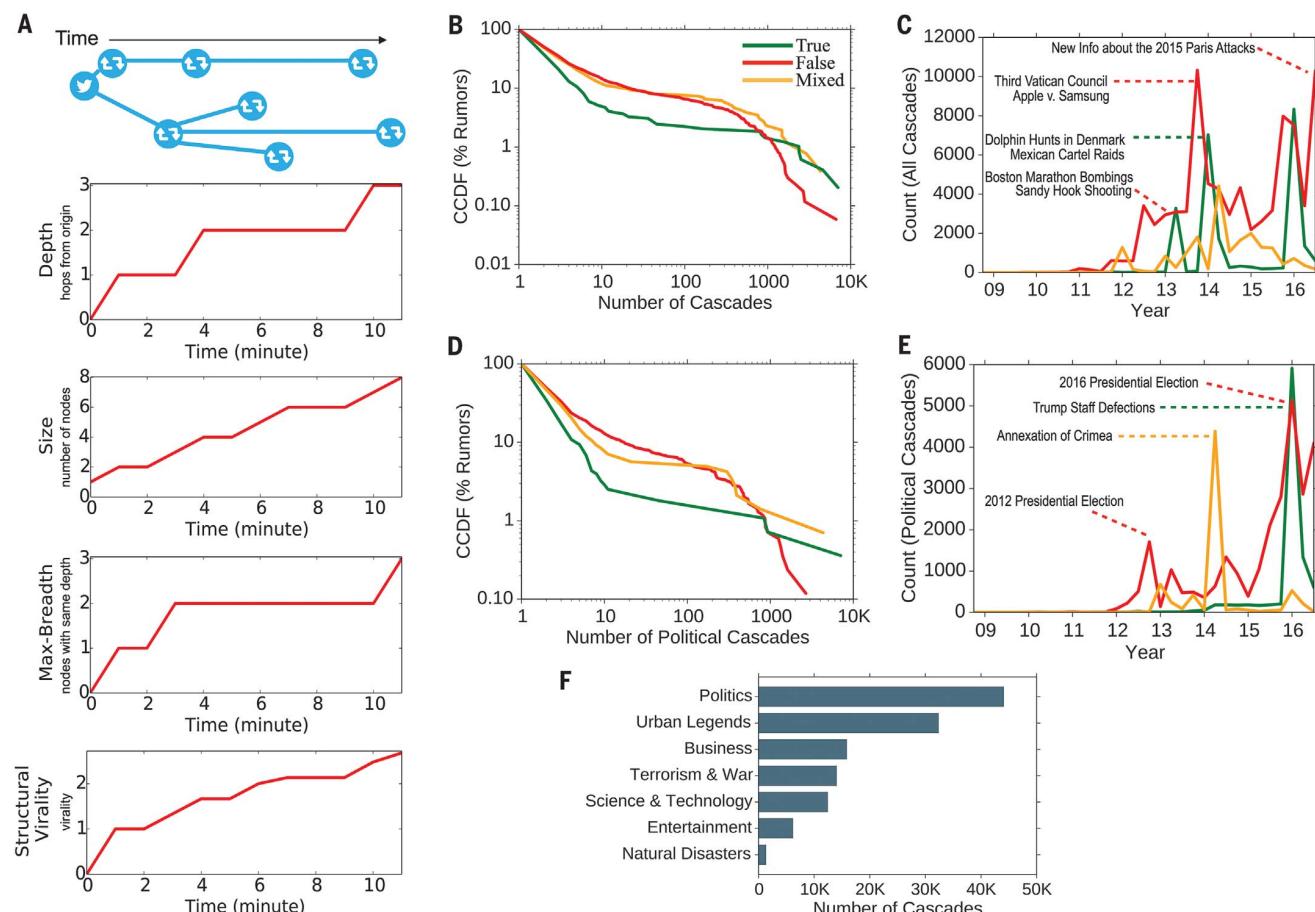


Fig. 1. Rumor cascades. (A) An example rumor cascade collected by our method as well as its depth, size, maximum breadth, and structural virality over time. “Nodes” are users. (B) The complementary cumulative distribution functions (CCDFs) of true, false, and mixed (partially true and partially false) cascades, measuring the fraction of rumors that exhibit a given number of cascades. (C) Quarterly counts of all true, false, and mixed rumor cascades

that diffused on Twitter between 2006 and 2017, annotated with example rumors in each category. (D) The CCDFs of true, false, and mixed political cascades. (E) Quarterly counts of all true, false, and mixed political rumor cascades that diffused on Twitter between 2006 and 2017, annotated with example rumors in each category. (F) A histogram of the total number of rumor cascades in our data across the seven most frequent topical categories.

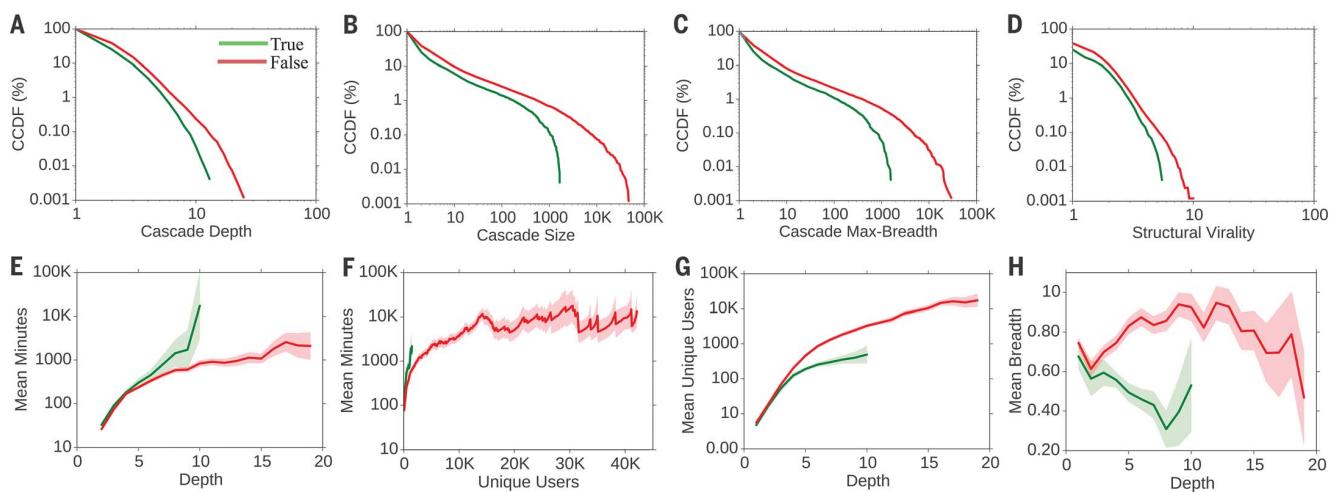


Fig. 2. Complementary cumulative distribution functions (CCDFs) of true and false rumor cascades. (A) Depth. (B) Size. (C) Maximum breadth. (D) Structural virility. (E and F) The number of minutes it takes for true and false rumor cascades to reach any (E) depth and (F) number of unique Twitter users. (G) The number of unique Twitter

users reached at every depth and (H) the mean breadth of true and false rumor cascades at every depth. In (H), plot is lognormal. Standard errors were clustered at the rumor level (i.e., cascades belonging to the same rumor were clustered together; see supplementary materials for additional details).

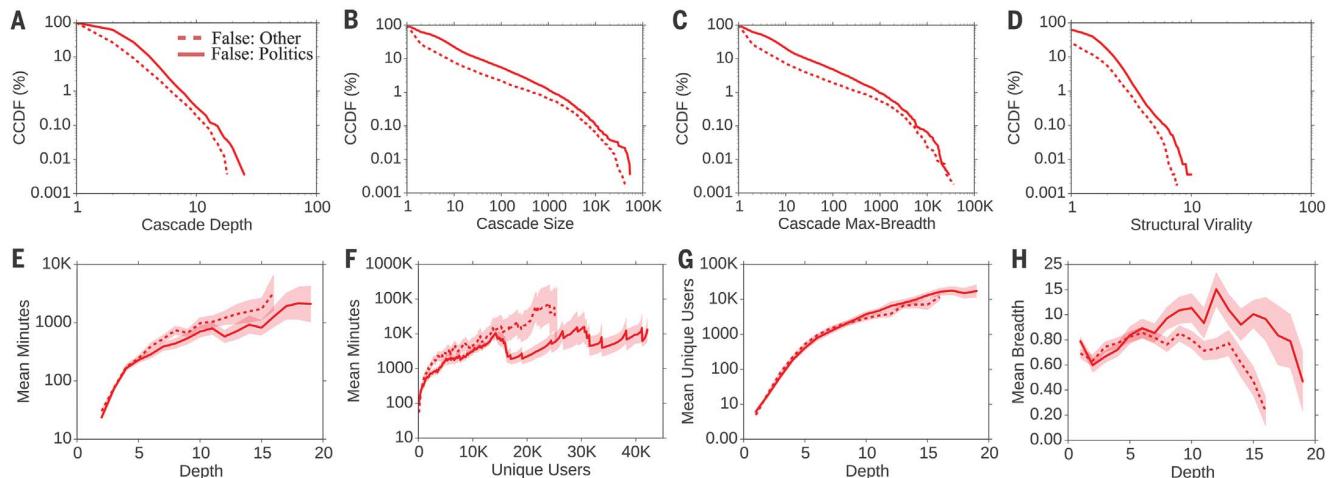


Fig. 3. Complementary cumulative distribution functions (CCDFs) of false political and other types of rumor cascades. (A) Depth. (B) Size. (C) Maximum breadth. (D) Structural virility. (E and F) The number of minutes it takes for false political and other false news cascades to reach

any (E) depth and (F) number of unique Twitter users. (G) The number of unique Twitter users reached at every depth and (H) the mean breadth of these false rumor cascades at every depth. In (H), plot is lognormal. Standard errors were clustered at the rumor level.

greater than 19 hops from the origin tweet (Fig. 2A). Falsehood also reached far more people than the truth. Whereas the truth rarely diffused to more than 1000 people, the top 1% of false-news cascades routinely diffused to between 1000 and 100,000 people (Fig. 2B). Falsehood reached more people at every depth of a cascade than the truth, meaning that many more people retweeted falsehood than they did the truth (Fig. 2C). The spread of falsehood was aided by its virality, meaning that falsehood did not simply spread through broadcast dynamics but rather through peer-to-peer diffusion characterized by a viral branching process (Fig. 2D).

It took the truth about six times as long as falsehood to reach 1500 people (Fig. 2F) and 20 times as long as falsehood to reach a cascade depth of 10 (Fig. 2E). As the truth never diffused beyond a depth of 10, we saw that falsehood reached a depth of 19 nearly 10 times faster than the truth reached a depth of 10 (Fig. 2E). Falsehood also diffused significantly more broadly (Fig. 2H) and was retweeted by more unique users than the truth at every cascade depth (Fig. 2G).

False political news (Fig. 3D) traveled deeper (Fig. 3A) and more broadly (Fig. 3C), reached more people (Fig. 3B), and was more viral than any other category of false information (Fig. 3D). False po-

litical news also diffused deeper more quickly (Fig. 3E) and reached more than 20,000 people nearly three times faster than all other types of false news reached 10,000 people (Fig. 3F). Although the other categories of false news reached about the same number of unique users at depths between 1 and 10, false political news routinely reached the most unique users at depths greater than 10 (Fig. 3G). Although all other categories of false news traveled slightly more broadly at shallower depths, false political news traveled more broadly at greater depths, indicating that more-popular false political news items exhibited broader and more-accelerated diffusion dynamics

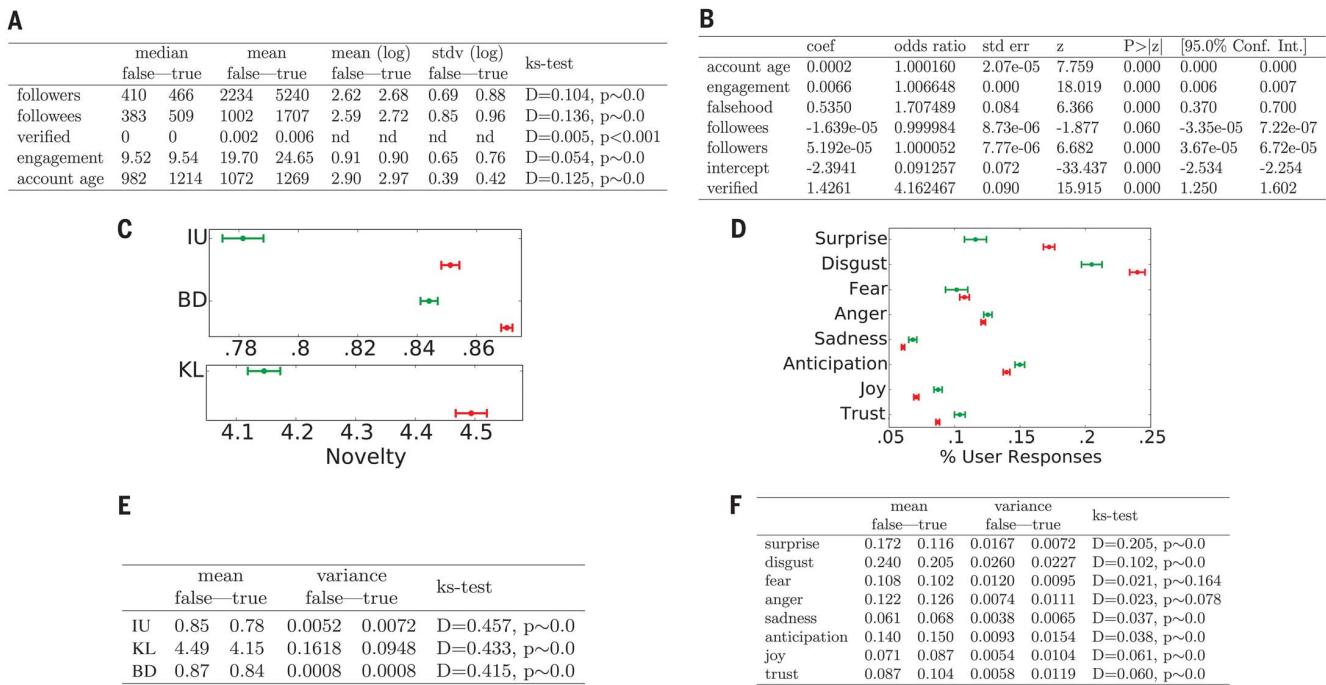


Fig. 4. Models estimating correlates of news diffusion, the novelty of true and false news, and the emotional content of replies to news.

(A) Descriptive statistics on users who participated in true and false rumor cascades as well as K-S tests of the differences in the distributions of these measures across true and false rumor cascades. (B) Results of a logistic regression model estimating users' likelihood of retweeting a rumor as a function of variables shown at the left. coeff, logit coefficient; z, z score. (C) Differences in the information uniqueness (IU), scaled Bhattacharyya distance (BD), and K-L divergence (KL) of true (green) and false (red) rumor tweets compared to the corpus of prior tweets the user was exposed to in the 60 days before retweeting the rumor tweet. (D) The emotional

content of replies to true (green) and false (red) rumor tweets across seven dimensions categorized by the NRC. (E) Mean and variance of the IU, KL, and BD of true and false rumor tweets compared to the corpus of prior tweets the user has seen in the 60 days before seeing the rumor tweet as well as K-S tests of their differences across true and false rumors. (F) Mean and variance of the emotional content of replies to true and false rumor tweets across seven dimensions categorized by the NRC as well as K-S tests of their differences across true and false rumors. All standard errors are clustered at the rumor level, and all models are estimated with cluster-robust standard errors at the rumor level.

(Fig. 3H). Analysis of all news categories showed that news about politics, urban legends, and science spread to the most people, whereas news about politics and urban legends spread the fastest and were the most viral in terms of their structural virality (see fig. S11 for detailed comparisons across all topics).

One might suspect that structural elements of the network or individual characteristics of the users involved in the cascades explain why falsity travels with greater velocity than the truth. Perhaps those who spread falsity "followed" more people, had more followers, tweeted more often, were more often "verified" users, or had been on Twitter longer. But when we compared users involved in true and false rumor cascades, we found that the opposite was true in every case. Users who spread false news had significantly fewer followers (K-S test = 0.104, $P \sim 0.0$), followed significantly fewer people (K-S test = 0.136, $P \sim 0.0$), were significantly less active on Twitter (K-S test = 0.054, $P \sim 0.0$), were verified significantly less often (K-S test = 0.004, $P < 0.001$), and had been on Twitter for significantly less time (K-S test = 0.125, $P \sim 0.0$) (Fig. 4A). Falsehood

diffused farther and faster than the truth despite these differences, not because of them.

When we estimated a model of the likelihood of retweeting, we found that falsehoods were 70% more likely to be retweeted than the truth (Wald chi-square test, $P \sim 0.0$), even when controlling for the account age, activity level, and number of followers and followees of the original tweeter, as well as whether the original tweeter was a verified user (Fig. 4B). Because user characteristics and network structure could not explain the differential diffusion of truth and falsity, we sought alternative explanations for the differences in their diffusion dynamics.

One alternative explanation emerges from information theory and Bayesian decision theory. Novelty attracts human attention (24), contributes to productive decision-making (25), and encourages information sharing (26) because novelty updates our understanding of the world. When information is novel, it is not only surprising, but also more valuable, both from an information theoretic perspective [in that it provides the greatest aid to decision-making (25)] and from a social perspective [in that it conveys so-

cial status on one that is "in the know" or has access to unique "inside" information (26)]. We therefore tested whether falsity was more novel than the truth and whether Twitter users were more likely to retweet information that was more novel.

To assess novelty, we randomly selected ~5000 users who propagated true and false rumors and extracted a random sample of ~25,000 tweets that they were exposed to in the 60 days prior to their decision to retweet a rumor. We then specified a latent Dirichlet Allocation Topic model (27), with 200 topics and trained on 10 million English-language tweets, to calculate the information distance between the rumor tweets and all the prior tweets that users were exposed to before retweeting the rumor tweets. This generated a probability distribution over the 200 topics for each tweet in our data set. We then measured how novel the information in the true and false rumors was by comparing the topic distributions of the rumor tweets with the topic distributions of the tweets to which users were exposed in the 60 days before their retweet. We found that false rumors were significantly more

novel than the truth across all novelty metrics, displaying significantly higher information uniqueness ($K\text{-}S$ test = 0.457, $P \sim 0.0$) (28), Kullback-Leibler ($K\text{-}L$) divergence ($K\text{-}S$ test = 0.433, $P \sim 0.0$) (29), and Bhattacharyya distance ($K\text{-}S$ test = 0.415, $P \sim 0.0$) (which is similar to the Hellinger distance) (30). The last two metrics measure differences between probability distributions representing the topical content of the incoming tweet and the corpus of previous tweets to which users were exposed.

Although false rumors were measurably more novel than true rumors, users may not have perceived them as such. We therefore assessed users' perceptions of the information contained in true and false rumors by comparing the emotional content of replies to true and false rumors. We categorized the emotion in the replies by using the leading lexicon curated by the National Research Council Canada (NRC), which provides a comprehensive list of ~140,000 English words and their associations with eight emotions based on Plutchik's (31) work on basic emotion—anger, fear, anticipation, trust, surprise, sadness, joy, and disgust (32)—and a list of ~32,000 Twitter hashtags and their weighted associations with the same emotions (33). We removed stop words and URLs from the reply tweets and calculated the fraction of words in the tweets that related to each of the eight emotions, creating a vector of emotion weights for each reply that summed to one across the emotions. We found that false rumors inspired replies expressing greater surprise ($K\text{-}S$ test = 0.205, $P \sim 0.0$), corroborating the novelty hypothesis, and greater disgust ($K\text{-}S$ test = 0.102, $P \sim 0.0$), whereas the truth inspired replies that expressed greater sadness ($K\text{-}S$ test = 0.037, $P \sim 0.0$), anticipation ($K\text{-}S$ test = 0.038, $P \sim 0.0$), joy ($K\text{-}S$ test = 0.061, $P \sim 0.0$), and trust ($K\text{-}S$ test = 0.060, $P \sim 0.0$) (Fig. 4, D and F). The emotions expressed in reply to falsehoods may illuminate additional factors, beyond novelty, that inspire people to share false news. Although we cannot claim that novelty causes retweets or that novelty is the only reason why false news is retweeted more often, we do find that false news is more novel and that novel information is more likely to be retweeted.

Numerous diagnostic statistics and manipulation checks validated our results and confirmed their robustness. First, as there were multiple cascades for every true and false rumor, the variance of error terms associated with cascades corresponding to the same rumor will be correlated. We therefore specified cluster-robust standard errors and calculated all variance statistics clustered at the rumor level. We tested the robustness of our findings to this specification by comparing analyses with and without clustered errors and found that, although clustering reduced the precision of our estimates as expected, the directions, magnitudes, and significance of our results did not change, and chi-square ($P \sim 0.0$) and deviance (d) goodness-of-fit tests ($d = 3.4649 \times 10^{-6}$, $P \sim 1.0$) indicate that the models are well specified (see supplementary materials for more detail).

Second, a selection bias may arise from the restriction of our sample to tweets fact checked by the six organizations we relied on. Fact checking may select certain types of rumors or draw additional attention to them. To validate the robustness of our analysis to this selection and the generalizability of our results to all true and false rumor cascades, we independently verified a second sample of rumor cascades that were not verified by any fact-checking organization. These rumors were fact checked by three undergraduate students at Massachusetts Institute of Technology (MIT) and Wellesley College. We trained the students to detect and investigate rumors with our automated rumor-detection algorithm running on 3 million English-language tweets from 2016 (34). The undergraduate annotators investigated the veracity of the detected rumors using simple search queries on the web. We asked them to label the rumors as true, false, or mixed on the basis of their research and to discard all rumors previously investigated by one of the fact-checking organizations. The annotators, who worked independently and were not aware of one another, agreed on the veracity of 90% of the 13,240 rumor cascades that they investigated and achieved a Fleiss' kappa of 0.88. When we compared the diffusion dynamics of the true and false rumors that the annotators agreed on, we found results nearly identical to those estimated with our main data set (see fig. S17). False rumors in the robustness data set had greater depth ($K\text{-}S$ test = 0.139, $P \sim 0.0$), size ($K\text{-}S$ test = 0.131, $P \sim 0.0$), maximum breadth ($K\text{-}S$ test = 0.139, $P \sim 0.0$), structural virality ($K\text{-}S$ test = 0.066, $P \sim 0.0$), and speed (fig. S17) and a greater number of unique users at each depth (fig. S17). When we broadened the analysis to include majority-rule labeling, rather than unanimity, we again found the same results (see supplementary materials for results using majority-rule labeling).

Third, although the differential diffusion of truth and falsity is interesting with or without robot, or bot, activity, one may worry that our conclusions about human judgment may be biased by the presence of bots in our analysis. We therefore used a sophisticated bot-detection algorithm (35) to identify and remove all bots before running the analysis. When we added bot traffic back into the analysis, we found that none of our main conclusions changed—false news still spread farther, faster, deeper, and more broadly than the truth in all categories of information. The results remained the same when we removed all tweet cascades started by bots, including human retweets of original bot tweets (see supplementary materials, section S8.3) and when we used a second, independent bot-detection algorithm (see supplementary materials, section S8.3.5) and varied the algorithm's sensitivity threshold to verify the robustness of our analysis (see supplementary materials, section S8.3.4). Although the inclusion of bots, as measured by the two state-of-the-art bot-detection algorithms we used in our analysis, accelerated the spread of both true and false news, it affected their spread roughly equally. This suggests that false

news spreads farther, faster, deeper, and more broadly than the truth because humans, not robots, are more likely to spread it.

Finally, more research on the behavioral explanations of differences in the diffusion of true and false news is clearly warranted. In particular, more robust identification of the factors of human judgment that drive the spread of true and false news online requires more direct interaction with users through interviews, surveys, lab experiments, and even neuroimaging. We encourage these and other approaches to the investigation of the factors of human judgment that drive the spread of true and false news in future work.

False news can drive the misallocation of resources during terror attacks and natural disasters, the misalignment of business investments, and misinformed elections. Unfortunately, although the amount of false news online is clearly increasing (Fig. 1, C and E), the scientific understanding of how and why false news spreads is currently based on ad hoc rather than large-scale systematic analyses. Our analysis of all the verified true and false rumors that spread on Twitter confirms that false news spreads more pervasively than the truth online. It also overturns conventional wisdom about how false news spreads. Though one might expect network structure and individual characteristics of spreaders to favor and promote false news, the opposite is true. The greater likelihood of people to retweet falsity more than the truth is what drives the spread of false news, despite network and individual factors that favor the truth. Furthermore, although recent testimony before congressional committees on misinformation in the United States has focused on the role of bots in spreading false news (36), we conclude that human behavior contributes more to the differential spread of falsity and truth than automated robots do. This implies that misinformation-containment policies should also emphasize behavioral interventions, like labeling and incentives to dissuade the spread of misinformation, rather than focusing exclusively on curtailing bots. Understanding how false news spreads is the first step toward containing it. We hope our work inspires more large-scale research into the causes and consequences of the spread of false news as well as its potential cures.

REFERENCES AND NOTES

1. L. J. Savage, *J. Am. Stat. Assoc.* **46**, 55–67 (1951).
2. H. A. Simon, *The New Science of Management Decision* (Harper & Brothers Publishers, New York, 1960).
3. R. Wedgwood, *Noûs* **36**, 267–297 (2002).
4. E. Fehr, U. Fischbacher, *Nature* **425**, 785–791 (2003).
5. C. E. Shannon, *Bell Syst. Tech. J.* **27**, 379–423 (1948).
6. S. Bikhchandani, D. Hirshleifer, I. Welch, *J. Polit. Econ.* **100**, 992–1026 (1992).
7. K. Rapoza, "Can 'fake news' impact the stock market?" *Forbes*, 26 February 2017; www.forbes.com/sites/kenrapoza/2017/02/26/can-fake-news-impact-the-stock-market/.
8. M. Mendoza, B. Poblete, C. Castillo, in *Proceedings of the First Workshop on Social Media Analytics* (Association for Computing Machinery, ACM, 2010), pp. 71–79.
9. A. Gupta, H. Lamba, P. Kumaraguru, A. Joshi, in *Proceedings of the 22nd International Conference on World Wide Web* (ACM, 2010), pp. 729–736.

10. K. Starbird, J. Maddock, M. Orand, P. Achterman, R. M. Mason, in *iConference 2014 Proceedings* (iSchools, 2014).
11. J. Gottfried, E. Shearer, "News use across social media platforms," Pew Research Center, 26 May 2016; www.journalism.org/2016/05/26/news-use-across-social-media-platforms-2016/.
12. C. Silverman, "This analysis shows how viral fake election news stories outperformed real news on Facebook," *BuzzFeed News*, 16 November 2016; www.buzzfeed.com/craigsilverman/viral-fake-election-news-outperformed-real-news-on-facebook/.
13. M. De Domenico, A. Lima, P. Mougel, M. Musolesi, *Sci. Rep.* **3**, 2980 (2013).
14. O. Oh, K. H. Kwon, H. R. Rao, in *Proceedings of the International Conference on Information Systems* (International Conference on Information Systems, ICIIS, paper 231, 2010).
15. M. Tambusco, G. Ruffo, A. Flammini, F. Menczer, in *Proceedings of the 24th International Conference on World Wide Web* (ACM, 2015), pp. 977–982.
16. Z. Zhao, P. Resnick, Q. Mei, in *Proceedings of the 24th International Conference on World Wide Web* (ACM, 2015), pp. 1395–1405.
17. M. Gupta, P. Zhao, J. Han, in *Proceedings of the 2012 Society for Industrial and Applied Mathematics International Conference on Data Mining* (Society for Industrial and Applied Mathematics, SIAM, 2012), pp. 153–164.
18. G. L. Ciampaglia et al., *PLOS ONE* **10**, e0128193 (2015).
19. A. Frigeri, L. A. Adamic, D. Eckles, J. Cheng, in *Proceedings of the International Conference on Weblogs and Social Media* (Association for the Advancement of Artificial Intelligence, AAAI, 2014).
20. M. Del Vicario et al., *Proc. Natl. Acad. Sci. U.S.A.* **113**, 554–559 (2016).
21. A. Bessi et al., *PLOS ONE* **10**, e0118093 (2015).
22. Frigeri et al. (19) do evaluate two metrics of diffusion: depth, which shows little difference between true and false rumors, and shares per rumor, which is higher for true rumors than it is for false rumors. Although these results are important, they are not definitive owing to the smaller sample size of the study; the early timing of the sample, which misses the rise of false news after 2013; and the fact that more shares per rumor do not necessarily equate to deeper, broader, or more rapid diffusion.
23. S. Goel, A. Anderson, J. Hofman, D. J. Watts, *Manage. Sci.* **62**, 180–196 (2015).
24. L. Itti, P. Baldi, *Vision Res.* **49**, 1295–1306 (2009).
25. S. Aral, M. Van Alstyne, *Am. J. Sociol.* **117**, 90–171 (2011).
26. J. Berger, K. L. Milkman, *J. Mark. Res.* **49**, 192–205 (2012).
27. D. M. Blei, A. Y. Ng, M. I. Jordan, *J. Mach. Learn. Res.* **3**, 993–1022 (2003).
28. S. Aral, P. Dhillon, "Unpacking novelty: The anatomy of vision advantages," Working paper, MIT-Sloan School of Management, Cambridge, MA, 22 June 2016; https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2388254.
29. T. M. Cover, J. A. Thomas, *Elements of Information Theory* (Wiley, 2012).
30. T. Kailath, *IEEE Trans. Commun. Technol.* **15**, 52–60 (1967).
31. R. Plutchik, *Am. Sci.* **89**, 344–350 (2001).
32. S. M. Mohammad, P. D. Turney, *Comput. Intell.* **29**, 436–465 (2013).
33. S. M. Mohammad, S. Kiritchenko, *Comput. Intell.* **31**, 301–326 (2015).
34. S. Vosoughi, D. Roy, in *Proceedings of the 10th International AAAI Conference on Weblogs and Social Media* (AAAI, 2016), pp. 707–710.
35. C. A. Davis, O. Varol, E. Ferrara, A. Flammini, F. Menczer, in *Proceedings of the 25th International Conference Companion on World Wide Web* (ACM, 2016), pp. 273–274.
36. For example, this is an argument made in recent testimony by Clint Watts—Robert A. Fox Fellow at the Foreign Policy

Research Institute and Senior Fellow at the Center for Cyber and Homeland Security at George Washington University—given during the U.S. Senate Select Committee on Intelligence hearing on "Disinformation: A Primer in Russian Active Measures and Influence Campaigns" on 30 March 2017; www.intelligence.senate.gov/sites/default/files/documents/os-cwatts-033017.pdf.

ACKNOWLEDGMENTS

We are indebted to Twitter for providing funding and access to the data. We are also grateful to members of the MIT research community for invaluable discussions. The research was approved by the MIT institutional review board. The analysis code is freely available at <https://goo.gl/forms/AK1lZujpepxhNTy33>. The entire data set is also available, from the same link, upon signing an access agreement stating that (i) you shall only use the data set for the purpose of validating the results of the MIT study and for no other purpose; (ii) you shall not attempt to identify, reidentify, or otherwise deanonymize the data set; and (iii) you shall not further share, distribute, publish, or otherwise disseminate the data set. Those who wish to use the data for any other purposes can contact and make a separate agreement with Twitter.

SUPPLEMENTARY MATERIALS

www.sciencemag.org/content/359/6380/1146/suppl/DC1
Materials and Methods
Figs. S1 to S20
Tables S1 to S39
References (37–75)

14 September 2017; accepted 19 January 2018
10.1126/science.aap9559

外 文 译 文

真假新闻的在线传播

Soroush Vosoughi, Deb Roy, Sinan Aral

麻省理工学院

第一章 概述

1.1 概述

决策、合作、通信和市场领域的基础理论全都将对真实或准确度的概念化作为几乎一切人类努力的核心。然而，不论是真实信息还是虚假信息都会于在线媒体上迅速传播。定义什么是真、什么是假成了一种常见的政治策略，而不是基于一些各方同意的事实的争论。我们的经济也难免遭受虚假信息传播的影响。虚假流言会影响股价和大规模投资的动向，例如，在一条声称巴拉克·奥巴马在爆炸中受伤的推文发布后，股市市值蒸发了 1300 亿美元。的确，从自然灾害到恐怖袭击，我们对一切事情的反应都受到了扰乱。

新的社交网络技术在使信息的传播速度变快和规模变大的同时，也便利了不实信息（即不准确或有误导性的信息）的传播。然而，尽管我们对信息和新闻的获取越来越多地收到这些新技术的引导，但我们仍然对他们在虚假信息传播上的作用知之甚少。尽管媒体对假新闻传播的轶事分析给予了相当多的关注，但仍然几乎没有针对不实信息扩散或其发布源头的大规模实证调查。目前，虚假信息传播的研究仅仅局限于小的、局部的样本的分析上，而这些分析忽略了两个最重要的科学问题：真实信息和虚假信息的传播有什么不同？哪些人类判断中的因素可以解释这些不同？

$$\max_W \text{tr}(\mathbf{W}^\top \mathbf{X} \mathbf{X}^\top \mathbf{W}) \quad \text{式 (外 1-1)}$$

的字。我只是为了把第二章挤到下一页而凑的字。我只是为了把第二章挤到下一页而凑的字。

第二章 我也不知道是什么

新的社交网络技术在使信息的传播速度变快和规模变大的同时，也便利了不实信息（即不准确或有误导性的信息）的传播。然而，尽管我们对信息和新闻的获取越来越多地收到这些新技术的引导，但我们仍然对他们在虚假信息传播上的作用知之甚少。尽管媒体对假新闻传播的轶事分析给予了相当多的关注，但仍然几乎没有针对不实信息扩散或其发布源头的大规模实证调查。目前，虚假信息传播的研究仅仅局限于小的、局部的样本的分析上，而这些分析忽略了两个最重要的科学问题：真实信息和虚假信息的传播有什么不同？哪些人类判断中的因素可以解释这些不同？

新的社交网络技术在使信息的传播速度变快和规模变大的同时，也便利了不实信息（即不准确或有误导性的信息）的传播。然而，尽管我们对信息和新闻的获取越来越多地收到这些新技术的引导，但我们仍然对他们在虚假信息传播上的作用知之甚少。尽管媒体对假新闻传播的轶事分析给予了相当多的关注，但仍然几乎没有针对不实信息扩散或其发布源头的大规模实证调查。目前，虚假信息传播的研究仅仅局限于小的、局部的样本的分析上，而这些分析忽略了两个最重要的科学问题：真实信息和虚假信息的传播有什么不同？哪些人类判断中的因素可以解释这些不同？

新的社交网络技术在使信息的传播速度变快和规模变大的同时，也便利了不实信息（即不准确或有误导性的信息）的传播。然而，尽管我们对信息和新闻的获取越来越多地收到这些新技术的引导，但我们仍然对他们在虚假信息传播上的作用知之甚少。尽管媒体对假新闻传播的轶事分析给予了相当多的关注，但仍然几乎没有针对不实信息扩散或其发布源头的大规模实证调查。目前，虚假信息传播的研究仅仅局限于小的、局部的样本的分析上，而这些分析忽略了两个最重要的科学问题：真实信息和虚假信息的传播有什么不同？哪些人类判断中的因素可以解释这些不同？

$$\max_{\mathbf{W}} \text{tr}(\mathbf{W}^T \mathbf{X} \mathbf{X}^T \mathbf{W}) \quad \text{式 (外 2-1)}$$

新的社交网络技术在使信息的传播速度变快和规模变大的同时，也便利了不实信息（即不准确或有误导性的信息）的传播。然而，尽管我们对信息和新闻的获取越来越多地收到这些新技术的引导，但我们仍然对他们在虚假信息传播上的作用知之甚少。尽管媒体对假新闻传播的轶事分析给予了相当多的关注，但仍然几乎没有针对不实信息扩散或其发布源头的大规模实证调查。目前，虚假信息传播的研究仅仅局限于小的、局部的样本的分析上，而这些分析忽略了两个最重要的科学问题：真实信息和虚假信息的传播有什么不同？哪些人类判断中的因素可以解释这些不同？

新的社交网络技术在使信息的传播速度变快和规模变大的同时，也便利了不实信息（即不准确或有误导性的信息）的传播。然而，尽管我们对信息和新闻的获取越来越多地收到这些新技术的引导，但我们仍然对他们在虚假信息传播上的作用知之甚少。尽管媒体对假新闻传播的轶事分析给予了相当多的关注，但仍然几乎没有针对不实信息扩散或其发布源头的大规模实证调查。目前，虚假信息传播的研究仅仅局限于小的、局部的

样本的分析上，而这些分析忽略了两个最重要的科学问题：真实信息和虚假信息的传播有什么不同？哪些人类判断中的因素可以解释这些不同？

新的社交网络技术在使信息的传播速度变快和规模变大的同时，也便利了不实信息（即不准确或有误导性的信息）的传播。然而，尽管我们对信息和新闻的获取越来越多地收到这些新技术的引导，但我们仍然对他们在虚假信息传播上的作用知之甚少。尽管媒体对假新闻传播的轶事分析给予了相当多的关注，但仍然几乎没有针对不实信息扩散或其发布源头的大规模实证调查。目前，虚假信息传播的研究仅仅局限于小的、局部的样本的分析上，而这些分析忽略了两个最重要的科学问题：真实信息和虚假信息的传播有什么不同？哪些人类判断中的因素可以解释这些不同？

北京邮电大学

本科毕业设计（论文）开题报告

学院	计算机学院	专业	计算机科学与技术	班级	2018211303
学生姓名	马嘉骥	学号	2018211149	班内序号	17
指导教师姓名	方维	所在单位	北邮计算机学院	职称	副教授
设计（论文）题目					
(中文) 基于漏洞知识图谱的可视化系统的设计与实现					
(英文) Design and Implementation of Visualization System Based on Vulnerability Knowledge Graph					

一、选题的背景和意义

1.1 选题背景

近年来，随着互联网产业迅速发展，互联网安全漏洞问题显著性也急剧增加。根据公共漏洞和暴露(Common Vulnerabilities and Exposures, CVE) 等权威漏洞数据档案的数据，自 1999 年安全漏洞首次披露以来，互联网安全漏洞数量呈现增长趋势，2019 年全年，新增漏洞两万余个；2020 年全年粗略估计，新增漏洞三万五千余个。对攻击者而言，不仅漏洞攻击的学习成本和难度下降，其可以利用的漏洞数量也明显增多；对企业与开发者而言，随着开源化逐渐成为一种趋势，各类互联网产品对开源系统与组件的依赖性逐步升高，正如近期 Java Log4j 日志组件漏洞造成大规模安全隐患，互联网产业蓬勃发展的同时也面临与日俱增的安全挑战。

1.2 项目意义

本课题针对上述问题，提出一种漏洞知识图谱可视化系统。基于知识图谱、图数据库等技术，对互联网漏洞数据包括受影响产品、可利用代码及补丁等信息进行收集与分析，形成知识结构；对多个数据源抽取的知识进行融合、抽取漏洞的实体及关系、建立漏洞的图数据库、形成漏洞知识图谱；基于漏洞知识图谱搭建基于 B/S 架构的可视化系统，提供易于使用的交互接口进行展示、知识筛选等操作。

本系统采用自动化的方式，实现对漏洞信息的持续收集与整理，极大节省了人力资源的消耗。结合抽取关键信息，对漏洞间关联性进行分析、构建漏洞知识图谱，将离散的漏洞信息转化为相互联系的图结构，为开发者提供项目依赖安全性参考、为互联网安全研究人员提供数据与服

务支撑，促进构建更高效安全的互联网环境。

二、研究的基本内容和拟解决的主要问题

2.1 研究的基本内容

- 1、调研互联网安全漏洞数据信息的来源与收集渠道
- 2、调研 Scrapy、Selenium、Puppeteer 等数据获取方案
- 3、调研知识图谱的思想、基本原理、构建技术
- 4、调研 Neo4j 等图数据库存储方案
- 5、调研针对图数据的 Web 技术方案，进行后端 Django/Flask/Springboot 框架技术选型、
进行 Vue/React/Angular 等前端框架技术选型
- 6、设计实现基于数据爬取、规则或机器学习方法的漏洞知识图谱构建系统
- 7、设计实现基于图数据的漏洞知识图谱存储系统
- 8、设计实现基于漏洞知识图谱的可视化系统
- 9、对上述系统进行系统测试、排错、功能扩展、优化，编写文档记录

2.2 拟解决的主要问题

本课题系统包含数据采集、数据分析、图谱构建、数据库存储、前后端搭建等内容，经前期调研，提出主要问题如下：

1、安全漏洞信息数据来源：数据采集是整个系统的源头。如何获得可靠、准确、高质量的漏洞信息，对数据分析过程实施、最终系统可用性具有决定性作用。本课题需调研互联网安全漏洞数据源，并提出具备可行性的信息采集方案。

2、基于多源异构漏洞数据的安全漏洞信息抽取：互联网相关漏洞、威胁情报、漏洞利用代码等信息分布在各大平台，杂乱且不够全面。本课题需提出一种采用多源异构数据融合的方案，从原始语料提取实体、关系、属性等知识要素，再经知识融合、消除歧义，得到一系列基本的事实表达，实现漏洞信息的完整收集与格式化。

3、漏洞知识图谱的构建方法：知识图谱是本项目核心所在。根据采集整理的格式化漏洞信息，进行本体构建、知识推理、质量评估的知识加工过程，最终获得结构化、网络化的知识体系。同时，还应考虑知识图谱的迭代更新过程实现。

4、漏洞知识图谱的可视化平台设计与实现：构建知识图谱、实现基于图数据的漏洞知识图

谱存储后，本系统应具有易于访问与交互的界面供用户使用。本课题需调研通过基于 B/S 架构的 WebApp 与前述图数据库的交互式访问与可视化图结构呈现方案，实现漏洞知识图谱的可视化系统。

三、研究方法及措施

1、安全漏洞信息数据来源

目前国际上有公共漏洞和暴露(Common Vulnerabilities and Exposures, CVE) 数据库、中国国家信息安全漏洞共享平台(China National Vulnerability Database, CNVD) 、中国国家信息安全漏洞库(China National Vulnerability Database of Information Security, CNNVD) 、美国国家漏洞库(National Vulnerability Database, NVD) 、中国关键基础设施安全应急响应中心、美国工业控制系统网络应急响应小组等。以上公开漏洞信息发布来源都是信息安全领域内最具权威性的平台。

本课题计划以上述公开漏洞信息源作为漏洞知识的主要数据来源。原始数据收集拟由本基于漏洞知识图谱的可视化系统的数据采集模块完成，拟采用定时增量方式从多个数据源进行信息收集。①对于漏洞相关数据，拟从上述漏洞信息平台、CPE（公共平台枚举）、CWE（常见缺陷列表）等数据源收集；②对于 POC/EXP 漏洞验证/利用代码，拟从 packet storm、exploit-db、github 上进行收集。

拟采用 Scrapy、Selenium 或 Puppeteer 框架进行漏洞数据爬取。例如对于提供 CVE 数据下载的 <https://www.cve.org/Downloads>，使用爬虫框架或 wget 命令直接访问该页面 <https://cve.mitre.org/data/downloads/allitems.csv> 链接进行数据下载，再使用 diff 等命令进行差异比对从而获取漏洞增量数据。对于需要从网页内容获取信息的数据源，使用 Scrapy 配合 XPath 获取页面信息，或采用 Selenium、Puppeteer 框架对爬取网页的 DOM 树进行操作获取需要的信息。根据数据更新频度需求，可使用 crontab 命令定时进行数据增量爬取。

2、基于多源异构漏洞数据的安全漏洞信息抽取

当前大多数的漏洞信息平台中涵盖了各种类别的漏洞信息，尽管有些漏洞信息平台会有行业分类，但其中的漏洞信息不够全面，存在交叉，不利于安全研究人员对行业内安全漏洞及相关威胁情报信息的获取和分析，亦不利于开发者增强漏洞防范本领。

信息抽取是知识图谱构建的第一步，其中关键问题是从业务数据源中自动抽取信息得到候选只是但愿。信息抽取是一种自动化地从半结构化和无结构数据中抽取实体、关系、实体属性等结

构化信息的技术，包括实体抽取、关系抽取、属性抽取等。本课题拟基于从多个数据源采集的数据，进行去重、融合、关键信息抽取，整理漏洞标题、对策、CVSS、CWE、CPE 等信息，从而实现漏洞信息的格式化。

去重：对于多源异构的数据，大多数的漏洞都有 CVE-ID，保证了其唯一性，对于没有 CVE-ID 的漏洞数据，可以通过相关链接，受影响平台及 CVSS 等信息结合人工分析比对现有的漏洞数据，判断是否为重复数据。

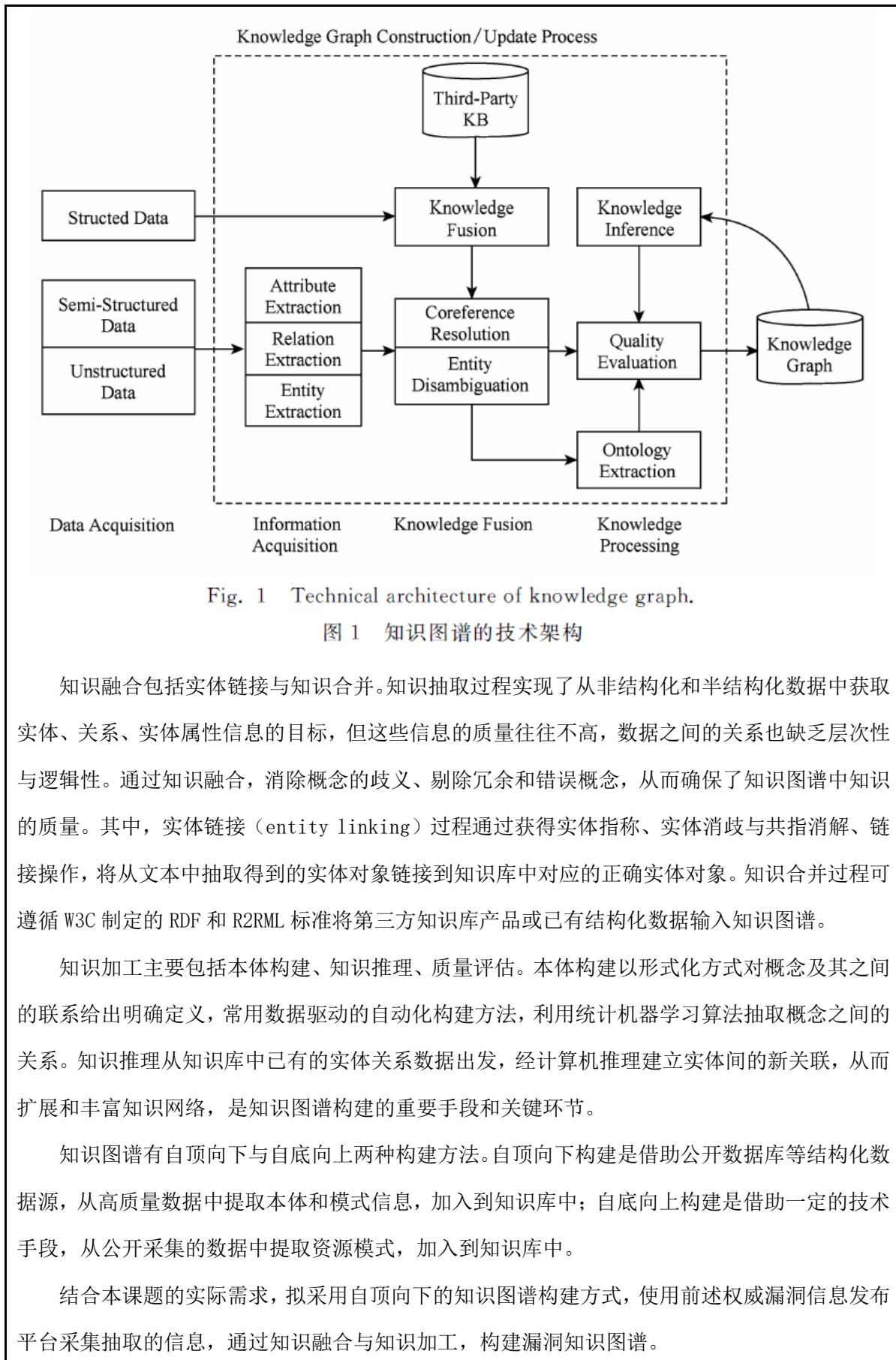
融合：结合各大漏洞平台的特点，针对漏洞的不同属性，参照其规范程度从不同源头选取，例如：漏洞标题及对策从 JVNDB 中选取，对于一些标准信息如 CVSS、CWE-ID 等从 NVD 中选取，相关链接则直接合并并去除重复链接。同时，利用 CVSS 及 CWE 的详细信息进一步扩充漏洞本体。对于威胁情报这种非结构化数据，通过 CVE-ID 和受影响软件和相关链接信息关联相应的漏洞。

关键信息抽取：通过规则匹配的方式从已有的漏洞数据中获取训练数据，针对漏洞描述信息，可以进行翻译、去除非文本、词形还原、转化为小写和删除停用词等数据清洗后训练 NER（命名实体识别）模型。使用训练好的模型抽取信息后，为保证漏洞信息的准确性和完整性，系统会对使用模型扩充的漏洞信息进行标注，随后进行人工审核和纠正。

3、漏洞知识图谱的构建方法

知识图谱的定义：知识图谱是结构化的语义知识库，用于以符号形式描述物理世界中的概念及其相互关系。它本身是一个具有属性的实体通过关系链接而成的网状知识库，知识图谱将互联网中积累的信息组织起来，成为可以被利用的知识。知识图谱的应用价值在于它能①通过推理实现概念检索、②以图形化方式向用户展示经过分类整理的结构化知识，从而使人从人工筛选过滤网页寻找答案的模式中解放出来。

知识图谱的构建过程如图所示，从原始数据出发，采用自动或半自动手段，从原始数据中提取知识要素，将其存入知识库的数据层和模式层，不断迭代更新。每轮迭代包含三个阶段：信息抽取、知识融合、知识加工。



4、漏洞知识图谱的存储与可视化平台设计与实现

构建知识图谱的工作实现了安全漏洞知识图谱中节点和关系的构建，形成了逻辑上的知识图谱。还需将信息进行存储以实现对知识图谱的实际利用。拟选用 Neo4j 图数据库存储该图谱。Neo4j 数据库具有高效、成熟、稳定、接口丰富、扩展型强等优点。若使用 Django/Flask 作为后端框架可使用 py2neo 库、若使用 SpringBoot 作为后端框架可使用 Neo4j Java Driver 进行数据库的驱动。此外 Neo4j 也支持使用 Neo4j-import 一次性批量导入 csv 数据文件。

设计并实现安全漏洞情报平台，能够将全面的漏洞情报信息及时响应给安全研究人员，并提供多种功能帮助安全研究人员做分析。本课题拟搭建一套基于 B/S 架构的漏洞知识图谱可视化系统，用户通过浏览器访问 Web 应用，通过 RESTful API 与后端进行通信，从图数据库中获取查询结果，并由 Web 应用以可视化的方式展现。

知识图谱的可视化系统首先需要建立多维度检索功能以确定展示范围。使用图结构，通过相似度度量与漏洞体量评估的方式，来展示漏洞之间的关联；展示信息应包含漏洞标题、漏洞发现日期、漏洞 CVE-ID、CWE-ID、CVSS 与 CWE 详细信息、CPE、对策等。此外，拟实现可交互的可视化展示效果，以便用户更清晰地梳理漏洞之间逻辑关联、查看漏洞详细信息。漏洞情报平台可以实现对漏洞和相关情报的及时告警，通过邮件等方式将新增漏洞信息发送给用户。

本系统实现难点在于对于大量漏洞关系数据的高效检索与展示。进一步调研将确定后端 Django/Flask/SpringBoot 框架技术选型、前端 Vue/React/Angular 等框架技术选型，以及 D3.js 等可视化方案的技术选型。

四、研究工作的步骤与进度

2021/12/12 – 2021/12/24（两周） 明确任务，了解课题背景，制定计划，查找相关论文资料，对课题的研究方法形成大体框架并提交开题报告。

2021/12/27 – 2022/01/14（三周） 学习基本 Web 知识和 Scrapy 爬虫框架的使用，爬取公开漏洞、CPE、CWE、POC 等数据。

2022/03/01 – 2022/03/13（两周） 学习 SpringBoot 和 Vue 框架，分析漏洞知识间的关系；分析系统功能，完成系统的需求分析。

2022/03/14 – 2022/03/26（两周） 使用基于规则或机器学习的方法抽取漏洞实体及关系，构建漏洞数据图模型，建立图数据库。完成系统的概要设计。

2022/03/27 – 2022/04/17（三周） 完成系统详细设计以及部分代码实现，完成系统前端漏洞知识图的可视化、知识筛选等基本功能。接受中期检查。

2022/04/19 – 2022/05/02（两周） 完善前端功能，完成前后端全部代码。

2022/05/03 – 2022/05/16（两周） 进行系统测试、排错，及功能扩展、优化，编写文档记录。

2022/05/17 – 2022/05/30（两周） 撰写毕业论文，完成毕业设计。

主要参考文献：

[1] 张吉祥,张祥森,武长旭,赵增顺.知识图谱构建技术综述[J/OL].计算机工程:1-16[2021-11-06].<https://doi.org/10.19678/j.issn.1000-3428.0061803>.

[2] 陶耀东,贾新桐,吴云坤.一种基于知识图谱的工业互联网安全漏洞研究方法[J].信息技术与网络安全,2020,39(01):6-13+18.

[3] Han Z, Li X, Liu H, et al. DeepWeak: Reasoning common software weaknesses via knowledge graph embedding[C]// 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2018.

[4] Yan Jia, Yulu Qi, Huaijun Shang, Rong Jiang, Aiping Li. A Practical Approach to Constructing a Knowledge Graph for Cybersecurity[J].Engineering,2018,4(1):53-60.

[5] <https://d3js.org/>

[6] <https://neo4j.com/docs/>

[7] <https://www.cve.org/>

允许进入论文撰写环节：是 <input checked="" type="checkbox"/> 否 <input type="checkbox"/>		指导教师 签字	
日期	2022 年 3 月 2 日		

注：可根据开题报告的长度加页。

北京邮电大学
本科毕业设计（论文）中期进展情况检查表

学院	计算机学院（国家示范性软件学院）	专业	计算机科学与技术	班级	2018211303
学生姓名	马嘉骥	学号	2018211149	班内序号	17
指导教师姓名	方维	所在单位	计算机学院（国家示范性软件学院）	职称	副教授
设计 (论 文)题 目	(中文) 基于漏洞知识图谱的可视化系统的设计与实现				
	(英文) Design and Implementation of Visualization System Based on Vulnerability Knowledge Graph				

目前已完成任务

一、 任务概述:

1. 项目背景:

近年来，随着互联网产业迅速发展，互联网安全漏洞问题显著性也急剧增加。本课题针对上述问题，提出一种漏洞知识图谱可视化系统。基于知识图谱、图数据库等技术，对互联网漏洞数据包括受影响产品、可利用代码及补丁等信息进行收集与分析，形成知识结构；对多个数据源抽取的知识进行融合、抽取漏洞的实体及关系、建立漏洞的图数据库、形成漏洞知识图谱；基于漏洞知识图谱搭建基于 B/S 架构的可视化系统，提供易于使用的交互接口进行展示、知识筛选等操作。

本系统采用自动化的方式，实现对漏洞信息的持续收集与整理，极大节省了人力资源的消耗。结合抽取关键信息，对漏洞间关联性进行分析、构建漏洞知识图谱，将离散的漏洞信息转化为相互联系的图结构，为开发者提供项目依赖安全性参考、为互联网安全研究人员提供数据与服务支撑，促进构建更高效安全的互联网环境。

2. 开发任务及要求:

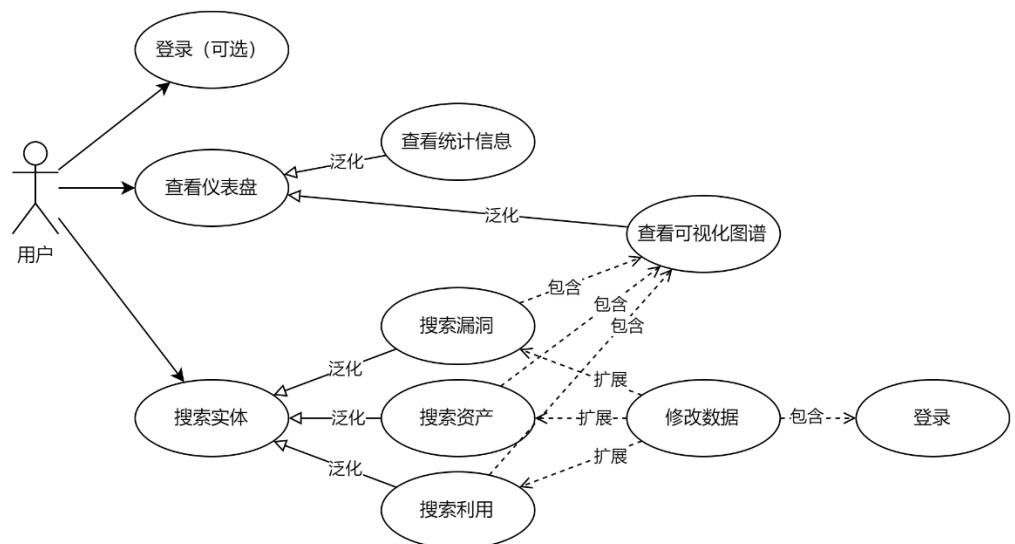
- 1). 设计并实现漏洞数据采集系统；
- 2). 设计并实现基于规则或机器学习方法的漏洞知识图谱构建系统；
- 3). 设计并实现基于图数据的漏洞知识图谱存储系统；
- 4). 设计并实现基于上述漏洞知识图谱的可视化系统；
- 5). 对上述系统进行系统测试、排错、功能扩展、优化，编写文档记录。

二、 使用到的技术和工具:

数据获取	Scrapy、Pandas
图谱构建	Rule-based、Bi-LSTM-CRF
后端	Python-Flask
前端	Vue.js、Vuetify、D3.js
持久化	MongoDB、Neo4j

三、 已经完成的工作:

1. 需求分析:



图：系统用例图

系统输入：来自多种数据源的漏洞相关信息数据。

系统输出：以图形化显示的知识图谱信息。

a) 功能性需求：

爬虫：自动化数据采集及处理。

持久化：结构化与非结构化数据存储。

数据处理：知识图谱生成。

后端：数据库交互、绘图数据生成。

前端：图展示。

b) 非功能性需求：

1> 性能：

系统具备处理大量图数据的能力，并能应对不断增长的数据量。

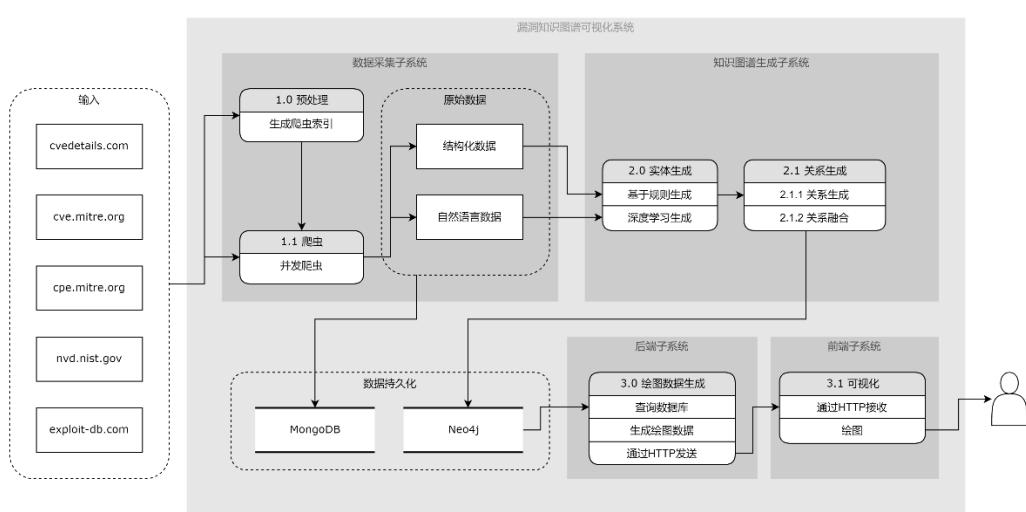
2> 兼容性：

可视化系统应在多种浏览器上运行。

2. 系统概要设计

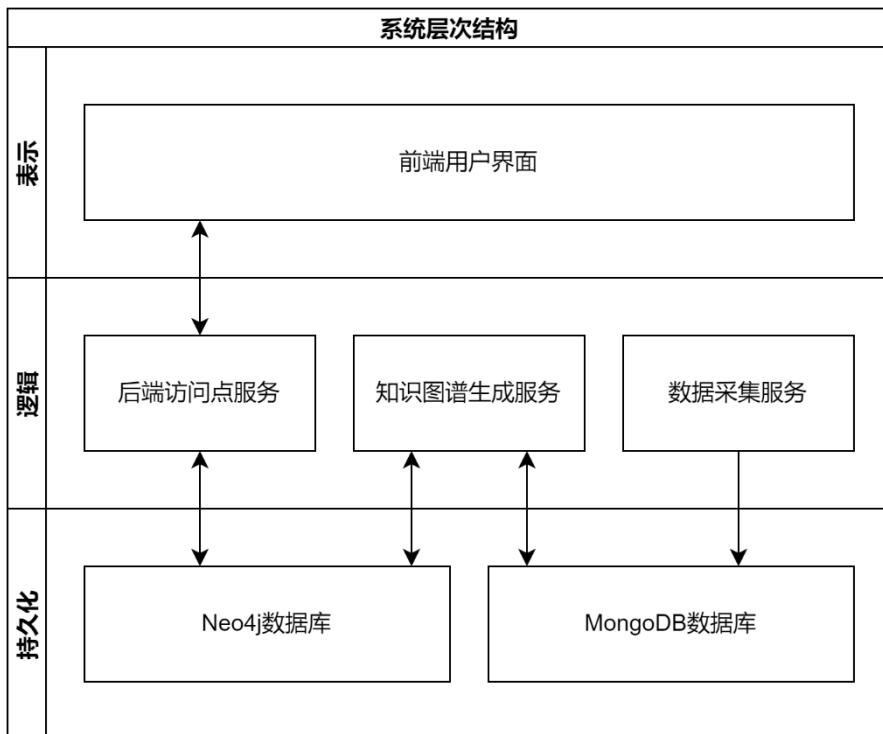
1) 系统总体设计

a) 系统数据流图



图：系统数据流图

b) 系统层次结构设计



图：系统层次结构设计

系统可以分为三层，表示层、服务层和持久化层。表示层包含前端用户界面。持久化层包含 Neo4j 与 MongoDB 两个数据库。逻辑层包含后端服务、知识图谱生成服务、数据采集服务。

数据采集服务通过生成索引、爬虫、数据清洗，向持久化层输入数据，知识图谱生成服务与持久化层进行交互，首先通过基于规则的实体构建与基于神经网络的命名实体识别，再经基于规则的关系生成、关系融合，将采集数据转化为知识图谱。后

端服务与持久化层、表示层进行交互，处理并传递图数据信息及用户控制信息。

c) 系统技术选型：

1> 前端用户界面：Vue.js+Vuetify（界面组件库）+Vuex（数据控制器）+Axios
(异步网络通信) +D3.js (可视化)

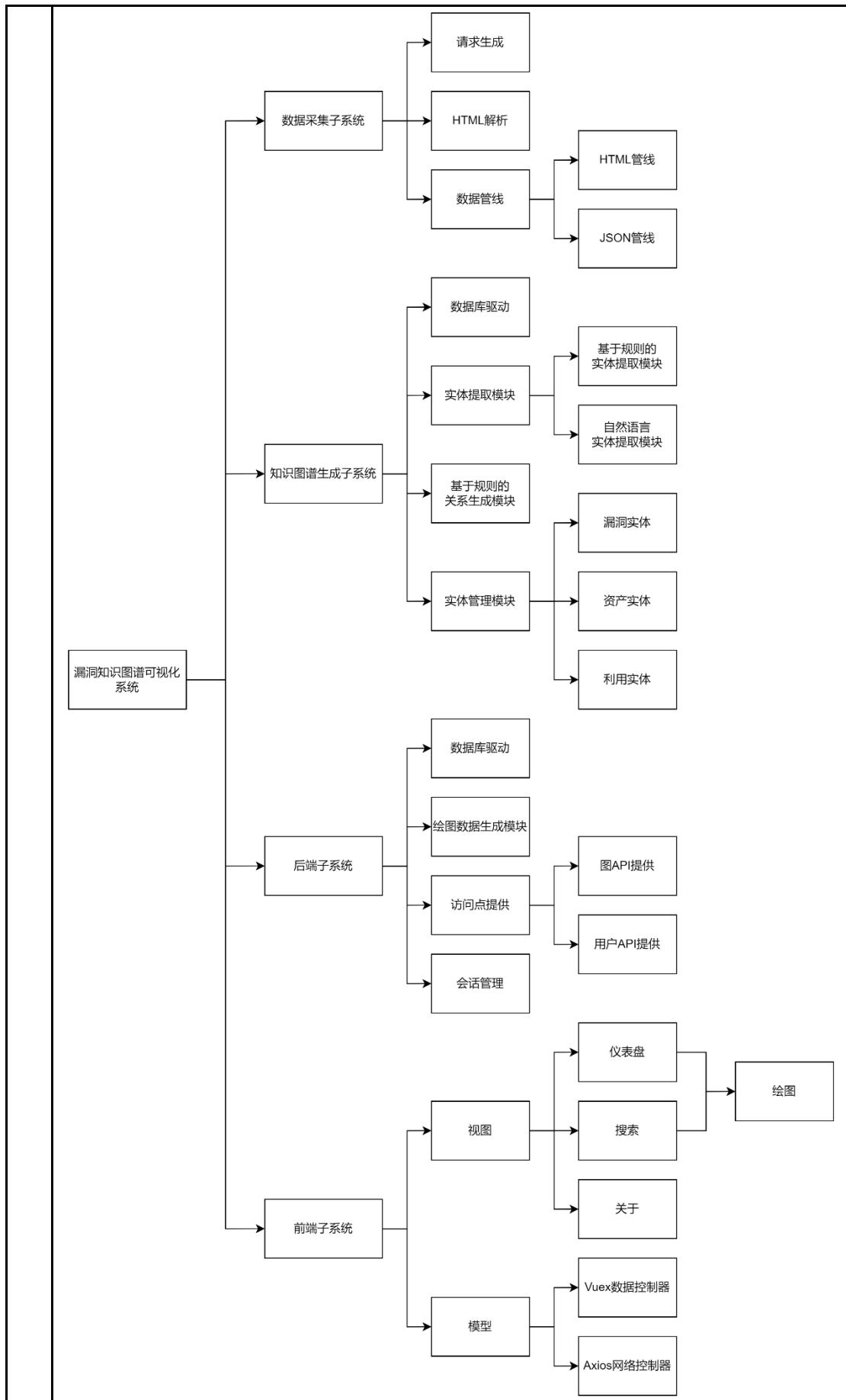
2> 后端 Web 服务：Python-Flask

3> 知识图谱生成服务：基于规则的实体生成与关系生成、Bi-LSTM-CRF

4> 数据采集服务：Scrapy、Pandas

5> 数据库：MongoDB、Neo4j

2) 系统功能模块设计



图：系统功能模块设计

系统可以分为 4 个主要子系统。

数据采集子系统： 主要包含 Web 爬虫服务，包括请求生成器、HTML 解析器、数据管道三个模块，其中数据管道模块包含 HTML 管道和 JSON 管道两个子模块。

请求生成器根据 csv 索引列表生成爬虫请求，HTML 解析器用于从不同页面中解析目标数据，数据管道将解析的数据格式化封装并存入数据库持久化，形成原始数据。

知识图谱生成子系统： 包含数据库驱动、信息提取器、基于规则的关系生成器、实体管理四个模块，其中信息提取器包含基于规则的结构化数据提取器和基于神经网络的自然语言数据提取器，实体管理包括漏洞实体、资产实体、利用实体三种。数据库驱动实例化一个单例数据库对象，并向其他模块暴露对数据库的访问方法。信息提取器用于从原始数据中抽取实体，提取相关属性，实例化实体类。关系生成器用于生成实体对象之间的关系，并将其存入图数据库中。实体管理则管理图数据库中已有的实体和关系，在处理新的数据时防止重复生成实体和关系。

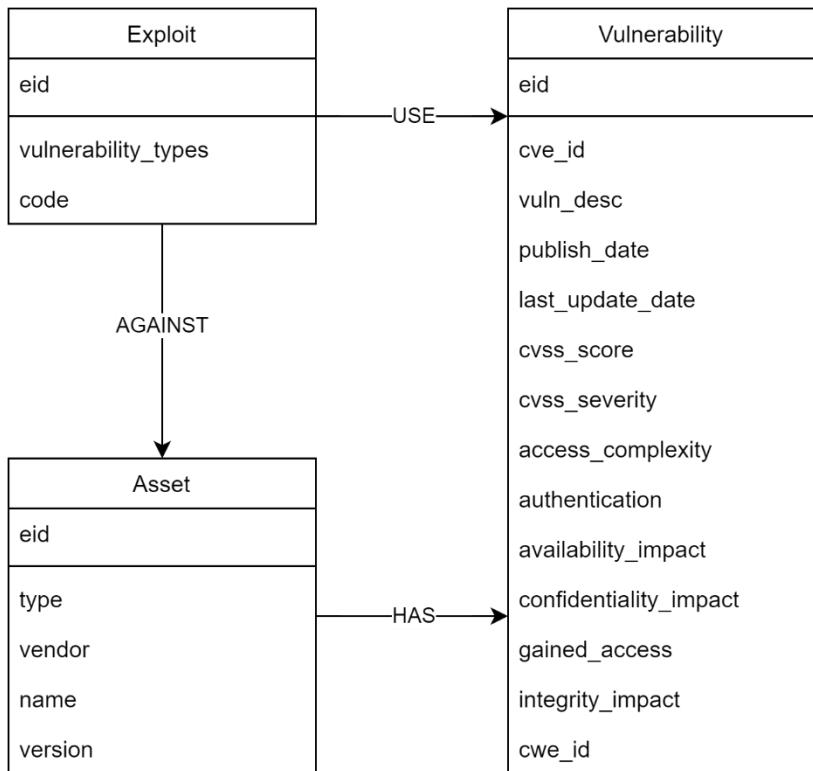
后端子系统： 包含数据库驱动、绘图信息生成器、Web 服务访问点、会话管理四个模块。数据库驱动实例化一个单例数据库对象，并向其他模块暴露对数据库的访问方法，并将其挂载到请求处理的上下文对象中。绘图信息生成器用于将图数据库中存储的信息根据用户请求生成 D3.js 可以处理的格式化节点及关系数据。Web 服务访问点通过 Flask 提供一系列遵循 RESTful 格式的 API，主要包含图处理 API 及用户 API。图处理 API 提供图数据交互，满足可视化展示、搜索、信息修改等功能。用户 API 提供用户管理相关功能。会话管理用于维护前后端间通信的会话，为用户系统提供保证。

前端子系统： 包含视图与模型两个模块。视图模块包含仪表盘、搜索、关于三个子模块，其中仪表盘子模块为用户提供知识图谱信息概览，包含列表数据展现及可视化数据统计。搜索模块支持用户在图谱中搜索感兴趣的内容，并将搜索结果以可视化图的形式展现。此外还提供数据修改等功能。关于子模块向用户展示本系统相关信息及操作指南。模型模块包含 Vuex 数据控制器与 Axios 网络控制器两个子模块。Vuex 数据控制器子模块是前端子系统中数据仓库，为视图模块响应式显示提供数据源，还包含在前端对数据进行操作处理功能。Axios 网络控制器子模块封装 httpClient 对象，提供向后端通信的方法。

3) 数据库设计

cve_item_html	cve_item_json
cve_id	cve_id
content	content cve_id vuln_desc publish_date last_update_date cvss_score cvss_severity access_complexity authentication availability_impact confidentiality_impact gained_access integrity_impact cwe_id affected_products[] reference[]

图: MongoDB 数据模式



图：Neo4j 数据模式

4) 系统接口设计（暂定）：

localhost:5000/api/graph/<op> 图数据接口

localhost:5000/api/user/<op> 用户接口

localhost:5000/<path> 前端路径

3. 系统详细设计

1) 开发环境配置：

操作系统: Windows 11 Pro, Ubuntu 20.04 LTS

数据采集子系统: Python 3.8, scrapy 2.6

图谱生成子系统: Python 3.8, py2neo 2021.2.3, pymongo 3.12, uuid 1.3

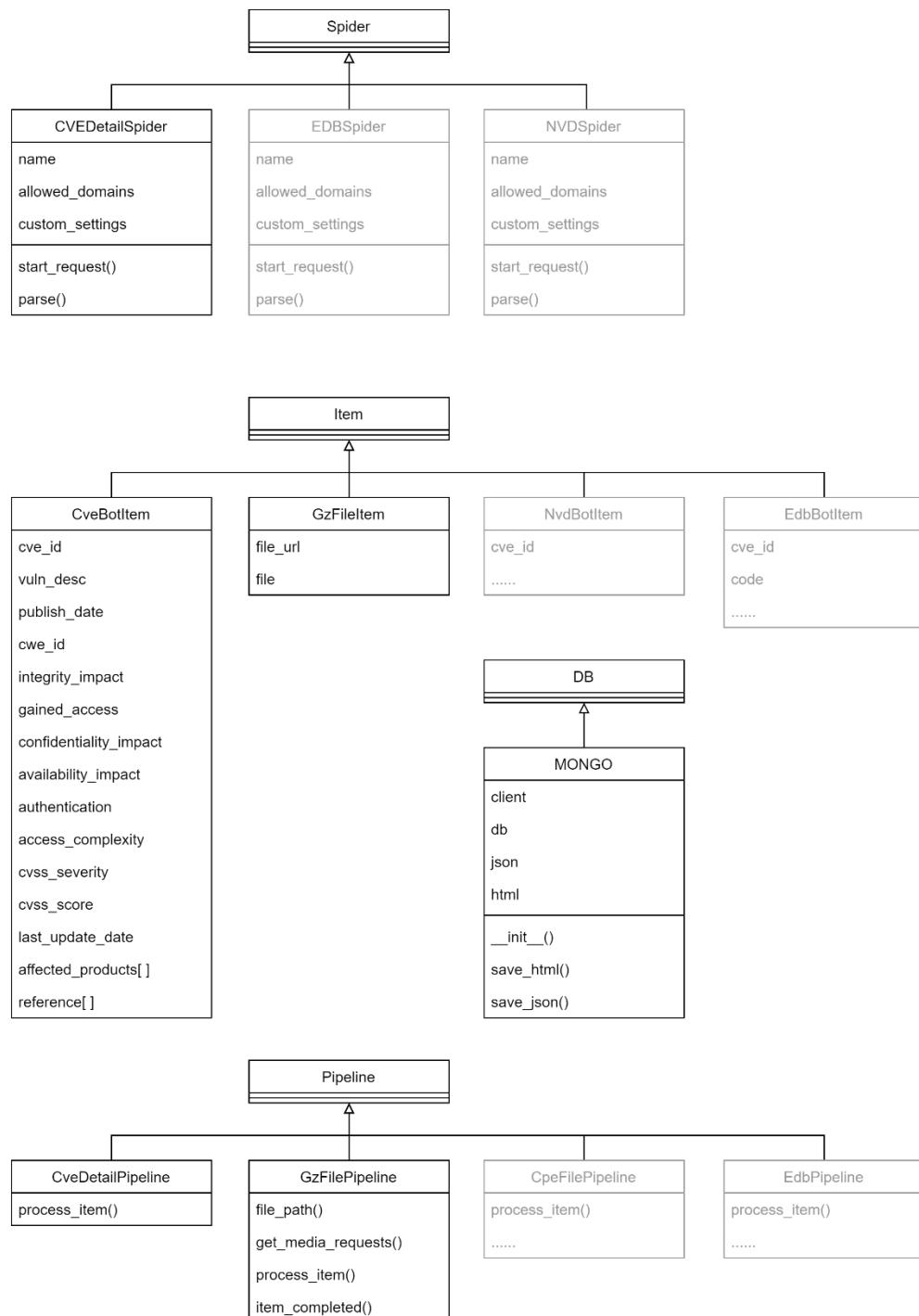
后端子系统: Python 3.8, flask 2.0, flask-cors 3.0, neo4j 4.4

前端子系统: JavaScript ES6, vue 2.6, d3 7.3, vue-axios 3.4, vuetify 2.6, vuex 3.4, force-graph 1.42

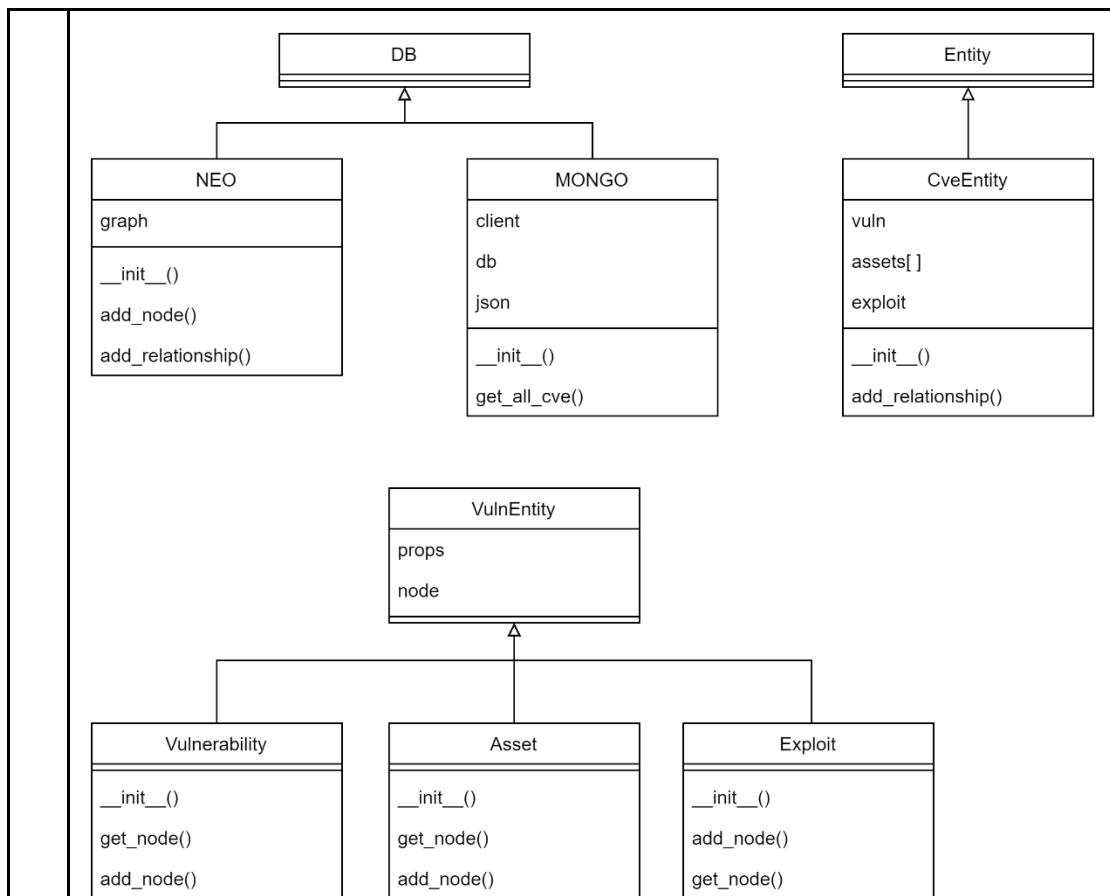
持久化子系统: MongoDB 5.0, Neo4j 4.4

2) 各核心功能模块的详细设计与实现

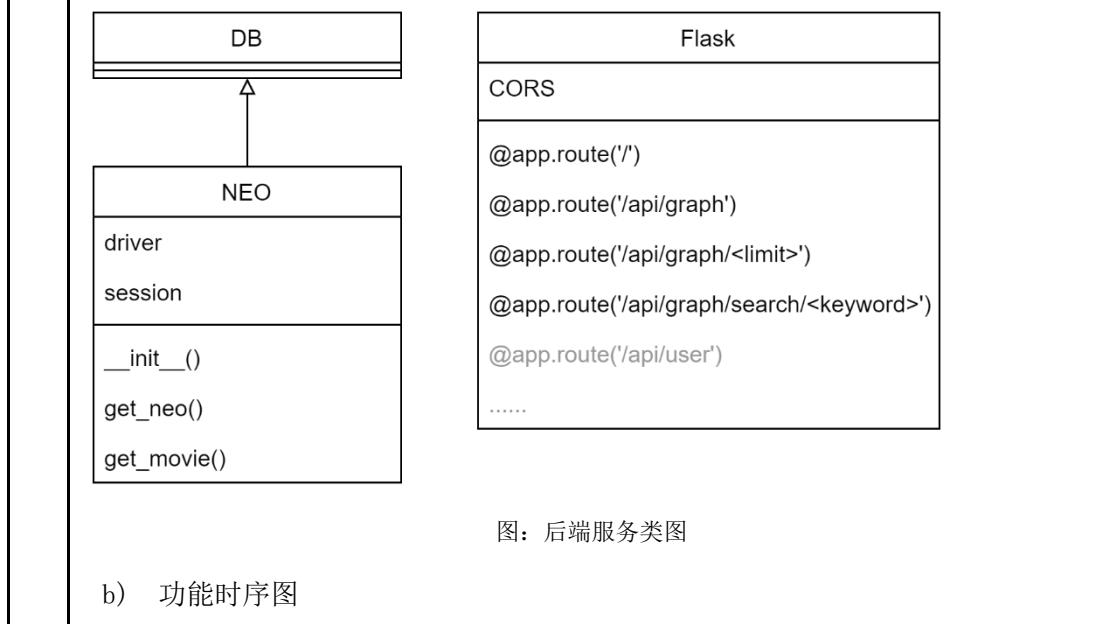
a) 类图



图：数据采集服务类图

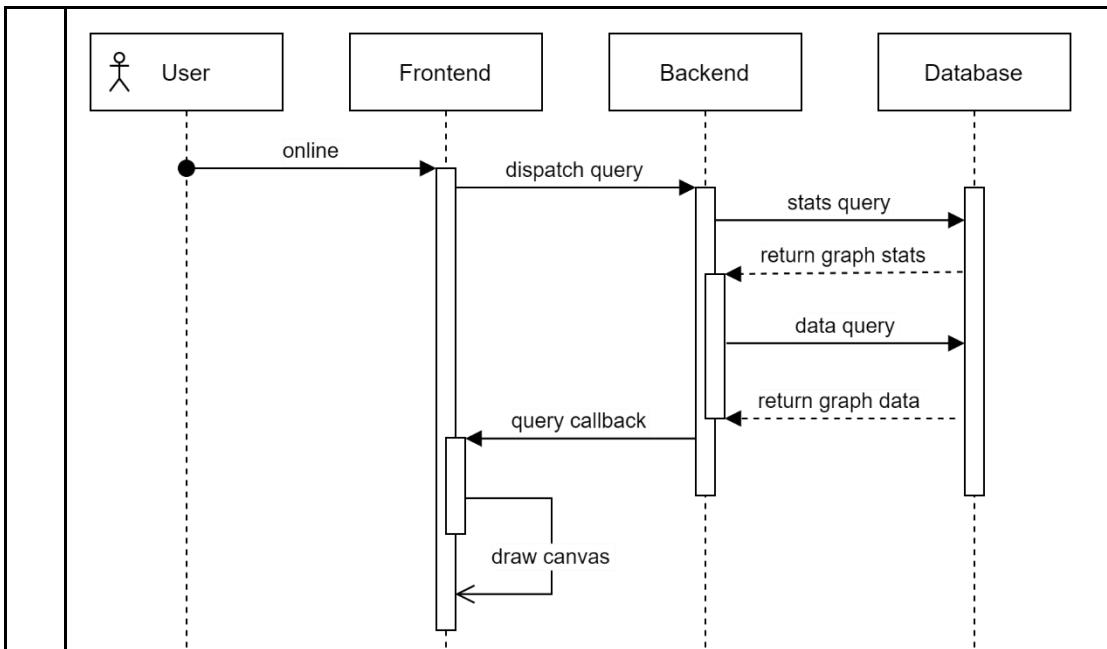


图：知识图谱生成服务类图



图：后端服务类图

b) 功能时序图



图：用户请求时序图

四、 已经完成的工作测试：

1. 数据采集模块获取数据：

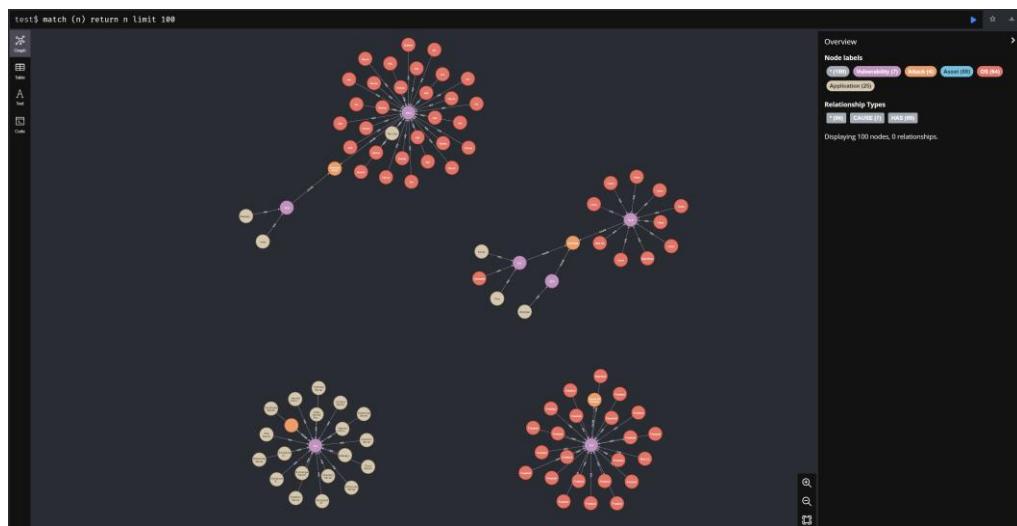
通过 Scrapy 爬虫 cvedetails.com、cpe.mitre.org、cve.mitre.org，获得约 16 万条 cve 信息及其 html 页面，清洗整理获得超 400 万个 cve 属性。

```

> _id: ObjectId("620d45286e07ac4435f26c8f")
  cve_id: "CVE-1999-0001"
  < content: Object
    cve_id: "CVE-1999-0001"
    vuln_desc: "ip_input.c in BSD-derived TCP/IP implementations allows remote attacke..."
    publish_date: "1999-12-30"
    last_update_date: "2010-12-16"
    cvss_score: 5
    cvss_severity: "Medium"
    < confidentiality_impact: Object
      text: "None"
      desc: "There is no impact to the confidentiality of the system."
    < integrity_impact: Object
      text: "None"
      desc: "There is no impact to the integrity of the system."
    < availability_impact: Object
      text: "Partial"
      desc: "There is reduced performance or interruptions in resource availability..."
    < access_complexity: Object
      text: "Low"
      desc: "Specialized access conditions or extenuating circumstances do not exist..."
    < authentication: Object
      text: "Not required"
      desc: "(Authentication is not required to exploit the vulnerability.)"
    gained_access: "None"
    vulnerability_types: "Denial Of Service"
    cwe_id: 20
  > affected_products: Array
  < references: Array
    0: "http://www.osvdb.org/5707"
    1: "https://www.openbsd.org/errata23.html#tcpfix"
  
```

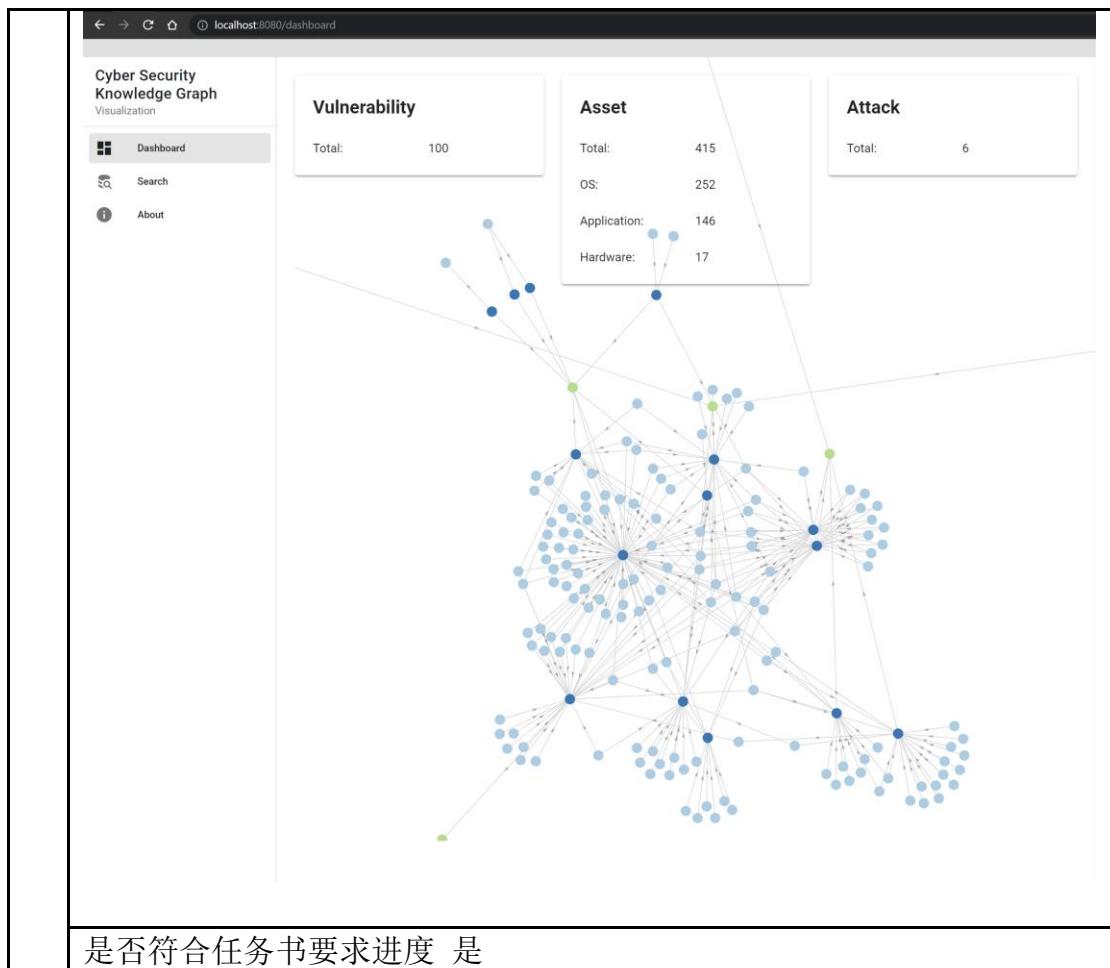
2. 知识图谱生成测试 (Neo4j Browser):

通过 MongoDB 的 cve 信息，使用前 100 条目，编写基于规则的实体与关系生成模块，进行生成漏洞、资产、利用实体测试。



3. 前端可视化展示测试:

使用 Vue.js 与 Flask 搭建前后端，Vuetify 作为前端组件框架，d3.js 作为绘图库，实现前端可视化展示与根据 cve-id 字符串进行简单搜索功能。



尚需完成的任务	<p>数据采集子系统:</p> <ol style="list-style-type: none"> 1. 根据数据库索引信息比对进行增量爬虫 2. 异构数据源采集 <p>知识图谱生成子系统:</p> <ol style="list-style-type: none"> 1. 完善网络安全实体与属性的设计，完善提取规则 2. 实现基于 Bi-LSTM-CRF 模型的深度学习命名实体识别模块 3. 完善实体融合模块的设计与实现 <p>后端系统:</p> <ol style="list-style-type: none"> 1. 实现用户认证、连接管理系统 2. 拓展图数据 API 功能，延伸可视化系统能力 3. Neo4j 数据库访问性能优化 <p>前端系统:</p> <ol style="list-style-type: none"> 1. 完善 UI 设计 2. 完善可视化展示功能、可视化数据修改功能 3. 增加用户系统 	
	能否按期完成设计（论文） 是	
存在 问题	存 在 问 题	<p>1. Flask 连接 Neo4j 驱动系统启动初次查询响应慢。</p> <p>2. 图数据库结点量为百万级，需要性能优化。</p>

和 解 决 办 法	拟 采 取 的 办 法	<p>1. 经查证此为 Neo4j 在 Windows 平台上的 bug，后期服务器部署将采用 Linux 系统。</p> <p>2. 在数据库中使用多种索引加快查询速度；在后端限制单次查询最大结点数量；在前端限制同时展示最大结点数量，为用户提供更细分的查询逻辑，使用 LocalStorage 缓存数据。</p>
指导教师签字		方淮 日期 2022 年 4 月 13 日
检查小组评分及意见		评分：23 (总分：25) 通过中期检查，按反馈意见修改。 组长签字： 方淮 2022年4月15日

注：可根据长度加页。