Computer Science and Application 计算机科学与应用, 2022, 12(6), 1587-1601
Published Online June 2022 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2022.126160

Hans汉斯

分布式资产信息探测与漏洞检测引擎的设计与实现

董鑫达，葛　艳，杜军威，陈　卓

青岛科技大学信息科学技术学院，山东　青岛

收稿日期：2022年5月17日；录用日期：2022年6月16日；发布日期：2022年6月27日

摘　要

随着新一代网络信息技术的迅速发展，信息资产的数量、企业数据的规模呈爆炸性增长，企业内部人员很难全面地了解资产的信息及业务系统当前的安全状况。虽然目前各企业购买大量安全产品，诸如WAF、防火墙等已进入常态化，但是传统的网络安全设备仅能提供网络级别的防护，无法发现和识别攻击者专业、灵活、多样的应用层攻击。为了尽早发现网络中的安全隐患，降低可能产生的影响和损失，漏洞检测技术应运而生。它模拟攻击者的攻击手段，在不影响目标系统正常运行的前提下，检测目标系统可能存在的脆弱性，协助安全管理人员进行安全隐患整改和清除，极大增强了网络环境的安全性。但是常规的漏洞检测引擎只会针对特定的目标进行检测，不会主动发现未知的资产信息，也不会自动扩大检测的覆盖面，并且往往采用单节点部署，扫描效率低下，不能满足及时发现安全风险的要求。本文针对漏洞产生的原理和检测方法进行研究，设计并实现一种综合资产信息探测技术、网络爬虫技术、漏洞检测技术和分布式技术的漏洞检测引擎。通过资产信息探测为漏洞检测提供更全面的信息支持，提高漏洞检测的准确率；通过宽度优先遍历策略和布隆过滤器算法提高爬取目标交互点的精准度；通过分布式引擎架构提高漏洞检测的速率和稳定性。为了有效验证该引擎漏洞检测的准确率和速率，搭建了测试环境；同时在保障基础配置资源环境一致的前提下，选取了多款漏洞扫描器进行对比测试。测试结果表明，该引擎在扫描速率和准确率上占据优势。

关键词

资产信息探测，漏洞检测，网络爬虫，宽度优先遍历，布隆过滤器算法，分布式技术

Design and Implementation of Distributed Asset Information Detection and Vulnerability Detection Engine

Xinda Dong, Yan Ge, Junwei Du, Zhuo Chen

College of Information Science and Technology, Qingdao University of Science and Technology, Qingdao

文章引用: 董鑫达, 葛艳, 杜军威, 陈卓. 分布式资产信息探测与漏洞检测引擎的设计与实现[J]. 计算机科学与应用, 2022, 12(6): 1587-1601. DOI: 10.12677/csa.2022.126160

Shandong

Received: May 17th, 2022; accepted: Jun. 16th, 2022; published: Jun. 27th, 2022

Abstract

With the rapid development of the new generation of network information technology, the number of information assets and the scale of enterprise data are explosive growth. It is difficult for enterprise internal personnel to comprehensively understand the information of assets and the current security status of the business system. At present, enterprises purchase a large number of security products such as WAF and firewall. However, traditional network security devices can only provide network-level protection and cannot detect and identify professional, flexible, and diversified application-layer attacks. In order to find the potential security risks in the network as soon as possible and reduce the possible impact and loss, vulnerability detection technology arises at the historical moment. It simulates the attacker's attack means, detects the possible vulnerability of the target system without affecting the normal operation of the target system, and assists security managers in rectification and removal of security risks, greatly enhancing the security of the network environment. However, conventional vulnerability detection engines only detect specific targets, do not actively discover unknown asset information, and do not automatically expand the coverage of detection. In addition, single-node deployment is often adopted, and scanning efficiency is low, which cannot meet the requirements of timely detection of security risks. This paper studies the principle and detection method of vulnerability generation, designs and implements a vulnerability detection engine that integrates asset information detection technology, web crawler technology, vulnerability detection technology and distributed technology. Asset information detection can provide more comprehensive information support for vulnerability detection and improve the accuracy of vulnerability detection. The width first traversal strategy and Bloom filter algorithm were used to improve the accuracy of target interaction points. Improve the rate and stability of vulnerability detection through distributed engine architecture. In order to verify the accuracy and speed of vulnerability detection, a testing environment was built. At the same time, on the premise of ensuring the consistency of basic configuration resources and environment, several vulnerability scanners are selected for comparative testing. The test results show that the engine has advantages in scanning speed and accuracy.

Keywords

Asset Information Detection, Vulnerability Detection, Web Crawler, Width-First Traversal, Bloom Filter Algorithm, Distributed Technology

1. 引言

随着信息技术的快速发展和信息基础设施的大规模建设，信息资源已经成为国家重要的战略储备资源，是经济建设和社会发展不可或缺的基础性条件。网络与信息安全问题日益突出，信息安全保障工作正面临着巨大的困难和挑战[1]。为了提前发现和预防安全风险，网络安全防护的理念逐渐由被动防御转变为主动防御，漏洞检测的技术逐渐被广泛应用。漏洞检测主要通过模拟入侵者的攻击方法，批量扫描

目标资产的脆弱性。越来越多的信息化管理者希望通过主动的安全隐患检测，提前发现和解决问题，实现防患于未然的目的。但是随着信息化资产的快速增长，对于漏洞主动发现的及时性、有效性的要求也越来越严格，漏洞检测技术的瓶颈也逐渐出现。为了提升漏洞检测的时效性和准确率，与当前迫切的安全需求相匹配，本文针对漏洞检测技术展开了深入的研究。

国内外已有很多专家开展了对于漏洞检测的技术研究工作，奥地利的 Jovanovic，Kruegel，Kird 开创性的提出了静态源代码分析法，通过此方法研究数据流，确认污点数据是否得到完善处理，同时推出服务侧源码静态分析工具 PiXy [2] [3]；Stefan Kals，EnginKirda 等人根据渗透测试推出了自动 Web 应用安全检测工具 SecuBat [4]，主要包含网络爬虫、渗透测试、漏洞分析、测试样本库，对于现代漏扫工具的发展起到了深远影响；法国 Renaud Derasion 在其基础之上进行了优化改进，研发了 C/S 架构的 Nessus [5] 系统，为了满足漏洞库升级的需求，采用了插件技术，维护过程非常简单；付堂欢[6]基于动态交互点和网址在页面中的分布规律设计与实现了漏洞扫描器的表单爬虫，利用预设的阈值和窗口，优先爬取在包含新 URL 较多的页面下发现的 URL；黄从韬[7]针对漏洞发掘技术和利用的方法不断变换的特点，设计并实现了一种基于规则和插件的可扩展的漏洞检测方法。以上研究为现代漏洞扫描技术的发展提供了参考性和可行性依据。

目前存在的漏洞扫描器依然存在一些不足。例如随着漏洞扫描时间的变长，Nessus 系统负载越来越大，用户在进行扫描配置和等待扫描结束需要的时间较长；AWVS、WebInspect、AppScan 这些漏洞扫描器均采用单机部署，其扫描效率取决于所部署服务器的性能；部分系统漏洞检测范围较小，漏洞库不能进行有效扩充，当新漏洞产生，不能准确识别。我国针对漏洞检测技术的研究工作不够深入，没有很完善的高性能检测系统，很多安全从业人员依然使用国外的检测引擎。因此，继续深入研究漏洞检测技术，设计一种高效、准确、全面、易用的漏洞检测引擎十分迫切和必要。

本文针对漏洞产生的原理和检测方法进行研究，设计并实现一种综合资产信息探测技术、网络爬虫技术、漏洞检测技术和分布式技术的漏洞检测引擎。通过资产信息探测为漏洞检测提供更全面的信息支持，提高漏洞检测的准确率；通过宽度优先遍历策略和布隆过滤器算法提高爬取目标交互点的精准度；通过分布式引擎架构提高漏洞检测的速率和稳定性。测试结果表明，本文设计与实现的引擎具有更高的漏洞检测准确率和扫描速率。

2. 分布式引擎的设计

2.1. 分布式技术

Celery 是一个简单、灵活、可靠的分布式队列，是分布式技术实现的基础，一般借助它进行任务的调度。当任务执行失败或者在任务执行过程中连接超时，它会自动发起连接，并重新执行任务[8]。

Celery 主要包含以下几个组件：

1) Beat (任务调度器)：任务调度器会读取配置文件的内容，并向任务队列发送即将到期的任务。

2) Producer (任务生产者)：任务生产者主要负责生成具体的任务，并发送到任务队列中。

3) Broker (消息中间件)：消息中间件存储任务生产者产生的任务，并下发给任务执行者。

4) Worker (任务执行者)：执行任务的消费者，通常在多个服务器上运行多个消费者，以提高执行的效率。

5) Backend (任务结果存储)：将状态信息和结果保存，供任务结束后查询。

任务生产者周期性地将任务发往消息中间件，任务执行者负责从消息中间件中获取任务，任务执行者执行完任务后将结果保存在任务结果存储中。Celery 架构如图 1 所示。

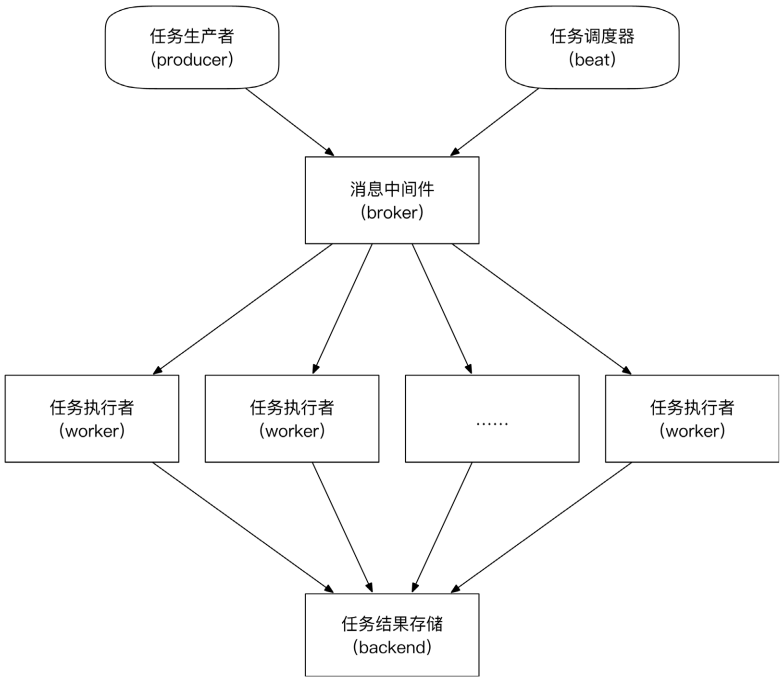

Figure 1. Celery architecture
图 1. Celery 架构

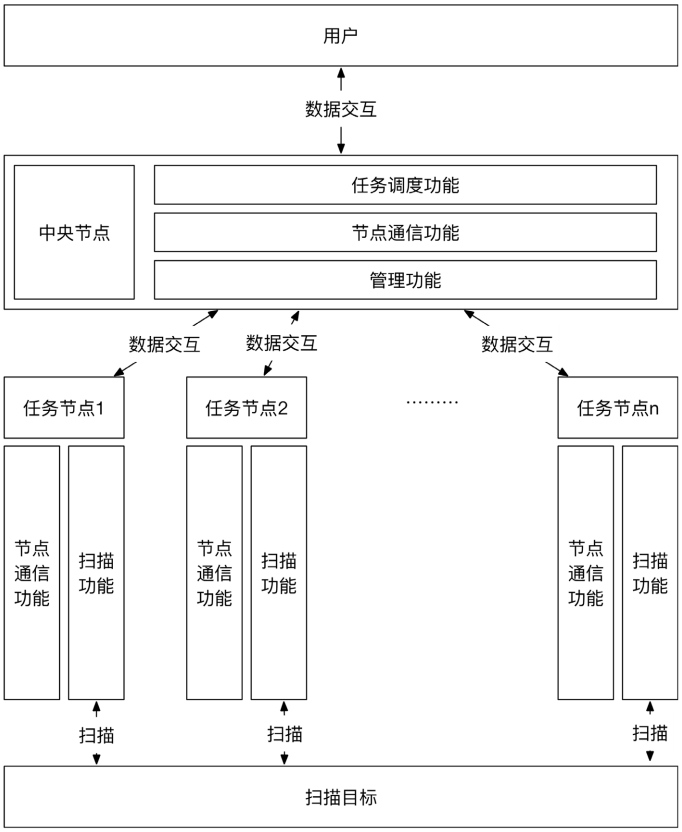

Figure 2. Distributed architecture diagram
图 2. 分布式架构图

2.2. 分布式引擎架构的设计

本文设计了分布式引擎，由中央节点和多个任务节点构成，采用可以进行全局控制多节点并行的主从式分布架构，中央节点对整个引擎的进行功能统筹与实时控制，进行整体的资源调度和扫描任务下发，并且对任务节点的 worker 进行数据监管，通过这种方式可以把控各任务节点的工作进展。任务节点主要负责网络爬虫、信息收集和漏洞检测三方面工作。其整体架构如图 2 所示。

2.3. 分布式引擎中央节点的设计

在主从式分布架构中，整个引擎的监管工作由中央节点执行，用户可以在中央节点配置扫描目标，中央节点向不同的任务节点发布检测任务，监管扫描状况，收集扫描数据。中央节点的各模块主要包括：前端交互模块、任务分配模块、消息监控模块、数据管理模块。中央节点的各功能模块及作用如表 1 所示。

Table 1. Function modules of the central node
表 1. 中央节点各功能模块

模块名称	作用
前端交互模块	用户在前端进行扫描配置，例如扫描目标、扫描节点配置等。扫描结束后，用户在前端可以查看扫描结果。
任务分配模块	根据用户配置的扫描目标，进行子任务划分，根据各任务节点中 worker 的负载情况，对各任务节点进行任务的调整与下发。
消息监控模块	实时与各任务节点进行通信，监管各任务节点 worker 的工作状态，若节点故障无法正常工作，则产生告警并将任务分配给其他节点。
数据管理模块	汇集各任务节点 worker 的检测数据，为用户提供条例清晰的检测报告。

2.4. 分布式引擎任务节点的设计

任务节点的功能模块包括：网络爬虫模块、信息探测模块和漏洞检测模块。任务节点的各个模块功能如表 2 所示。

Table 2. Function modules of the task node
表 2. 任务节点各功能模块

模块名称	作用
通信模块	保持与中央节点实时通信，获取中央节点分配的各项检测任务，并向中央节点反馈检测任务的执行结果。
网络爬虫模块	依据爬取规则获取目标网站的页面内容，提取网站中的交互点。
信息探测模块	根据配置好的检测规则，加载对应的信息探测插件，提交请求与解析响应，判断与收集目标网站的各项基本信息，辅助漏洞检测模块进行检测。
漏洞检测模块	根据配置好的检测规则，加载对应的漏洞检测插件，向检测目标提交带有 payload 的请求，解析响应，判断目标网站是否存在相应的漏洞。

2.5. 分布式引擎工作流程

分布式引擎的前后端为 B/S 结构，首先用户访问浏览器端，向中央节点下达检测任务并进行检测任务与检测规则配置。中央节点服务器在获取到分配好的检测任务后，将检测任务划分为不同的子任务，

并将子任务根据规则存入检测任务队列中，激活任务节点中的 worker。任务节点 worker 开启对目标的检测。在所有子任务检测完成后，将检测结果数据反馈到中央节点。中央节点负责收集与处理各任务节点的结果数据。分布式引擎的工作流程如图 3 所示。

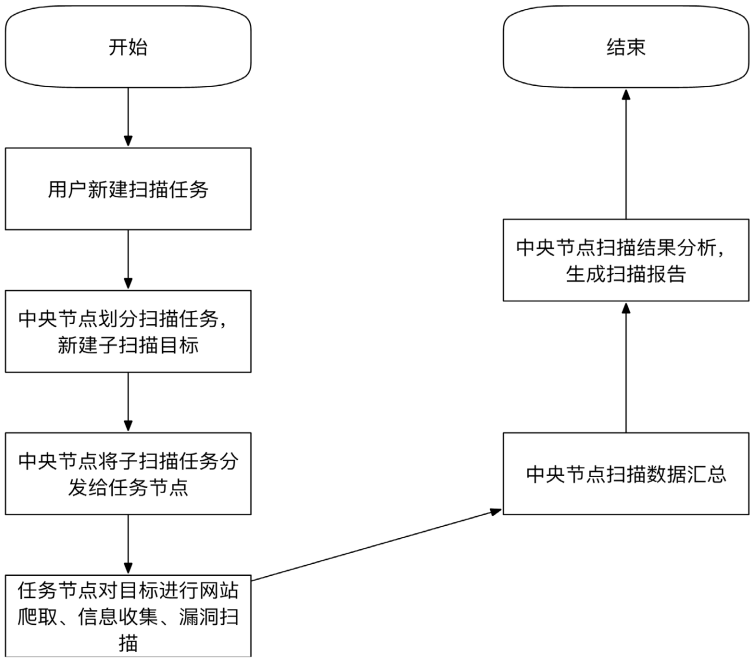

Figure 3. Workflow of a distributed engine
图 3. 分布式引擎的工作流程

3. 分布式引擎的实现

3.1. 分布式中央节点的实现

3.1.1. 任务分配模块的实现

在分布式引擎中，中央节点向每个任务节点的 worker 分配检测任务，到任务节点把检测任务执行完成并将检测数据反馈，整个检测过程包含了四部分：

1) 用户在中央节点进行检测任务下发并将主任务划分为子任务，然后根据子任务的检测级别分配到引擎的消息代理中。

2) 消息代理在接收到检测任务后，将不同的子任务分别压入不同的任务队列中。

3) 任务队列根据任务节点的运行情况，将任务分配给不同的任务节点的 worker，避免超出某一节点的性能负荷。

4) worker 在收到下发完成的任务后开始检测，在检测完成后将检测数据传输到任务结果存储中。

使用 Redis 数据库作为消息中间件，任务结果存储使用 MySQL 数据库，如图 4 和图 5 所示。

```
import djcelery
djcelery.setup_loader()
BROKER_URL='redis://127.0.0.1:6379'
```

Figure 4. Configure message-oriented middleware
图 4. 配置消息中间件

```
DATABASES = {
    'default':{
    'ENGINE':'django.db.backends.mysql',
    'NAME':'scanner',
    'HOST':'127.0.0.1'
    }
}
```

Figure 5. Configure task result storage
图 5. 配置任务结果存储

通过任务下发接口下发检测任务，任务队列根据各个任务节点服务器的性能进行最终的任务分配，如图 6 所示。

```
[14092] 06 Mar 21:58:39.995 # Server started, Redis version 3.2.100
[14092] 06 Mar 21:58:40.017 * DB loaded from disk: 0.021 seconds
[14092] 06 Mar 21:58:40.017 * The server is now ready to accept connections on port 6379
[14092] 06 Mar 22:03:41.060 * 10 changes in 300 seconds. Saving...
[14092] 06 Mar 22:03:41.066 * Background saving started by pid 14280
[14092] 06 Mar 22:03:41.366 # fork operation complete
[14092] 06 Mar 22:03:41.367 * Background saving terminated with success
[14092] 06 Mar 22:09:07.620 * 10 changes in 300 seconds. Saving...
[14092] 06 Mar 22:09:07.624 * Background saving started by pid 14128
[14092] 06 Mar 22:09:08.024 # fork operation complete
[14092] 06 Mar 22:09:08.025 * Background saving terminated with success
[14092] 06 Mar 22:24:09.096 * 1 changes in 900 seconds. Saving...
[14092] 06 Mar 22:24:09.116 * Background saving started by pid 14856
[14092] 06 Mar 22:24:09.416 # fork operation complete
[14092] 06 Mar 22:24:09.417 * Background saving terminated with success
[14092] 06 Mar 22:36:25.676 * 10 changes in 300 seconds. Saving...
[14092] 06 Mar 22:36:25.681 * Background saving started by pid 16400
```

Figure 6. Assignment of tasks to task queues
图 6. 任务队列的任务分配

3.1.2. 消息监控模块的实现

当消息监控模块监控任务的执行情况时，分布式引擎通过任务优先级调度的方法将更多的节点资源分配给优先级更高的任务，并提高任务执行的效率。同时为了避免节点中的 worker 运行负荷过高，通过令牌桶算法[9]来进行任务调度。令牌桶算法是速率限制(Rate Limiting)中最常用的算法之一，如果令牌桶中存在令牌，则允许发送流量，任务节点中的 worker 接收并执行任务；而如果令牌桶中不存在令牌，则不允许发送流量，任务节点中的 worker 停止接收任务。

当消息监控模块对节点的运行情况进行监控时，不需要对部署任务节点的服务器进行资源监控，仅需要判断任务节点中的 worker 是否能够继续接收任务即可。如果任务节点中的 worker 能够正常完成各项检测任务，则说明任务节点负载正常，如果 worker 在执行任务过程中长时间无正常响应，则说明任务节点资源异常，此时将此 worker 承担的任务分配给其他正常 worker。

3.2. 分布式任务节点的实现

3.2.1. 网络爬虫模块的实现

网络爬虫模块[10]主要包括解析模块和提取模块。解析模块负责爬取目标 url，并构造 http 请求，接收与解析目标 url 反馈的响应包，判断目标 url 是否存活。提取模块负责从解析 url 之后的页面中提取交互点。

网络爬虫采用宽度优先遍历策略，在爬取过程中先完成对根结点的遍历，再依次对所有的直接子结点进行遍历，通过这种策略尽可能扩大资产的爬取范围。基于宽度优先遍历的网络爬虫的工作流程如图 7 所示：

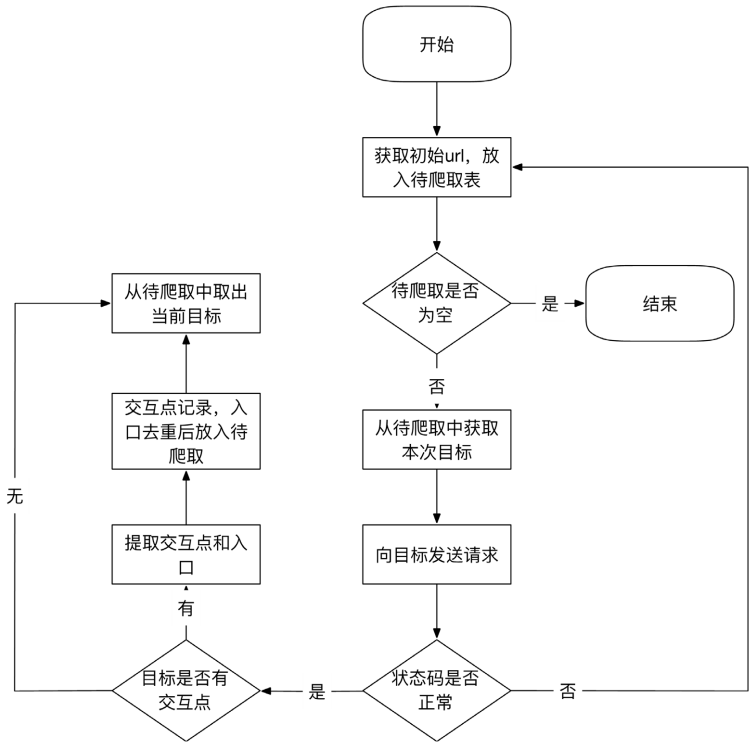

Figure 7. Workflow of crawler
图 7. 爬虫的工作流程

1) 首先是获取初始 url，将初始 url 放入待爬取列表，并且基于此 url 进行进一步的地址资源搜索；

2) 当在爬取过程中发现新的 url 时，将其放入待爬取列表；

3) 从待爬取列表中取出 url 并发送请求，通过响应的状态码判断 url 是否存活。若存活则继续对爬取的页面进行动态交互点提取；

4) 页面爬取完成后，将此 url 从待爬取列表中删除，然后取出新的 url 重复交互点的爬取，直至待爬取列表为空。

爬虫过程中需要对爬取的 url 进行去重，这里通过布隆过滤器算法[11]进行去重，定义了两个方法，一个负责向布隆过滤器添加元素，另一个负责判断元素是否位于布隆过滤器中，在调用布隆过滤器进行去重时，将 Hash 函数个数预先设定为 3，将二进制数组的长度设置为 10,000。添加元素和检查元素的核心代码如图 8、图 9 所示。

```
def add(self,msg):
    #add a string to bloomfilter
    if not isinstance(msg,str):
        msg = str(msg)
    positions,hashsum_positions =[],[]
    hash_sum =0
    for _hash_value in self._hashes(msg):
        hash_sum+=_hash_value % self.num_of_bits
        positions.append(_hash_value % self.num_of_bits)
    for pos in sorted(positions):
        self.data_store[pos] = True
    for _hash_sum_value in self._hashes(str(hash_sum)):
        hashsum_positions.append(_hash_sum_value % self.num_of_bits)
    for hashsum_pos in sorted(hashsum_positions):
self._data_store_hash[hashsum_pos]=True
```

Figure 8. The code to add elements
图 8. 添加元素的代码

```
def check(self,msg):
    #check if a string is in bloomfilter
    if not isinstance(msg,str):
        msg = str(msg)
    positions,hashsum_positions =[],[]
    hash_sum = 0
    for _hash_value in self._hashes(msg):
        hash_sum += _hash_value % self.num_of bits
        positions.append(_hash_value % self.num_of_bits)
    for position in sorted(positions):
        if self._data_store[position] == False:
            return False
    for _hash_sum_value in self._hashes(str(hash_sum)):
        hashsum_positions.append(_hash_sum_value % self.num_of_bits)
    for hashsum_pos in sorted(hashsum_positions):
        if self._data_store_hash[hashsum_pos]== False:
            return False
    return True
```

Figure 9. The code to examine elements
图 9. 检查元素的代码

3.2.2. 资产信息探测模块的实现

资产信息探测模块主要负责对目标进行信息收集，一方面为用户提供信息参考，另一方面当获取到新的动态交互点可以添加到待检测目标中，增加漏洞检测的准确率。该模块包含了多种信息收集子模块，例如可以针对一些可能包含敏感信息的文件进行扫描，针对目标地址开放的端口进行扫描，针对目标服务器响应包中的 banner 进行扫描以获取系统指纹信息等。其中包含的信息探测模块如图 10 所示。

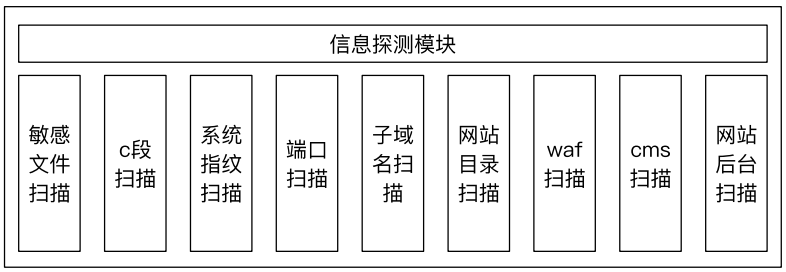

Figure 10. Structure of information detection module
图 10. 信息探测模块的结构

1) 敏感文件扫描

敏感文件是指具有网站敏感信息的文件，例如网站源文件、网站备份文件、包含用户数据的文件等。在没有任何外部访问控制的情况下，一旦被攻击者获取，极易被利用，进一步开展对服务器的攻击。针对敏感文件的扫描，首先是构建敏感文件的后缀规则库，将后缀与检测目标的目录进行拼接，然后将构造的 http 请求发送给目标服务器，根据 http 响应的状态代码确定是否存在敏感文件。

2) c 段扫描

c 段扫描主要是为了检测处于同一 c 段的存活的主机，资产信息探测模块在接收到检测目标后，对 c 段中的每个目标发起主动连接，并监听是否有响应信息返回。若存在响应，则证明此 ip 存活，并将存活的 ip 录入到数据库中，继续对其他 ip 进行遍历，未返回响应则证明不存在活跃主机。

3) 系统指纹扫描

获取存活主机返回的指纹信息，从指纹信息中可以获取软件开发商，软件名称、版本、服务类型等信息。从这些信息中可以判断目标系统是否存在脆弱性，借助某些工具可以直接调用对应的 exp 进行攻击。

4) 端口扫描

端口扫描主要是为了发现目标服务器开放了哪些服务，该功能的实现主要是通过 socket 套接字进行探测，向目标服务器的端口不断发送请求建立连接的数据包。如果对方存在此服务则会应答，全端口的探测可以通过增加线程进行快速探测，资产较多的情况下通过分布式探测增加探测速度。

5) 子域名扫描

通过子域名扫描可以发现目标存在的其他服务，增加发现漏洞的可能性。首先构造子域名字典，对目标主域名进行字典爆破来发现可能存在的子域名。对爆破发现的子域名进行 dns 解析，如果可以解析为存活的真实 ip，则说明目标存在该子域名，将发现的子域名保存到数据库中；如果解析不成功，则说明不存在，进行其他测试，直至字典爆破结束。

6) 网站目录扫描

网站目录扫描主要是为了发现更多有价值的信息，比如网站的管理后台、网站的配置信息、网站的源码备份等。网站目录扫描的工作流程为：首先，获取目标的初始 url 并将其存储在数据库中，然后在初始页面中寻找新的 url，将获取的新 url 和数据库中的 url 进行比较，以确定它是否已存储，如果没有存储则将其录入数据库中，并对该 url 的页面进行爬取。爬取结束后，数据库中的 url 则为获取的网站目录。

7) waf 扫描

目前大多数网站都会被 waf 防护，导致漏洞检测无法正常工作，进行 waf 扫描主要是为了判断目标是否被 waf 防护。如果被防护则根据返回的信息判断 waf 的类型，进行简单的绕过，保持漏洞检测引擎的正常工作。不同厂商的 waf 具备不同的特征，这些特征可能会在响应数据包的 header 头部信息中显示。进行扫描时，首先主动向目标发送请求，在获取的响应数据包中找出相应的字段并与已有的特征进行比对。本系统对网络中常见的 waf 进行了特征收集，尽可能判定目标的 waf 类型[12]。

8) cms 扫描

通过对 cms 的扫描可以发现目标正在使用的 cms 以及使用的版本，基于以上信息，可以通过检索当前版本 cms 已暴露出来的漏洞进行攻击利用。目前市面上使用的 cms 种类非常多，想要识别目标使用的 cms 版本，需要采集不同 cms 的特征进行比对。目前已收集了 1400 多种 cms 特征，主要通过识别 MD5 和正则表达式匹配的方式来判单判断目标的 cms 类型。cms 的指纹如下所示：

```
{
    "url": "/kindeditor/license.txt",
    "re": "",
    "name": "T-Site 建站系统",
    "md5": "b0d181292c99cf9bb2ae9166dd3a0239"
}
```

9) 网站后台扫描

对网站进行后台扫描，主要是为了发现网站开发后台，并进行版本漏洞利用。扫描模块中包含构造好的后台字典，然后将常见的后台访问链接与目标地址逐一拼接，根据能否正常访问来判断是否存在相应的网站后台。

3.2.3. 漏洞检测模块的实现

漏洞检测是引擎的核心功能，从结构上看，漏洞检测模块包括交互点提取、漏洞特征识别、漏洞扫描模块。同时为了更好的与任务节点配合，需要实时与节点通信模块保持通信。为了扩大漏洞检测的范围，增加漏洞检测的准确率，需要接收来自信息探测模块采集的相关信息。以上几个模块共同为引擎的漏洞检测功能服务，结构如图 11 所示。

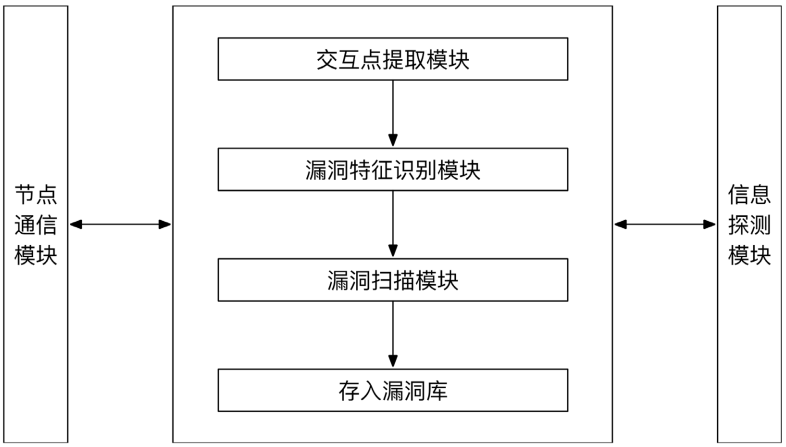

Figure 11. Vulnerability detection module structure
图 11. 漏洞检测模块结构

当进行漏洞检测时，首先用户下发检测任务，检测目标被漏洞检测模块获取，根据针对目标收集的基础信息，调用对应的漏洞检测插件，检测插件会向目标交互点发送请求，请求中的 POC 中包含攻击载荷。检测模块会监听目标的响应，它主要通过判断响应的信息来确定目标是否存在对应的漏洞。若存在则将漏洞信息存储到数据库中，不存在则结束当前检测，开始下一项漏洞的检测。漏洞扫描的流程如图 12 所示。

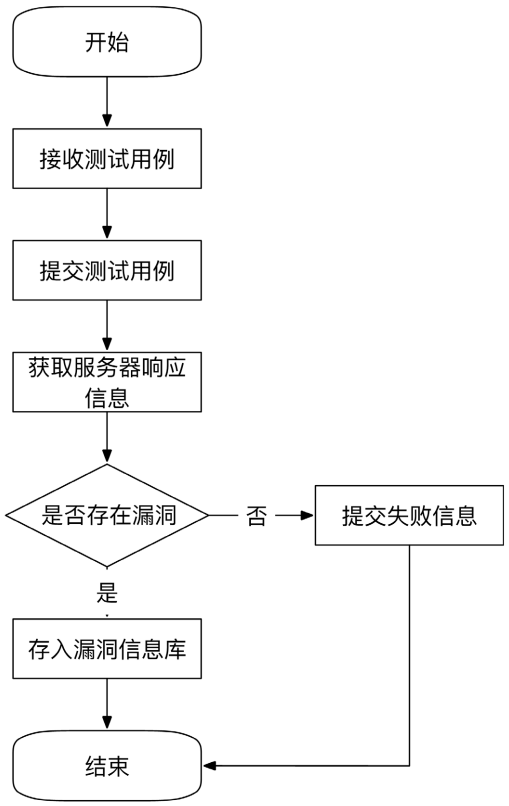

Figure 12. Vulnerability scanning workflow
图 12. 漏洞扫描工作流程

为了增强漏洞检测模块的扩展性，除了包含针对常见漏洞进行检测的模块，还有一个可以扩展的漏洞检测插件模块。本文对所有的漏洞检测插件进行规范化，所有扫描插件都有相同的编码标准[13]，使得检测引擎能够及时识别出现的新漏洞，编写格式如图 13 所示。

```
from pocsuite.poc import POCBase, Output
from pocsuite.utils import register


class TestPOC(POCBase):
    name = ''
    vulID = '12345'  # https://www.seebug.org/vuldb/ssvid-12345
    author = ['HELLOWORLD']
    vulType = ''
    version = '1.0'    # default version: 1.0
    references = ['https://seebug.org/']
    desc = '''......'''

    vulDate = 'XXXX-XX-XX'
    createDate = 'XXXX-XX-XX'
    updateDate = 'XXXX-XX-XX'

    appName = ''
    appVersion = ''
    appPowerLink = ''
    samples = ['']

    def _attack(self):
        '''attack mode'''
        return self._verify()

    def _verify(self):
        '''verify mode'''
        result = {}

        ...

        result['VerifyInfo'] = {}
        result['VerifyInfo']['URL'] = self.url
        return self.parse_output(result)

    def parse_output(self, result):
        output = Output(self)
        if result:
            output.success(result)
        else:
            output.fail('Internet nothing returned')
        return output


register(TestPOC)
```

Figure 13. Plug-in writing format
图 13. 插件编写格式

4. 测试结果及分析

4.1. 测试目标选取

鉴于网络安全法中明确禁止个人从事未经授权的漏洞探测和任何其他危害公共系统安全的行为，此次测试将个人搭建的三个网站作为测试目标。本次测试采用了对比测试的方法，进行测试的漏洞检测引擎包括常用的漏洞扫描工具 AWVS、APPSCAN、WDScanner [14]和本系统。其中 AWVS、APPSCAN 为目前白帽子使用的主流漏洞扫描器，WDScanner 为 ted 团队开发的具有代表性的分布式扫描器，通过对比以上几款扫描器的检测结果，可以更直观的反映本系统的真实性能。

4.2. 测试环境搭建

检测引擎配置在 5 台虚拟机上，其中 1 台部署中央节点，其他 4 台部署任务节点，保持每台虚拟机的内存、cpu、硬盘等基础配置相同，其他检测引擎根据要求部署在同样配置的虚拟机中，虚拟机中的软件信息如表 3 所示。

Table 3. Environment configuration for a distributed engine
表 3. 分布式引擎的环境配置

软件名称	版本	用途
Redis	5.0.14	存储队列
python	3.8	编程语言
mysql	5.7.4	存储扫描数据
django	3.0	检测引擎的 B/S 架构
celery	4.1.1	分布式架构框架

4.3. 性能测试

在检测引擎部署完成后，分别使用四种检测引擎对三个目标进行扫描，统计扫描的时间和发现的漏洞数量，对比四款引擎的漏洞检测效率。统计结果如表 4 所示。

Table 4. Comparison of test results
表 4. 检测结果对比

扫描工具	A 组		B 组		C 组		平均扫描时间	平均漏洞数量
	扫描时间	漏洞数量	扫描时间	漏洞数量	扫描时间	漏洞数量		
AWVS	38 min 11 s	65	10 min 20 s	23	20 min 3 s	47	22 min 48 s	45
APPSCAN	30 min 6 s	52	9 min 53 s	13	18 min 29 s	33	19 min 30 s	32
WDScanner	26 min 7 s	55	8 min 49 s	19	15 min 33 s	39	16 min 48 s	37
本系统	20 min 40 s	59	5 min 6 s	21	12 min 6 s	43	12 min 36 s	41

1) 通过对比扫描的平均时间可以发现，本系统的扫描速度最快：

本系统 < WDScanner < APPSCAN < AWVS

2) 在发现的平均漏洞数量上，AWVS 发现的漏洞数量最多，本系统第二：

AWVS > 本系统 > WDScanner > APPSCAN

3) 在查看漏洞信息发现，由于本系统的漏洞库不够完善，部分漏洞未检出，所以发现漏洞数量偏低。

为了统计以上几款检测引擎的漏洞检测准确率，对发现的漏洞进行手工验证，验证结果如表 5 所示：

Table 5. Accuracy comparison
表 5. 准确率对比

扫描工具	A 组		B 组		C 组		平均误报数量	平均准确率
	误报数量	准确率	误报数量	准确率	误报数量	准确率		
AWVS	13	80%	3	86.9%	8	82.9%	8	83.2%
APPSCAN	22	57.6%	6	53.8%	17	48.4%	15	53.2%
WDScanner	16	70.9%	4	78.9%	13	66.6%	11	72.1%
本系统	15	74.5%	3	85.7%	10	76.7%	9	78.9%

1) 对比统计的平均误报数量，AWVS 误报数量最少，APPSCAN 误报数量最多，本系统的误报数量略高于 AWVS。

AWVS < 本系统 < WDScanner < APPSCAN

2) 对比计算的平均准确率，AWVS 准确率最高，本系统准确率第二位。

AWVS > 本系统 > WDScanner > APPSCAN

漏洞检测引擎不能保证将所有漏洞检出，存在漏报的情况，因此计算漏报率是衡量漏洞检测引擎性能的另一个重要指标。本文采取的方法是，将所有引擎检测的有效漏洞的并集作为漏洞检测的最终结果，经过比对发现本次检测发现的 A、B、C 三组的有效漏洞数分别为 58、31、43 个，统计的漏报情况如表 6 所示：

Table 6. Comparison of false alarm rate
表 6. 漏报率对比

扫描工具	A 组		B 组		C 组		平均漏报数量	平均漏报率
	漏报数量	漏报率	漏报数量	漏报率	漏报数量	漏报率		
AWVS	6	10.3%	11	35.4%	4	9.3%	7	18.3%
APPSCAN	28	48.2%	24	77.4%	27	62.7%	26	62.7%
WDScanner	19	32.7%	16	51.6%	17	39.5%	17	41.2%
本系统	14	24.1%	13	41.9%	10	23.2%	12	29.7%

1) 对比统计的平均漏报数量，AWVS 漏报数量最少，APPSCAN 漏报数量最高，本系统漏报数量略高于 AWVS。

AWVS < 本系统 < WDScanner < APPSCAN

2) 对比计算的平均漏报率，AWVS 漏报率最低，本系统漏报率略高于 AWVS。

AWVS < 本系统 < WDScanner < APPSCAN

结合以上数据分析得出如下结果：

1) AWVS 检测的准确率最高，本系统次之，APPSCAN 检测的准确率最低；

2) 本系统检测的速度最快，AWVS 检测的速度最慢；

3) AWVS 检测的漏报率最低，本系统次之，APPSCAN 检测的漏报率最高。

从准确率、速率、漏报率三个指标综合分析发现：APPSCAN 的综合性能最差，AWVS 的综合性能比较突出，但是检测的时间太长，本系统的检测准确率和漏报率最接近 AWVS，在检测速率上占据上风。

综上所述，与其他检测引擎相比，本系统可以快速准确的对目标进行漏洞检测，满足设计的要求与目的。

5. 结语

针对目前漏洞检测技术中存在的不足，基于前人研究的基础上，本文对漏洞检测相关技术进行了应用和改进，设计与实现了一种分布式资产信息探测与漏洞检测引擎。该分布式引擎通过宽度优先遍历策略提高了漏洞检测的全面性；通过布隆过滤器算法对爬虫获取的 url 进行去重，提高了漏洞检测的效率；通过资产信息探测辅助漏洞检测，提高了漏洞检测的准确率；通过分布式架构极大提高了漏洞检测的速率。该分布式引擎与其他三款漏洞扫描器相比，提高了漏洞检测的准确率和速率，可以很好地应用到实际生产环境中，为主动发现网络环境的安全隐患提供了有力保障。

致 谢

向本文在完成过程中给予指导和帮助的老师以及给本文提出指导意见的评审专家表示感谢。

基金项目

本课题得到山东省自然科学基金项目(ZR2021MF092)资助。

参考文献

[1] 邓景辉. 浅议计算机网络信息安全现状及防护措施[J]. 信息与电脑(理论版), 2010(6): 1

[2] Jovanovic, N., Kruegel, C. and Kirda, E. (2018) Pixy: A Static Analysis Tool for Detecting Webapplication Vulnerabilities. http://www.seclab.tuwien.ac.at/papers/pixytechreport.pdf

[3] Pixy. http://pixybox.seclab.tuwien.ac.at/pixy

[4] http://www.iseclab.org/papers/secubat.pdf

[5] http://www.tenable.com/products/nessus

[6] 付堂欢. 基于网络的 Web 应用漏洞扫描系统的分析与设计[D]: [硕士学位论文]. 北京: 北京邮电大学, 2012.

[7] 黄从韬. 面向 Web 服务安全的漏洞扫描器的设计与实现[D]: [硕士学位论文]. 长沙: 中南大学, 2012.

[8] Genre Coulouri, Jean Dollimore, Tim Kindberg, 等. 分布式系统概念与设计[J]. 计算机教育, 2013(12): 30-35.

[9] 刘振宇. 基于令牌桶算法的网络流量控制技术的研究与实现[D]: [硕士学位论文]. 呼和浩特: 内蒙古大学, 2012.

[10] Cho, W.-S., Lee, J.-E. and Choi. C.-H. (2013) Refresh Cycle Optimization for Web Crawlers. *Journal of the Operational Research Society*, **13**, 30-39. https://doi.org/10.5392/JKCA.2013.13.06.030

[11] Kirsch, A. and Mitzenmacher, M. (2008) Less Hashing, Same Performance: Building a Better Bloom Filter. *Random Structures and Algorithms*, **33**, 187-218. https://doi.org/10.1002/rsa.20208

[12] EnableSecurity.github. https://github.com/EnableSecurity/wafw00f

[13] Pocsuite.github. https://github.com/knownsec/Pocsuite/wiki/How-to-get-started-with-writing-a-POC-module

[14] TideSec.github. https://github.com/TideSec/WDScanner