中国研究生网络安全创新大赛

作品报告

作品名称：MKWAY 白名单规则自动化构建工具

提交日期：2024 年 9 月 20 日

队伍名称：银河护卫队

队员姓名：杨昌澄、赵文博、张雪怡、李太森

填写说明

1. 所有参赛项目必须为一个基本完整的设计。作品报告书旨在能够清晰准确地阐述（或图示）该参赛队的参赛项目（或方案）。

2. 作品报告采用A4纸撰写。除标题外，所有内容必需为宋体、小四号字、1.5倍行距。

3. 作品报告中各项目说明文字部分仅供参考，作品报告书撰写完毕后，请删除所有说明文字。(本页不删除)

4. 作品报告模板里已经列的内容仅供参考，作者可以在此基础上增加内容或对文档结构进行微调。

5. 为保证网评的公平、公正，作品报告中应避免出现作者所在学校、院系和指导教师等泄露身份的信息。一经发现，取消作品参赛资格。

目　录

图 目录

表 目录

摘要

在当前数字化时代，系统安全面临着严峻挑战，系统被入侵后，如何**有效识别和防御异常命令和异常文件访问**成为关键问题。根据进程和文件访问的正常行为构建的白名单可以有效阻止系统中的异常命令与访问，但是因为操作系统中的进程和文件访问的数量级庞大，因此如何设计一种**无监督的自动学习算法**来生成白名单规则成为一个核心问题。本方案提出了一种**基于 MiniBatchK-Means 聚类算法的白名单规则自动化构建工具**，旨在自动学习并生成系统进程和文件访问的白名单规则。本方案通过聚类方法降低白名单规则数量，以提高管理效率和监控时的异常发现速度，同时确保规则的粒度适当，避免漏掉潜在的攻击行为。

本方案以 MiniBatchK-Means 聚类算法为核心，巧妙融合了掩码技术以降低白名单规则的复杂度。通过自动学习系统中的进程与文件访问模式，进行正常行为模式的识别，并将正常行为模式最终转化为高效的白名单规则。这些规则**不仅数量显著减少，而且能够精确地去除潜在的异常行为**，从而在不影响系统性能的前提下，大幅提升了系统的安全监控能力。此外，算法还特别考虑了**规则粒度的平衡**，避免了过于粗糙的规则可能导致的安全隐患，确保了在降低规则数量的同时，不会遗漏对攻击行为的检测。

通过对不同规模的数据集进行测试，验证了本方案的有效性。测试结果表明，本工具可以将给定的 fileaccessdata 样例数据集中的 50347 条白名单规则简化为 269 条，且可以确保降低后的**白名单覆盖率达到 99.89%**。同时，该模型具有良好的性能，对规则数达到 20000 的数据集进行聚类所**花费的时间仅为 1.51 秒**。

关键词：白名单规则　无监督学习　聚类算法　异常检测

Abstract

In the current digital age, system security is facing severe challenges. After a system is compromised, how to effectively identify and defend against abnormal commands and file access becomes a key issue. Whitelist rules based on the normal behavior of processes and file access in the operating system can effectively organize abnormal commands and access within the system. However, due to the large number of processes and file accesses in the operating system, designing an unsupervised automatic learning algorithm to generate whitelist rules becomes a core problem. This proposal introduces an automated whitelist rule construction tool based on the MiniBatchK-Means clustering algorithm, aiming to automatically learn and generate whitelist rules for system processes and file access. This tool uses clustering methods to reduce the number of whitelist rules, thereby improving management efficiency and the speed of anomaly detection during monitoring, while ensuring the appropriate granularity of rules to avoid missing potential attack behaviors.

The tool, with MiniBatchK-Means clustering algorithm at its core, ingeniously integrates masking techniques to reduce the complexity of whitelist rules. By automatically learning the process and file access patterns in the system, it identifies normal behavior patterns and ultimately transforms them into efficient whitelist rules. These rules not only significantly reduce in number but also precisely eliminate potential abnormal behaviors, thereby greatly enhancing the system's security monitoring capabilities without affecting system performance. In addition, the algorithm also specially considers the balance of rule granularity to avoid security risks that may arise from overly coarse rules, ensuring that the detection of attack behaviors is not missed while reducing the number of rules.

Through testing on datasets of different scales, the effectiveness of this solution has been verified. The test results show that the model can simplify the 50,347 whitelist rules in the given fileaccessdata sample dataset to 269, while ensuring that the reduced whitelist coverage rate reaches 99.89%. The model also has good performance, taking only 3.71 seconds to cluster a dataset with 20,000 rules.

Keywords： Whitelist rules; Unsupervised learning; Clustering algorithm;Anomaly detection

第一章　作品概述

1.1 研究背景

在数字化时代背景下，网络安全问题已经成为全球范围内的共同挑战。随着网络技术的不断进步，网络攻击手段也日益多样化和复杂化[1]，从传统的病毒、木马到高级持续性威胁（APT）和零日漏洞攻击，这些攻击行为对个人隐私、企业运营甚至国家安全构成了严重威胁，表 1.1 展示了常见的漏洞类型。在这样的大环境下，传统的入侵检测系统[2]已经难以应对快速变化的攻击模式，因此，如何有效地识别和防御这些攻击行为，成为了网络安全领域急需解决的问题。

表 1.1　2023 年漏洞数据汇总表

序号	漏洞类型	所占比例
1	恶意软件感染	27.4%
2	非法外联通信	21.0%
3	僵尸网络感染	16.3%
4	挖矿木马行为	14.4%
5	木马后门感染	8.8%
6	暴力破解	6.4%
7	网站web注入	1.9%
8	漏洞利用	1.4%
9	其他	2.4%

在众多的安全防护策略中，白名单机制[3]因其独特的优势而备受关注。白名单机制通过预先定义一系列被认为安全的进程和文件访问规则，仅允许这些规则内的活动发生，从而有效过滤掉不在白名单中的可疑行为。这种机制能够显著提高系统的安全性，因为它能够阻止未知的或未经授权的行为，减少潜在的安全风险。然而，由于操作系统中进程和文件的访问关系可能非常庞大，直接管理和维护这些规则变得极其复杂和繁琐。为了解决这一问题，引入了掩码技术，它通过使用通配符简化规则，减少规则的数量，使得规则管理变得更加高效。

尽管掩码技术在一定程度上简化了规则管理，但如何设计一个能够自动学习并生成这些白名单规则的算法，仍然是一个挑战。这正是本次比赛的核心任务——设计并

实现一个无监督学习算法[4]，它能够从大量的系统进程和文件访问数据中自动学习并生成有效的白名单规则。该算法需要利用聚类等方法，以减少规则的数量，同时保持高准确度，确保能够快速有效地检测到异常行为。此外，算法还需要在规则的粒度和数量之间找到恰当的平衡点，避免因规则过粗而漏掉真正的攻击，或因规则过细而导致过多的误报。

在这一过程中，算法的设计和实现需要考虑多个方面的因素。首先，算法需要具备良好的聚类能力，能够从复杂的数据中识别出有意义的模式和关系。其次，算法需要具备较高的准确度，以确保生成的白名单规则能够有效地覆盖正常的系统行为，同时准确地识别出异常行为。此外，算法还需要具备良好的适应性，能够适应不同的系统环境和应用场景，以应对不断变化的网络威胁。最后，算法的性能也是一个重要的考量因素，它需要在保证准确度的前提下，尽可能地提高学习效率，减少学习时间。

1.1.1 异常命令与访问

在系统被入侵后，攻击者会采取多种手段来获取或修改系统中的数据，其中关于异常命令与异常文件访问的有以下几种：

(1) 未知或不常用的命令：系统日志中出现未知或不常用的命令，尤其是那些通常不由用户直接执行的系统级命令。

(2) 频繁的命令执行：短时间内频繁执行某些命令，尤其是那些用于数据访问或系统配置更改的命令。

(3) 命令的参数异常：命令的参数与正常使用时不同，例如使用非标准路径或文件名。

(4) 访问敏感文件：尝试访问或修改通常不对普通用户开放的敏感文件，如系统配置文件、密码文件或关键数据文件。

(5) 用户或进程的文件访问模式发生改变，例如访问通常不访问的文件类型或目录。

(6) 未经授权更改文件权限，使得更多的用户或进程能够访问敏感文件。

针对上述几种攻击方式，系统可以通过日志分析[5]来定期检查系统和应用程序日志来寻找异常命令和文件访问的迹象。在遇到异常用户行为时，对用户的行为模式进行分析，识别与正常行为不同的活动。也可以使用机器学习[6]等技术对异常行为进行分析，并建立正常行为的基线来识别偏离这些基线的异常行为。检测到入侵行为后，

入侵防御可以根据配置的响应动作进行自动处置，包括产生告警、丢弃数据包、阻止来自源地址的流量或重置连接。

常见入侵防御流程如图 1.1 所示：

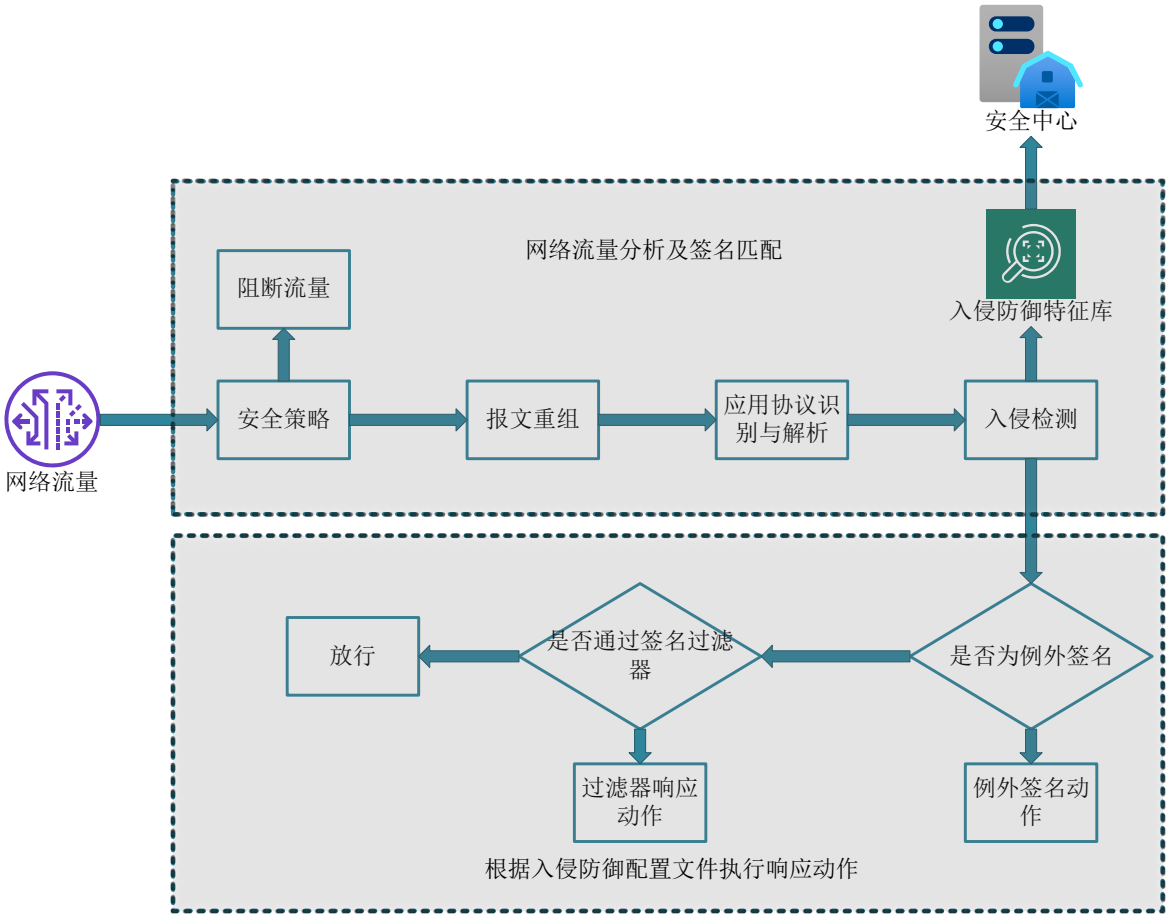

图 1.1　入侵防御流程

1.1.2 白名单机制

白名单是一种保护系统安全的机制，白名单机制允许经过批准的电子邮件地址、IP 地址和应用程序，并阻止其他地址。该机制可以确保用户只能访问已经进行明确授权的项目，以此来增强计算机和网络的安全性。白名单仅向预先批准的实体提供访问权限，在名单内的项目可以合法使用系统资源，其他项目的访问权限会被制止。白名单的流程如图 1.2 所示，当程序或命令试图在系统内运行时，必须根据白名单对其进行验证，只有当其在白名单中时，才能获得访问权限。

本方案中白名单可以明确指定哪些进程可以访问哪些文件或目录，确保只有经过审核的操作才能被执行。通过限制未授权的访问，白名单可以有效防止攻击者利用系统漏洞进行恶意操作。

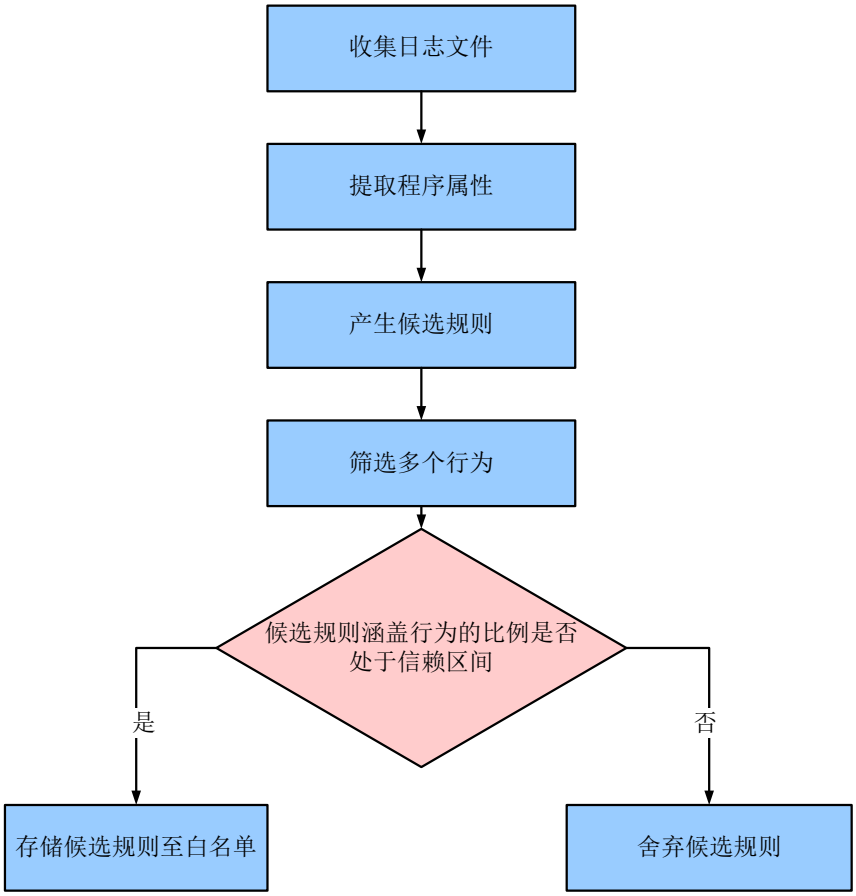

图 1.2　白名单流程

白名单是实现系统安全的重要组成部分，此策略会自动授予名单中命令或软件的访问权限，可以避免在每次尝试访问时对权限进行检查。白名单的目的是确保只有安全和必要的资源采可以访问，从而保护系统免受潜在的安全威胁。在对系统进行保护的过程中，白名单机制主要发挥以下的几个作用：

(1) 用户权限管理：在多用户的环境下，白名单可以定义用户的访问权限，可以指定哪些用户可以访问哪些程序，这种功能有助于防止数据泄露和未授权访问。

(2) 程序控制：在终端安全中，白名单可以限制哪些应用可以在用户的设备上运行，这可以有效防止恶意软件的执行，只有那些在白名单中的程序才会被允许执行。

(3) 文件和资源访问：在文件系统级别，白名单可以用于控制哪些进程和用户可以读取、写入或执行特定的文件，这有助于保护关键数据不被未授权修改。

(4) 自动化任务和脚本执行：在自动化环境中，白名单可以用于限制哪些脚本或自动化任务可以被执行。这有助于防止恶意脚本的运行，确保自动化流程的安全性。

白名单机制的有效性依赖于其准确性，随着系统环境的变化，新的规则需要被添加到白名单中，而不再安全的规则需要被废除。在实际的部署中，白名单机制通常与

其他安全机制结合使用，这种方法可以更加有效的保护系统免受各种威胁的侵害。

1.1.3 掩码技术

掩码技术指在数据存储、传输或处理过程中，使用特定的模式或规则来隐藏或保护某些信息的技术。在不同的上下文中，掩码技术可以有不同的应用和实现方式。在系统保护的领域中，掩码技术通常用于数据脱敏、访问控制和网络流量控制等方面。在展示或共享敏感数据时，使用掩码技术来隐藏部分信息，例如显示信用卡号时只显示后四位，其余用星号代替。在用于访问控制时，掩码技术可以用来定义访问规则。而在网络层面，掩码技术可以用来定义数据包过滤规则，比如通过 IP 地址和端口号的掩码来控制数据流。

在本方案中使用掩码技术主要是为了提高安全性、简化管理以及增强灵活性。掩码技术可以通过限制敏感数据来防止未授权访问和数据泄露。在进行控制列表管理时，可以使用该技术来减少需要单独配置的规则数量。本方案使用掩码技术来进行文件访问控制，通过使用文件路径的掩码规则来定义哪些进程可以访问哪些文件或目录。在生成白名单规则时，可以使用掩码技术来合并相似的规则，减少规则的数量，并且保持规则的有效性和安全性。在无监督学习算法中，掩码技术可以用于从大量的访问记录中自动学习并生成通用的访问控制规则。

1.1.4 聚类算法

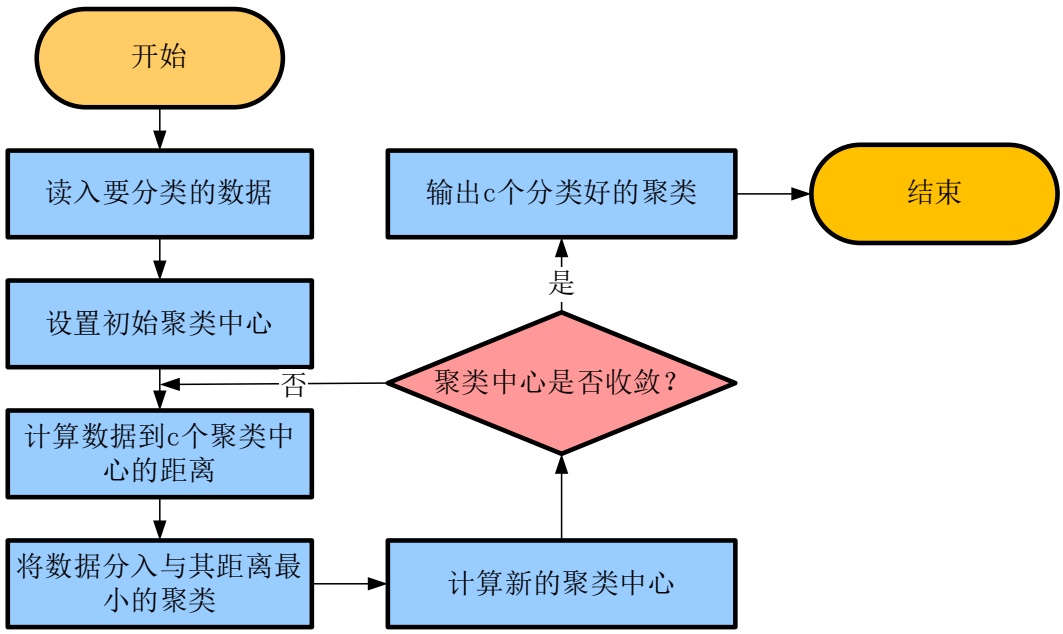

图 1.3　聚类算法流程图

聚类算法[7]是一类无监督学习算法，其目的是将数据集中的样本划分成若干个彼此相似的组或“簇”。这些算法不需要事先指定类别标签，而是直接从数据中发现内在的结构和模式，常见聚类算法的流程如图 1.3 所示。聚类算法通常以相似性为基础进行分析，在一个聚类中的模式之间比不在同一聚类中的模式之间具有更高的相似性。聚类算法通常在特征空间中工作，每个数据点由一组特征去描述。这种算法需要一种方法去衡量数据点之间的相似度，常见的度量方法有欧几里得距离、曼哈顿距离、余弦相似度、KL 距离等，其公式如下表 1.2 所示：

表 1.2　聚类算法距离公式

类型	公式
欧几里得度量	$$d(x,y)=\sqrt{(x_1-y_1)^2+(x_2-y_2)^2+\cdots+(x_n-y_n)^2}=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$$
曼哈顿度量	$$d(x,y)=\lvert x_1-y_1\rvert+\lvert x_2-y_2\rvert+\cdots+\lvert x_n-y_n\rvert=\sum_{i=1}^{n}(x_i-y_i)$$
夹角余弦相似度	$$cos(\theta)=\frac{\sum_{k=1}^{n}x_{1k}x_{2k}}{\sqrt{\sum_{k=1}^{n}x_{1k}^2}*\sqrt{\sum_{k=1}^{n}x_{2k}^2}}=\frac{a^Tb}{\lvert a\rvert\lvert b\rvert}$$
KL 距离	$$D(P\lvert\lvert Q)=\sum_{x}P(x)log(\frac{p(x)}{Q(x)})$$

可以将聚类算法简单的分为几种大类，第一种为划分法，划分法给定一个有 N 个元组或记录的数据集，并使用分裂法将其构造为 k 个分组，每个分组就代表一种聚类，其中满足 K<N。大部分划分法是基于距离的，划分法通常先进行一个初始划分，采用一种迭代的重定位技术，将对象从一个组移动到另一个组进行划分。第二种为层次法，该方法对给定的数据集进行层次分解，知道满足某种条件。第三张为密度算法，基于密度的算法不是根据距离而是根据密度，这样可以克服基于距离的算法只能发现“类圆形”的聚类缺点。图论聚类发和网格算法也是常见的聚类算法。

本方案中的聚类算法使用 K-Means 算法[8]，该方法也可以称为 K-均值或 K-平均算法。在输入样本 T=x_1,x_2,…x_m后，选择初始化的 k 个类别中心a_1,a_2,…a_k，对每个

样本x_i将其标记位距离类别中心a_j最近的类别 j，更新每个类别的中心店a_j为隶属该类别的所有样本的均值，再重复上面的两步操作，直到达到某个中止条件。记 K 个簇中心分别为$a_1,a_2,\dots a_k$，其中每个簇的样本数量为$N_1,N_2,\dots N_k$。

使用欧几里得距离公式，具体公式为：

$$J(a_1,a_2,\dots a_k)=\frac{1}{2}\sum_{j=1}^{k}\sum_{i=1}^{N_j}(x_i-a_j)^2 \tag{1-1}$$

如果要获取最优解，对 J 函数求偏导数簇中心点 a 更新的公式为：

$$\frac{\partial J}{\partial a_j}=\sum_{i=1}^{N_j}(x_i-a_j)\overset{令}{\rightarrow}0\Rightarrow a_j=\frac{1}{N_j}\sum_{i=1}^{N_j}x_i \tag{1-2}$$

对 K-Means 算法进行可视化，如图 1.4 所示,先对需要进行分类的数据进行展示，首先选择 k 个初始聚类中心点，这里设置为四个聚类中心点.针对每一个数据点，计算其与类聚中心点之间的距离，并将其归为距离最近的聚类点所在的类。计算每个聚类中心点所在类的数据点的平均值，将其作为新的聚类中心点。再根据欧氏距离来度量数据点之间的相似性，重新划分为四个簇。最后重复上述步骤，知道聚类中心点不再变化，即已经将数据分为了 k 个不同的类。

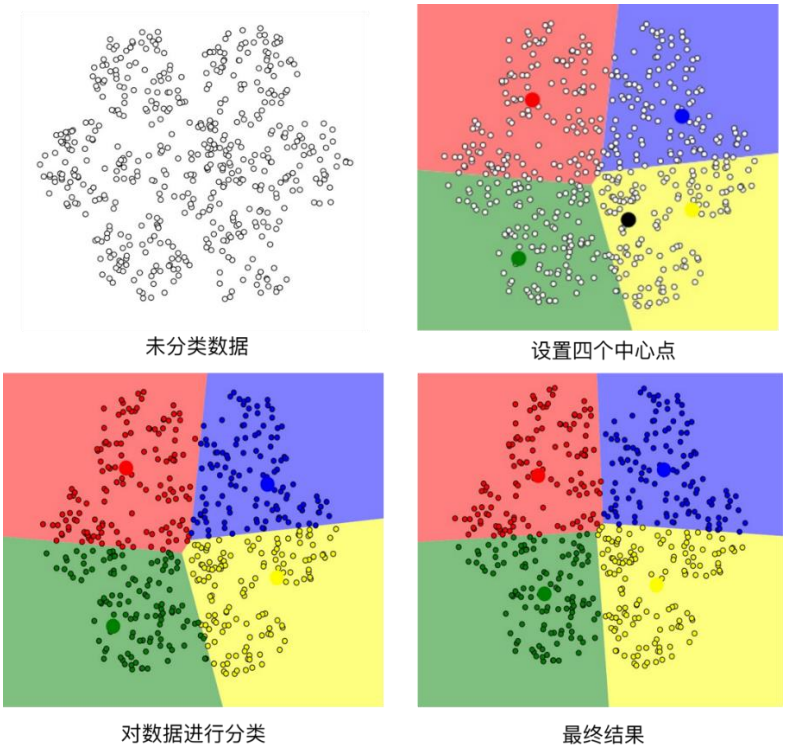

图 1.4　K-Means 算法可视化过程

K-Means 算法存在一定的缺点，在该算法中用于进行分类的 k 值是用户给定的，在没有进行数据处理之前，无法得知 k 值为多少并且当 k 值不同时分类的结果也会出现较大的区别。并且当存在特殊值时会对模型产生比较大的影响。但是其在处理数据量较大的数据集时，可以确保有良好的伸缩性。

1.2 相关工作

随着网络技术的发展，企业和组织中的重要信息区域信息化和数字化，但是目前的安全防御策略主要针对各种外部威胁。在系统被入侵后，往往难以对内部的异常命令或异常访问进行识别和防御。系统被入侵后，攻击者会对权限进行修改、并篡改部分配置信息，以此来获取系统中的重要数据或对系统进行破坏。

彭豪辉等人提出了一种基于用户行为的内部威胁检测方法研究[9],该方案提出了户击键行为、网络行为及文件访问行文的用户模型，并提出了一种基于时间序列的内部威胁检测模型。但是在实际应用中系统可能会存在误报漏报现象，且方案中的模型只是在特定的数据集和环境下表现良好。张锐等人提出了一种基于文件访问行为的内部威胁异常检测模型研究[10]，论文通过威胁研究的现状、特点和分类，提出一种结合个体行为的异常检测模型，可以有效解决个体的异常行为。该模型存在一定的缺点，例如过分依赖文本的准确性且对用户行为的假设太过于简单，且模型不能反应用户最新的行为模型。孙国基等人提出了一种基于文件访问监控的主机异常入侵检测系统[11]，该论文主要对系统中恶意活动表现出的异常文件访问模式进行了分析，并提出了一种基于文件访问你的关系树模型用于建立正常的文件访问行为模型，以此来对异常访问行为进行检测。但是该模型在高频文件访问的环境中需要占用的系统性能较多，且对完全未知的攻击模式的检测能力有限。

王翎霁等人提出了一种基于木马本机文件访问行为检测的方法[12]，该论文分析了木马的植入技术、隐藏技术和自启动技术，并提出了一种通过木马特定行为模式序列来检测木马的方法，该方法值针对木马的特定行为模式进行检测，对于未知或变种木马可能检测能力受限。孙超等人提出了一种基于用户行为和关系的内部风险分析[13]，该论文提出了一种基于日志文件分析的方法，通过比较用户行为和历史行为来对异常行为进行识别，通过分析用户行为的异常性和一致性提出了两种典型的共谋问题好对应的共谋风险分析方法。该方案高度依赖日志文件的完整性和准确性，如果日志文件不全或存在偏差，可能会影响分析结果的准确性。成双等人提出了一种基于 LSTM 网

络的额异常操作行为检测方法[14]，通过使用长短期记忆网络来检测网络中的异常行为，通过分析威胁行为的序列关系来定义行为模式，并挖掘了攻击者与正常用户在行为模式中的差异。但是本方案使用的 LSTM 模型虽然在序列数据处理上有着出色的表现，但是在实时监测的场景中也需要对模型进行优化来满足实时性要求。

1.3 特色描述

1.3.1 规则生成效率较高

本方案通过算法流程和优化后的数据处理策略来实现更高的效率。选择 MiniBatchK-Means[15]作为聚类算法来对大数据集进行处理，与传统的 KMeans 算法相比，MiniBatchK-Means 可以在不牺牲聚类质量的前提下显著提升聚类速度。

在规则生成阶段，自动地利用文件路径信息来生成访问规则，并通过掩码技术来简化规则的表示，通过智能识别和移除冗余规则来避免不必要的计算和存储开销，从而提升模型整体的运行效率。并且代码的模块化设计使得每个步骤可以独立执行，这种解耦合方式可以提高代码的易读性和可维护性，进一步提升处理速度。

1.3.2 自动化处理、提取生成规则

本方案通过集成数据处理、特征提取、聚类分析、规则生成和冗余规则优化等一系列自动化步骤，实现了从原始数据到具体应用规则的转换。首先，利用　自动化地将文本数据（如文件路径）转换为数值特征向量，为聚类分析做准备。接着，采用 MiniBatchK-Means 算法对特征化的数据进行高效聚类，自动将数据划分为预定义数量的簇。然后，根据聚类结果，自动化生成描述文件访问模式的规则，并进一步通过模式匹配和规则简化技术移除冗余，确保规则集的简洁性和实用性。最后，通过计算规则的覆盖率和匹配数，自动化评估聚类效果和规则的准确性，从而实现整个聚类和规则生成过程的闭环优化。这一自动化流程不仅提高了数据处理的效率和准确性，而且通过减少人工干预，降低了出错率，增强了结果的可重复性和可靠性。

1.3.3 应用范围广

本方案实现的算法，可以应用在多个领域中。可以对系统实行异常检测和访问控制，通过生成访问规则，来限制未授权文件的访问增强系统的安全性。并且可以对日志进行聚类分析[16]，以此来识别常见的访问模式或异常事件，同时也能实现日志数据的简化和管理，提高日志的分析效率。在进行系统管理时，也可以用此算法来自动生成系统配置规则，如文件访问权限并以此来简化系统管理。

1.4 应用前景

1.4.1 异常命令和文件访问识别

随着网络攻击手段的多样化和复杂化，越来越多的攻击难以被识别和防控。本方案可以有效的阻止系统中的异常命令和异常文件访问。通过对系统日志和网络流量进行分析，自动化聚类算法可以识别与一直正常行为模式不同的异常行为。本方案可以自动生成并优化访问控制列表[17]，生成用户或进程白名单，通过细化访问权限来减少潜在攻击，以此提高系统的安全性该方案可以持续性监控系统活动，快速识别并响应异常事件，对事件源头进行定位。

综上所述，本方案在系统安全领域有着广泛的应用前景，能为系统提供一种高效、安全且自动化的方法来增强其安全防御机制，降低安全风险，提高对威胁行为[18]的响应能力。

1.4.2 良好的可扩展性

本方案中的算法主要针对具有明确结构的文本数据，但其聚类算法和特征提取方法可以进行调整以处理其他类型的数据，例如数值数据或混合类型数据。随着聚类算法的发展，本方案也可以集成新的聚类算法来实行更加复杂的数据分布。并且可以对方案的特征提取功能进行拓展，来提升聚类结果的准确性和可靠性。在规则数量增加时，本方案也可以进一步有优化规则生成和简化过程，来减少冗余并提高规则的覆盖率。

第二章　作品设计与实现

2.1 系统方案设计

针对系统中的进程和文件访问关系过多的问题，本章提出了一种基于无监督学习的白名单规则生成方法——MKWAY。本方案通过分析正常操作期间的系统进程和文件访问数据，识别出常见的安全访问模式。通过聚类算法进行提取，以减少需要管理的规则数量，同时对生成的规则进行简化处理。本方案旨在降低白名单的复杂性，使其更易于管理和维护，同时提高系统监控时发现异常的速度和准确性。

该模型要实现的功能是自动学习系统中的进程和文件访问关系，输出白名单规则。在保证规则粒度适当的情况下，保证算法学习效率，以提高系统监控时快速发现异常的速度。

该模型的主要流程如图 2.1 所示，在进行数据的收集和预处理之后，通过聚类算法进行特征的提取和分析，最终对生成的规则进行简化处理后输出结果。

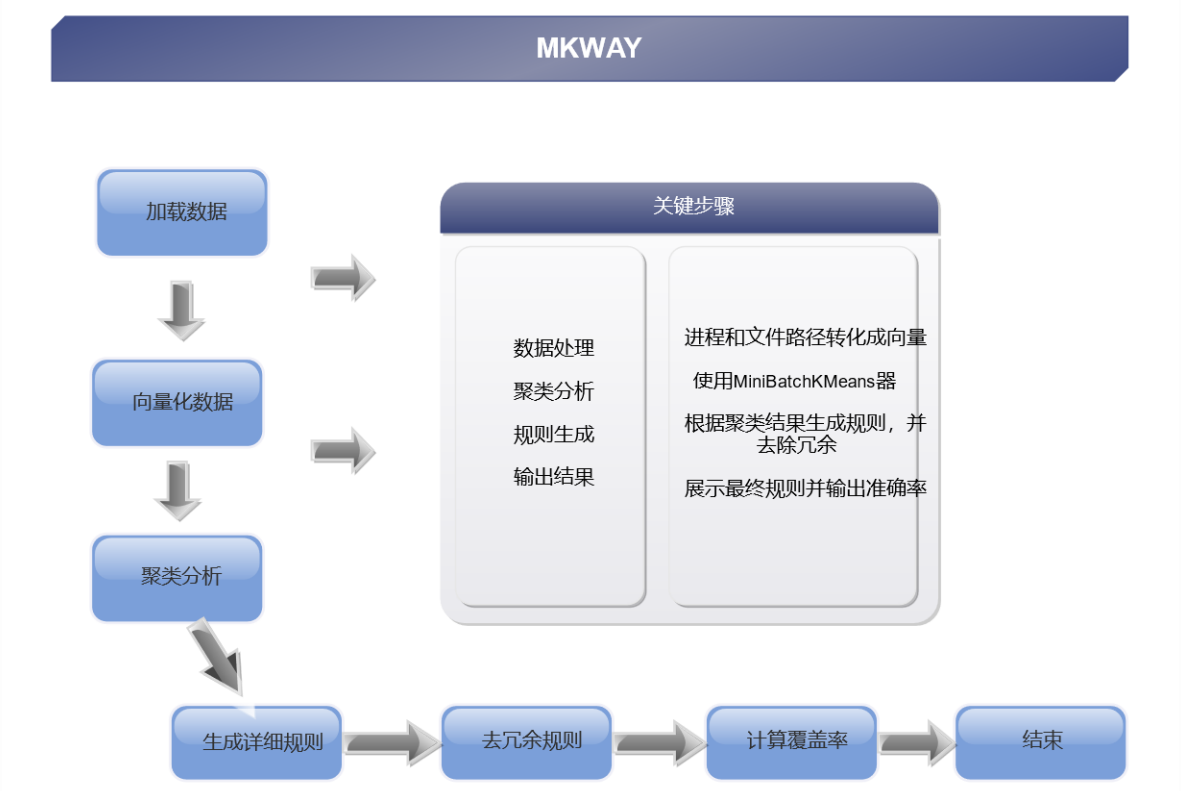

图 2.1　MKWAY 算法核心流程图

该模型的核心功能如下：

(1) 行为模式识别

算法通过分析正常操作期间的系统进程和文件访问数据，识别出常见的安全访问模式。这是通过无监督学习方法实现的，它不需要预先标记的数据来识别正常行为模式。

(2) 聚类分析

利用聚类算法将具有相似访问特征的进程和文件访问行为分为一组。聚类有助于发现访问模式的内在结构，并减少需要管理的规则数量。

(3) 白名单规则生成和简化

根据聚类结果，算法自动生成白名单规则。这些规则定义了哪些进程可以访问哪些文件，以及在什么条件下可以进行访问，从而帮助系统仅允许符合预定义模式的行为。通过合并相似规则和移除冗余来简化规则集。这有助于降低白名单的复杂性，使其更易于管理和维护。

(4) 生成报告和日志记录

算法能够生成详细的报告和日志，帮助安全团队了解系统访问模式的变化，并为安全审计提供必要的记录。

其整体方案思路如图 2.2 所示：

图 2.2 核心功能整体思路图

本解决方案的设计遵循以下四个方面：

(1) 根据系统检测需求，明确目标是生成一套涵盖正常进程文件访问关系的白名

单规则。目标是减少规则数量，并确保规则的准确性，以快速过滤异常行为。

(2) 通过无监督学习算法，如聚类分析，对不同进程与文件访问的关系进行分析。通过比较各类访问行为，选择合适的掩码模式（例如通配符匹配），从而减少白名单规则的粒度。

(3) 生成测试集并对规则进行详细评估，确保规则既能高效覆盖正常行为，也能及时发现异常。评估结果将包括规则数量、检测准确度和生成规则的效率。

(4) 提供详细的交付文档，包括规则生成算法的源代码、运行说明以及测试报告。同时，确保交付的白名单规则具有较高的适应性，可以适应不同系统环境中的进程文件访问行为。

基于 scikit-learn 库中的 CountVectorizer 类的向量转化方案实现如下：1）创建 CountVectorizer 实例，使用'/'分割文本，将每个路径部分（如 usr、bin、bash）视为一个独立的词汇，为了防止向量的特征数过多问题，将使用" max_features "参数将向量的最大特征数设为 50，提高算法的合理性；2）向量化进程和文件路径，fit_transform 方法在 data['Process']和 data['File']上被调用，该方法学习词汇表（即所有出现的路径部分），并将文本数据转换为稀疏矩阵，最后将稀疏矩阵转换为一个常规的 numpy 数组；3）最后，函数返回两个数组：process_vec 和 file_vec，分别代表进程路径和文件路径的数值向量表示。

其向量转化方案过程如图 2.3 所示：

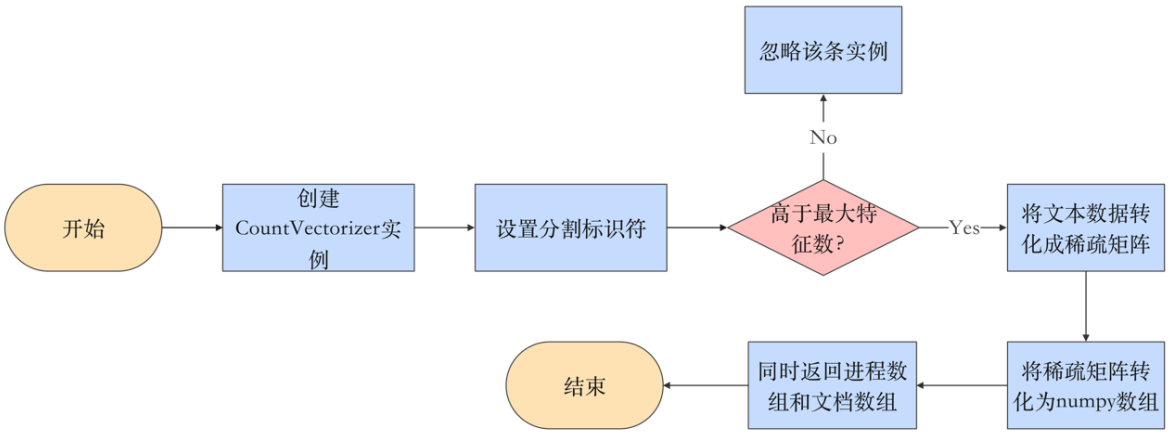

图 2.3　融合 CountVectorizer 类的向量转化方案

总的来说，本架构采用了以 MiniBatchK-Means 聚类技术为主的无监督学习方法，来自动学习系统中的进程和文件访问关系的白名单规则。使用 pandas 库读取 csv 格式

的文件，使用 sklearn 库的 CountVectorizer 类进行向量的转化，再通过聚类算法的对比和优化确定最终的聚类算法，聚类结束后生成的白名单规则经过自动化处理后，通过赛题方给的测试集，以及在 ubuntu 系统生成的多个测试集进行准确度的测试，并对规则进行详细评估，确保规则既能高效覆盖正常行为，也能及时发现异常。

2.2 自动化白名单规则生成

本工具通过无监督学习方法自动从系统进程和文件访问数据中提取特征，自动化生成白名单规则。通过对海量日志数据进行深入分析，本方案能够高效识别正常行为模式，并将其转化为简洁、准确的白名单规则。该节涵盖了数据预处理、特征提取、聚类算法的应用、规则的生成与简化，以及覆盖率的计算等多个关键步骤，现了从原始日志数据到具体白名单规则的高效转换。

2.2.1 基于 CountVectorizer 的数据预处理

(1) 数据预处理

数据预处理是自然语言处理（NLP）和文本挖掘中的一个重要步骤，它涉及将原始文本数据转换成适合机器学习模型处理的格式，如图 2.4 所示：

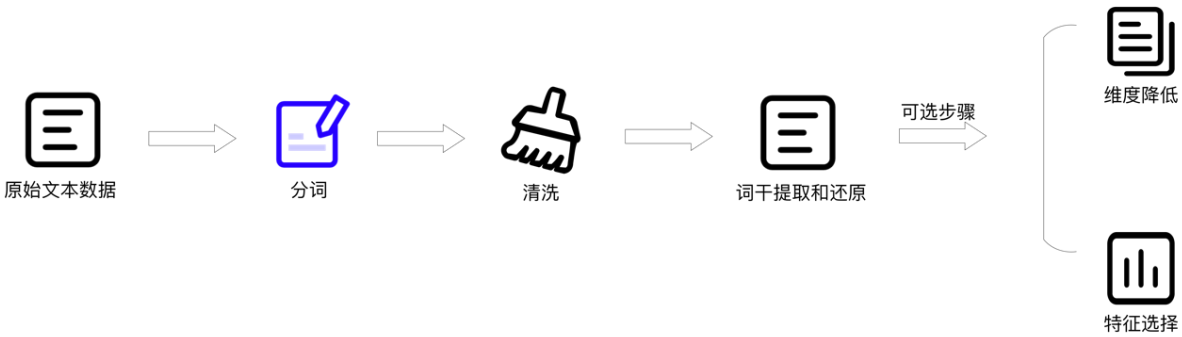

图 2.4 nlp 数据预处理流程图

在 MKWAY 算法中，从文件中加载原始数据。通常是一个 CSV 或日志文件，其中包含进程和文件访问记录，将文本数据（进程和文件路径）转换为数值特征向量之前，需要先进行分词，结合赛题需求，本算法采用'/'作为分隔符，目的是为了将进程和文件路径分割成单独的路径字符串，以便接下来的文本向量化使用。

(2) 文本向量化

词袋模型是自然语言处理中的一种经典技术，它将文本转换为单词出现次数的向量。对于路径分割的情况，每个路径元素可以视为一个“单词”。

如果有一个文本集合$D = \{d_1, d_2, \ldots, d_n\}$，其中$d_i$是一个文本文档，词袋模型将

每个文档转化为一个向量v_i向量的每个维度对应词汇表γ中的一个单词ω_j，并且$v_i[j]$表示单词ω_j中在文档d_i中的出现次数。

在本题中，$V_{(d_i)} = [f(\omega_1, d_i), f(\omega_2, d_i), \ldots, f(\omega_k, d_i)]$，$k \leq 50$为 MKWAY 算法预设的最大特征数。

Process_vec 和 file_vec 分别是两个二维数组（矩阵），每个样本对应一行，每个特征（词汇）对应一列，矩阵中的值为词汇的词频，用矩阵的形式表示：

$$\mathbf{M} = \begin{bmatrix} f(w_1, d_1) & f(w_2, d_1) & \ldots & f(w_k, d_1) \\ f(w_1, d_2) & f(w_2, d_2) & \ldots & f(w_k, d_2) \\ \vdots & \vdots & \ddots & \vdots \\ f(w_1, d_n) & f(w_2, d_n) & \ldots & f(w_k, d_n) \end{bmatrix}$$

例如，假如有三个文本文档，这些文档可以是文件路径或进程路径，具体通过词袋模型处理文本数据的步骤如下：

1.d_1 = "/usr/bin/bash"，d_2 = "/usr/bin/bash/ls"，d_3 = "/bin/bash"，首先将这些文档转化为特征向量。

2.构造一个词汇表，在该词汇表包含所有文档中出现的唯一的路径元素。γ = {"usr", "bin", "bash", "ls"}。本模型中假设注意 max_features=50，但在这个例子中，只有 5 个唯一的路径元素，满足限制条件；

3.现在，将每个文档转换为一个特征向量。函数 vectorize_data 中的 CountVectorizer 会计算每个路径元素在每个文档中的出现次数。每个文档转换后的特

表 2.1　文档转换后的向量特征

特征向量	代表含义
v_1 = [1, 1, 1, 0]	对应于 d_1="/usr/bin/bash"，"usr", "bin", "bash" 各出现 1 次，"ls" 没有出现
v_2 = [1, 1, 1, 1]	对应于 d_2="/usr/bin/bash/ls"，所有元素都出现了 1 次
v_3 = [0, 1, 1, 0]	对应于 d_3="/bin/bash"，"bin", "bash" 各出现 1 次，"usr" 和 "ls" 没有出现

4.在实际的 vectorize_data 函数中，process_vec 和 file_vec 将是二维数组，其中

每一行代表一个文档，每一列代表词汇表中的一个元素。如果将上述三个文档的特征向量放入一个矩阵中，如下所示：

$$M_{process} = \begin{bmatrix} [1,1,1,0] \\ [1,1,1,1] \\ [0,1,1,0] \end{bmatrix}$$

在这个矩阵中，每一行都是一个文档的词袋模型表示，每一列对应于词汇表中的一个路径元素，矩阵中的值表示该元素在文档中出现的频率。这个矩阵可以用于后续的机器学习任务，如聚类分析。

(3) 矩阵转换

在文本向量化过程中，通常会生成稀疏矩阵，因为大多数文本数据中的词汇在大多数文档中并不会出现。例如，在处理大量文档时，可能只有一小部分独特的词汇会在任何一个给定的文档中出现。稀疏矩阵是一种高效的存储方式，它仅存储存在的值（非零项），从而节省内存和计算资源，以对角存储矩阵为例，如图 2.5 所示：

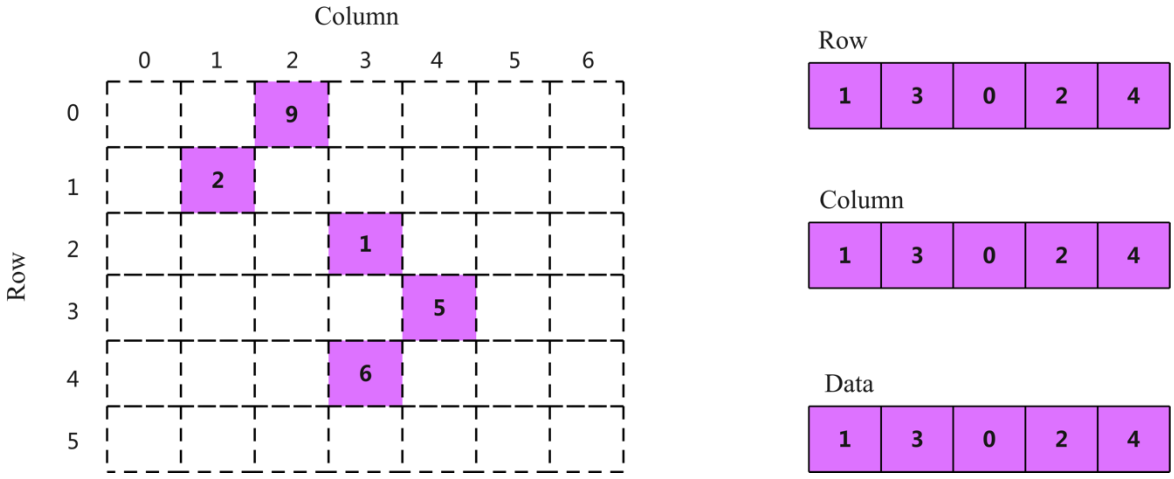

图 2.5 稀疏矩阵存储示意图

toarray()方法接受当前的稀疏矩阵并构建一个完整的 NumPy 数组。目的是将文件路径文本转换为数值特征向量，这些向量可以用于后续的机器学习任务，如聚类、分类等。通过这种方式，文本数据被转换成了可以进行数学运算的格式，从而使机器学习算法能够应用于文本数据。

融合了 CountVectorizer 类的向量转化方案是数据预处理的关键步骤，为后续的聚类分析和白名单规则生成奠定了基础。通过使用’/’为分隔符，分割进程和文件路径，

形成单独的路径字符串，利用 CountVectorizer 将这些文本数据转换为数值特征向量。通过设置最大特征数为 50，有效控制了特征数量，这一过程不仅提高了算法的合理性，避免了特征过多带来的问题。此外，稀疏矩阵的转换，将其转换为常规的 NumPy 数组，以便于后续的聚类处理。

2.2.2 基于路径分词的聚类特征提取方案

在将文本数据转换为数值特征向量后，需要将向量进行处理并用于聚类分析。为了识别出数据中的内在结构，将具有相似特征的数据点归类到一起，从而揭示出数据的分布特征和潜在模式。本方案采用了 K-Means 下的高效变种聚类算法 MiniBatchK-Means 来实现这一目标。

KMeans 聚类算法常用于探索性数据分析，当数据没有明确的标签或者类别未知时，聚类可以帮助发现数据的内在结构。分类算法适用于预测性任务，当需要根据已知数据对新数据进行分类时，分类算法可以提供预测模型。两者的分类如表 2.1 所示，故本方案采用无监督的聚类算法。

表 2.2　聚类分类算法对比

	聚类	分类
核心思想	把数据分成多个组，探索每个组的数据是否有联系	从分组的数据中去学习把新数据放到已经分好的组中去
学习类型	无监督，无需标签进行训练	有监督，需要标签进行训练
典型算法	K-Means, DBSCAN，层次聚类，光谱聚类	决策树，贝叶斯，逻辑回归
算法输出	聚类结果是不确定的，不一定总是能够反映数据的真实分类同样的聚类	分类结果是确定的分类的优劣是客观的不是根据业务或算法需求决定

K-Means 算法是无监督的聚类算法，也可以叫 K 均值聚类，随机选择 K 个簇中心点后，构成样本集样本被分配到离其最近的中心点，K-Means 常规流程如图 2.6 所示：

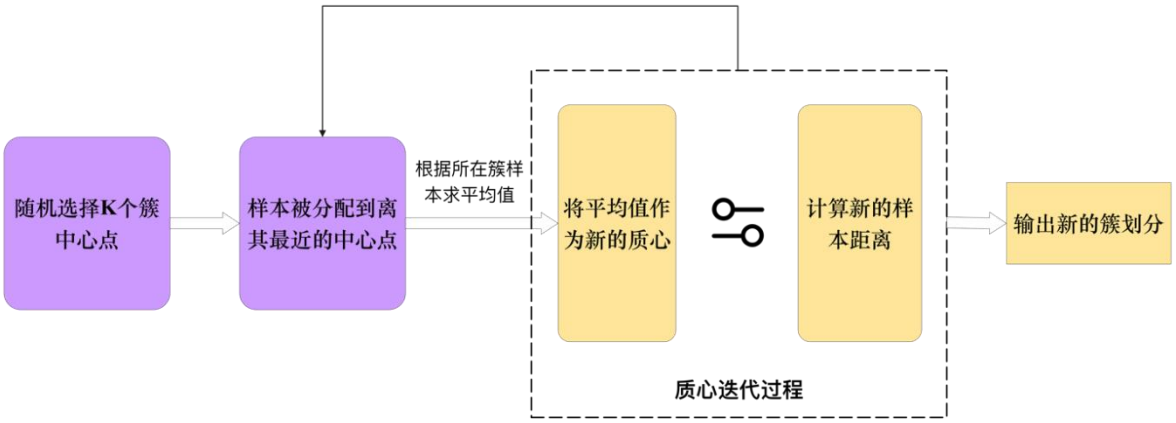

图 2.6　K-Means 常规流程图

假设样本集$D = \{x_1, x_2, .. x_m\}$，簇分为$C = \{C_1, C_2, .. C_k\}$，目标函数为最小化平方误差。

$$E = \sum_{i=1}^{k} \sum_{x \in C_i} \left\| x - \mu_i \right\|_2^2 \tag{2-1}$$

其中，μ_i为簇C_i的均值向量，即质心，表达式为：

$$\mu_i = \frac{1}{\left| C_i \right|} \sum_{x \in C_i} x \tag{2-2}$$

分别计算每个样本x_i和各个质心向量μ_i的距离d_{ij}

$$d_{ij} = \left| x_i - \mu_i \right|_2^2 \tag{2-3}$$

根据d_{ij}计算新的质心，输出改变后的簇划分$C = \{C_1, C_2, .. C_k\}$。

在传统的 K-Means 算法中，要计算所有的样本点到所有的质心的距离。如果样本量非常大，如特征达到 10 万以上，特征有 100 以上，此时用传统的 K-Means 算法非常耗时，在大数据时代，这样的场景越来越多。通过操作系统的访问关系生成白名单规则的过程中，由于访问关系复杂导致规则数量至少是数万级，故本方案采用 K-Means 的变种 MiniBatchK-Means，大大提高了算法的收敛速度。在 MiniBatchK-Means 中，本方案选择一个合适的批样本大小 batch size，为了增加算法的准确性，在生成白名单规则的过程中，通常会调整参数进行多次 MiniBatchK-Means 算法的结果对比，通过生成不同的随机采样集来得到聚类簇，选择其中最优的聚类簇，本聚类方法包括以下基本步骤，详细流程如图 2.7 所示。

(1) 数据加载

使用矢量化后的进程和文件数据作为输入，加载进程和文件的路径信息。

(2) 数据向量化

使用 CountVectorizer 对将进程和文件数据转换成数值特征向量，这一步骤包括分词处理，将文本数据转换为可以进行数学运算的格式。

(3) 合并特征向量

聚类白名单规则需要同时考虑进程和文件的特征，使用 numpy.hstack() 将进程向量 (process_vec) 和文件向量 (file_vec) 水平堆叠合并成一个特征矩阵 (combined_vec)，使得两个特征向量并排放置，形成一个新的二维数组，其中每一行代表一个数据点，列数是两个原始向量列数的总和。

(4) 参数优化

经过多轮的结果精确度的横向对比，以及对白名单规则的有效性分析，将簇数 (n_clusters)设置为 10，意味着算法会将数据分为 10 个簇；随机状态 (random_state) 设置为 42，确保每次运行算法时初始化的质心相同，保证结果的可复现性。

(5) 执行聚类

通过迭代优化过程，逐步调整簇质心，以最好地代表所属簇内的数据点，并且为每个数据点分配一个簇标签，表示它属于哪个簇。

(6) 返回簇标签

使用变量 labels 存储包含所有数据点簇标签的数组，为后续规则的生成提供基础数据。

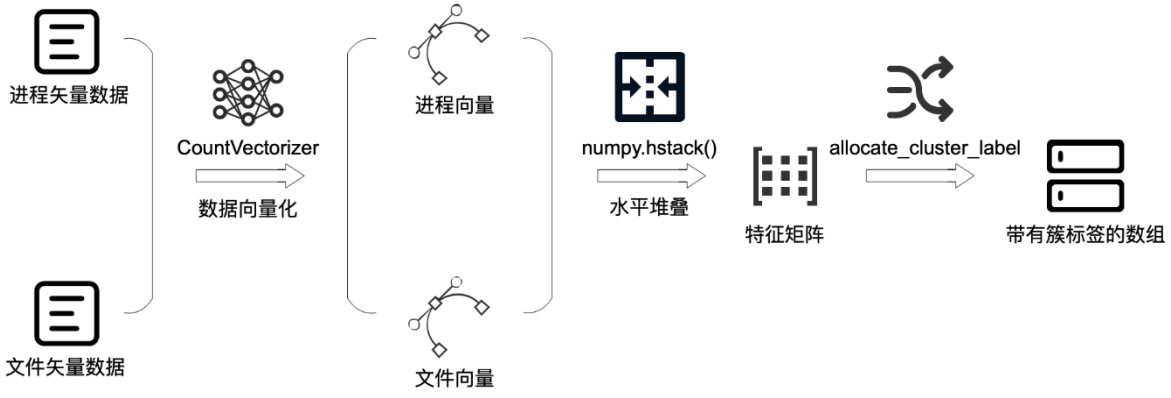

图 2.7 本方案聚类算法流程图

通过以上步骤，MiniBatchK-Means 聚类算法能够有效地处理大规模数据集，将系统进程和文件访问数据分组，通过识别和处理矢量化的数据，将输入的进程和文件数据进行簇的分类，最终生成完整的二维数组，为后续白名单规则的生成提供了原始

数据基础。这种方法不仅提高了数据处理的效率，还通过无监督学习揭示了数据的内在结构，为进一步的白名单策略生成提供了支持。

2.2.3 规则生成及去除冗余

本方案通过对聚类算法结果进行转化来生成一组具体的访问控制规则，这些规则用于描述每个聚类簇内进程和文件的典型访问模式，规则生成及简化规则的流程图图 2.8 所示。使用 set（labels）来获取数据集中的位移的聚类标签，对于聚类算法生成的聚类标签使用布尔索引来从原始数据中提取出该簇的子集。布尔索引依赖于逻辑运算符来创建条件表达式，使用表达式对数据集中的元素进行评估，并返回一个布尔值，布尔索引的基本流程如图 2.9 所示。

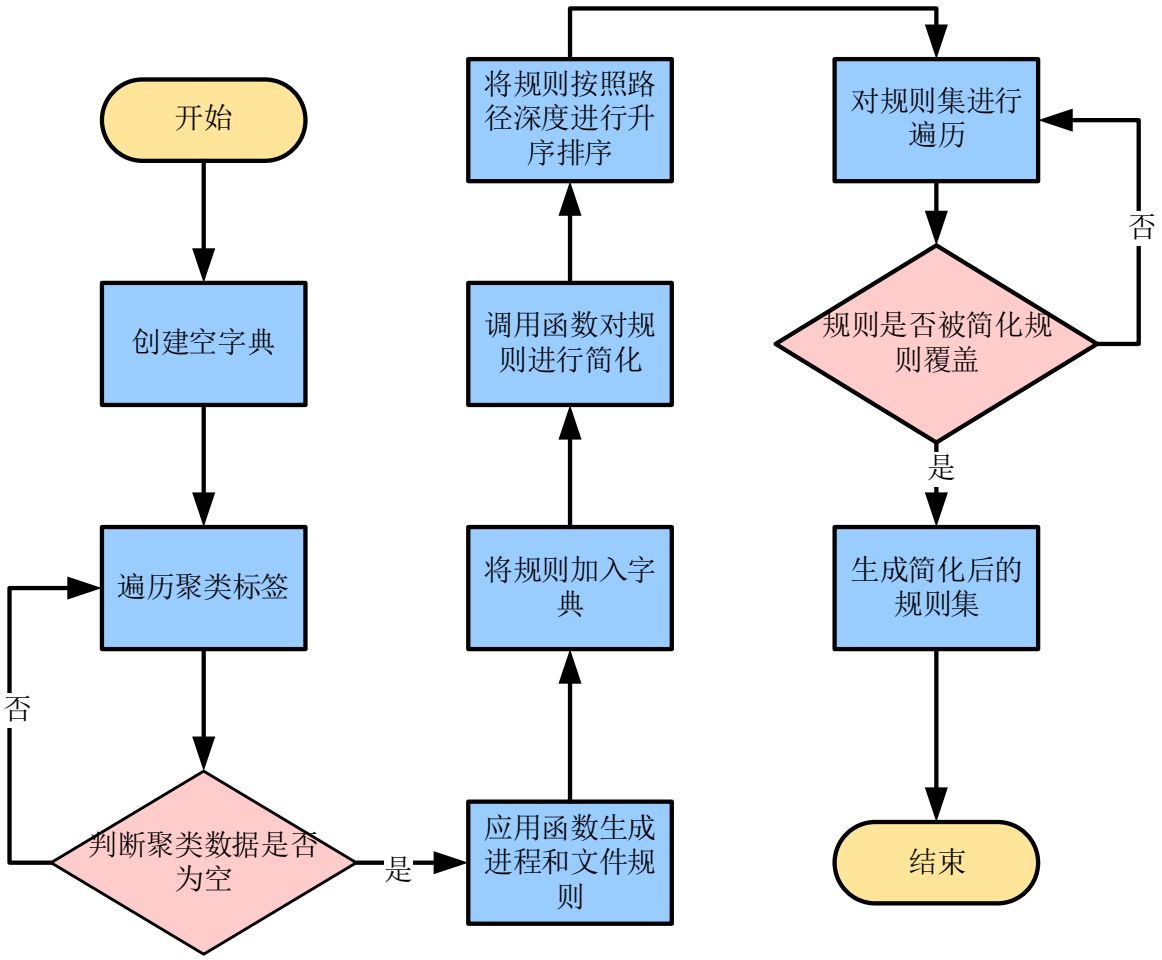

图 2.8　规则生成及简化规则的流程图

对于非空的簇，本模型使用 pandas 中的 apply 方法结合 lamda 函数进行处理，apply 函数可以对 DataFrame 中的列或行中的元素应用一个自定义的函数，在 axis 参数设置为默认值时，该函数会对 DataFrame 中的每一列应用指定函数，处理结果为与

原列相同长度的序列。通过 os.path 方法来获取每个进程或文件路径的目录部分，并通过添加掩码来降低规则数量，在生成规则后使用 unique 方法来确保生成的规则集合中不包含重复的规则，确保规则的唯一性。

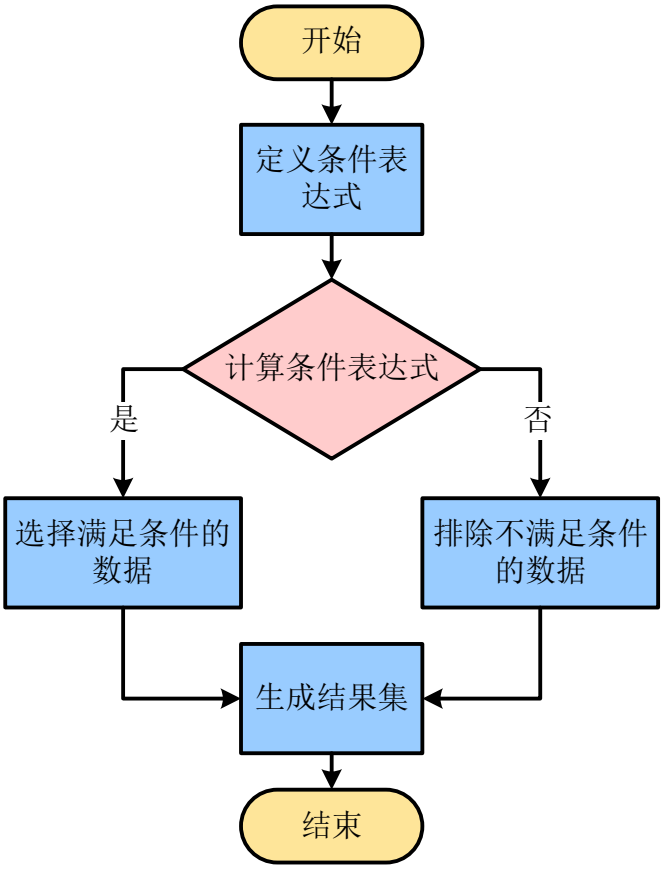

图 2.9 布尔索引流程图

在生成规则即后，需要去对其进行简化并去除冗余部分，其基本规则如下：

(1) 数据加载

遍历未简化规则集中的每个条目，其中每个条目都包含一组进程规则和文件规则。

(2) 规则排序

使用 sorted 函数将规则按照路径深度进行升序排序，可以确保在检查和处理规则时首先考虑更加具体的路径规则，使得算法可以更容易地识别和去除被更广泛规则覆盖的冗余规则，可以有效提升该模型的运算效率。因为更具体的规则，例如"/usr/local/bin/script/*"通常可以指代单一的文件或目录。

(3) 规则检测

对于每个规则，使用 any()函数结合生成器表达式来检查是否存在对应的简化规则。其中使用 fnmatch() 函数进行模式匹配，如果当前规则没有被任何已简化规则集

中的规则覆盖，则将该规则添加到集合中。

(4) 集合匹配

将每个唯一的进程规则与对应的简化后的文件规则集合配对，添加到列表中。

(5) 输出最终规则集

在通过规则集简化以及去除冗余后，输出最终规则集。图 2.10 为最终输出的部分简化规则集。

```
Process rule: /usr/libexec/*, File rule: /var/opt/kaspersky/kesl/private/*
Process rule: /usr/libexec/*, File rule: /var/lib/*
Process rule: /usr/libexec/*, File rule: 24014/*
Process rule: /usr/libexec/*, File rule: /var/www/html/*
Process rule: /usr/libexec/*, File rule: /var/log/*
Process rule: /usr/libexec/*, File rule: /var/opt/kaspersky/kesl/common/updates/data/data/*
Process rule: /usr/libexec/*, File rule: /run/log/journal/*
Process rule: /usr/sbin/*, File rule: ./mysql/*
Process rule: /usr/sbin/*, File rule: /etc/*
Process rule: /usr/sbin/*, File rule: /usr/local/hostguard/lib/*
Process rule: /usr/sbin/*, File rule: /var/tmp/*
Process rule: /usr/sbin/*, File rule: /usr/lib64/security/*
Process rule: /usr/sbin/*, File rule: /tmp/*
Process rule: /usr/sbin/*, File rule: /usr/lib64/elfutils/*
Process rule: /usr/sbin/*, File rule: /usr/local/hostguard/module/*
Process rule: /usr/sbin/*, File rule: /lib64/*
Process rule: /usr/sbin/*, File rule: /usr/share/zoneinfo/*
Process rule: /usr/sbin/*, File rule: /proc/irq/*
Process rule: /usr/sbin/*, File rule: ./yyb_tmp/*
Process rule: /usr/sbin/*, File rule: ./yyb/*
Process rule: /usr/sbin/*, File rule: /var/opt/kaspersky/kesl/private/*
Process rule: /usr/sbin/*, File rule: /var/lib/*
Process rule: /usr/sbin/*, File rule: 24014/*
Process rule: /usr/sbin/*, File rule: /var/www/html/*
Process rule: /usr/sbin/*, File rule: /var/log/*
Process rule: /usr/sbin/*, File rule: /var/opt/kaspersky/kesl/common/updates/data/data/*
Process rule: /usr/sbin/*, File rule: /run/log/journal/*
Process rule: /var/opt/kaspersky/kesl/10.1.1.6421_1566542505/opt/kaspersky/kesl/libexec/*, File
rule: ./mysql/*
Process rule: /var/opt/kaspersky/kesl/10.1.1.6421_1566542505/opt/kaspersky/kesl/libexec/*, File
```

图 2.10　部分简化规则集

在进行规则简化的步骤中，核心功能为使用 fnmatch()函数进行模式匹配，fnmatcht 提供了基于 Unix shell 的通配符匹配功能，可以在定义规则时使用掩码符号*这种通配符，在本方案中 fnmatch.fnmatch()函数用于检查一个具体文件路径是否符合特定模式，即通过将规则中的目录路径与通配符结合来实现。例如使用"/usr/local/bin/*" 匹配"/usr/local/bin"下的任何文件或子目录。在生成详细的访问控制规则后，fnmatch 用于识别和删除那些可以被带有掩码简化规则进行覆盖的规则，当一个规则能被已经简化的规则集中的任何规则通过 fnmatch 匹配成功时，认为它是冗余的，将其从规则集中去除。

其去除冗余的流程如图 2.11 所示：

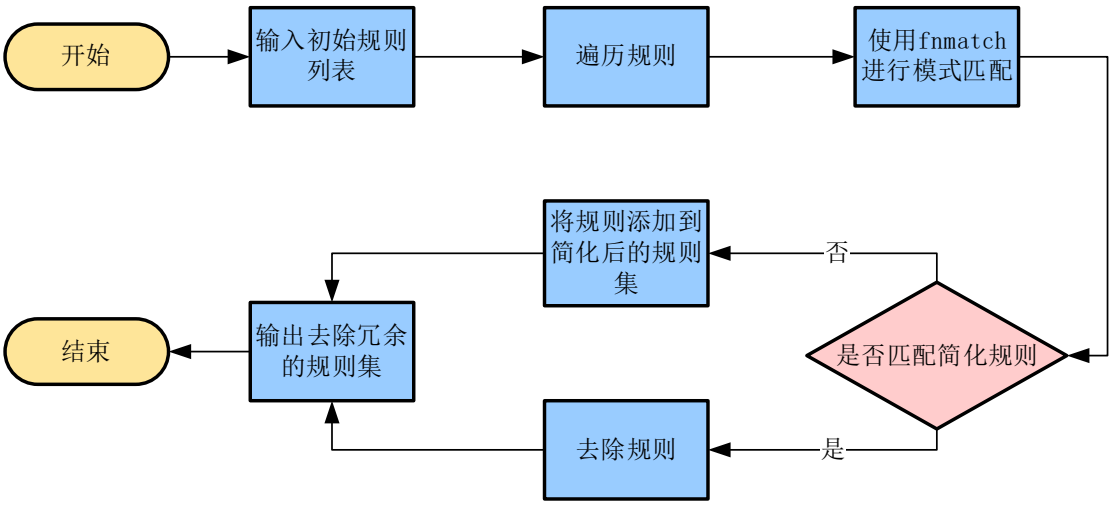

图 2.11 fnmatch 匹配流程图

2.2.4 基于 fnmatch 的覆盖率计算

本方案提出了一种计算白名单规则集覆盖率的方法，旨在评估规则集对已知数据集的适应性和全面性。通过对数据集中每一条记录进行规则匹配测试，并统计匹配成功的记录数，来计算规则集的覆盖率，具体流程如下：

(1) 初始化计数器

将用来记录匹配规则的变量 matches 初始化为 0，用于记录规则成功匹配数据条目的数量。

(2) 遍历数据集和规则集

使用 iterrows() 遍历数据集 data 中的每一行。iterrows() 为每一行返回索引和行数据。在遍历每个数据行时，都需要遍历整个规则集 rules 每个 process_rule 和对应的 file_rules 集合代表一组白名单规则。

(3) 匹配进程和文件规则

与 2.1.4 中简化规则的遍历方法相同，使用 fnmatch 分别判断进程规则 process_rule 和文件规则 file_fule 是否都分别包含在对应的规则集里。fnmatch.fnmatch()函数是 Python 标准库 fnmatch 模块中的一个函数，支持 Unix shell 风格的通配符，其中，'*'通配符与本题要求相同，代表任意数量的字符，例如，规则 "/usr/bin/*" 会匹配任何在 "/usr/bin/" 目录下的文件或目录，词函数是用于覆盖率和错误率判断的最优解。在成功匹配规则时，直接 break 终止本次循环，以节省资源，提高算法效率。

(4) 返回总覆盖率

在成功匹配规则时，通过变量 matches 来记录匹配成功的次数。最后，通过匹配总数 matches / len(data)来记录覆盖率，并返回匹配总数和覆盖率，总体流程如图 2.12 所示：

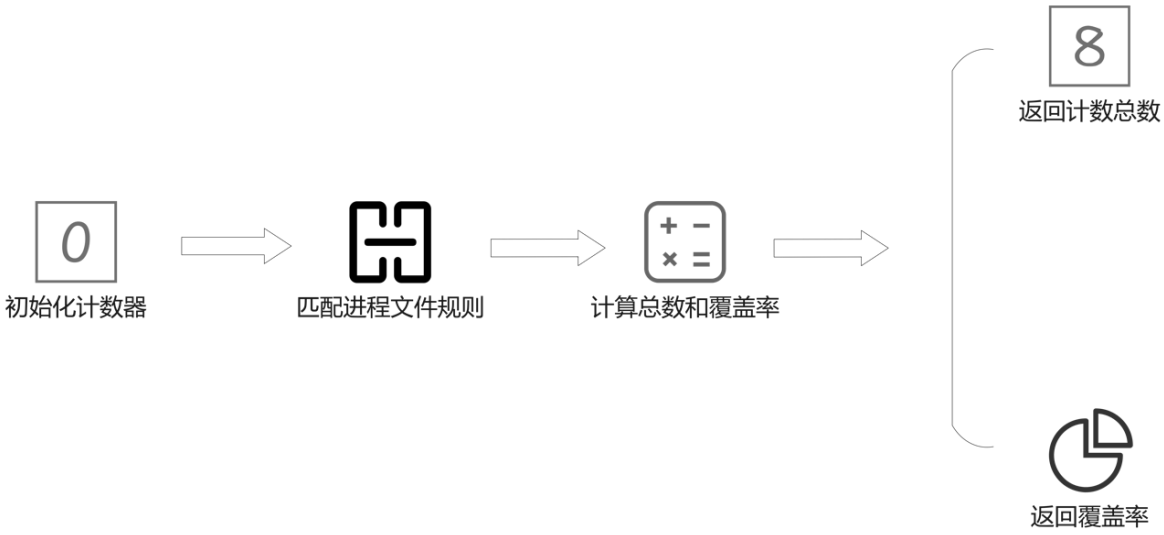

图 2.12　本方案聚类算法流程图

通过计算匹配数与数据总数的比值计算得到了白名单规则集的覆盖率，这一指标直观地反映了规则集的全面性和实用性。通过这一流程，能够对白名单规则集的有效性进行定量分析，并据此进行相应的调整和优化，以提高其在实际应用中的效能。

2.2.5 聚类结果的可视化分析

使用 matplotlib 库进行聚类结果的可视化绘制，目的是将聚类分析的结果以图形化的方式展现。通过可视化的方式帮助用户直观理解聚类结果，评估聚类算法的性能，如聚类的紧密性和分离度。

使用 matplotlib.pyplot 创建一个新的图形窗口，使用 scatter 函数绘制第一组数据点（进程向量）。数据点的位置由 data['Process_vec']中的向量坐标决定，颜色由对应的聚类标签（data['Cluster']）决定。类似地，使用 scatter 函数绘制第二组数据点（文件向量）和第三组数据点（聚类中心）。聚类中心由 centers 数组提供坐标，通常使用红色和较大的点（本方案中为 200 个单位大小）以及半透明（alpha=0.5）来突出显示，使得它们在图中明显区别于其他数据点。最后调用 legend()函数添加图例，说明不同颜色和标记代表的数据点类型（如进程、文件以及聚类中心），并使用 show()函数将最终的图形展示出来，使得用户可以看到聚类的结果和数据点的分布情况。具体流程如图 2.13 所示：

图 2.13　聚类结果可视化分析流程

2.3 功能与指标

2.3.1 功能指标

(1) **文本向量化：**文本数据（如文件路径和进程名称）转换为数值向量。这一步骤涉及分词处理，其中文本数据被分割成单独的元素或词汇，每个元素对应向量中的一个维度。

(2) **特征矩阵构建：**向量化后的数据整合成一个特征矩阵，其中每一行代表一个数据实例（如一个文件访问事件），每一列代表一个特征（如路径中的一个目录名）。

(3) **自动化白名单规则生成：**工具采用无监督学习算法，自动从系统进程和文件访问数据中提取特征，生成白名单规则。

(4) **报告生成功能：**工具在进行白名单规则生成的任务中，最终生成 output.txt 文件，记录了格式化的详细白名单规则，其中包括进程路径和文件路径

(5) **聚类结果可视化：**工具可根据聚类结果生成可视化图，可以直观地看到不同数据点如何被分成不同的组（聚类），以及每组的中心位置。

2.3.性能指标

(1) **算法性能：**本工具使用 MiniBatchK-Means 聚类算法，其可以快速处理大规模数据集且能有效识别数据中的异常行为。其次，规则生成算法能准确从聚类标签 labels 中提炼出访问控制规则，确保规则的精确性和实用性。

(2) 算法准确性：本工具的准确性集中体现在聚类算法和规则生成算法上，MiniBatchK-Means 将数据集划分为多个簇，使同一簇内样本尽量相似，该算法能生成

高覆盖率且准确性较高的规则集。规则生成算法基于聚类结果生成访问模式的规则，其规则可以精准的组织系统中的异常行为。

(3) **算法适应性：**针对不同的系统环境以及不同的数据集，本工具能在较短时间内得到较高的规则覆盖率，既本工具的适应性能确保其在不同的系统环境下均能得到较好的白名单规则。

第三章　作品测试与分析

3.1 测试方案

本算法主要有两大功能以及一个辅助功能，分别是：1）自动学习系统中的进程和文件访问关系白名单规则；2）通过聚类方法降低白名单规则数量；3）生成聚类后的白名单规则。其中第三个功能作为前两个功能的辅助化展示，在算法开发过程中已经得到很好的测试与使用。**本部分测试方案主要针对自动学习系统中的进程和文件访问关系白名单规则以及通过聚类方法降低白名单规则数量这两大功能进行功能测试，再针对系统的适应性进行性能测试。**具体测试方案如下：

3.1.1 白名单规则聚类测试

测试目标：验证通过本算法生成的白名单规则对原始进程与文件访问关系数据集的覆盖率，确保在减少白名单规则数量的同时不影响覆盖精度。

测试过程：使用 ausearch 工具生成的系统的进程和文件访问关系数据集和附件给定的数据集进行白名单规则的自动学习以及聚类。在聚类过程中引入掩码机制来实现白名单规则数量的降低。聚类完成后，将生成的白名单规则输出到一个 txt 文件。检查输出文件的格式是否正确，规则是否合理，聚类是否有效减少规则数量。

3.1.2 白名单规则覆盖率测试

测试目标：验证通过本算法生成的白名单规则对原始进程与文件访问关系数据集的覆盖率，确保在减少白名单规则数量的同时不影响覆盖精度。

测试过程：使用原始的进程与文件访问关系数据集作为基准，对比生成的白名单规则与原始数据集，计算规则的覆盖率。覆盖率计算公式为：覆盖率=(白名单规则覆盖的访问数/总访问数)×100%，将生成的白名单规则覆盖率输出。

3.1.3 性能测试

测试目标：验证本算法在不同规模的数据集下的性能表现，确保算法在大规模数据集上仍能快速实现白名单规则数量的降低。

测试过程：使用不同规模的数据集，分别包含 5000 条、10000 条、20000 条进程与文件访问记录，对每个数据集执行自动学习和规则数量降低过程，测量并记录每个数据集的运行时间，评估算法在不同规模数据集上的执行效率。

3.2 测试环境及设备

(1) 测试设备参数如表 3.1 所示：

表 3.1 测试设备参数

设备名称	DESKTOP-PKU13LU
处理器	Intel(R) 12th Gen Core i7-12700H-十六核
机带 RAM	16.0GB
硬件 ID	COMPUTER\{6DAAB560-C163-5555-8BD3-177780ADE8D2}
系统类型	Ubuntu 64 位操作系统, 基于 x64 的处理器

(2) 软件工具如表 3.2 所示：

表 3.2 软件工具

软件名称	实现功能
auditctl	记录系统进程与文件访问的审计日志。
ausearch	提取和解析审计日志的工具。
Python3.8	用于运行 getexefiles.py 脚本和实现 MKWAY 无监督算法。

(3) 数据集：

表 3.3 数据集

数据集名称	数据集来源
fileaccessdata	赛事方提供的数据集
test_data	通过审计工具 auditctl 从 ubuntu 系统中收集日志，并通过 ausearch -i 生成解析后的进程与文件访问数据记录，最终生成的自定义数据集。

3.3 功能测试结果

3.3.1 测试数据准备

为了验证本算法的普适性，需在多个数据集上进行测试，可以利用审计工具auditctl记录的审计日志来生成系统中的进程和文件访问关系数据集，以下是使用Linux ausearch以及给定的getexefiles.py脚本构建数据集的分析过程：

(1) 生成审计数据：

使用auditctl进行审计设置，接着运行系统服务和应用以产生进程和文件访问数据，然后使用ausearch命令将审计数据导出到文件。

通过在Ubuntu终端运行图3.1所示的命令，实现了使用auditctl工具生成系统的审计数据，并将生成的数据以文件的形式保存到桌面。

```
parallels@ubuntu-linux-22-04-02-desktop:~$ sudo service auditd start
parallels@ubuntu-linux-22-04-02-desktop:~$ cd Desktop
parallels@ubuntu-linux-22-04-02-desktop:~/Desktop$ pwd
/home/parallels/Desktop
parallels@ubuntu-linux-22-04-02-desktop:~/Desktop$ ausearch -i > /home/parallels
/Desktop/test_log
```

图 3.1 使用 auditctl 生成审计数据示意图

图 3.2 是生成的部分审计数据示意图，其中包括了时间戳、事件类型、执行的用户、受影响的文件路径等关键信息，这些审计日志提供了系统活动信息，用来对异常行为进行分析。

```
----
type=PROCTITLE msg=audit(09/10/24 21:28:08.272:494460) : proctitle=/usr/bin/dbus-daemon --session --address=systemd: --
nofork --nopidfile --systemd-activation --syslog-only
----
type=PROCTITLE msg=audit(09/10/24 21:28:08.280:494461) : proctitle=/usr/bin/dbus-daemon --session --address=systemd: --
nofork --nopidfile --systemd-activation --syslog-only
type=PATH msg=audit(09/10/24 21:28:08.280:494461) : item=0 name=/usr/share/icons/Yaru/scalable/ui/pan-down-symbolic.svg
inode=2892894 dev=08:02 mode=file,644 ouid=root ogid=root rdev=00:00 nametype=NORMAL cap_fp=none cap_fi=none
cap_fe=0 cap_fver=0 cap_frootid=0
type=CWD msg=audit(09/10/24 21:28:08.280:494461) : cwd=/home/parallels
type=SYSCALL msg=audit(09/10/24 21:28:08.280:494461) : arch=aarch64 syscall=openat success=yes exit=16 a0=AT_FDCWD a1=
0xaaaaebb9a730 a2=O_RDONLY a3=0x0 items=1 ppid=2049 pid=20428 auid=parallels uid=parallels gid=parallels euid=parallels
suid=parallels fsuid=parallels egid=parallels sgid=parallels fsgid=parallels tty=(none) ses=4 comm=gedit exe=/usr/bin/gedit
subj=unconfined key=(null)
----
type=PROCTITLE msg=audit(09/10/24 21:28:08.280:494462) : proctitle=/usr/bin/dbus-daemon --session --address=systemd: --
nofork --nopidfile --systemd-activation --syslog-only
type=PATH msg=audit(09/10/24 21:28:08.280:494462) : item=0 name=/usr/share/icons/Yaru/scalable/ui/tab-new-symbolic.svg
inode=2892903 dev=08:02 mode=file,644 ouid=root ogid=root rdev=00:00 nametype=NORMAL cap_fp=none cap_fi=none
cap_fe=0 cap_fver=0 cap_frootid=0
type=CWD msg=audit(09/10/24 21:28:08.280:494462) : cwd=/home/parallels
type=SYSCALL msg=audit(09/10/24 21:28:08.280:494462) : arch=aarch64 syscall=openat success=yes exit=16 a0=AT_FDCWD a1=
0xaaaaebe327c0 a2=O_RDONLY a3=0x0 items=1 ppid=2049 pid=20428 auid=parallels uid=parallels gid=parallels euid=parallels
suid=parallels fsuid=parallels egid=parallels sgid=parallels fsgid=parallels tty=(none) ses=4 comm=gedit exe=/usr/bin/gedit
```

图 3.2 生成审计数据部分示意图

根据生成的审计数据，可以清楚的看到系统中进程运行的时间，打开文件的路径以及对文件进行的操作等信息。

(2) 将审计日志转化为数据集

在使用了auditctl工具生成审计日志之后，接下来需要将生成的审计数据转化成一个简洁的系统进程和文件访问关系的数据集，本文在附件给定的getexefiles.py脚本的基础上改进了代码，具体改进点包括以下几个方面：

首先，去掉了对itemList长度的限制，而是改为直接遍历每一行日志。这样，脚本

可以灵活地寻找日志中出现的exe=和name=字段，并在找到这些字段后立即输出相关结果。通过这种方式，简化了日志处理流程，使得处理更加高效。

其次，改进后的脚本会在找到一组可执行文件和文件名后，立即进行输出，并随后重置exe和filename，以便处理下一组日志。这种改进使得日志的解析和输出过程更加流畅，避免了对日志条目数量的依赖。

最后，新的脚本增强了对日志格式的处理灵活性。现在，脚本对日志条目的数量和顺序不再有严格要求。只要日志中包含 exe=和 name=字段，脚本都会对其进行解析并输出结果。这一改进使得脚本在处理各种日志格式时更加适应和可靠。

```python
import sys

def getValue(line, str):
    value = ""
    index = line.find(str)
    if index != -1:
        end = line.find(" ", index)
        if end == -1:
            end = len(line)
        value = line[index + len(str) : end]
    return value

if len(sys.argv) != 2:
    print("Usage: python3 getexefiles.py ausearchFile")
    sys.exit()

with open(sys.argv[1], 'r') as f:
    lines = f.readlines()
```

图 3.3　脚本改进示意图

在运行了改进后的审计数据处理脚本之后，将生成的系统中进程和文件的访问关系数据集保存为文件的形式，图 3.4 是生成的数据集部分示意图。

```
/usr/bin/gedit    /usr/share/icons/Yaru/scalable/ui/pan-down-symbolic.svg
/usr/bin/gedit    /usr/share/icons/Yaru/scalable/ui/tab-new-symbolic.svg
/usr/bin/gedit    /usr/share/icons/Yaru/scalable/actions/open-menu-symbolic.svg
/usr/bin/gedit    /usr/share/icons/Yaru/scalable/ui/window-minimize-symbolic.svg
/usr/bin/gedit    /usr/share/icons/Yaru/scalable/ui/window-restore-symbolic.svg
/usr/bin/gedit    /usr/share/icons/Yaru/scalable/ui/window-close-symbolic.svg
/usr/bin/gedit    /usr/share/icons/Yaru/scalable/ui/pan-down-symbolic.svg
/usr/bin/gedit    /usr/share/icons/Yaru/scalable/actions/go-up-symbolic.svg
/usr/bin/gedit    /usr/share/icons/Yaru/scalable/actions/go-down-symbolic.svg
/usr/bin/dash     /usr/lib/parallels-tools/lib/tls/aarch64/atomics/libc.so.6
/usr/bin/dash     /usr/lib/parallels-tools/lib/tls/aarch64/libc.so.6
/usr/bin/dash     /usr/lib/parallels-tools/lib/tls/atomics/libc.so.6
/usr/bin/dash     /usr/lib/parallels-tools/lib/tls/libc.so.6
/usr/bin/dash     /usr/lib/parallels-tools/lib/aarch64/atomics/libc.so.6
/usr/bin/dash     /usr/lib/parallels-tools/lib/aarch64/libc.so.6
/usr/bin/dash     /usr/lib/parallels-tools/lib/atomics/libc.so.6
/usr/bin/dash     /usr/lib/parallels-tools/lib/libc.so.6
/usr/bin/dash     /etc/ld.so.cache
/usr/bin/dash     /lib/aarch64-linux-gnu/libc.so.6
/usr/bin/lpstat   /usr/lib/parallels-tools/lib/tls/aarch64/atomics/libcups.so.2
/usr/bin/lpstat   /usr/lib/parallels-tools/lib/tls/aarch64/libcups.so.2
/usr/bin/lpstat   /usr/lib/parallels-tools/lib/tls/atomics/libcups.so.2
/usr/bin/lpstat   /usr/lib/parallels-tools/lib/tls/libcups.so.2
/usr/bin/lpstat   /usr/lib/parallels-tools/lib/aarch64/atomics/libcups.so.2
```

图 3.4　生成数据集部分示意图

根据图 3.4 可以清晰的看到，审计日志数据已经成功被转化为验证本算法所有功能所需要的输入形式。

3.3.2 白名单规则聚类测试

白名单规则聚类测试主要包括白名单规则学习与聚类降维两个方面。首先，测试的目的是确保算法能够正确学习系统进程和文件访问的白名单规则并且在聚类的过程中引入掩码机制，实现对白名单规则数量的降低。具体步骤包括使用fileaccessdata样例数据集以及转换后的审计数据作为输入，运行无监督学习算法以生成聚类后的白名单规则。

如图3.5，在给定的fileaccessdata样例数据集上运行本算法的结果输出。**在聚类前，白名单的规则数量为50347条，通过本算法聚类后的白名单规则数量为269条。**显然，本算法在聚类系统中进程与文件访问关系白名单规则中具有非常良好的效果。

```
Generated Rules:
Process rule: /usr/local/hostguard/bin/*, File rule: /usr/local/hostguard/lib/*
Process rule: /usr/local/hostguard/bin/*, File rule: /usr/local/hostguard/lib/tls/*
Process rule: /usr/local/hostguard/bin/*, File rule: /usr/local/hostguard/conf/*
Process rule: /usr/local/hostguard/bin/*, File rule: /usr/local/hostguard/module/*
Process rule: /usr/local/hostguard/bin/*, File rule: /usr/local/hostguard/lib/x86_64/*
Process rule: /usr/local/hostguard/bin/*, File rule: /usr/local/hostguard/module/x86_64/*
Process rule: /usr/local/hostguard/bin/*, File rule: /usr/local/hostguard/module/tls/x86_64/*
Process rule: /usr/local/hostguard/bin/*, File rule: /usr/libexec/sysmonitor/*
Process rule: /usr/local/hostguard/bin/*, File rule: /usr/local/hostguard/lib/tls/x86_64/*
Process rule: /usr/local/hostguard/bin/*, File rule: /usr/lib/locale/*
Process rule: /usr/local/hostguard/bin/*, File rule: /usr/local/hostguard/module/tls/*
Process rule: /usr/sbin/*, File rule: /usr/local/hostguard/lib/*
Process rule: /usr/sbin/*, File rule: /usr/local/hostguard/lib/tls/*
Process rule: /usr/sbin/*, File rule: /usr/local/hostguard/conf/*
Process rule: /usr/sbin/*, File rule: /usr/local/hostguard/module/*
Process rule: /usr/sbin/*, File rule: /usr/local/hostguard/lib/x86_64/*
Process rule: /usr/sbin/*, File rule: /usr/local/hostguard/module/x86_64/*
Process rule: /usr/sbin/*, File rule: /usr/local/hostguard/module/tls/x86_64/*
Process rule: /usr/sbin/*, File rule: /usr/libexec/sysmonitor/*
Process rule: /usr/sbin/*, File rule: /usr/local/hostguard/lib/tls/x86_64/*
Process rule: /usr/sbin/*, File rule: /usr/lib/locale/*
Process rule: /usr/sbin/*, File rule: /usr/local/hostguard/module/tls/*
Process rule: /usr/bin/*, File rule: /usr/local/hostguard/lib/*
```

图 3.5 fileaccessdata 数据集聚类结果部分示意图

为了验证本算法的普适性，本文在构建的数据集上也进行了算法的聚类效果测试，图3.6为在构建的数据集test_log_output上的聚类结果部分示意图。

```
Generated Rules:
Process rule: /usr/bin/*, File rule: /usr/lib/parallels-tools/lib/tls/aarch64/atomics/*
Process rule: /usr/bin/*, File rule: /usr/lib/parallels-tools/lib/atomics/*
Process rule: /usr/bin/*, File rule: /usr/lib/parallels-tools/lib/*
Process rule: /usr/bin/*, File rule: /usr/lib/parallels-tools/lib/aarch64/*
Process rule: /usr/bin/*, File rule: /usr/lib/parallels-tools/lib/tls/aarch64/*
Process rule: /usr/bin/*, File rule: /usr/lib/parallels-tools/lib/aarch64/atomics/*
Process rule: /usr/bin/*, File rule: /usr/lib/parallels-tools/lib/tls/atomics/*
Process rule: /usr/bin/*, File rule: /usr/lib/parallels-tools/lib/tls/*
Process rule: /usr/lib/systemd/*, File rule: /usr/lib/aarch64-linux-gnu/tracker-miners-3.0/aarch64/atomics/*
Process rule: /usr/lib/systemd/*, File rule: /usr/lib/aarch64-linux-gnu/tracker-3.0/tls/atomics/*
Process rule: /usr/lib/systemd/*, File rule: /lib/aarch64/*
Process rule: /usr/lib/systemd/*, File rule: /lib/*
Process rule: /usr/lib/systemd/*, File rule: /usr/lib/aarch64-linux-gnu/gedit/atomics/*
Process rule: /usr/lib/systemd/*, File rule: /usr/lib/aarch64-linux-gnu/atomics/*
Process rule: /usr/lib/systemd/*, File rule: /usr/lib/aarch64-linux-gnu/gedit/tls/aarch64/*
Process rule: /usr/lib/systemd/*, File rule: /usr/lib/aarch64-linux-gnu/gedit/girepository-1.0/*
Process rule: /usr/lib/systemd/*, File rule: /usr/lib/aarch64-linux-gnu/tracker-3.0/tls/aarch64/*
Process rule: /usr/lib/systemd/*, File rule: /lib/systemd/tls/aarch64/*
Process rule: /usr/lib/systemd/*, File rule: /usr/lib/aarch64-linux-gnu/tracker-miners-3.0/tls/aarch64/*
Process rule: /usr/lib/systemd/*, File rule: /lib/aarch64-linux-gnu/atomics/*
Process rule: /usr/lib/systemd/*, File rule: /usr/lib/aarch64-linux-gnu/gedit/*
Process rule: /usr/lib/systemd/*, File rule: /lib/aarch64-linux-gnu/aarch64/atomics/*
Process rule: /usr/lib/systemd/*, File rule: /lib/tls/atomics/*
```

图 3.6　test_log_output 数据集聚类结果部分示意图

通过聚类结果的可视化绘制，我们可以直观地观察到系统中进程和文件访问关系的白名单规则的分布情况。在可视化图中，每个数据点表示为一个点，其位置由其在特征空间中的坐标决定。聚类中心表示为红色且大小较大的点，它们在图中以半透明的形式突出显示，以便于识别。通过聚类结果的可视化，能够帮助我们评估聚类算法的效果。图 3.7 和图 3.8 分别是样例数据集和测试集的聚类结果可视化绘图。

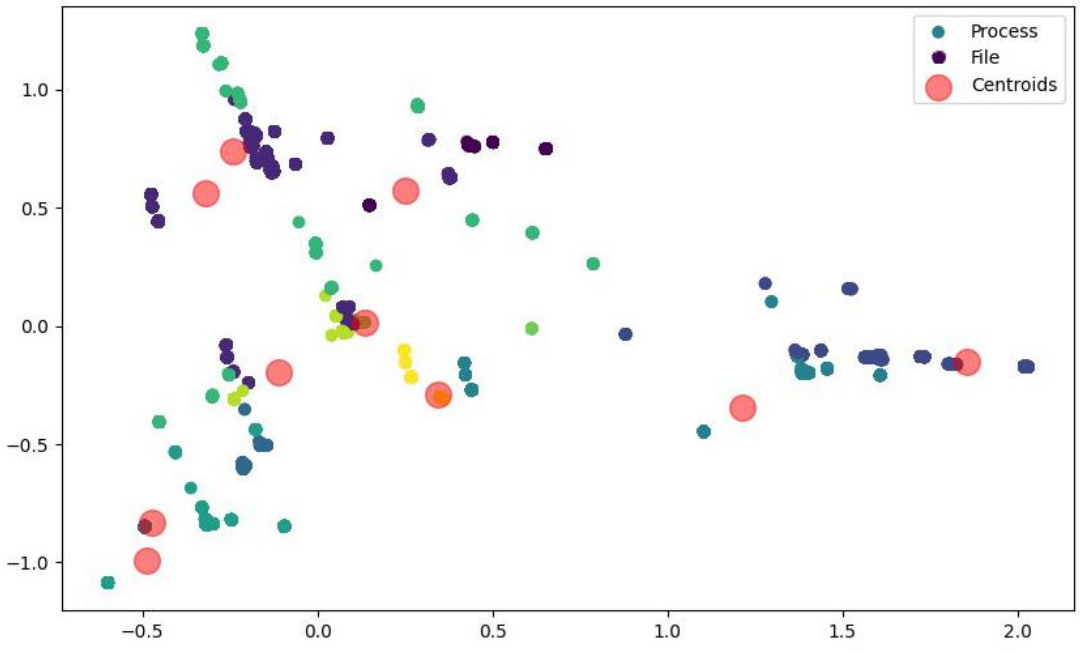

图 3.7　fileaccessdata 数据集聚类结果可视化绘图

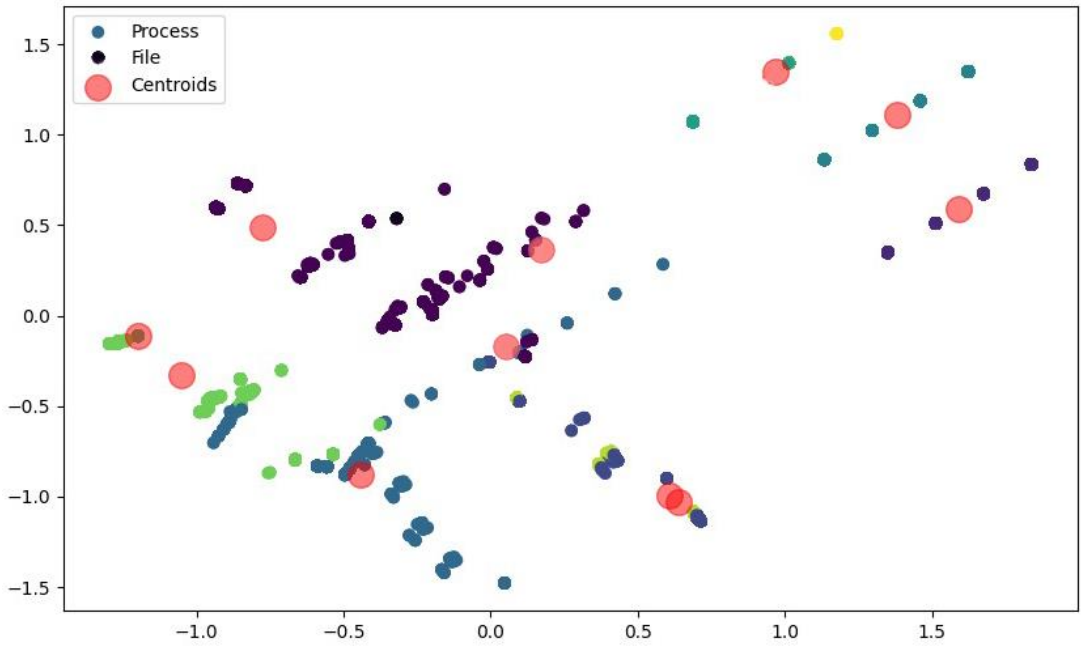

图 3.8　test_log_output 数据集聚类结果可视化绘图

通过直方图，可以直观地看到不同聚类中数据点的分布情况，这有助于理解聚类算法的效果，以及数据在不同聚类中的聚集程度。如果某个聚类的数据点数量显著多于其他聚类，即该聚类规则包含了大多数的文件与进程的访问关系，以测试集 fileaccessdata 为例，聚类结果直方图如图 3.9 所示：

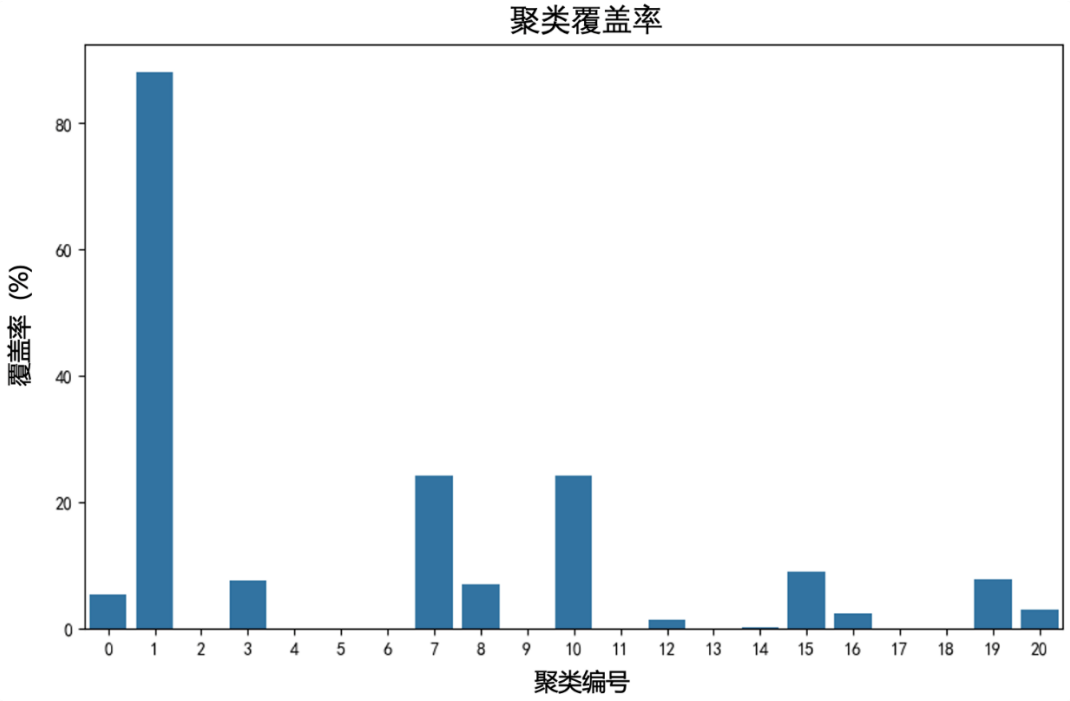

图 3.9　聚类结果直方图

白名单规则覆盖率测试的主要目标是测试本算法生成的白名单规则对原始进程与

文件访问关系数据集的覆盖率，确保在减少白名单规则数量的同时不影响覆盖精度。首先使用原始的进程与文件访问关系数据集作为基准，对比生成的白名单规则与原始数据集，计算规则的覆盖率。最后，将生成的白名单规则的覆盖率结果进行输出，以确保规则的精确度和有效性在规则数量减少的情况下得到保证。

图3.7所示的是**算法在给定样例数据集fileaccessdata上的覆盖率，可以看到降低后的白名单规则能够达到99.89%的覆盖率，这表明本算法能够在减少白名单规则数量的同时不影响覆盖精度。**

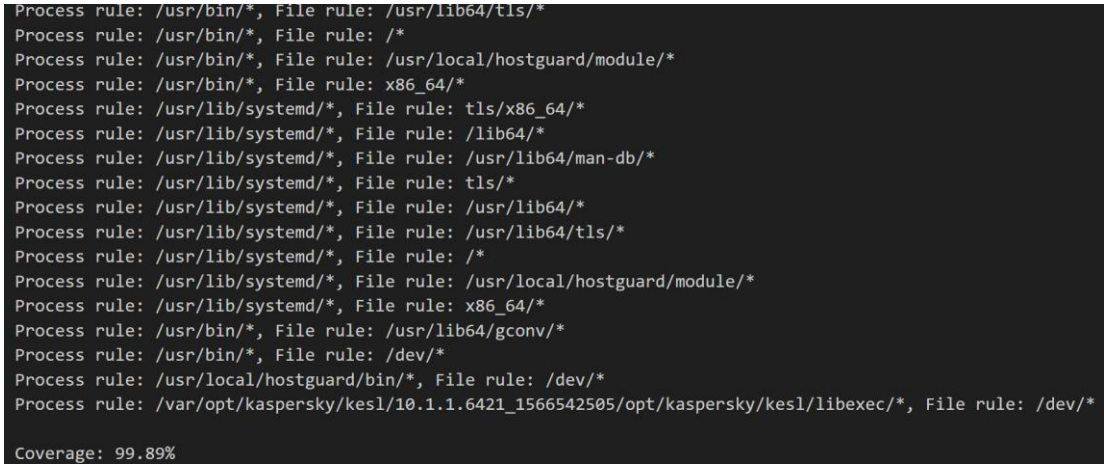

图 3.10　fileaccessdata 数据集覆盖率结果示意图

图3.8所示的是算法在本文构建的数据集test_log_output上的覆盖率，可以看到降低后的白名单规则依旧能够达到99.87%的覆盖率，这表明本算法在保证高覆盖率的同时也具有普适性。

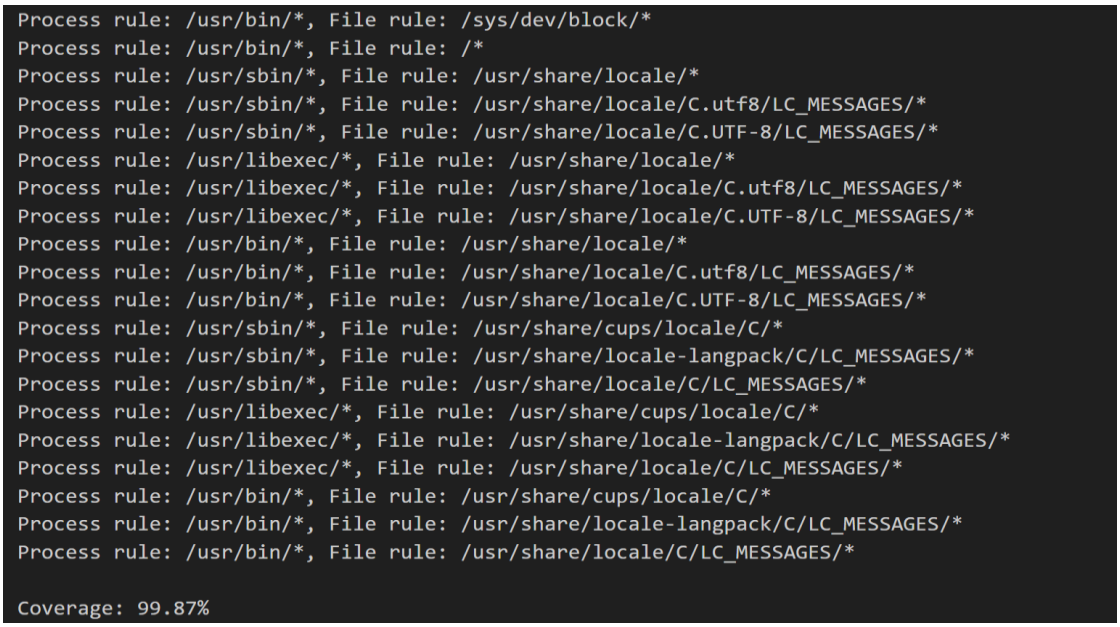

图 3.11　test_log_output 数据集覆盖率结果示意图

3.4 性能测试结果

性能测试的目的是为了验证本算法在不同规模的数据集下的性能表现，确保算法在大规模数据集上仍能快速实现白名单规则数量的降低。在5000条、10000条以及20000条规则的数据集上使用本算法进行白名单规则的聚类，根据算法的运行时间评估算法的效率。

表3.1为5000条、10000条以及20000条规则的数据集上使用本算法进行白名单规则的聚类时算法的运行时间、算法的覆盖率以及聚类后的规则数量对比表。

表 3.4　不同数据规模下的算法性能对比

规则数量	覆盖率	聚类后规则数量	运行时间
5000	99.92%	145	0.38s
10000	99.93%	205	0.74s
20000	99.89%	239	1.51s

可以看到，本算法能够在不同规模的数据集上保持良好的性能，能够有效的聚类降低白名单规则数量并保持较高的规则覆盖率。

第四章　创新性说明

4.1 自动化生成规则与优化

本工具基于 MiniBatchK-Means 算法实现聚类，并且可以自动化生成和简化白名单规则，其存在以下几点优势：

(1) 应用自动化聚类算法：本工具基于 MiniBatchK-Means 算法实现自动化聚类，该算法与传统的 KMeans 算法相比具有更快的计算速度，在处理大规模数据集时通过每次只处理一个子集来减少内存的消耗。并且算法通过在多个小批量上迭代更新质心来减少对单个数据点异常值的敏感性，以此来提升算法的鲁棒性。

(2) 多维度特征提取：本工具从多不同角度和层次捕捉数据的关键属性，构建了一个全面的特征表示方法。这种全面的数据捕捉方法可以更精细地区分不同的数据模式，还能通过降维数据来优化计算效率，减少特征选择和模型训练过程中的计算负担。

(3) 冗余规则自动识别与去除：本工具可以实现对白名单规则的冗余识别与去重，通过自动化的白名单规则简化，可以减少对存储以及处理资源的需求。并且去除冗余规则可以减少规则检查过程中的计算量，加快访问决策的速度，从而整体提升系统的性能。

4.2 擅长处理大规模数据集

本工具使用的 MiniBatchK-Means 聚类算法是 KMeans 算法的一种改进算法，相对于其他聚类算法，其存在如下几种优势：

(1) 处理大规模数据集的能力：MiniBatchK-Means 算法通过每次处理数据集的一个小批量（mini-batch），显著减少了内存消耗。这种处理方式能够处理太大而无法在内存中完全加载的大规模数据集。

(2) 提高计算效率：由于每次迭代仅使用数据的一个子集，MiniBatchK-Means 算法减少了每次迭代所需的计算资源，提高了算法的计算效率。这种效率的提升在处理大数据集时尤为明显。

(3) 早期停止机制：算法提供了早期停止的机制，当连续多个小批量没有带来聚类质心的显著改进时，算法可以提前结束，从而节省计算时间。这种机制有助于避免不必要的计算，提高算法的运行效率。

(4) 灵活性和参数调整：MiniBatchK-Means 算法允许用户调整 mini-batch 的大小

和其他参数，如聚类数量（n_clusters）、随机初始化次数（n_init）等，以适应不同的数据集和应用需求。这种灵活性使得算法能够在效率和准确性之间进行平衡，更好地适应特定的业务场景。

(5) 部分拟合能力：支持部分拟合（partial_fit）方法，使得 MiniBatchK-Means 算法能够逐步接收新数据并更新模型。这对在线学习场景以及处理动态数据集十分重要，该机制可以使算法在新数据输入时持续地优化聚类结构，而不需要初始化模型，从而提高数据地处理效率以及对新数据集地适应性。

综上所述，MiniBatchKMeans 聚类算法通过其部分拟合能力来实现逐步接收并处理新数据的功能，这确保了在实时监控和响应系统访问行为中进行数据处理的高效性。此外，算法的灵活性和适应性使其能够适应未知应用系统进程文件访问行为，学习到高质量的白名单规则。同时算法的快速学习时间确保了其在动态环境中能够快速产生可靠的结果。

4.3 模块化设计以及可扩展性

本工具的可扩展性和动态环境适应性是其核心创新点，这些特性可以确保本工具能够适应不断变化的技术需求以及业务场景。其存在以下几点优势：

(1) 模块化设计：

本工具采用模块化设计，使得各个组件（如数据预处理、特征提取、聚类算法、规则生成等）之间可以独立更新和扩展。随着新技术的出现，如新的机器学习算法或数据处理技术，本工具可以轻松集成这些技术，以提高其性能和准确性。

(2) 灵活的配置选项：

提供了灵活的配置选项，允许用户根据具体需求调整参数，如聚类的数量、特征提取的方法、规则简化的策略等。这种灵活性使得本工具能够适应不同的数据集和业务需求。同时不仅限于处理特定类型的数据，而是设计为能够处理多种数据类型，包括文本、数值、图像等。这种多数据类型支持确保了方案在不同领域的广泛应用。

(3) 实时数据处理：

本工具能够处理实时数据流，使其能够适应动态变化的环境，如实时监控系统状态，快速响应新的安全威胁。且实现了有效的反馈机制，允许系统根据实时的运行结果（如规则的覆盖率和误报率）自动调整参数，以优化性能。

(4) 自动化规则调整：

采用持续学习的策略，方案能够不断从新的数据集中学习，自动更新规则集，以适应用户的行为变化或新的安全威胁。本工具中使用的规则生成算法能够自动调整，以反映最新的系统访问模式，确保规则集始终处于最新的状态，并且能够覆盖所有正常的访问模式。

本工具通过其模块化架构、参数化配置、多模态数据处理能力、实时数据处理与反馈机制以及自动化规则优化等核心创新特性，实现了卓越的可扩展性和动态环境适应性，确保了在不断变化的技术需求和业务场景中的高效性和灵活性。

第五章　总结

MKWAY 白名单规则自动化构建工具采用**无监督学习算法**，能够高效处理大规模数据集，从系统进程和文件访问数据中**自动学习并生成白名单规则。**本算法平衡了**白名单规则的准确性与粒度，**在确保规则具有**较高准确度**的同时，保证了算法具有**较高的学习效率。**

MKWAY 工具的核心功能包括自动化特征提取、聚类分析、规则生成和简化、计算规则覆盖率以及输出精简的白名单规则集。该工具的数据预处理模块**利用 CountVectorizer 对文本数据进行向量化处理**，转为数值特征向量，为后续分析提供标准化基础。特征提取模块进一步筛选关键特征，以降低数据维度并提高处理效率。聚类分析模块**采用 MiniBatchK-Means 算法，快速对特征向量进行聚类**，确保了处理速度和聚类质量。

最终测试结果表明，MKWAY 工具**达到了题目给定的要求。在 1s 的运行时间内能够处理超过 10000 条白名单规则。并且在实现超过 99%的规则覆盖率的同时，显著减少白名单规则的数量。**

参考文献

[1] 初龙丰,高鹏,邱志远.网络安全隐患及漏洞挖掘技术应用分析[J].网络安全技术与应用,2024,(09)：25-27.

[2] M. F. Hyder and T. Fatima, "Towards Crossfire Distributed Denial of Service Attack Protection Using Intent-Based Moving Target Defense Over Software-Defined Networking," in IEEE Access, vol. 9, pp. 112792-112804, 2021, doi: 10.1109/ACCESS.2021.3103845.

[3] 邬坤露.结合白名单与机器学习的工控网络入侵检测方法研究[D].辽宁工程技术大学,2023.DOI：10.27210/d.cnki.glnju.2023.000326.

[4] 甘井中,杨秀兰,吕洁,等.人工智能中无监督学习算法综述[J].海峡科技与产业,2019,(01)：134-135.

[5] 杨光.基于深度学习的日志分析检测研究[D].西安工业大学,2024.DOI：10.27391/d.cnki.gxagu.2024.000575.

[6] 林松.基于机器学习的网络流量分类和异常检测技术研究与实现[D].南京邮电大学,2023.DOI：10.27251/d.cnki.gnjdc.2023.000112.

[7] R. Zhang, Y. Duan, F. Nie, R. Wang and X. Li, "Unsupervised Deep Embedding for Fuzzy Clustering," in IEEE Transactions on Fuzzy Systems, 2024,doi: 10.1109/TFUZZ.2024.3462545.

[8] Dan Xu, Mingrui Zhou, and Meng Yuan. 2024. Optimization Research of K-Means Clustering Algorithm Based on Big Data. In Proceedings of the 5th International Conference on Computer Information and Big Data Applications (CIBDA 24). Association for Computing Machinery, New York, NY, USA, 1021–1025. https://doi.org/10.1145/3671151.3671329

[9] 彭豪辉. 基于用户行为的内部威胁检测方法研究[D].北京交通大学,2020.

[10]张锐.基于文件访问行为的内部威胁异常检测模型研究[D].北京交通大学,2015.

[11]崔洁,蔡忠闽,孙国基.一种基于文件访问监控的主机异常入侵检测系统[J].微电子学与计算机,2005,(04)：57-62.DOI：10.19304/j.cnki.issn1000-7180.2005.04.016.

[12]王翎霁.木马本机文件访问行为监测方法研究[D].华中科技大学,2015.

[13]孙超.基于用户行为和关系的内部风险分析[D].山东大学,2015.

[14]成双,郭渊博.基于 LSTM 网络的异常操作行为检测方法[J].信息工程大学学报,2019,20(01)：122-128.

[15]Newling J , Fleuret F.Nested Mini-Batch K-Means[J]. 2016.DOI：10.48550/arXiv.1602.02934.

[16]X. Wang and Y. Xing, "Research on Web Log Data Mining Technology Based on Optimized Clustering Analysis Algorithm," 2021 International Conference on Artificial Intelligence and Blockchain Technology (AIBT), Beijing, China, 2021, pp. 6-11, doi: 10.1109/AIBT53261.2021.00008.

[17]Sascha Kaven and Volker Skwarek. 2023. Poster: Attribute Based Access Control for IoT Devices in 5G Networks. In Proceedings of the 28th ACM Symposium on Access Control Models and Technologies (SACMAT 23). Association for Computing Machinery, New York, NY, USA, 51–53. https://doi.org/10.1145/3589608.3595081.

[18]H. Ma, C. Wang and H. Qi, "Anomaly Behavior Detection for the Web Application Based on LSTM," 2021 IEEE Conference on Telecommunications, Optics and Computer Science (TOCS), Shenyang, China, 2021, pp. 553-559, doi: 10.1109/TOCS53301.2021.9688720.