

基于大语言模型的AppArmor安全策略自动生成方法

沙倚天¹, 刘少君¹, 钱欣¹, 吴越¹, 陈鹏¹, 刘行²

(1. 国网江苏省电力有限公司南京供电分公司, 江苏 南京 210016; 2. 南京南瑞信息通信科技有限公司, 江苏 南京 210016)

摘要: 在计算机信息系统的整体安全性中, 操作系统安全扮演着至关重要的角色, 而操作系统安全的基础是信息系统的安全。目前, 已经有SELinux、AppArmor等基于强制访问控制机制的安全方法被提出, 来增强操作系统的安全性。然而, 这类安全方法在实际应用中存在的主要问题, 例如安全策略配置的复杂性和对专业知识的高度依赖, 使安全策略的生成变得困难, 并且防护粒度相对较粗。因此, 对AppArmor的安全策略的生成过程进行进一步的优化, 并提出了一种基于大语言模型 (large language model, LLM) 的访问控制策略自动生成方法。该方法对目标应用程序进行静态分析, 获取部分安全策略规则; 并全面运行应用程序, 以获得最大代码覆盖率的相关日志; 最后结合日志信息和静态分析结果以及少样本学习, 运用大模型为应用自动生成安全策略, 减少了对安全专家的依赖, 从而显著降低了人工成本以及减少了人为配置安全策略的主观性和复杂性。通过将所提方法生成的AppArmor安全策略与默认策略进行对比, 对策略的正确性、完整性和精简性进行了评估。实验结果显示, 所提方法生成的策略既完整又精简, 并且不会影响应用程序的正常运行。这表明了所提方法在简化安全策略生成过程和提高策略质量方面的有效性。

关键词: 强制访问控制; AppArmor; 安全策略; 静态分析

中图分类号: TP309.1

文献标志码: A

DOI: 10.11959/j.issn.2096-109x.2024046

Automatic generation of AppArmor security policies based on large language models

SHA Yitian¹, LIU Shaojun¹, QIAN Xin¹, WU Yue¹, CHEN Peng¹, LIU Xing²

1. State Grid Nanjing Power Supply Company, Nanjing 210016, China

2. Nanjing Nanrui Information and Communication Technology Limited Company, Nanjing 210016, China

Abstract: Operating system (OS) security has been considered as a critical layer within the comprehensive security framework of computer information systems. Mandatory access control (MAC) mechanisms such as SELinux and AppArmor have been employed to strengthen OS security. However, significant challenges have been encountered in the application of MAC in practice, primarily involving the complexity of security policy configuration, which

收稿日期: 2024-04-18; 修回日期: 2024-06-10

通信作者: 沙倚天, 973923801@qq.com

基金项目: 国网江苏电力有限公司科技项目 (J2023110)

Foundation Item: State Grid Jiangsu Electric Power Co., Ltd. Science and Technology Project (J2023110)

引用格式: 沙倚天, 刘少君, 钱欣, 等. 基于大语言模型的AppArmor安全策略自动生成方法[J]. 网络与信息安全学报, 2024, 10(3): 143-155.

Citation Format: SHA Y T, LIU S J, QIAN X, et al. Automatic generation of AppArmor security policies based on large language models[J]. Chinese Journal of Network and Information Security, 2024, 10(3): 143-155.

has demanded specialized expertise and often resulted in coarse-grained protection measures. Aimed at enhancing the process of generating security policies for AppArmor, an automated access control policy generation method was proposed based on large language models (LLM). The process began with a static analysis of the target application to extract preliminary security policy rules. Subsequently, the application was extensively executed to gather logs that achieved maximum code coverage. Large models, in conjunction with the gathered log information, static analysis outcomes, and few-shot learning, were utilized to autonomously generate security policies for applications. This approach significantly reduced the dependence on security experts, lowered manual labor costs, and diminished the subjectivity and complexity associated with manual security policy configurations. The efficacy of this methodology was demonstrated through a comparative analysis between the AppArmor security policies generated by this framework and the default policies, focusing on the policies' correctness, completeness, and succinctness. The experimental findings reveal that the generated policies are not only comprehensive and succinct but also do not impede the normal operation of the application. These results underscore the effectiveness of the proposed method in streamlining the security policy generation process and enhancing the quality of the policies.

Keywords: mandatory access control, AppArmor, security policy, static analysis

0 引言

在信息技术时代, 确保操作系统的安全性对维护计算机信息系统的整体安全至关重要。操作系统安全是信息系统安全的基础, 更是确保数据完整性、可靠性和可用性的前提。随着网络攻击手段的日益复杂和多样化, 传统的安全防护措施已经难以满足日益增长的安全需求。因此, 如何建立更加高效、智能的安全策略生成机制, 实现灵活的访问控制, 成为操作系统安全保障的关键。

强制访问控制 (mandatory access control, MAC) 机制是操作系统安全架构的核心, 其通过比较主体 (如用户或进程) 与对象 (如文件或设备) 的安全属性, 来决策是否允许某一访问行为的发生。AppArmor 是基于 MAC 机制的安全框架, 为操作系统提供了强大的安全保障基础, 其依靠预定义的安全策略, 拒绝对资源的未授权访问, 从而保护操作系统及其上运行的应用程序不受恶意攻击和不当行为的威胁。

具体来说, AppArmor 提供了基于路径的 MAC 解决方案, 它将访问控制属性与目标软件 (而非用户) 相关联, 一定程度上降低了策略复杂性。其安全策略中详细规定了对应用程序中潜在访问路径的控制方式, 以及应拒绝哪些类型的未授权访问。为辅助策略生成, AppArmor 提供了审计和规则判断机制, 当应用程序在 Linux 安全模块 (linux security module, LSM) 的访问点

出现异常行为时, 审计消息会被记录到日志文件中。然后, 系统管理员可根据这些审计日志, 选择授予的权限, 生成相应的策略。策略配置文件的生成基于应用程序被允许的操作记录, 未被列出的操作将被默认拒绝。由于现代软件系统以及安全需求复杂, AppArmor 安全策略的制定仍需要安全专家人工参与, 配置和管理成本较高。

为降低相关成本, 促进各种新开发应用的安全策略配置, 近年来自动化的安全策略生成技术受到持续关注。然而, 这一领域面临着以下关键挑战: ①程序运行期间的行为模式动态多样, 难以收集覆盖全面的日志信息等刻画程序行为, 导致生成的规则覆盖率低, 甚至影响程序正常执行; ②安全策略规则形式复杂, 且配置过程高度依赖安全领域专家知识, 现有方法智能化水平不足, 基于有限规则生成的配置冗余程度高、防护粒度粗。这些挑战严重限制了现有安全配置生成技术在真实场景中的应用。

为此, 本文提出了基于大语言模型的 AppArmor 安全策略自动生成方法, 从二进制目标程序 (无须源代码) 出发, 自动分析其行为并生成对应的安全策略, 旨在避免人工配置大量规则, 减轻安全专家的负担。提出的方法首先在沙箱中动态执行目标程序, 获取最大化代码覆盖率的运行日志, 并通过预处理得到低冗余的切分结果, 分段送入大模型。为了应对日志信息不全的挑战, 本文利用静态分析技术从目标二进制程序中提取部分策略规则, 补充运行日志得到更全面

的程序行为描述。在此基础上, 为了实现更智能、更高效的安全策略生成, 在安全语境下充分发挥通用大模型在文本模式识别和内容归纳等方面的突出能力, 本文将预定义的通用安全规则以及少量安全配置生成示例导入大语言模型, 增强大模型的领域知识, 最终为应用程序自动生成包含 File、Capabilities、Network、DBus、PTrace 和 Signal 这 6 类 AppArmor 安全策略。通过系统的实验评估, 证明提出的方法能在不影响目标程序正确执行的前提下, 有效简化安全策略生成过程并提高生成策略的质量, 满足复杂的安全需求。

本文的主要贡献包括 4 个方面:

1) 提出了基于大语言模型的 AppArmor 安全策略自动生成方法, 大大降低了人工安全策略配置的复杂性和对专业性的要求。

2) 提出融合程序静态分析的安全策略生成辅助技术, 通过结合静态分析结果和动态执行日志, 为安全策略生成过程提供全面完整的程序行为信息。

3) 设计了语境学习辅助的大语言模型 AppArmor 安全策略增强方法, 通过引入 AppArmor 策略通用模板、少样本示例等领域特定上下文知识, 实现了更智能和高效生成安全策略。

4) 设计并进行实验, 比较了自动生成与专家生成的安全策略, 结果验证了本文提出方法的有效性和实用性。

1 相关工作

近年来, 国内外有一些关于安全策略生成方法的研究。Han 等^[1]提出以能力依赖图和 MaxSAT 为核心的自动化方法, 加强了安全策略的配置。这一方法致力于识别并修正现行安全策略中的漏洞。能力依赖图为每项操作所必要的条件和结果提供了预先定义的框架, 阐释了潜在的攻击路径可能对系统造成的风险。通过全面的静态代码分析, 确保生成的安全策略可以全面包括应用所需的各项访问权限。Lachmund 等^[2]采用基于调用图进行静态分析的技术, 可以区分出由应用程序自主发起和由用户操作触发的资源访问请求, 通过该技术生成的安全策略未能涵盖需要用户直接交互的访问权限部分。Centonze 等^[3]结合

静态分析和动态分析的技术手段, 实现了对应用程序访问控制策略的自动化精确配置。Rauter 等^[4]通过使用二进制分析的方法为软件必要的权限进行了高层次的定义, 基于这种定义实现了 AppArmor 安全策略的自动化生成。然而, 这种以二进制文件分析为基础的方法构建的调用树并不完备, 特别是在处理面向对象编程的语言时, 会导致所产生的安全策略存在缺陷。Schreuders 等^[5]通过对程序的依赖关系、管理信息以及文件系统的内容进行分析, 实现了安全策略的自动化制定和规范化。在直接对应用程序进行分析之外, 还有一些研究通过分析包含有应用执行信息的审计日志来生成安全策略。Sanders 等^[6-7]使用机器学习技术从审计日志中提取信息, 实现了安全策略的自动化生成。首个专注于自动化分析和优化 SEAndroid 策略的平台 EASEAndroid^[8], 它借助半监督学习技术对审计日志进行分析。SELinux 发布的工具 Audit2allow 可以把审计日志转化为规则, 进而可以实现 SELinux 安全策略的自动化生成和优化。但类似于 Lachmund 面临的相同问题, Audit2allow 机械化的转换过程, 很难捕捉到用户真实的操作意图, 会出现超出权限的问题。Nadkarni 等^[9]旨在通过用户定制的安全策略示例来预测和制定新应用的安全策略, 它能够根据提供的示例来决定特定情况下对资源的访问权限应该是允许还是拒绝。Enck 等^[10]通过自动分析应用程序的单元测试并提取其中的领域知识, 用于定义 SEAndroid 策略规则的潜在攻击向量, 从而丰富了语义功能测试的应用。马铭鑫等^[11]提出适用于分布式系统的多级安全访问控制策略, 并对策略进行安全性和灵活性分析。

在安全策略生成方法的研究领域, 上述研究的内容涵盖了静态分析、动态分析、二进制文件分析、审计日志分析等多种方法。这些研究显著提高了安全策略配置的自动化程度和准确性, 但通常依赖于深厚的安全知识, 仍需要人为介入, 且在处理动态环境和面向对象编程语言时存在局限性。相比现有方法, 本文提出的基于大语言模型的自动化方法, 在继承前述优点的基础上, 基于大语言模型, 通过结合静态分析、动态行为日志与少样本学习, 显著减少了对专业知识的依

赖,提高了策略的生成效率和适应性。这一方法不仅优化了安全策略的覆盖率、细粒度和适用性,还降低了策略配置的主观性和复杂性,使安全策略的生成更加高效和准确。

2 AppArmor 安全策略背景知识

如图 1 所示,在该 AppArmor 安全策略 profile 示例中,共列出了 9 个内容。①为“include”相关内容,通过引入通用的预定义规则,从而简化安全策略的编写和管理,在该示例中,分别引入全局变量定义、基础的系统访问权限和网络服务相关的规则定义。②为路径变量定义,在策略中定义变量@{exec_path},用于表示受限制的应用程序的执行路径。这提供了一种灵活的方式来指定和引用应用程序的位置。③为安全策略定义,示例中定义了一个名为 sample-application 的安全策略,适用于 @{exec_path} 指定的程序。这个部分是策略的核心,其中定义了所有具体的访问控制规则。④为 capability 规则,这些规则明确了程序所需的特定系统能力,如 setgid、net_bind_service、dac_override 等,允许程序执行相关的系统操作。这有助于实施最小权限原则,确保程序只能执行其必须的操作。⑤为 network 规则,规定了应用程序的网络访问权限。示例中允许使用 inet 协议进行流式通信,使用 inet6 协议进行数据报文通信。这限制了程序可能使用的网络接口和协议。⑥为文件操作规则,定义了对文件系统的访问权限。示例中为允许程序可以读写/tmp/logs/myapp.log、只读访问/etc/myapp/config.json 以及执行/usr/bin/myapp。这些规则细化了程序对文件系统的访问范围。⑦为 dbus 规则,指定了程序通过 DBus 发送消息的能力,包括目标总线、路径和接口。这允许程序与其他使用 DBus 通信的程序交互,同时限制了这种交互的范围和方式。⑧为 signal 规则,控制了程序发送信号的权限。示例允许向任何未被 AppArmor 策略限制的进程发送信号。这种规则用于细化程序间通信的权限。⑨为 ptrace 规则,定义了程序使用 ptrace 进行跟踪或调试其他进程的权限。在本例中,只允许跟踪/usr/bin/myapp。这是对程序调试能力的一个重要限制,有助于防止恶意软件或用户滥用 ptrace 造成安全风险。

```
#include<tunables/global> ①
#a comment naming the application to confine
@{exec_path} = @{bin}/sample-application ②
profile sample-application @{exec_path} ③ {
    #include <abstractions/base> ①
    #include <abstractions/namespace> ①

    capability setgid, ④
    capability net_bind_service, ④
    capability dac_override, ④

    network inet stream, ⑤
    network inet6 dgram, ⑤

    /tmp/logs/myapp.log rw, ⑥
    /etc/myapp/config.json r, ⑥
    /usr/bin/myapp ix, ⑥

    dbus send ⑦
        bus=session
        path=/org/example/MyApp
        interface=org.example.MyAppInterface,

    signal (send) peer=unconfined, ⑧

    ptrace (trace) peer=/usr/bin/myapp, ⑨
}
```

图 1 AppArmor 安全策略 profile 示例

Figure 1 Example of AppArmor security policy profile

3 AppArmor 安全策略自动生成方法

本文假设程序代码定义的所有行为都为合法行为,仅考虑攻击者通过潜在的程序漏洞劫持程序访问未经授权的系统资源的威胁。检测程序本身是否为恶意程序或包含后门代码超出了本文研究的范围,这类检测可以通过匹配已知的恶意代码特征、监控恶意程序行为等技术来完成。

为了解决引言部分所述问题,本文提出融合程序动静态分析的大语言模型安全策略自动生成方法。系统输入方面,本文选择从二进制目标应用程序而非源码出发,这主要是出于以下考虑:一方面,目标程序可能本身没有提供正确版本的源码,即使从源码进行编译,也将面临编译过程时特定于平台的问题,不能精确还原实际部署的版本,这将阻碍生成正确的、通用的安全策略;另一方面,源码的声明与实现通常分散在多个文件中,而编译过程又将涉及源码不可见的宏定义或符号,这些因素显著加大了分析的难度。

如图 2 所示,日志获取及处理模块首先多次

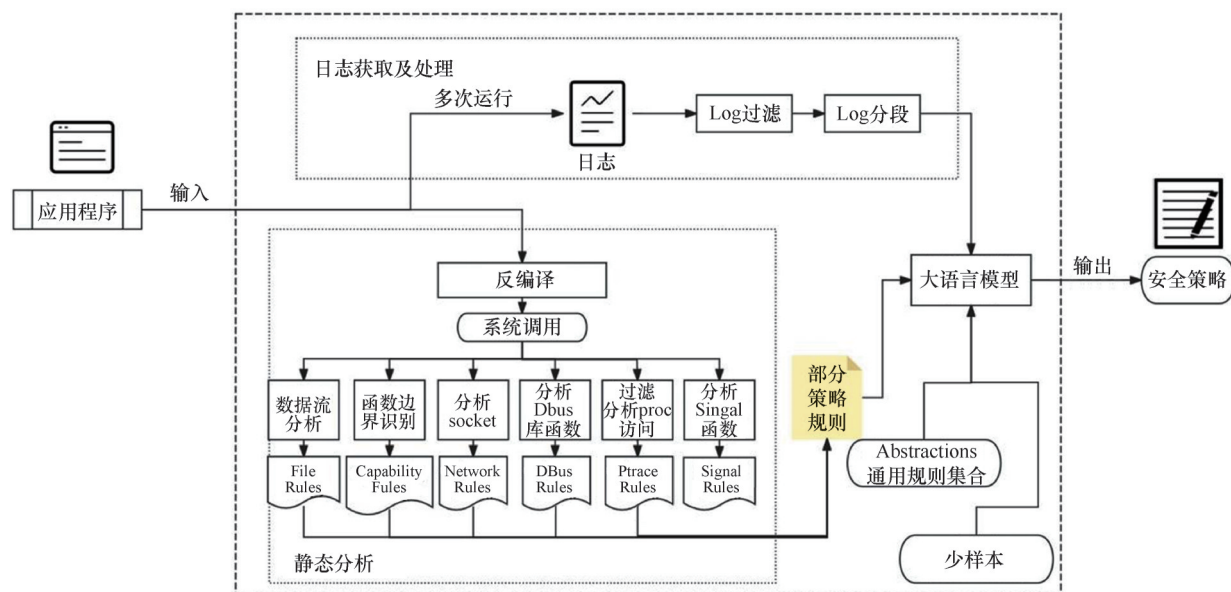


图2 AppArmor安全策略自动生成流程

Figure 2 Flowchart of automatic generation of appArmor security policies

执行目标应用程序以收集尽可能全面的原始日志，之后执行包括重复信息过滤和最优化上下文切分的日志预处理过程，得到适合大语言模型处理的日志输入。为了解决日志信息不全面的问题，静态分析模块综合利用多种工具对二进制文件格式的应用程序进行反编译和分析，根据结果中的系统调用、socket通信、数据流等情况匹配AppArmor策略中的不同规则，整合得到部分安全策略规则，提高输入信息覆盖率。为了解决策略生成不够智能和高效的问题，策略生成模块一方面利用Abstraction预定义通用规则集合与少量高质量样本，增强了通用大语言模型任务相关领域的知识，进而基于大模型将预处理日志信息和静态分析结果转换为完整的安全策略。

3.1 日志获取及处理模块

系统日志包含了应用程序的运行信息，是生成安全策略的重要数据源。要分析应用程序的行为信息和正常执行所需访问的资源，关键在于如何获取尽量全面的系统日志。此外，由于原始日志文件过于冗长，还必须对原始日志进行预处理，过滤其中的重复信息并适当分段，在适应大语言模型上下文长度限制的同时提高其对日志中关键信息的记忆和理解能力。

3.2.1 原始日志收集与过滤

收集日志前，首先把AppArmor的状态切换

到Complain模式，并且为目标程序生成一个不含任何规则的安全策略配置文件，由于AppArmor以白名单机制执行访问控制，此时应用程序的任何资源访问行为均会被记录。为了获得全面的系统日志，需要尽可能触发应用程序的所有可达系统调用，提高运行过程中的代码覆盖率，因此，本文设计了一套详细的手动执行策略。首先，根据应用程序的用户标准使用手册和功能列表，识别所有包含的功能模块；然后，对所有可执行操作进行多次独立的执行，确保涵盖所有主要功能和边界条件，触发不同的执行路径。除了单独测试，还应设计并执行模块间的不同组合场景。这包括连续执行相关功能模块以模拟真实用户的使用流程，确保在模块交互时能实现产生覆盖更全面的日志。特别关注那些可能引发安全事件的操作，如文件访问、网络通信等，以确保日志中包含足够的安全相关数据，从而提供对应用行为的全面视角，获取更多更全面的日志信息。图3给出了两条“dhclient”程序执行“recvmsg”系统调用时的日志记录，可以发现它们仅在“audit”序列号处存在差异，其他部分完全相同。因此，接下来对类似这种情况的原始日志进行过滤，以提高输入为语言模型信息的丰富度。

3.2.2 面向大语言模型的日志切分

大语言模型是指在大规模文本语料库上进行

```

type=AVC msg=audit(1711882389.221:1317): apparmor="AL-
LOWED" operation="recvmsg" profile="/sbin/dhclient" pid=4323
comm="dhclient" family="netlink" sock_type="raw" protocol=0
requested_mask="receive" denied_mask="receive"

type=AVC msg=audit(1711882389.221:1318): apparmor="AL-
LOWED" operation="recvmsg" profile="/sbin/dhclient" pid=4323
comm="dhclient" family="netlink" sock_type="raw" protocol=0
requested_mask="receive" denied_mask="receive"

```

图3 日志记录

Figure 3 Log records

训练的语言模型，大多采用了 Transformer 架构^[12]。如图4所示，Transformer 是基于自注意力机制的编码—解码架构，自注意力机制通过设置注意力矩阵，可以有效地捕捉输入序列的依赖关系，位置编码机制则通过编码位置信息，更加准确地理解输入序列的顺序关系。大语言模型利用 Transformer 的强大建模能力，采用大规模无监督预训练的方式学习文本序列的表示，最终生成具有语法合理性和语义连贯性的文本。在大语言模型的训练过程中，通常使用交叉熵损失（cross-entropy）函数来衡量模型生成的文本序列与真实文本序列之间的差异。给定一个文本序列的真实分布和模型生成的文本序列的预测分布，损失函数定义如下：

$$\text{Cross-Entropy}(P, Q) = -\sum_x P(x) \log(Q(x)) \quad (1)$$

其中， x 表示序列中的每个元素， $P(x)$ 是真实分布中 x 的概率， $Q(x)$ 是模型预测 x 的概率。

上下文是大语言模型获得高精度结果所需的关键信息。上下文长度和大模型解答能力之间的关系类似正态分布，如图5所示。过短的上下文存在背景知识不足、信息歧义等诸多限制，而随着文本长度的增加，大语言模型难以在非常稀疏的信息分布中捕捉重要的上下文信息，在处理长程依赖时可能会出现梯度消失或梯度爆炸问题。

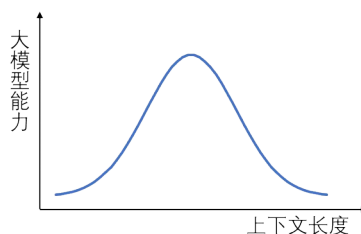


图5 大模型能力与上下文长度的关系示意

Figure 5 Schematic diagram of the relationship between large model capabilities and context length

因此，将输入序列按照最优的上下文 token 数进行切分，将最大化保证大模型的解答能力，但如何找到该最优解，尚未完全获得探索。假设

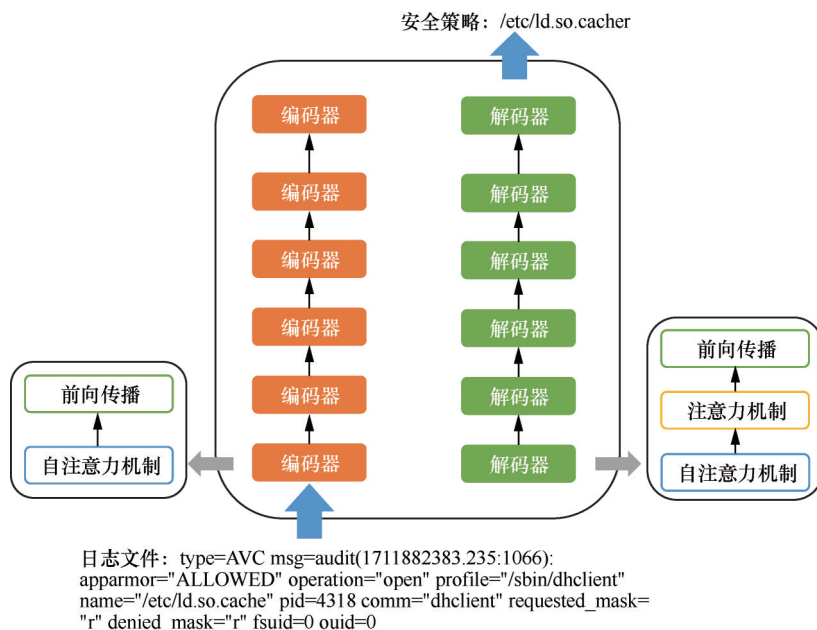


图4 Transformer 结构示意图

Figure 4 Schematic diagram of the transformer architecture

输入序列为 S ，按照设定的token数。将其划分为 l 段，即 $S = (s_1, s_2, s_3, \dots, s_l)$ ，本文将其建模为一个优化问题：

$$J(l) = \text{Eval}(\text{Concat}(\sum_{i=0}^l \text{LLM}(s_i)), G) \quad (2)$$

其中， $\text{LLM}(\cdot)$ 表示大语言模型， $\text{Concat}(\cdot)$ 表示将大语言模型的所有输出结果拼接， G 表示理想的文本序列（可以来自数据集或者真人解答）， $\text{Eval}(\cdot)$ 表示对于生成文本和理想文本的评价函数（如对比损失（contrastive loss），KL散度损失（KL divergence loss）等）。

因此，本文的目标是找到最优的上下文切分段落数 $l^* = \arg \max_l J(l)$ 。为了防止陷入局部最优解，结合模拟退火算法的原理求解 l^* ，具体如算法1所示。

算法1 基于模拟退火的最优上下文划分算法

输入 S : 输入序列; r : 降温系数; T_0 : 初始温度; $T_{\min}$: 下限温度; l_0 : 初始切分段落数; $l_{\max}$: 最大切分段落数; n : 状态数; k_1, k_2 : 比例系数

输出 l^* : 最优的划分段数

```

1: Initialization  $T$ ,  $l$  as  $T_0$ ,  $l_0$ 
2: while  $T > T_{\min}$  do
3:   for  $i=1$  to  $n$  do
4:      $l_{\text{new}} = l + \text{randomInt}(k_1, k_2) * T$  // 获得新的划分段落
5:      $dE = J(l_{\text{new}}) - J(l)$ 
6:     // 按照式(2)计算 $J(l_{\text{new}})$ 和 $J(l)$ 
7:     if  $dE \geq 0$  then
8:        $l_{\text{new}} = l$ 
9:     else
10:      if  $\exp(-dE/T) > \text{random}(0,1)$  then
11:         $l_{\text{new}} = l$ 
12:      end if
13:    end for
14:     $T = r * T$  // 降温退火
15:  end while
```

通过在多个日志文件上测试发现， l^* 的选取与文件长度有关，单次送入的token数取6 000~9 000时，大模型的推理效果最优。基于此，本

文将过滤后的日志文件切分若干子段，作为最终输入大语言模型的日志信息。

3.2 静态分析模块

3.1节获取的日志信息中仍然可能遗漏目标程序的部分资源访问行为。为进一步提高覆盖率，在静态分析模块中利用多种静态分析工具，向大语言模型追加额外视图以补充目标程序的潜在行为。

AppArmor的安全策略规则分为6类，因此本模块的设计目的即从目标程序中提取出每类规则相关的系统调用，分析处理得到部分安全规则。为了确保静态分析结果的完整性，将在对目标程序的静态分析过程中递归地分析其依赖的外部动态链接库中的函数。具体来说，在静态分析时首先会使用readELF工具导出目标程序使用的所有动态链接库函数，然后分析这些函数所对应的动态链接库。对于基础的标准系统库，如libc.so，本文的静态分析方法将事先分析出所有函数对应的系统调用类型；对于非标准的本地动态链接库，分析的方法则和对目标程序本身类似。

下面逐一介绍如何利用静态分析结果提取6类AppArmor策略规则。

1) File规则：AppArmor中的文件规则涉及目标程序运行过程中对特定文件目录的访问和读写权限。本文使用IDAPython对目标程序进行数据流分析，追踪Open系统调用参数中文件名字符串的数据流动方向，确定应用程序可能会访问哪些文件路径，将其添加到文件规则中。

2) Capabilities规则：AppArmor中的Capabilities规则基于Linux对Capabilities的相关定义^[13]。每种Capability都涉及一种或多种系统调用的使用限制。因此，首先使用二进制分析工具Angr^[14]对目标程序进行函数边界识别和跳转解析等静态分析，过滤出应用程序中包含的系统调用的类型。之后与Linux对Capabilities规则的定义手册进行对应，以此确定目标程序的Capabilities规则。

3) Network规则：AppArmor的Network规则限制目标程序所有基于socket系统调用的操作。因此，使用IDA Pro反汇编目标程序，分析socket系统调用的各个参数，依次是协议簇、套

接字类型和协议类型，以此确定 Network 规则。

4) Dbus 规则：在 AppArmor 的安全策略中，涉及 Dbus 的相关规则建立在目标程序使用 Dbus 相关库函数的基础之上。沿用上面的方法，仍然借助反汇编工具 IDA Pro 提取此类规则，通过分析目标程序中 Dbus 相关库函数的调用和参数情况，确定目标程序的 Dbus 规则。

5) Ptrace 规则：根据 Linux 官方文档^[15]对 proc 的相关说明，对 proc 目录下一些特定文件的访问需要 Ptrace 的特定权限，这一特点符合 AppArmor 中 Ptrace 相关规则的使用特征。本文方法对目标程序是否访问 proc 目录下这些特定文件进行过滤分析，由此确定 Ptrace 的相关规则。

6) Signal 规则：同 Dbus 规则类似，使用信号机制的目标程序的前提条件是使用了 Linux 标准库函数中的 Signal 相关函数，并且添加了与对应信号相关的处理函数。本文对 Signal 规则的分析方法也建立在使用 IDA Pro 工具分析目标程序中 Signal 函数的调用情况，可以得出 Signal 规则。

最终将这 6 类规则进行整合，得出静态分析这一步骤的结果——部分策略规则，并在后续步骤中将结果提供给大语言模型进行处理。

3.3 策略生成模块

策略生成模块负责利用上述预处理后的日志以及静态分析得到的部分安全策略规则，引导大语言模型为目标应用程序生成高质量的 AppArmor 安全策略。虽然通用大语言模型在理解和转换自然语言方面表现出色，但它们可能缺乏将程序行为描述转换为精确安全策略所需的特定领域知识。因此，在这一模块中，利用少量可用的二元组（审计日志—默认安全策略）以及 Abstractions 预定义的通用规则集合，为大语言模型补充任务直接相关的上下文，以实现更精简、更细粒度的策略生成。

少样本学习（few-shot learning）是一种机器学习方法，旨在让模型从有限的样本数据中学习并做出准确的预测。由于编写安全策略依赖的领域知识复杂烦琐，且与具体威胁场景高度相关，使用少样本学习方法有助于显著提高 AppArmor 安全策略的生成效率和准确性，而无须设计烦琐且片面的规则。利用少量可用的审计日志，通过

人工匹配找到它们与安全专家编写的 AppArmor 默认安全策略之间的对应关系，作为样本输入大语言模型。这一步骤虽然需要人工的参与，但相比完全手动编写安全策略，工作量大大减少。大语言模型通过对这些高质量但数量有限的样本对进行归纳总结，可以隐式地提炼出应用程序行为模式与安全策略之间的映射关系，自动学习到从应用程序行为日志生成相应的安全策略的任务的特定知识。

具体而言，策略生成过程首先将少量样本和 Abstractions 预定义的通用规则集合传入大语言模型完成少样本学习，并且掌握预定义规则的大模型可被视为掌握了一定的知识和经验，因而能将后续输入的 AppArmor 安全事件日志有效地转化为相应策略规则，并与静态分析结果和预定义通用规则合并生成目标应用程序的完整安全策略。

4 实验与评价

下面所设计的基于大语言模型的 AppArmor 安全策略自动生成方法加以实现和验证，并对该方法进行测试，结合测试结果进行分析和评价。

4.1 实验内容及具体实现

在 Ubuntu 18.04.6 LTS 环境中基于 Apparmor-utils 2.12-4ubuntu5.3 软件包实现了本文中 AppArmor 安全策略自动生成方法的原型。具体是通过 Shell 脚本来实现收集和预处理与 AppArmor 相关的日志，以及通过使用 GPT4 模型来自动生成安全策略。本文中目标应用程序的 AppArmor 相关日志储存在 /var/log/audit/audit.log 中，在运行目标应用前，脚本读取当前日志中最后一条的“msg=audit (……)”以此作为标记，收集日志将从该标记后的位置开始读取并收集日志。当收集完日志后，将按照 2.2 节中所述对日志进行预处理，得到最终要输入大模型的日志数据。

本文首先创建了一个测试的目标应用程序。其中涉及 AppArmor 安全策略中的 6 类规则。编译运行该测试应用，得到对应的日志。首先将日志直接输入 GPT4 模型，得到大模型根据日志生成的 AppArmor 安全策略，如图 6 所示。对该测试应用采用基于大语言模型的 AppArmor 安全策略自动生成方法，得到对应的 AppArmor 安全策略如图 7 所示。

```

profile /home/jak/local/test {
    /etc/ld.so.cache r,
    /lib/x86_64-linux-gnu/libdbus-1.so.3.19.4 rm,
    /lib/x86_64-linux-gnu/libc-2.27.so rm,
    /lib/x86_64-linux-gnu/libpthread-2.27.so rm,
    /lib/x86_64-linux-gnu/libsystemd.so.0.21.0 rm,
    /lib/x86_64-linux-gnu/librt-2.27.so rm,
    /lib/x86_64-linux-gnu/libzma.so.5.2.2 rm,
    /usr/lib/x86_64-linux-gnu/liblz4.so.1.7.1 rm,
    /lib/x86_64-linux-gnu/libgcrypt.so.20.2.1 rm,
    /lib/x86_64-linux-gnu/libdl-2.27.so rm,
    /lib/x86_64-linux-gnu/libgpg-error.so.0.22.0 rm,
    /etc/machine-id r,
    /run/user/1000/bus wr,
    /tmp/test/ c,
    /tmp/test/test1 wc,
    /bin/ls px,
    /usr/bin/dbus-launch px,
    /usr/bin/dbus-daemon px,
    network inet stream
    ptrace (trace) peer="/home/jak/local/test",
    dbus send
    bus=session
    path=/org/freedesktop/DBus
    interface=org.freedesktop.DBus
    member=Hello
    peer=(label="/home/jak/local/test/null-/usr/bin/dbus-launch/null-/usr/bin/dbus-daemon"),
}

```

图6 大模型根据日志生成的安全策略

Figure 6 Security policy directly generated by the large mode

由图7可知，大模型仅根据日志生成的安全策略不够完整，除相关文件操作规则不全之外，还缺少network、signal、capability规则，并且通过本文方法生成的安全策略还包含（include）预定义的通用规则集合，可以减少策略文件的复杂性。

对该测试目标应用程序进行静态分析和源码分析，结合AppArmor安全策略的语法规则和专家知识，对比大模型仅通过日志生成的安全策略与本文方法生成的安全策略，对比该目标应用程序所需6类规则的完备性，结果如图8所示，可以看出仅基于日志生成安全策略对几类规则都有不足，甚至完全缺失。

之后，本文选择一组共计18个应用程序进行评估，基于以下关键标准。

```

#include <tunables/global>
profile /home/jak/local/test flags=(complain) {
    #include <abstractions/base>
    #include <abstractions/dbus>
    #include <abstractions/dbus-session>
    /etc/ld.so.cache r,
    /lib/x86_64-linux-gnu/* rm,
    /usr/lib/x86_64-linux-gnu/* rm,
    /etc/machine-id r,
    /dev/null rw,
    /tmp/* rwk,
    /run/user/1000/bus rw,
    /run/user/1000/gdm/Xauthority r,
    @{PROC}/[0-9]*/fd/ r,
    /etc/passwd r,
    /etc/group r,
    /etc/nsswitch.conf r,
    /usr/share/zoneinfo/* r,
    network inet stream,
    network inet6 stream,
    network bind stream ip4 127.0.0.1:18080,
    network connect stream ip4 127.0.0.1:18080,
    /bin/ls px,
    /usr/bin/dbus-launch px,
    /usr/bin/dbus-daemon px,
    dbus (send)
        bus=session
        path=/org/freedesktop/DBus
        interface=org.freedesktop.DBus
        member=Hello
        peer=(label="/home/jak/local/test/null-/usr/bin/dbus-launch/null-/usr/bin/dbus-daemon"),
    dbus (receive)
        bus=session
        path=/org/freedesktop/DBus
        interface=org.freedesktop.DBus
        member=Hello
        peer=(label=unconfined),
    owner @{(HOME)}/.dbus/session-bus/* rwk,
    /home/jak/.dbus/ rw,
    /home/jak/.dbus/session-bus/* rw,
    /home/jak/.dbus/session-bus/* rw,
    /home/jak/.dbus/session-bus/* rw,
    /sys/kernel/security/apparmor/access rw,
    /proc/sys/kernel/cap_last_cap r,
    ptrace (trace) peer=@(profile_name),
    signal(receive) set = (SIGINT)
    /usr/share/dbus-1/* r,
    /etc/dbus-1/* r,
    /var/lib/snapd/dbus-1/* r,
    capability dac_read_search,
    capability sys_ptrace,
    capability chown,
}

```

图7 本文提出方法生成的安全策略

Figure 7 Security policy generated by the method proposed in this study

① 应用程序应在其领域内占有重要的市场份额或展现出显著的技术影响；

② 应用程序应具备较长的开发历史或较大的项目规模，并且能够找到较为完善的专家撰写的安全策略作为基准供参考

③ 应用程序的默认安全策略中覆盖了AppArmor安全策略规则中的所有类别。

基于这些准则，本文选定了atd、biosde-

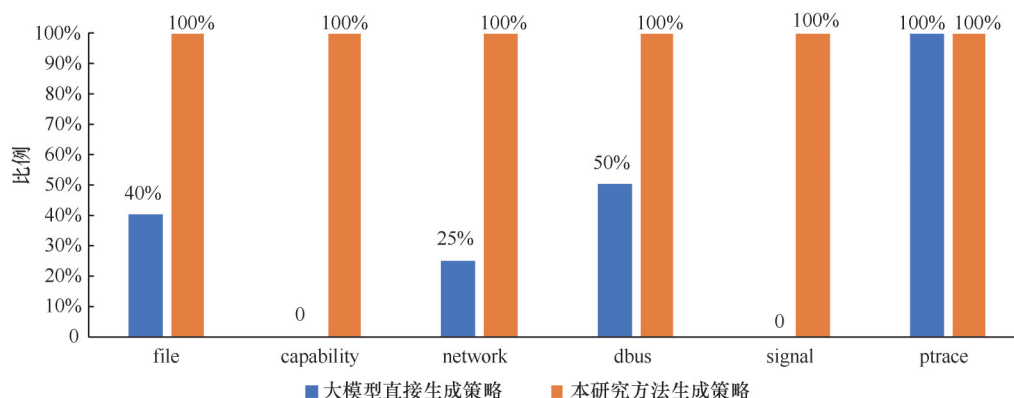


图8 安全策略完备性对比结果

Figure 8 Comparison results of security policy completeness

code、cgrulesengd、chage、dhclient、dig、dumpcap、fping、gpasswd、hddtemp、kerneloops、last、nethogs、nslookup、passwd、pkexec、pkttyagent、top 作为实验的目标应用程序。

4.2 实验评估

本节首先使用本文方法为上节 18 个应用程序生成安全策略，同时也通过大模型根据日志为 18 个应用程序生成安全策略。为验证正确性，18 个应用程序均部署了空白的安全策略配置文件，并确保 AppArmor 处于 enforce 模式，因为 AppArmor 遵循的是白名单机制，此时运行应用程序，任何操作都会被当成违规或错误操作，所以都被拒绝执行。之后分别使用本文生成的安全策略和仅根据日志生成的安全策略进行部署，再次运行 18 个应用，以此验证所有应用程序的使用是否受影响，对比结果如表 1 所示。

接着将采用本文方法为真实世界中 18 个应

表 1 正确性对比结果

应用程序	默认策略	本研究方法生成策略	大模型直接生成策略
atd	√	√	√
biosdecode	√	√	√
cgrulesengd	√	√	×
chage	√	√	×
dhclient	√	√	√
dig	√	√	×
dumpcap	√	√	√
fping	√	√	√
gpasswd	√	√	√
hddtemp	√	√	√
kerneloops	√	√	√
last	√	√	×
nethogs	√	√	×
nslookup	√	√	√
passwd	√	√	√
pkexec	√	√	√
pkttyagent	√	√	√
top	√	√	√

用程序自动生成的安全策略、大模型仅根据日志生成的安全策略与 AppArmor 预置的现有默认安全策略进行比较。默认策略是由安全专家参与制定的，既具有权威性又有经典性。考虑大多数用户不具有更改这些默认安全策略的知识储备和能力，所以基本上都用默认安全策略，因此这种比较方法是有意义的。除前文中所提对比正确性，还对精简性和生成策略数量进行了对比，精简性即为所包含的预定义通用规则集合的条数，生成策略数量为生成安全策略与默认策略的百分比。对比结果如表 2 和表 3 所示。其中表 3 展示了自动生成策略与默认策略的数量比较。策略数量比例计算公式为：策略数量比例 = (自动生成策略数量) / (默认策略数量) × 100%。

表 2 精简性对比结果

应用程序	默认策略/条	本文方法生成策略/条	大模型直接生成策略/条
atd	3	3	0
biosdecode	1	1	0
cgrulesengd	2	2	0
chage	3	3	0
dhclient	2	2	0
dig	3	3	0
dumpcap	4	4	0
fping	2	2	0
gpasswd	3	3	0
hddtemp	1	1	0
kerneloops	3	3	0
last	4	4	0
nethogs	2	2	0
nslookup	3	3	0
passwd	5	5	0
pkexec	6	6	0
pkttyagent	5	5	0
top	4	4	0

分别针对每个应用的 3 个安全策略配置文件中的安全策略进行对比，比较安全策略的正确性、精简性和生成策略数量。由表 3 中数据可得，仅基于日志生成策略数量百分比有的超过

100%，经分析对比可知，主要是部分安全策略规则过于细化，且没有引入预定义通用规则集合，导致条数过多且复杂度较高，不够精简。本文方法生成策略数量相对而言百分比降幅明显，主要是通过上下文学习和导入Abstraction数据，有效降低复杂度，简化策略的编写和管理。仍高于100%的主要是根据日志生成更加细化，而专家编写的默认策略会更加粗粒度，通过通配符提高权限。比较对于文件、网络、能力等不同事件的具体策略，经过对比发现，默认的安全策略基本上都可以对应上自动生成方法所生成的安全策略，并且自动生成的安全策略更加细粒度，所以绝大多数自动生成的安全策略条数比默认数量更多。不同安全策略条数对比如图9所示。仅有两个应用程序Pkexec和Top通过本文所提出方法得到的安全策略生成策略数量百分比低于100%，经分析可能是获取日志仍不够完整和全面。

在第3节的实验评估中，证明了本文提出的基于大语言模型的AppArmor安全策略自动生成方法，成功满足了正确性、生成策略数量和精简性的标准。本文在第1节介绍的相关工作中已经讨论了多种现有的通过分析应用程序的依赖性 or 二进制文件来自动生成安全策略的研究。与这些研究不同，本文采用基于大语言模型结合静态分析来自动生成安全策略的方法，实现了在AppArmor策略生成过程中的自动化，使用户的操作更加便捷。

表3 生产策略数量比例对比结果

Table 3 Comparison results of the quantity ratio of production policies

程序	本文方法生成策略	大模型直接生成策略
atd	100.00%	177.27%
biosdecode	100.00%	166.67%
cgrulesengd	123.53%	200.00%
chage	172.73%	336.36%
dhclient	156.25%	237.50%
dig	184.62%	323.08%
dumpcap	132.00%	216.00%
fping	400.00%	700.00%
gpasswd	180.00%	346.67%
hddtemp	142.86%	257.14%
kerneloops	200.00%	700.00%
last	128.57%	371.43%
nethogs	283.33%	333.33%
nslookup	200.00%	283.33%
passwd	175.00%	400.00%
pkexec	79.31%	162.07%
pktyagent	190.00%	370.00%
top	50.00%	130.00%

5 结束语

本文介绍了基于大语言模型的AppArmor安全策略自动生成方法，目的在于优化操作系统安全性的关键支撑——安全策略生成过程。通过应用静态分析技术，本文方法能够实现对应用程序

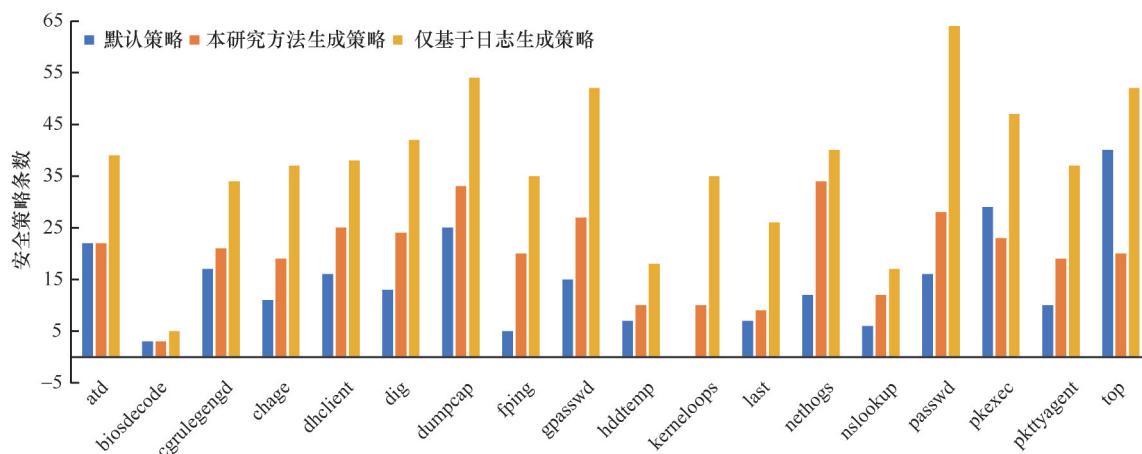


图9 安全策略条数对比

Figure 9 Comparing the number of security policies

的深入分析,从而获取部分安全策略。通过实现尽可能全面覆盖代码,获取详尽的日志数据作为生成安全策略的基础,通过少样本学习的大模型结合静态分析结果、日志以及预定义通用规则集合来自动化生成安全策略,有效减少了对安全专家的依赖,同时降低了安全策略配置的复杂性和主观性。实验结果表明,本文提出的自动生成策略方法不仅能够产生完整且精简的安全策略,而且这些策略不会干扰应用程序的正常运行。这说明了该方法在简化安全策略生成流程和提升策略效果方面的显著优势,展现了其在操作系统安全管理领域的实际应用潜力。

综上所述,本文通过自动化安全策略的生成,为降低安全专家人工参与安全策略生成的工作负担和提高系统安全性提供了新的思路。未来工作将探索如何进一步提高该方法的效率以及降低该方法的复杂度,进一步验证该方法的通用性和有效性,以期为操作系统安全提供更加全面和智能的自动化生成安全策略机制。

参考文献:

- [1] HAN Z, CHENG L, ZHANG Y, et al. Operating system security policy hardening via capability dependency graphs[C]//Proceedings of Information Security Practice and Experience: 11th International Conference (ISPEC 2015), Beijing, China: Springer, 2015: 3-17.
- [2] LACHMUND S. Auto-generating access control policies for applications by static analysis with user input recognition[C]//Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems. New York: ACM, 2010: 8-14.
- [3] CENTONZE P, FLYNN R J, PISTOIA M. Combining static and dynamic analysis for automatic identification of precise access-control policies[C]//Proceedings of the Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007). Piscataway: IEEE Press, 2007: 292-303.
- [4] RAUTER T, HÖLLER A, KAJTAZOVIC N, et al. Towards an automated generation of application confinement policies with binary analysis[C]//Proceedings of the 2015 International Symposium on Networks, Computers and Communications (ISNCC). Piscataway: IEEE Press, 2015: 1-6.
- [5] SCHREUDERS Z C, PAYNE C, MCGILL T. Techniques for automating policy specification for application-oriented access controls [C]//Proceedings of the 2011 Sixth International Conference on Availability, Reliability and Security. Piscataway: IEEE Press, 2011: 266-271.
- [6] SANDERS M W, YUE C. Minimizing privilege assignment errors in cloud services[C]//Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy. New York: ACM, 2018: 2-12.
- [7] SANDERS M W, YUE C. Mining least privilege attribute based access control policies[C]//Proceedings of the 35th Annual Computer Security Applications Conference. New York: ACM, 2019: 404-416.
- [8] WANG R W, ENCK W, REEVES D, et al. EASEAndroid: automatic policy analysis and refinement for security enhanced android via large-scale semi-supervised learning[C]//Proceedings of the 24th USENIX Conference on Security Symposium. New York: ACM, 2015: 351-366.
- [9] NADKARNI A, ENCK W, JHA S, et al. Policy by example: an approach for security policy specification[EB]. arXiv Preprint arXiv: 1707.03967, 2017.
- [10] WANG R W, AZAB A M, ENCK W, et al. SPOKE: scalable knowledge collection and attack surface analysis of access control policy for security enhanced android[C]//Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. New York: ACM, 2017: 612-624.
- [11] 马铭鑫, 史国振, 王亚琼, 等. 适用于分布式系统的多级安全访问控制策略[J]. 网络与信息安全学报, 2017, 3(8): 32-38.
- [12] MA M X, SHI G Z, WANG Y Q, et al. Multilevel secure access control policy for distributed systems[J]. Chinese Journal of Network and Information Security, 3(8): 32-38. 2017.
- [13] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[EB]. arXiv Preprint arXiv: 1706.03762, 2017.
- [14] HALLYN S E, MORGAN A G. Linux capabilities: making them work[J]. Linux Symposium, 2008, 1: 8.
- [15] WANG F, SHOSHITAISHVILI Y. Angr - the next generation of binary analysis[C]//Proceedings of the 2017 IEEE Cybersecurity Development (SecDev). Piscataway: IEEE Press, 2017: 8-9.
- [16] PETERSEN R, VUELAPLUMA S L. Linux: manual de referencia [M]. Madrid: McGraw-Hill, 2001.

作者简介



沙倚天 (1985-), 女, 江苏南京人, 国网江苏电力有限公司南京供电分公司高级工程师, 主要研究方向为电力信息通信、网络安全。



刘少君 (1980-), 男, 江苏南京人, 国网江苏电力有限公司南京供电分公司高级工程师, 主要研究方向为电力信息通信、网络安全。



钱欣（1989-），女，江苏南京人，国网江苏电力有限公司南京供电分公司高级工程师，主要研究方向为网络安全及数字化应用。



陈鹏（1995-），男，江苏泰州人，国网江苏电力有限公司南京供电分公司工程师，主要研究方向为电力信息通信。



吴越（1995-），男，江苏沭阳人，国网江苏电力有限公司南京供电分公司工程师，主要研究方向为电力信息通信专业。



刘行（1986-），男，江苏南京人，南京南瑞信息通信科技有限公司高级工程师，主要研究方向为电力信息通信、网络安全。