AUTOARMOR 解决两个问题：

1. 如何获得完整和细粒度的调用逻辑

提出了一种基于静态分析的请求提取机制，微服务可以从其源代码中提取所有可能的源代码调用。这种提取使用了服务间通信库的使用和语义模型来识别请求。之后，将收集调用的详细属性，用于细粒度访问控制。

1. 如何生成和更新访问控制策略。

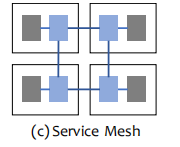
设计了一种新的基于图形的策略管理机制，它考虑了微服务的独特特征，并通过**权限图**接管访问控制策略的生成、更新和删除。更新： **设计集成到微服务的生命周期中，不需要对当前的应用程序代码和基础设施进行任何修改。**

**贡献：**

1. 提出了AUTOARMOR，第一个用于微服务的服务间访问控制的自动策略生成工具，它提高了当前服务级授权的可用性
2. 一种基于静态分析的请求提取机制，它使用程序切片和语义分析来提取具有详细的服务间调用逻辑
3. 一个基于图的策略管理机制，它将调用逻辑转换为细粒度的访问控制策略，并随着应用程序的发展继续更新它们。
4. 我们为两个应用最广泛的微服务基础设施实现了kubernetes和Istio的，并使用5个流行的微服务应用进行评估。在处理微服务的服务间访问控制的策略生成方面是可靠和实用的。

**背景：**

作为一个副作用，随着服务数量的增长，服务之间的调用关系变得很麻烦。为了解决这个问题，服务网格占据了舞台，并增强了微服务架构作为一个专用的通信基础架构层。它使用代理（图1(c)中的蓝色框）来管理微服务之间的所有网络流量，并透明地向服务间通信添加访问控制、流量管理和监控等功能。



在这种体系结构中，以前通过本地调用在单片应用程序中进行的通信现在可以通过网络进行调取，这就留下了潜在的攻击面。

虽然网络隔离在一定程度上提高了安全性，但通信通道仍然需要得到保护。目前采用两种方法保护业务间通信，一种是SSL/TLS，另一种是业务间访问控制。

**服务间访问控制：**

现有微服务大多基于容器运行，引入第三方机构会带来风险，现有调查显示，93%的容器并未为软件打漏洞补丁。因此，攻击者可以通过侵入相应的容器来破坏微服务。隐藏在受损服务的IP地址和证书后面，他们可以向其他微服务发送恶意请求，以发起攻击或窃取数据。当微服务共同完成复杂的功能时，它们自然的相互信任使得整个应用程序容易受到单一受损服务的影响

这就是服务间访问控制的切入点。通过指定服务的权限（即它们可以访问的资源），它可以规范微服务的行为并防止此类攻击

**策略生成间隙**。

现有较流行的微服务的访问控制机制：

策略是一个合法请求的列表，即一个白名单。由管理员发布后，策略将安装在相关服务对应的代理中。在运行时，代理根据已安装的策略验证每个传入请求，并返回授权结果以执行策略。

缺陷：手动配置 对于大规模微服务模型 不现实 服务升级可能会导致调用逻辑的变化，从而影响相关的访问控制策略。微服务的频繁迭代需要频繁的策略更新，这对于手动配置也具有挑战性。

**概述：**

危险模型：攻击者利用其容器中的漏洞来危及运行微服务。这些攻击者能够感知网络中的其他服务以及它们公开的api，也能够从被颠覆的服务发起任意请求。因此 对手可以对其他微服务进行非法访问

目标：具体来说，我们的目标是为服务自动生成最少特权的访问2控制策略，并随着应用程序的发展而保持它们的最新状态。

未考虑：由于本文关注的是服务间访问控制，因此抵御来自终端用户或针对主机或平台的攻击的安全策略超出了我们的范围。

安全条件：1.源代码良性，程序员不会恶意攻击 2. 可以获得微服务源代码

**关键问题：**

1. **寻找反映微服务正常行为的合适目标**

**定义 系统的正常行为 ：**

**现有的工作 集中在文档、系统日志和监控数据上。我们选择的目标必须完全和准确地反映服务的预期行为，这设置了方法的上界**

**Auto：微服务的代码是其行为的直接来源，比日志等其他文件更直接，其获取不需要完整的测试或预运行。当微服务向其他服务或外部网络启动与其代码不一致的请求时，视为意图违反，即恶意请求。**

1. **处理静态分析的固有限制。**

**直接静态分析 会导致状态爆炸。需要减少空间搜索同时确保完整和健全的结果。**

**解决：仅考虑与网络调用有关的代码，关注与服务间通信相关的程序切片，从而缩小分析空间。选择了5个流行的开源微服务应用程序，并观察用于服务间的协议和库**