usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION
30TH USENIX SECURITY SYMPOSIUM
网络安全研究国际学术论坛
International Forum for Security Research

微服务间访问控制策略的自动生成

Automatic Policy Generation for Inter-Service Access Control of Microservices

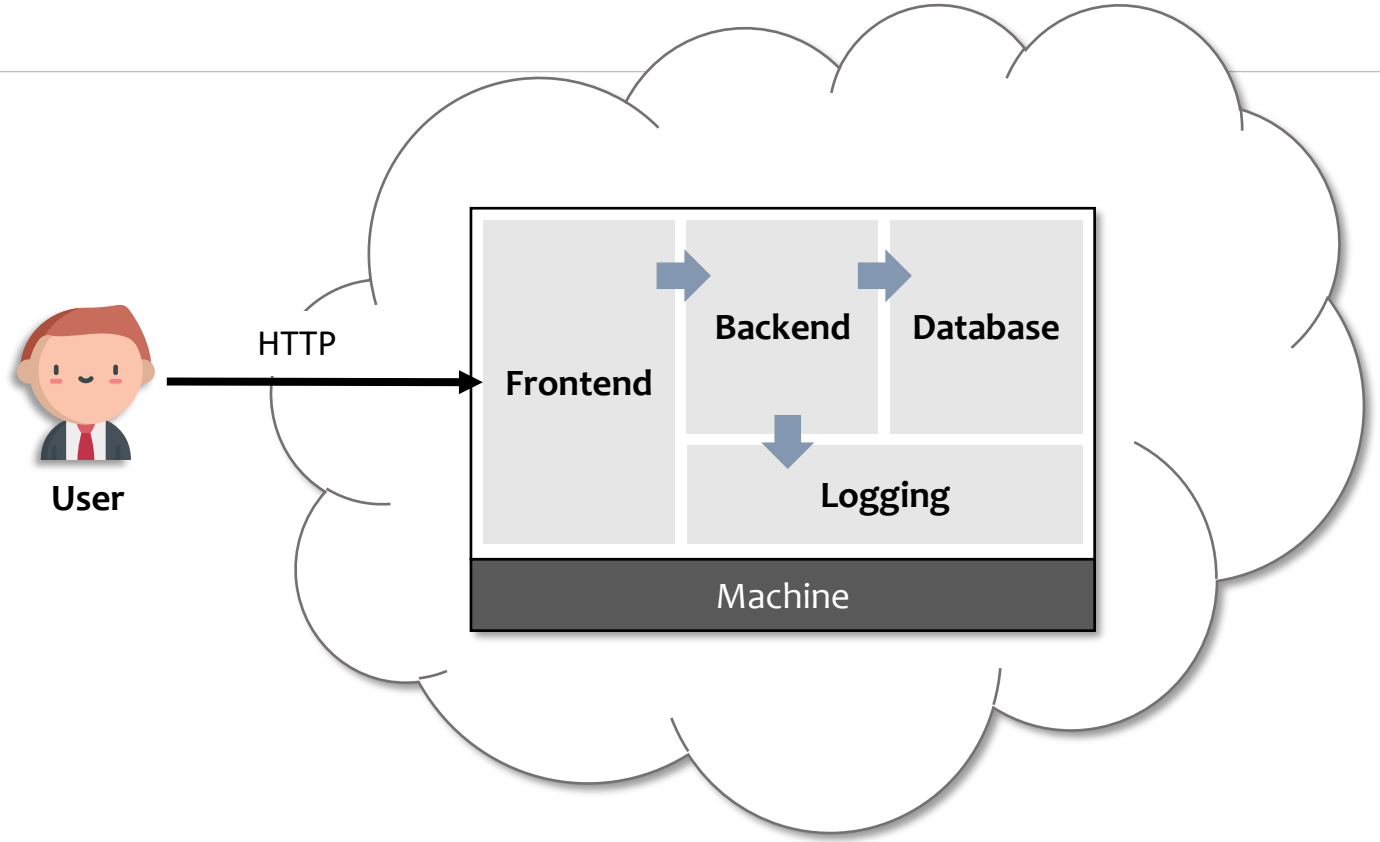
Xing Li[1,2], Yan Chen[2], Zhiqiang Lin[3], Xiao Wang[2], and Jim Hao Chen[2]

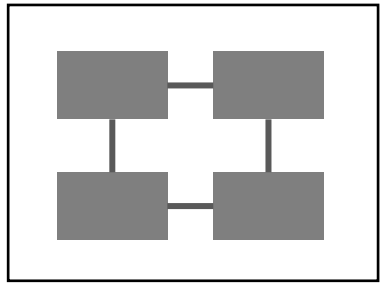
[1]Zhejiang University, [2]Northwestern University, [3]The Ohio State University

研究背景

云应用架构的演进

传统单体架构





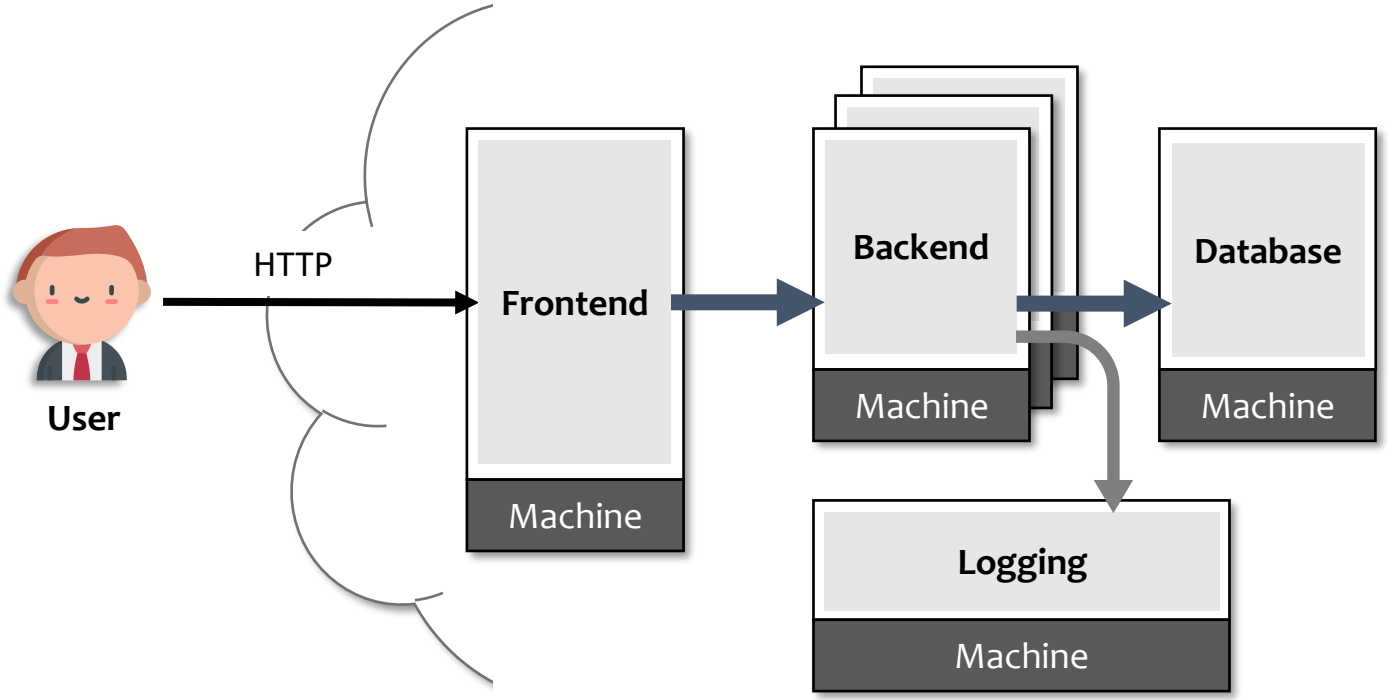
(a) 单体架构

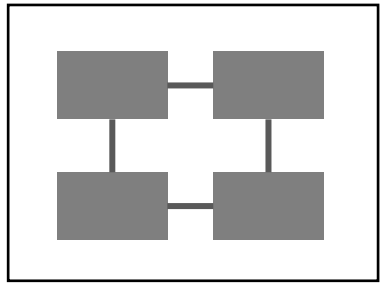
- 模块紧耦合
- 开发/测试/上线周期长
- 新功能迭代缓慢
- 不易维护和扩展

研究背景

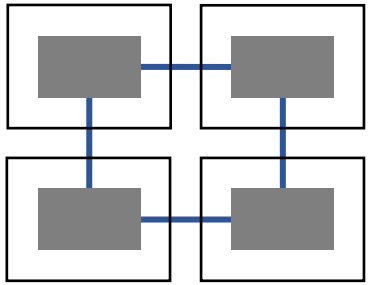
云应用架构的演进

微服务架构





(a) 单体架构



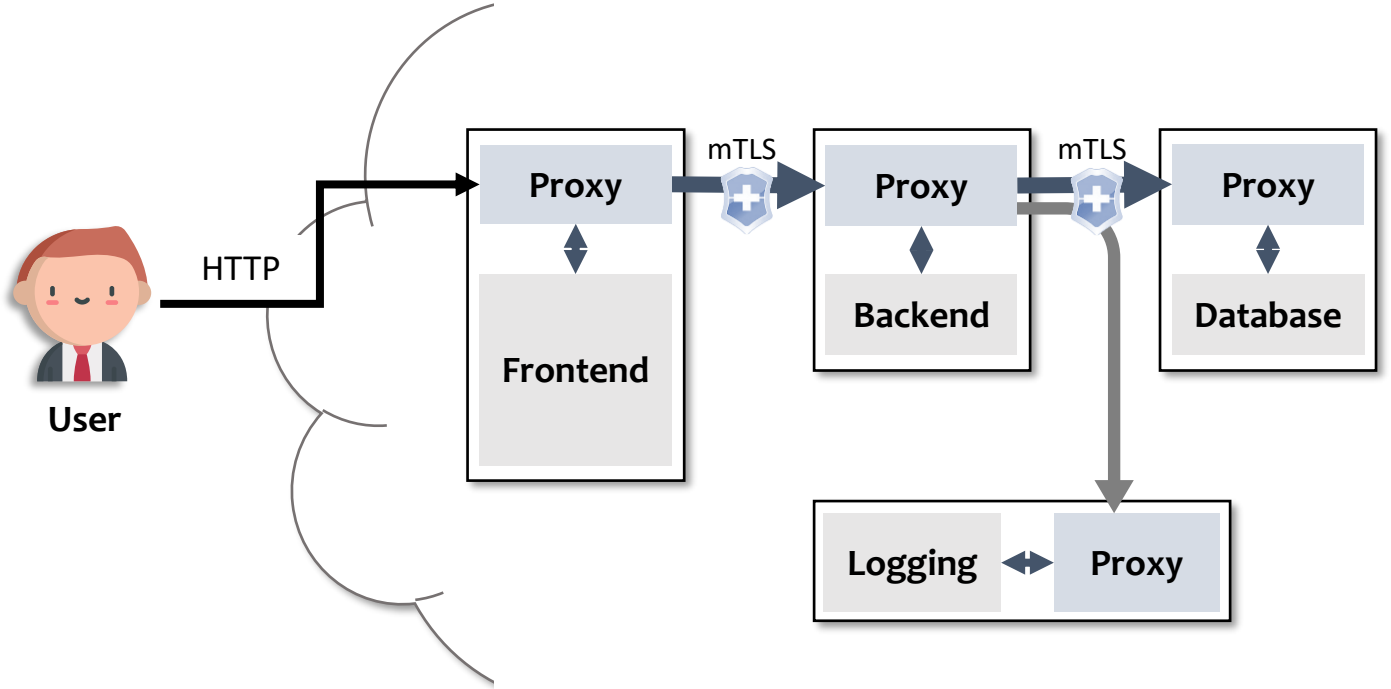
(b) 微服务架构

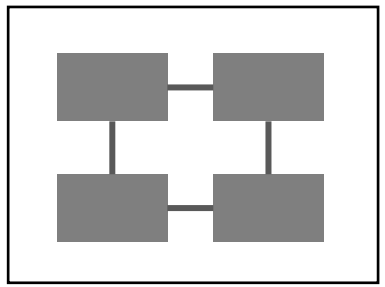
- 将模块按功能解耦
- 独立开发、部署
- 缩短开发/测试/上线周期
- 新功能迭代加快
- 易维护、易扩展

研究背景

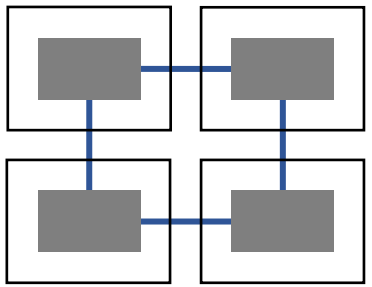
云应用架构的演进

服务网格

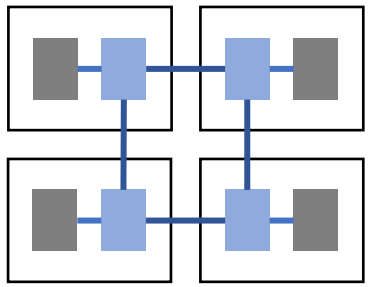




(a) 单体架构



(b) 微服务架构

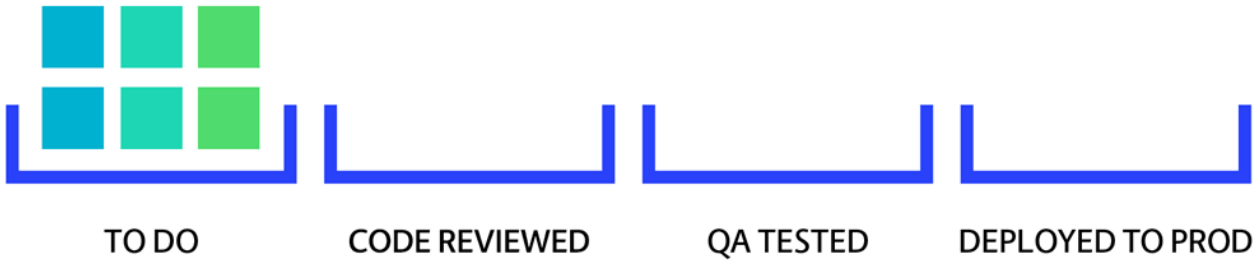


(c) 服务网格

- 统一管理服务间通信
- 透明地加入基础功能
- 可以专注于业务代码

研究背景

云应用架构的演进



微服务的生命周期

1. 持续集成/持续部署（Continuous Integration / Continuous Delivery，CI/CD）
2. 敏捷、渐进式的服务更新

微服务的发展现状

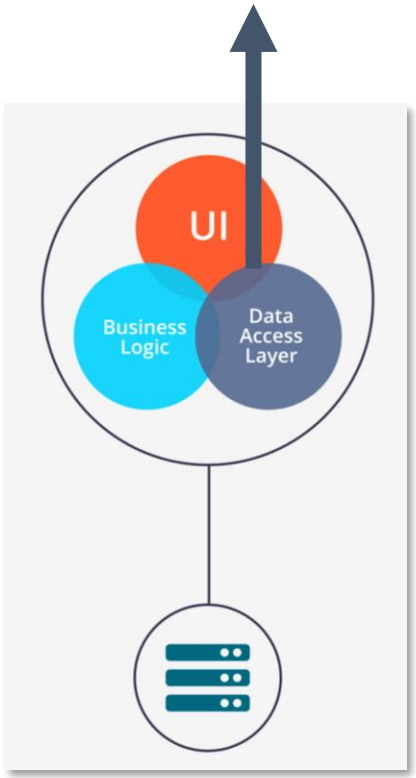
1. **获得了产业界的广泛应用**
   - 阿里巴巴、亚马逊、推特、Uber、eBay等公司均将软件系统升级为微服务架构
   - 2018年的调研报告显示，受访的353家企业中有**91%** 的公司正在使用或计划使用微服务架构
2. 流行的基础设施平台：Kubernetes 和 Istio

研究动机

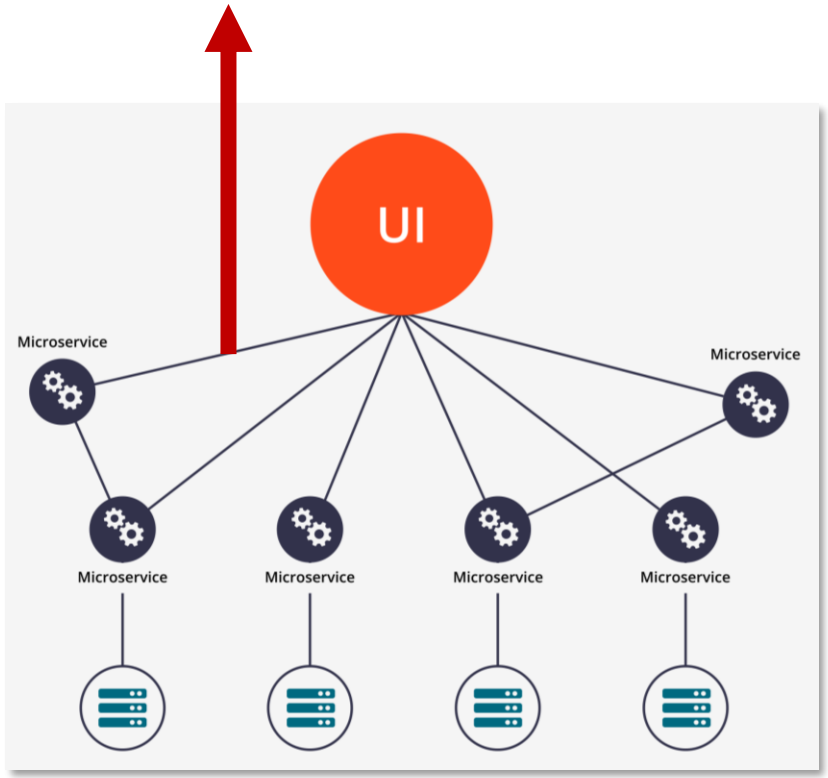
- **微服务间的通信方式带来新的攻击面**
- **不安全的容器image可能导致容器被渗透**

可信的程序内部调用

不可信的服务间网络通信



(a) Monolithic



(b) Microservice

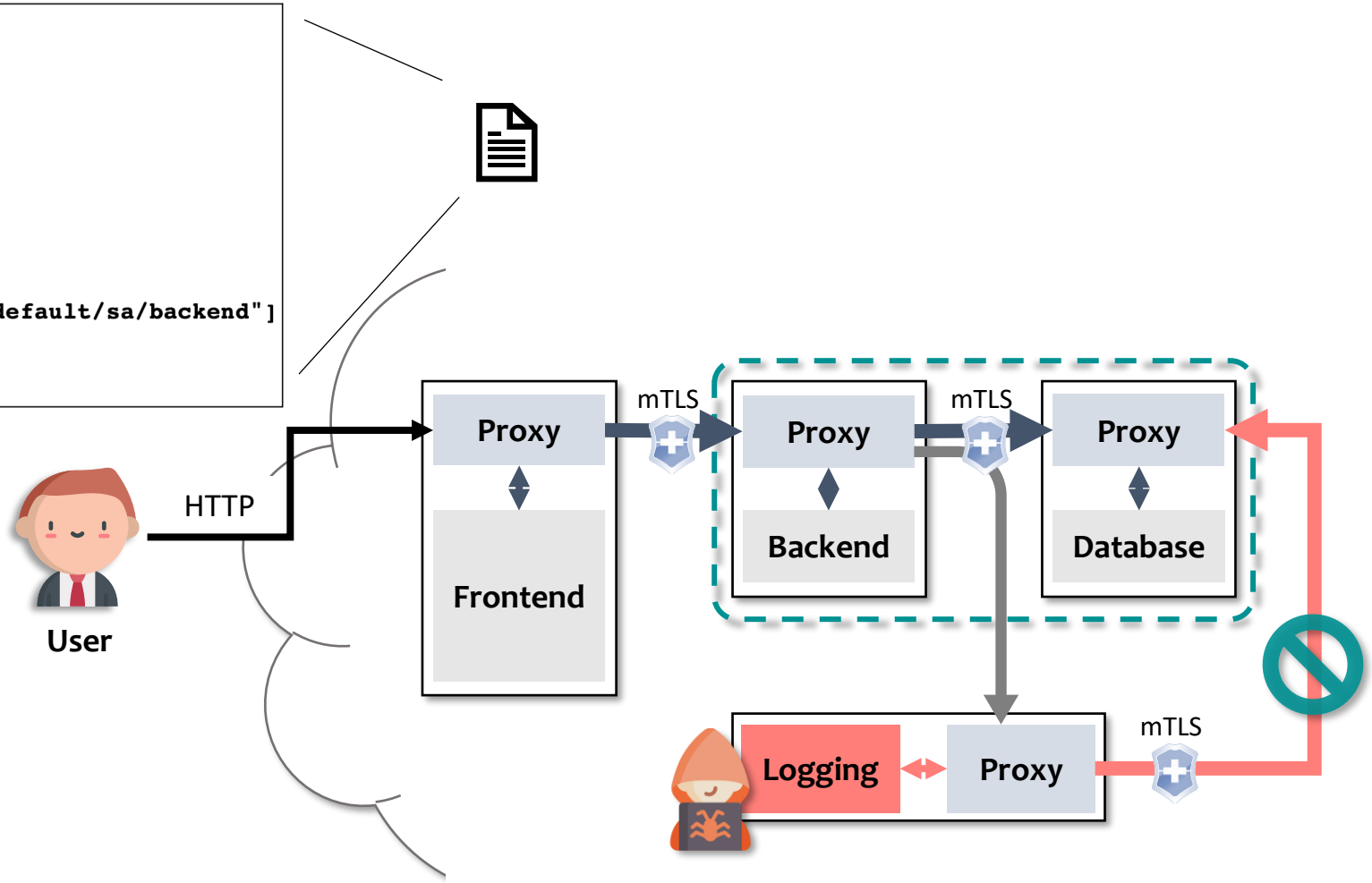
More than **92%** of the container images contain package vulnerabilitie [1].

[1] B. Tak, H. Kim, S. Suneja, C. Isci, and P. Kudva, “Security analysis of container images using cloud analytics framework,” in International Conference on Web Services. Springer, 2018, pp. 116–133

研究动机

- 威胁：被攻破的微服务可能通过恶意请求窃取数据或发起攻击
- 对策：服务间细粒度的访问控制

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: backend-v1-to-database
  namespace: default
spec:
  selector:
    matchLabels:
      app: database
  rules:
  - from:
    - source:
        principals:["cluster.local/ns/default/sa/backend"]
    to:
    - operation:
        ports: ["9000"]
```

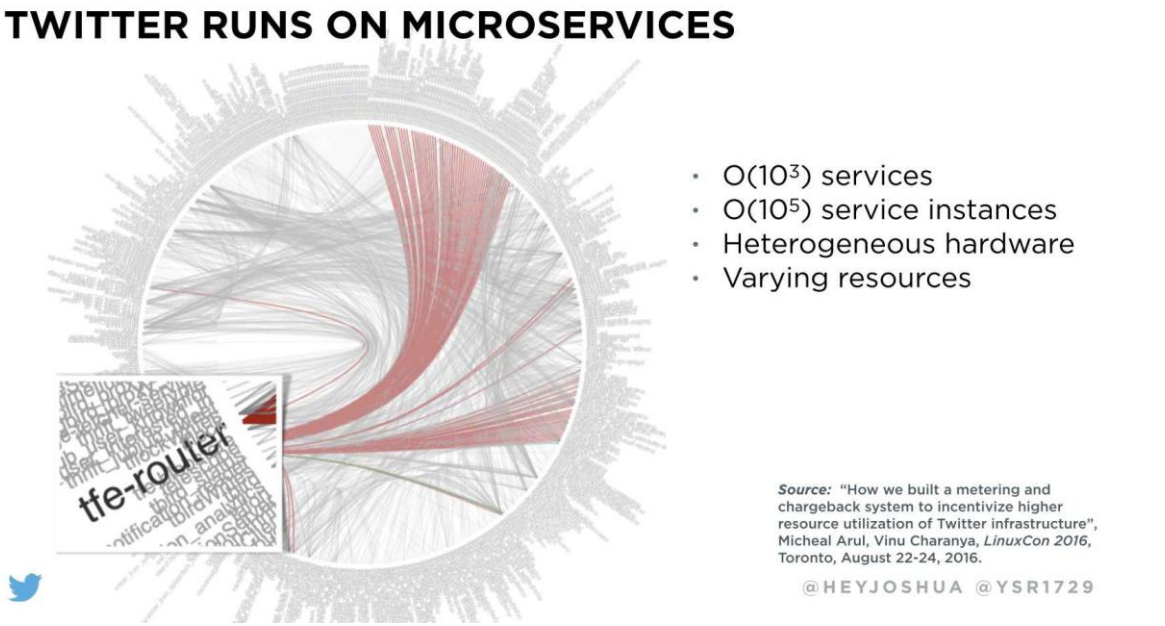


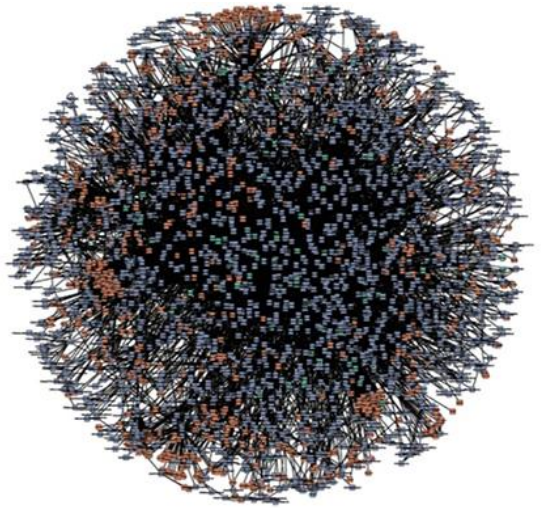
研究动机

微服务应用：规模庞大 + 频繁更新

- 手工配置访问控制策略？

  耗时、容易出错、不灵活



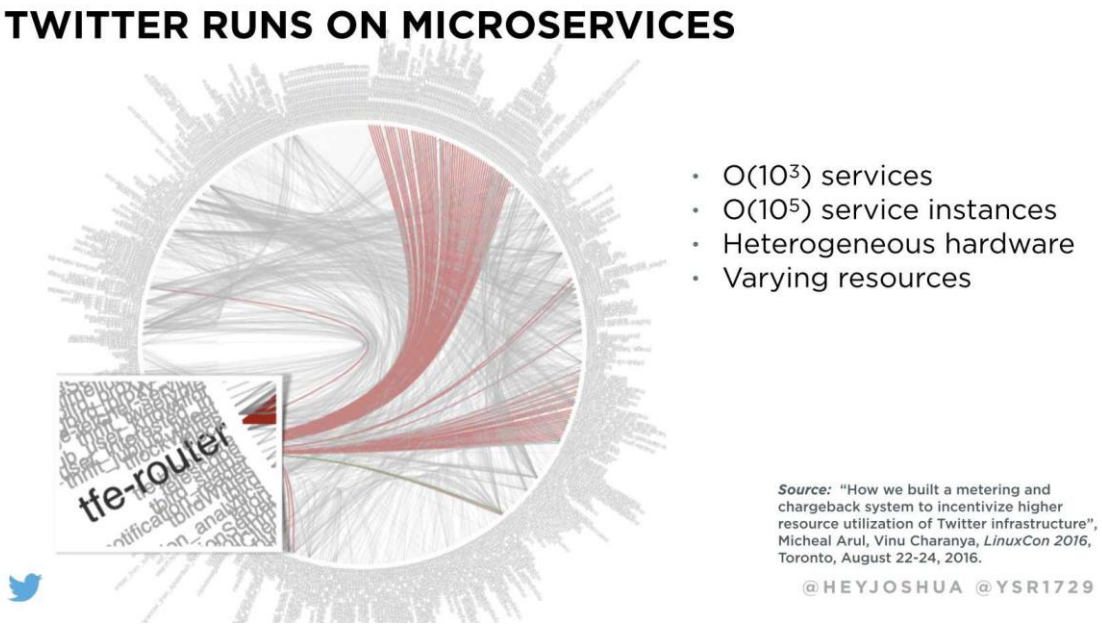


amazon.com

NETFLIX

研究动机

微服务应用：规模庞大 + 频繁更新



- 手工配置访问控制策略?

  耗时、容易出错、不灵活

- 分布式系统中的安全策略自动化方法?

方法	全面性	细粒度	敏捷性	可伸缩性
基于文档的方法	✘	✘	✓	✓
基于历史的方法	✘	✓	✓	✘
基于模型的方法	✓	✓	✘	✘

设计思路

微服务应用的特点

1. 单个微服务的**内部复杂度较小**
2. 单个应用内
   a. 微服务间的**调用方式相对统一**
   b. 涉及到的服务间调用协议和调用库**数量也十分有限**

Application	Protocol	# of Used Libraries	# of Initiated Requsets	Share of the Main Library
Bookinfo	HTTP	3	5	60%
	TCP	2	2	50%
Online Boutique	gRPC	2	20	95%
	TCP	1	7	100%
Sock Shop	HTTP	2	39	85%
	TCP	3	47	51%
Pitstop	HTTP	2	48	69%
	TCP	4	73	52%
Sitewhere	HTTP	1	4	100%
	gRPC	1	270	100%
	TCP	5	240	61%

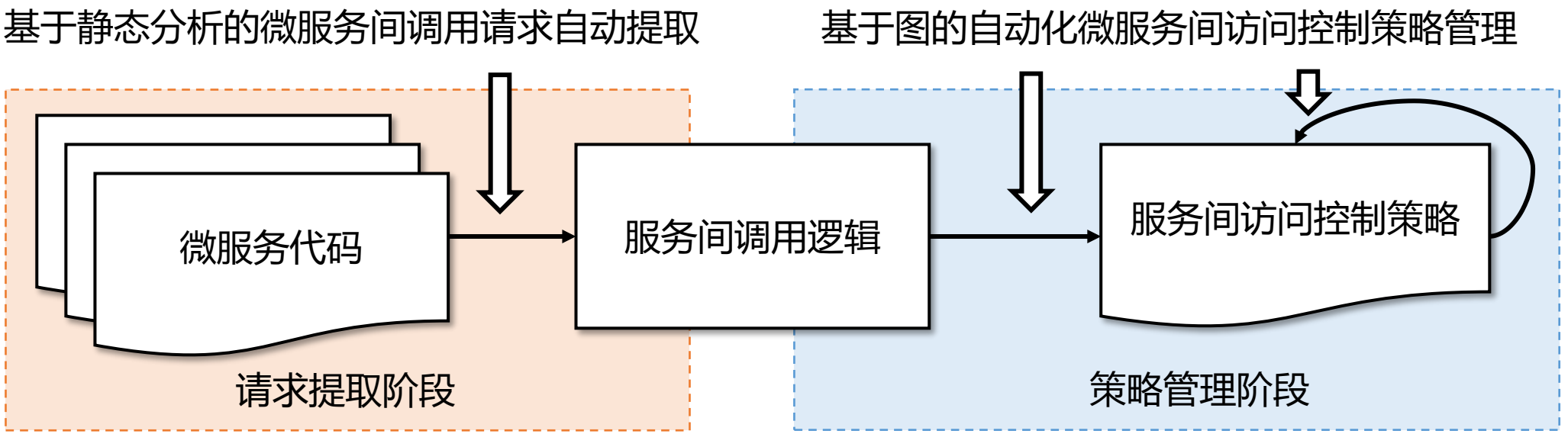
通过静态分析从微服务的代码中提取其正常系统行为

设计思路

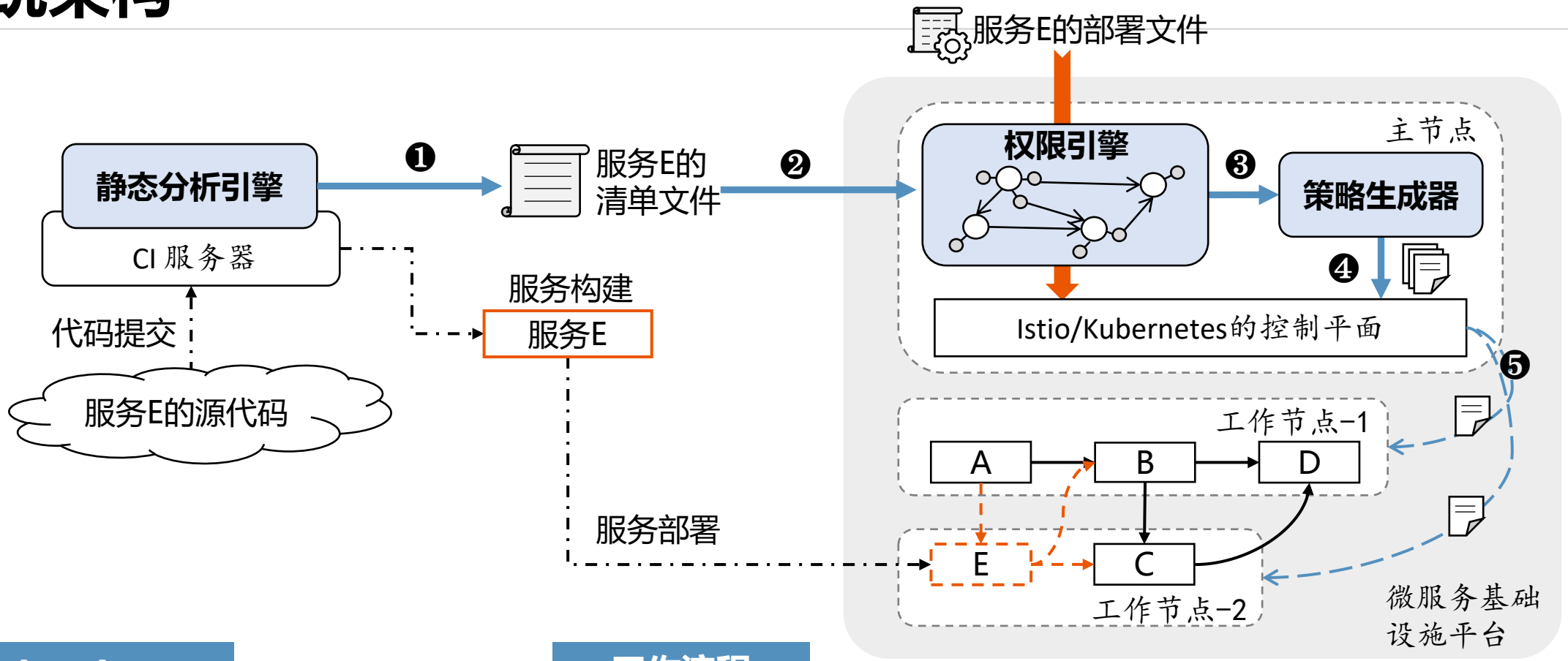
- **目标问题**：自动 **生成**、**维护**、**更新** 微服务的服务间访问控制策略

- **技术路线**：

  1. 提取服务间调用逻辑： 静态分析
  2. 据此生成和维护服务间访问控制策略：基于图的权限管理



系统架构



AutoArmor

1. 静态分析引擎
2. 权限引擎
3. 策略生成器

工作流程

1. 服务E代码提交，静态分析引擎生成清单文件描述其能发起的调用
2. 在E部署时，权限引擎获取其清单文件，生成权限节点加入权限图
3. 策略生成器根据权限图的变化生成并下发访问控制策略

基于静态分析的微服务间调用请求提取

Step I. 识别网络 API 调用语句，即发起服务间调用的关键语句。

- 静态类型语言：函数签名
- 动态类型语言：建模并扫描方法使用的全过程

语义模型

```
Library: requests
    Method: get(url, params=None, **kwargs)
                Semantics: HTTP-GET
                Key parameters: url (Semantics: HTTP-URL)
```

源代码

```
1   import requests
2   from flask import request, session
…
3   reviews = {
4           "name" : "http://reviews:9080",
5           "endpoint" : "reviews"
6   }
…
7   @app.route('/api/v1/products/<product_id>/reviews')
8   def reviewsRoute(product_id):
9           headers = getForwardHeaders(request)
10          user = session.get('user', '')
11          status, reviews = getProductReviews(product_id, headers)
…
12  def getProductReviews(product_id, headers):
13          try:
14              url = reviews['name'] + "/" + reviews['endpoint'] + "/" + str(product_id)
15      res = requests.get(url, headers=headers, timeout=3.0)
…
```

基于静态分析的微服务间调用请求提取

Step II. 以上述关键语句为起点，沿数据流反向进行污点传播，获得程序切片。

- 只关注可被用于访问控制的关键变量，以减少静态分析复杂性
- 创造程序切片副本以处理控制流分支

语义模型

```
Library: requests
    Method: get(url, params=None, **kwargs)
                Semantics: HTTP-GET
                Key parameters: url (Semantics: HTTP-URL)
```

源代码

```
1   import requests
2   from flask import request, session
…
3   reviews = {
4           "name" : "http://reviews:9080",
5           "endpoint" : "reviews"
6   }
…
7   @app.route('/api/v1/products/<product_id>/reviews')
8   def reviewsRoute(product_id):
9           headers = getForwardHeaders(request)
10          user = session.get('user', '')
11          status, reviews = getProductReviews(product_id, headers)
…
12  def getProductReviews(product_id, headers):
13          try:
14              url = reviews['name'] + "/" + reviews['endpoint'] + "/" + str(product_id)
15      res = requests.get(url, headers=headers, timeout=3.0)
…
```

基于静态分析的微服务间调用请求提取

Step III. 通过语义分析在程序切片中提取服务间调用的详细属性。

- 利用语义模型进行属性提取
- 从程序切片的末端沿控制流正向进行变量重构

语义模型

```
Library: requests
    Method: get(url, params=None, **kwargs)
                Semantics: HTTP-GET
                Key parameters: url (Semantics: HTTP-URL)
```

程序切片

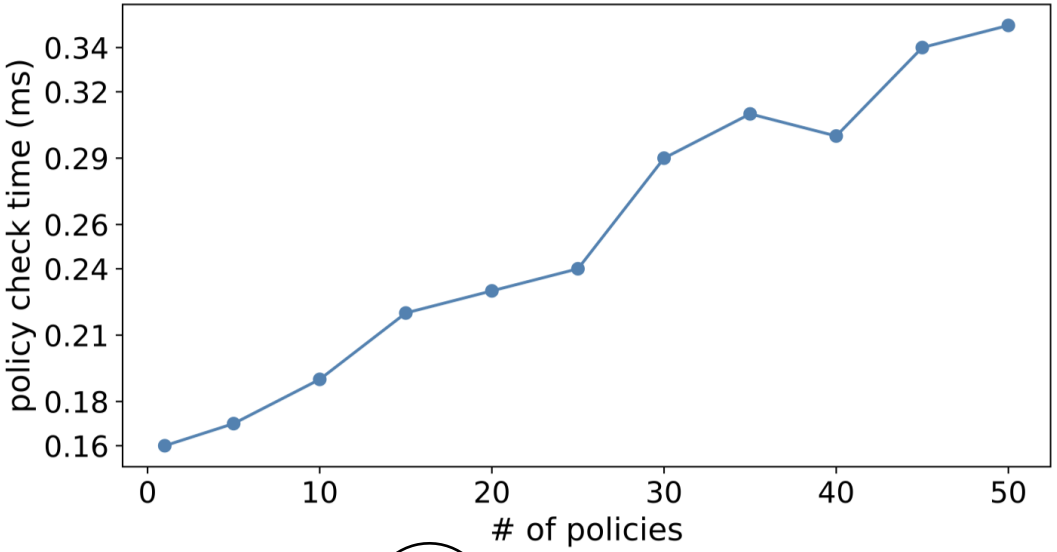
```
1  reviews = {
2          "name" : "http://reviews:9080",
3          "endpoint" : "reviews"
4  }
5  @app.route('/api/v1/products/<product_id>/reviews')
6  url = reviews['name'] + "/" + reviews['endpoint'] + "/" + str(product_id)
7  res = requests.get(url, headers=headers, timeout=3.0)
```

提取出的请求

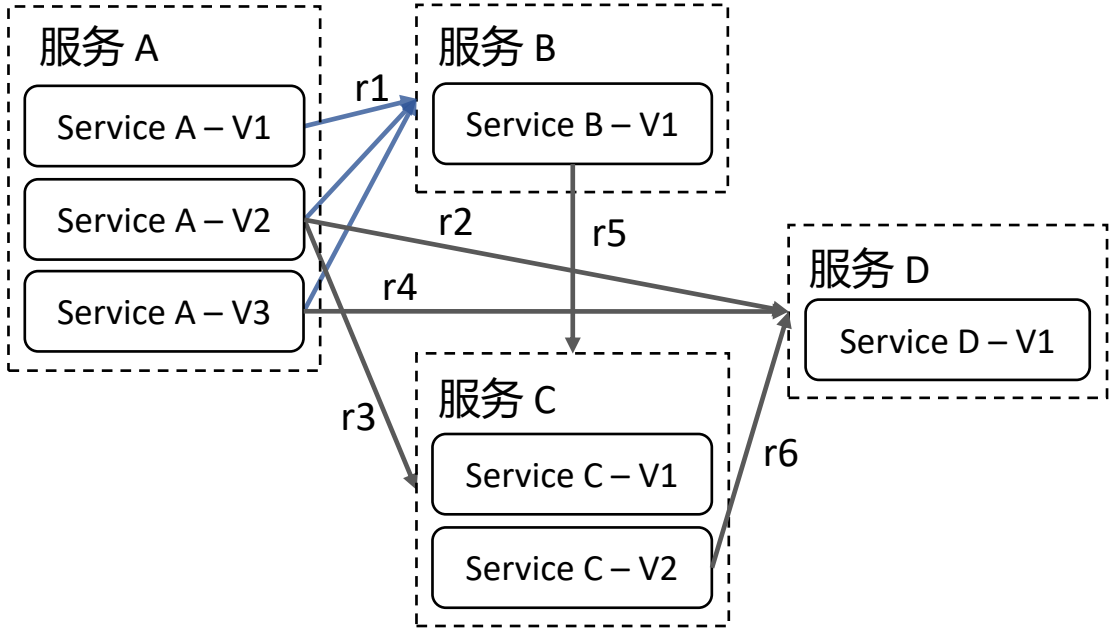
```
{
    "type": "HTTP",
    "url": "http://reviews:9080/reviews/*",
    "path": "/reviews/*",
    "method": "GET"
}
```

基于图的微服务间访问控制策略管理

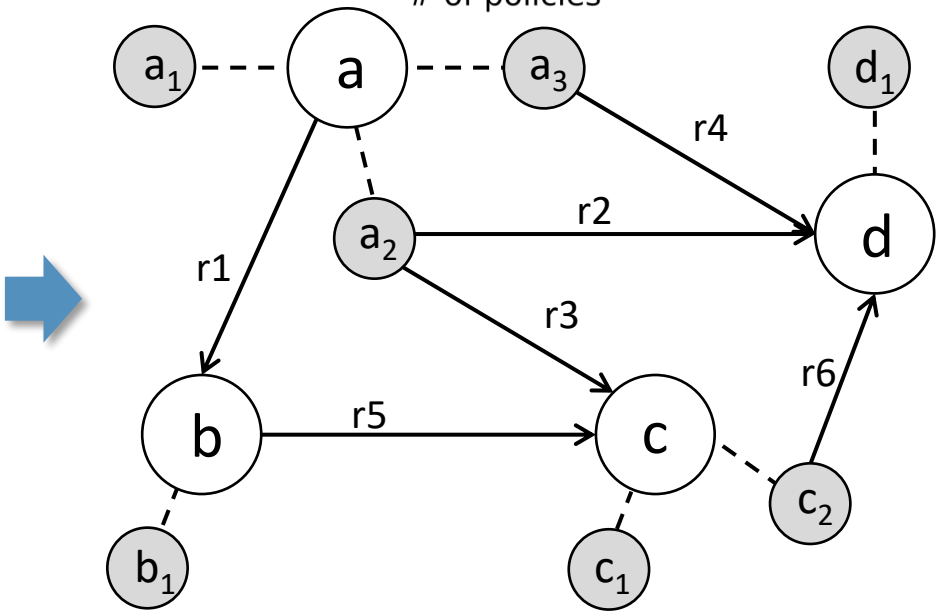
运行时的**策略检查时间**随着安装的服务间访问控制**策略数目**线性增加



服务间访问控制**策略的冗余**会造成整个微服务应用的**性能下降**



实际系统行为

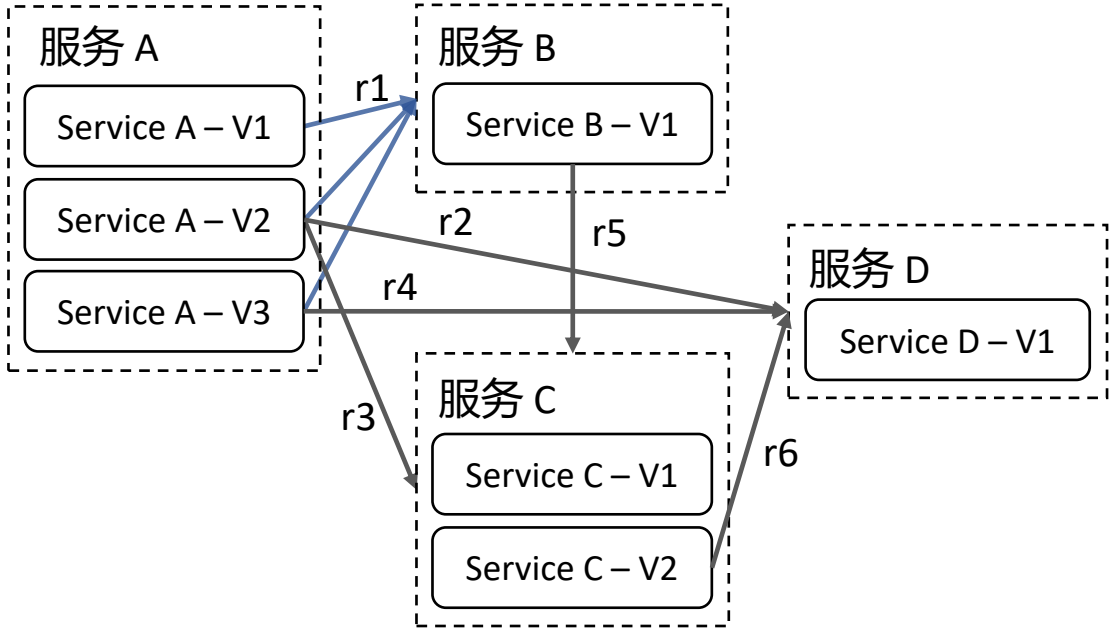


对应的权限图

基于图的微服务间访问控制策略管理

运行时的**策略检查时间**随着安装的服务间访问控制**策略数目**线性增加

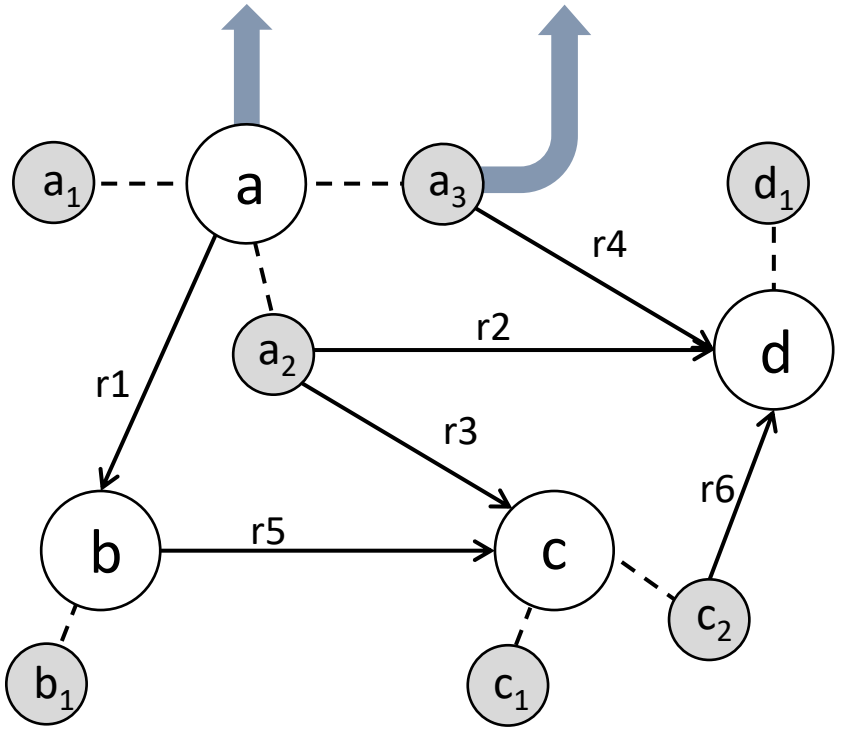
服务间访问控制**策略的冗余**会造成整个微服务应用的**性能下降**



实际系统行为

服务节点: 包含该服务各版本共有的权限

版本节点: 包含该版本独特的权限



对应的权限图

基于图的微服务间访问控制策略管理

优化思路：将同一服务各个版本共有的权限进行整合

- 消除了冗余，减少了策略总数
- 消除了不必要的策略更新

策略生成

① 将与当前权限节点相关的每个请求边翻译成一条服务间访问控制策略

② 如果一个调用请求的目标微服务尚未被部署在系统中，就不授予微服务相应的权限

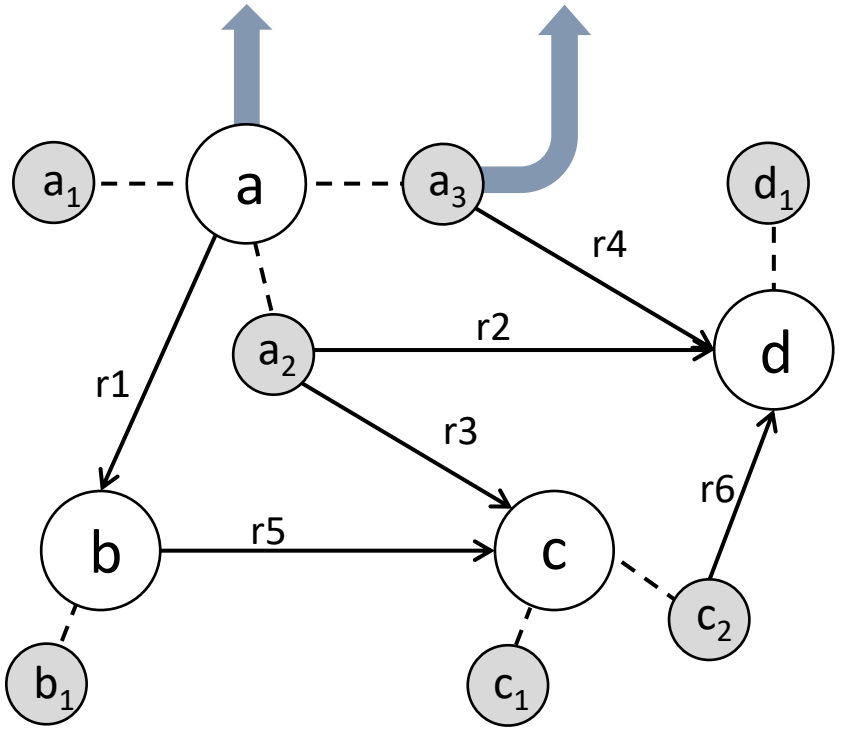
策略更新

① 版本更新 → 版本节点的**添加**、**删除**

② 版本回滚 → 版本节点的**删除**、**添加**

服务节点： 包含该服务各版本共有的权限

版本节点： 包含该版本独特的权限



对应的权限图

系统评估

- **评估对象：**5个流行的开源微服务应用

Name	# of Services	LoCs	Type	Multiple Languages	★ on GitHub
Bookinfo	6	2,702	Demo	✓	24.7k
Online Boutique	11	23,219	Demo	✓	8.8k
Sock Shop	13	20,150	Demo	✓	2.5k
Pitstop	13	45,028	Demo	✗	630
Sitewhere	21	53,751	Industrial	✗	717

- **原型实现：**
  - 包含约 16,500 行代码，覆盖了评估应用中涉及到的 6 种编程语言。
  - 静态分析引擎基于一系列现有的代码分析工具，包括 SonarJava、SonarJS、SonarPython、Roslyn、Go Tools 以及 Parser。

- **实验环境：**
  - 装有 Istio（v1.6.8） 的3 节点 Kubernetes 集群（v.1.18.6）。
  - 每个节点配有 8 个 2.30GHz 的 Intel ® Core ™ CPU（i58259U）以及 32 GB 内存

系统评估

实验一：
请求提取的有效性

服务间调用请求识别率：100%

调用请求的参数提取率：99.5%

实验二：
静态分析的时间

分析每个微服务平均需要 57s

Application	Microservice	Language	LoCs	Identified Requests			Extracted Attributes			Time
				HTTP	gRPC	TCP	URL	Method	Port	
Bookinfo	productpage	Python	2,061	3/3	-	-	3/3	3/3	N/A	21s
	details	Ruby	122	1/1	-	-	1/1	1/1	N/A	4s
	reviews	Java	301	1/1	-	-	1/1	1/1	N/A	27s
	ratings	JavaScript	218	-	-	2/2	2/2	N/A	2/2	27s
Online Boutique	frontend	Go	3,666	-	11/11	-	11/11	N/A	N/A	35s
	cartservice	C#	5,941	-	-	7/7	7/7	N/A	7/7	38s
	productcatalogservice	Go	2,460	-	-	-	N/A	N/A	N/A	18s
	currencyservice	JavaScript	359	-	-	-	N/A	N/A	N/A	25s
	paymentservice	JavaScript	343	-	-	-	N/A	N/A	N/A	26s
	shippingservice	Go	2,458	-	-	-	N/A	N/A	N/A	18s
	emailservice	Python	2,146	-	-	-	N/A	N/A	N/A	20s
	checkoutservice	Go	2,816	-	8/8	-	8/8	N/A	N/A	21s
	recommendationservice	Python	2,112	-	1/1	-	1/1	N/A	N/A	28s
	adservice	Java	918	-	-	-	N/A	N/A	N/A	29s
Sock Shop	front-end	JavaScript	9,922	33/33	-	-	33/33	33/33	N/A	125s
	orders	Java	2,187	6/6	-	2/2	4/8	6/6	2/2	55s
	payment	Go	863	-	-	-	N/A	N/A	N/A	11s
	user	Go	2,515	-	-	24/24	24/24	N/A	24/24	33s
	catalogue	Go	1,439	-	-	8/8	8/8	N/A	8/8	23s
	carts	Java	1,840	-	-	7/7	7/7	N/A	7/7	48s
	shipping	Java	929	-	-	3/3	3/3	N/A	3/3	34s
	queue-master	Java	926	-	-	3/3	3/3	N/A	3/3	31s
Pitstop	webapp	C#	40,461	16/16	-	-	16/16	16/16	N/A	52s
	customermanagementapi	C#	423	-	-	5/5	5/5	N/A	5/5	19s
	vehiclemanagementapi	C#	451	-	-	5/5	5/5	N/A	5/5	18s
	workshopmanagementapi	C#	1,563	4/4	-	20/20	24/24	4/4	20/20	46s
	workshopmanagementeventhandler	C#	685	10/10	-	14/14	24/24	10/10	14/14	30s
	auditlogservice	C#	136	1/1	-	2/2	3/3	1/1	2/2	7s
	notificationservice	C#	511	7/7	-	12/12	19/19	7/7	12/12	42s
	invoiceservice	C#	641	9/9	-	14/14	23/23	9/9	14/14	45s
	timeservice	C#	157	1/1	-	1/1	2/2	1/1	1/1	7s
Sitewhere	web-rest	Java	6,648	-	215/215	-	215/215	N/A	N/A	242s
	instance-management	Java	4,069	-	-	35/35	35/35	N/A	35/35	99s
	event-sources	Java	6,619	-	1/1	3/3	4/4	N/A	3/3	130s
	inbound-processing	Java	825	-	2/2	4/4	6/6	N/A	4/4	49s
	device-management	Java	6,381	-	-	74/74	74/74	N/A	74/74	156s
	event-management	Java	4,799	-	4/4	60/60	64/64	N/A	60/60	204s
	asset-management	Java	5,993	-	-	10/10	10/10	N/A	10/10	142s
	schedule-management	Java	1,964	-	-	10/10	10/10	N/A	10/10	77s
	batch-operations	Java	2,122	-	6/6	16/16	22/22	N/A	16/16	105s
	device-registration	Java	1,075	-	10/10	4/4	14/14	N/A	4/4	57s
	device-state	Java	1,739	-	1/1	7/7	8/8	N/A	7/7	61s
	event-search	Java	769	4/4	-	-	4/4	4/4	N/A	34s
	label-generation	Java	1,379	-	10/10	-	10/10	N/A	N/A	66s
	rule-processing	Java	1,091	-	2/2	2/2	4/4	N/A	2/2	50s
	command-delivery	Java	3,417	-	6/6	3/3	9/9	N/A	3/3	123s
	streaming-media	Java	736	-	-	10/10	10/10	N/A	10/10	49s
	outbound-connectors	Java	4,125	-	13/13	2/2	15/15	N/A	2/2	145s
Total	48 unique services	6 languages	-	96/96	290/290	369/369	751/755	96/96	369/369	-

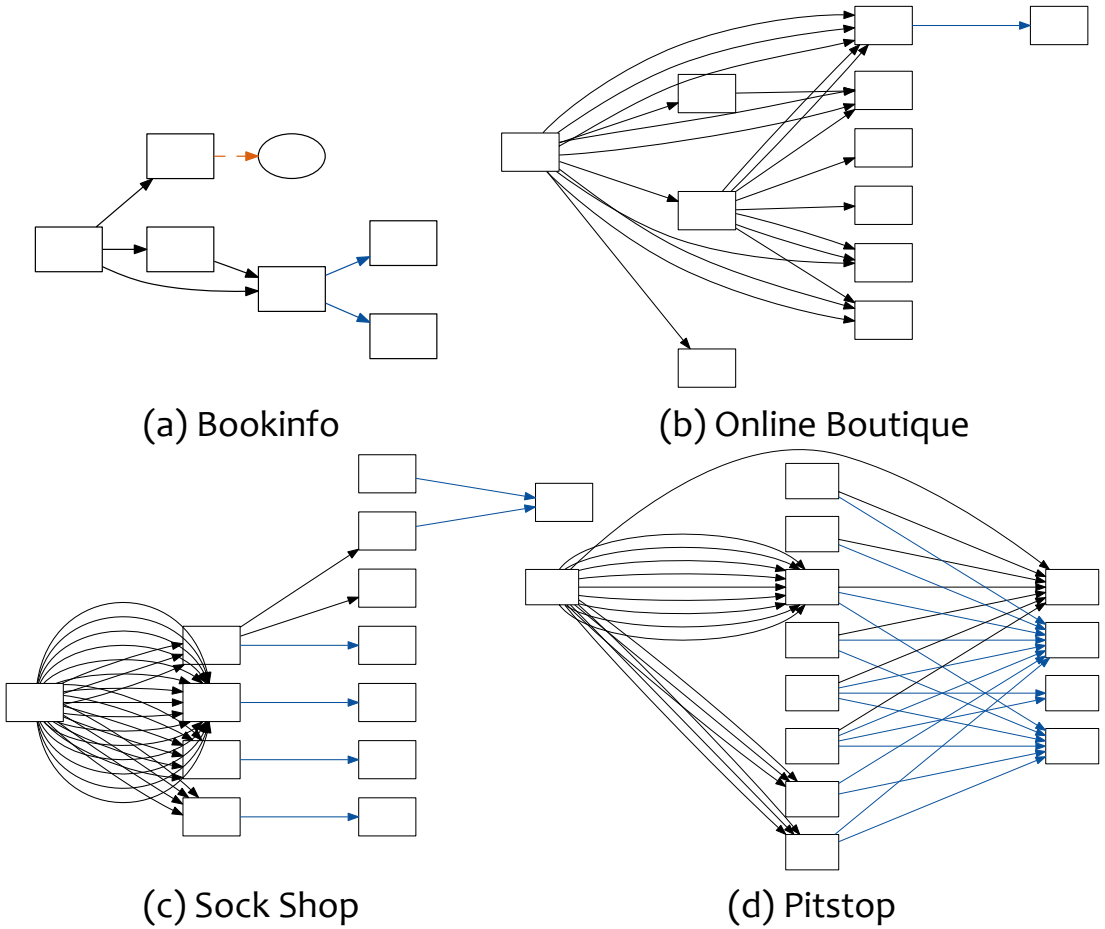
系统评估

实验三：策略生成的正确性及安全性能

没有正常请求被错误地拒绝，也没有未授权的请求绕过了服务间访问控制策略的保护。

基准微服务应用		BookInfo		Online Boutique		Sock Shop	
是否装有 AutoArmor		✗	✓	✗	✓	✗	✓
攻击 A1	越权请求数及其分布	140 (78%)		316 (79%)		2,092 (85%)	
	被阻挡的请求数	0	140	0	316	0	2,092
攻击 A2	越权请求数及其分布	24 (13%)		42 (11%)		290 (12%)	
	被阻挡的请求数	0	24	0	42	0	290
攻击 A3	越权请求数及其分布	16 (9%)		40 (10%)		78 (3%)	
	被阻挡的请求数	0	16	0	40	0	78
总计	越权请求数及其分布	180 (100%)		398 (100%)		2460 (100%)	
	被阻挡的请求数	0	180	0	398	0	2,460

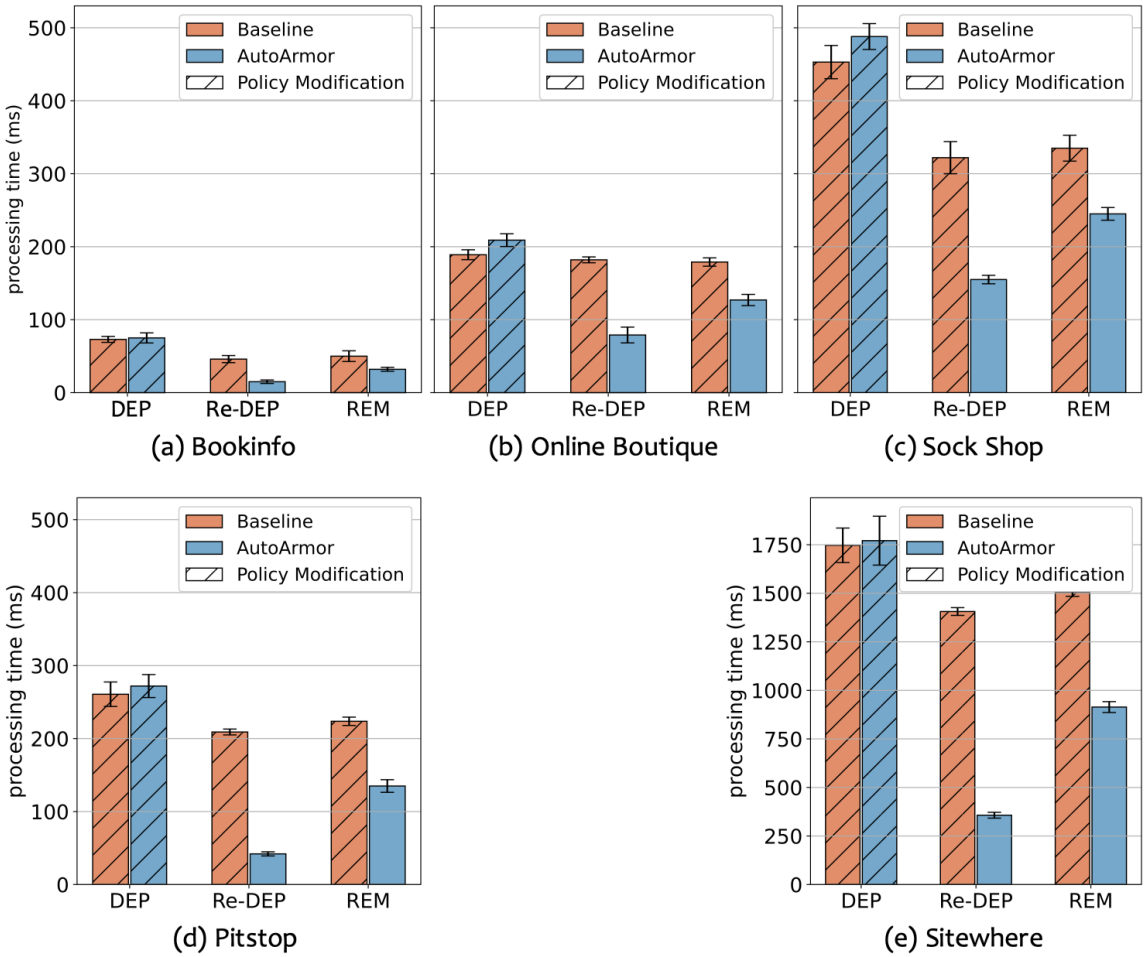
攻击 A1：访问未授权的微服务。攻击 A2：访问未授权的资源。
攻击 A3：进行未授权的操作。



系统评估

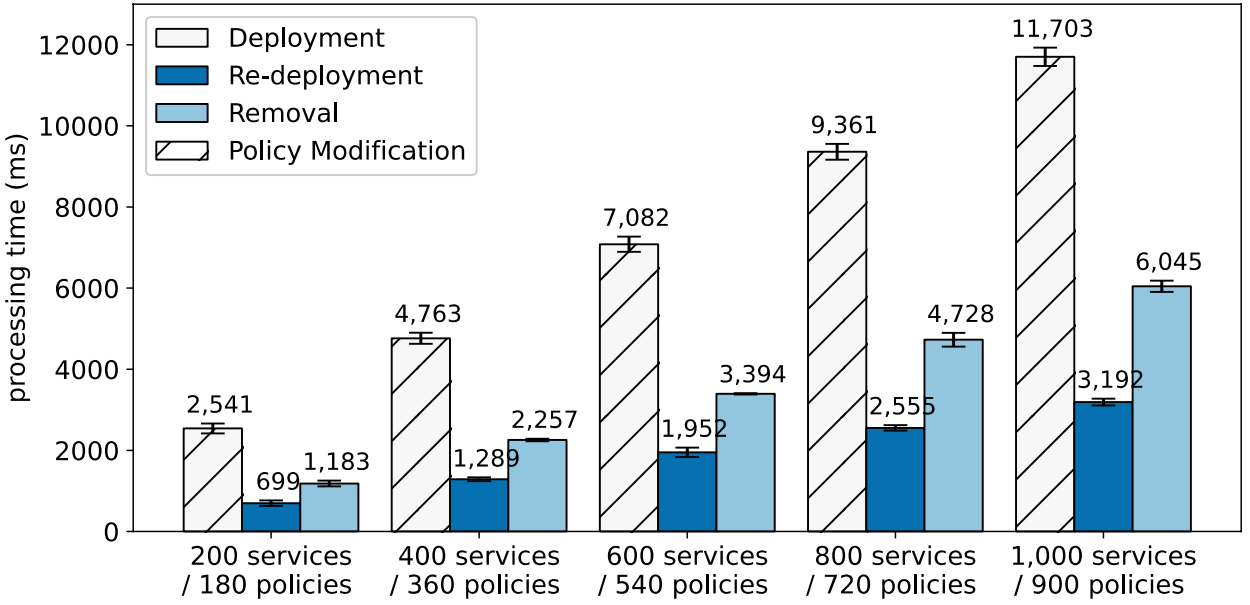
实验四：策略管理的性能

- 单个微服务的策略生成时间均小于2s.
- 策略生成时间与微服务能够发起的服务间调用请求数目正相关



实验五：面对大规模微服务应用的可伸缩性

为包含1,000个微服务的大型云应用生成900条策略的时间小于12s.



(Generated by istio_load_test mesh)

系统评估

实验六：对运行时端到端性能的影响

加速了微服务基础设施在运行时的策略检查， 使微服务应用实现了更好的端到端性能

Application	External Requests			Average End-to-End Latency				
	URL	Method	Quantity	w/o Policies	w/ Baseline Policies		w/ AutoArmor Policies	
Bookinfo	http://<bookinfo-url>/productpage	GET	100000	319ms	323ms	▲ 4ms(1.25%)	321ms	▼ 2ms (0.62%)
Online Boutique	http://<boutique-url>/	GET	4,400	82ms	86ms	▲ 4ms (4.88%)	84ms	▼ 2ms (2.33%)
	http://<boutique-url>/cart	GET	13,000	80ms	91ms	▲ 11ms (13.75%)	86ms	▼ 5ms (5.81%)
	http://<boutique-url>/cart	POST	13,000	139ms	158ms	▲ 19ms (13.67%)	144ms	▼ 14ms (8.86%)
	http://<boutique-url>/cart/checkout	POST	4,400	112ms	129ms	▲ 17ms (15.18%)	121ms	▼ 8ms (6.20%)
	http://<boutique-url>/product/*	GET	56,000	76ms	85ms	▲ 9ms (11.84%)	82ms	▼ 3ms (3.53%)
	http://<boutique-url>/setCurrency	POST	9,000	91ms	93ms	▲ 2ms (2.20%)	94ms	▲ 1ms (1.08%)
Sock Shop	http://<sockshop-url>/	GET	11,000	95ms	104ms	▲ 9ms (9.47%)	98ms	▼ 6ms (5.77%)
	http://<sockshop-url>/basket.html	GET	11,000	101ms	111ms	▲ 10ms (9.90%)	105ms	▼ 6ms (5.41%)
	http://<sockshop-url>/cart	DELETE	11,000	190ms	204ms	▲ 14ms (7.37%)	197ms	▼ 7ms (3.43%)
	http://<sockshop-url>/cart	POST	10,000	364ms	436ms	▲ 72ms (19.78%)	401ms	▼ 35ms (8.03%)
	http://<sockshop-url>/catalogue	GET	11,000	168ms	177ms	▲ 9ms (5.36%)	169ms	▼ 8ms (4.52%)
	http://<sockshop-url>/category.html	GET	11,000	96ms	105ms	▲ 9ms (9.38%)	98ms	▼ 7ms (6.67%)
	http://<sockshop-url>/detail.html?id=*	GET	11,000	95ms	105ms	▲ 10ms (10.53%)	98ms	▼ 7ms (6.67%)
	http://<sockshop-url>/login	GET	11,000	350ms	373ms	▲ 23ms (6.57%)	367ms	▼ 6ms (1.61%)
	http://<sockshop-url>/orders	POST	9,500	392ms	476ms	▲ 84ms (21.42%)	468ms	▼ 8ms (1.68%)

总结

AUTOARMOR： 第一个

微服务间访问控制策略自动生成解决方案

- 一个基于静态分析的微服务间调用请求自动提取机制
- 一个基于图的自动化微服务间访问控制策略管理机制
- 实验表明，它可以有效地实现微服务间访问控制策略的自动生成、维护和更新，仅引入极小的性能开销

usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

30TH USENIX SECURITY SYMPOSIUM

网络安全研究国际学术论坛

International Forum for Security Research

Thanks!

xing.li@zju.edu.cn