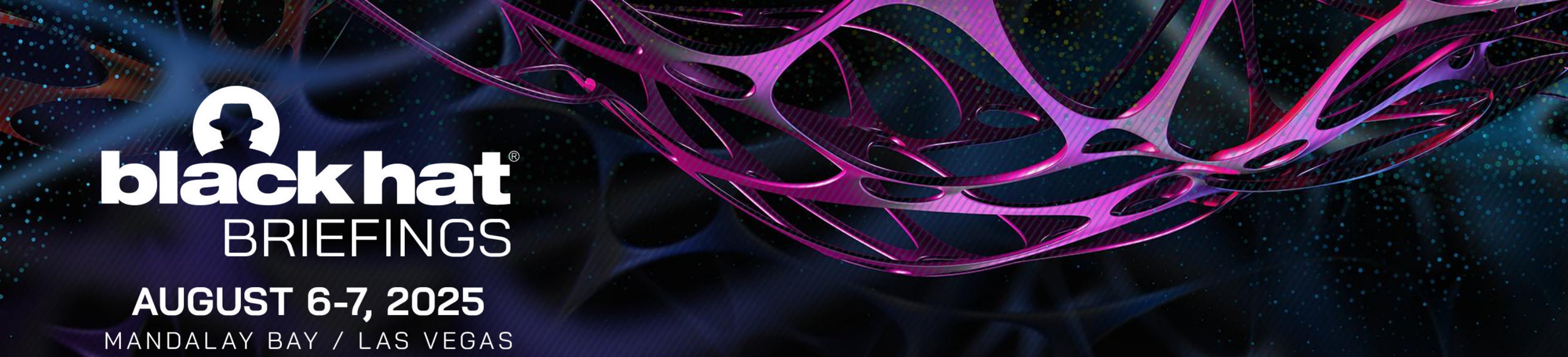


Detecting Taint-Style Vulnerabilities in Microservice-Structured Web Applications

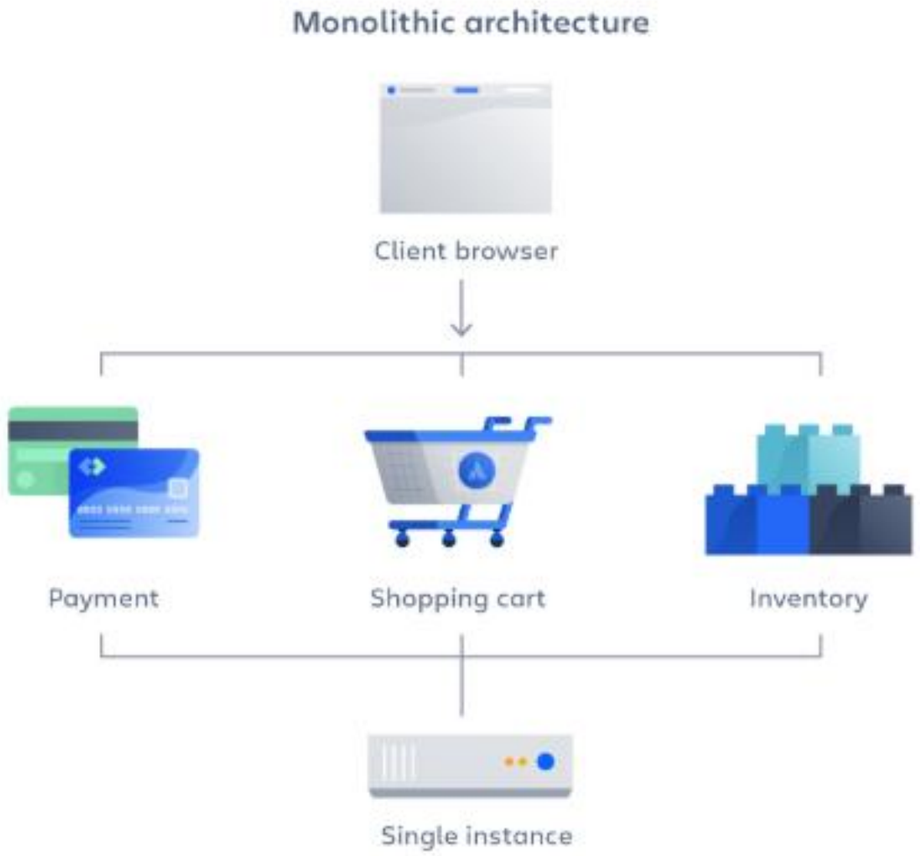
Speaker: Fengyu Liu (LFY)

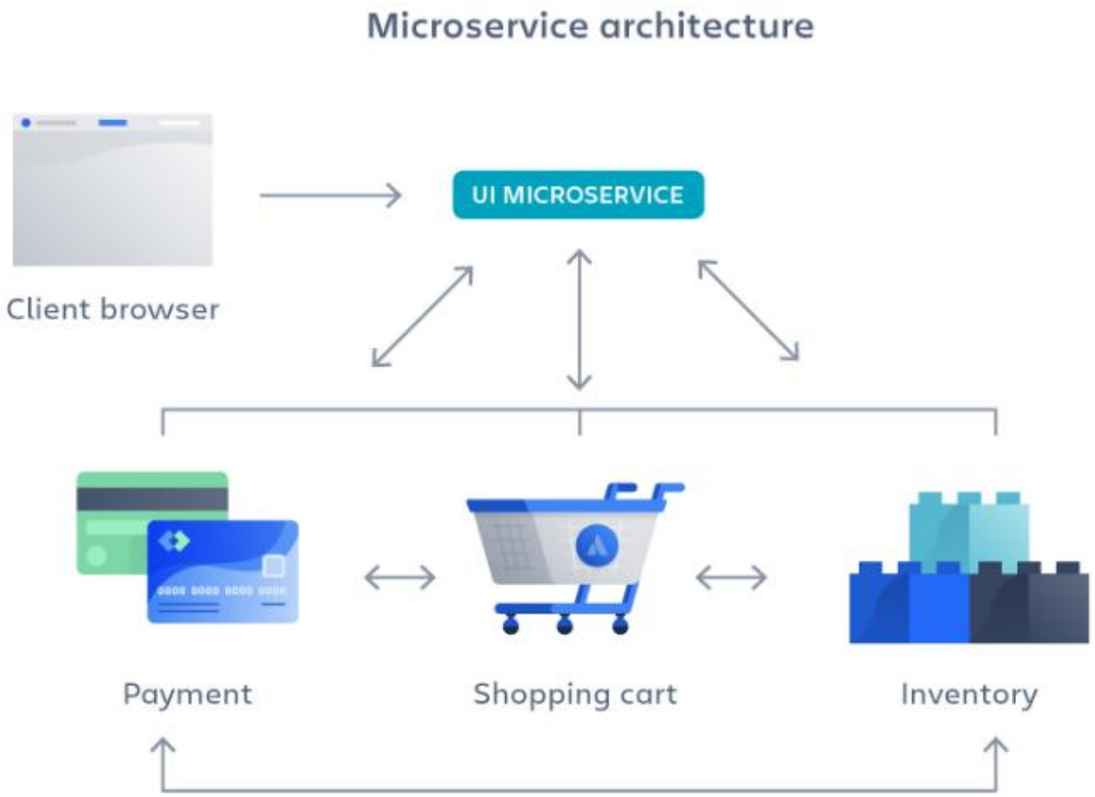
Contributors:

Agenda

- Warm-up & Industry Context
- The Attack Surfaces in Microservices
- Real Case Study
- How MScan Works
- Evaluation
- Conclusion & Takeaways

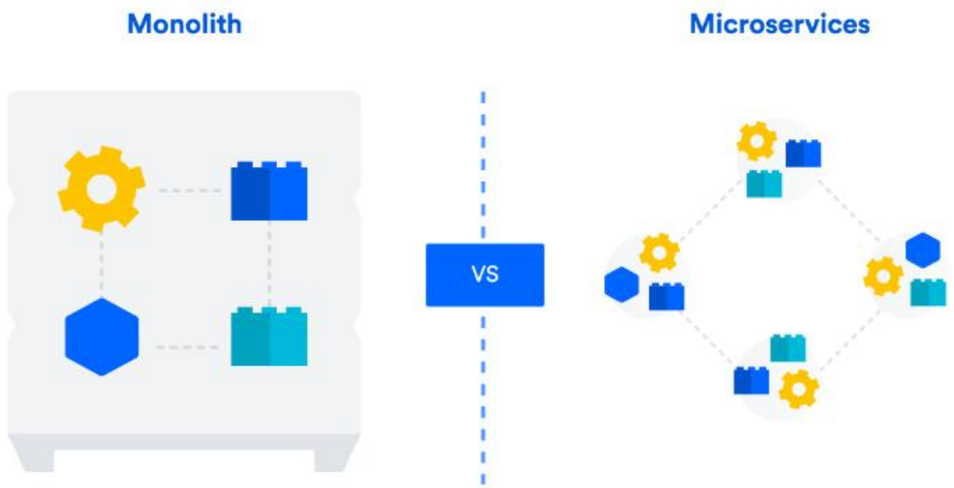
Modern Apps: From Monolith to Microservices





Modern Apps: From Monolith to Microservices

- Microservices dominate cloud-native architecture
- Decentralized, scalable, dynamic — but complex
- One user request may pass through 10+ services



Microservices: Gateway

- Central entry that **routes** user requests to internal services based on **routing rules**
- For example, it forwards requests to *Portal* but blocks access direct to *User*

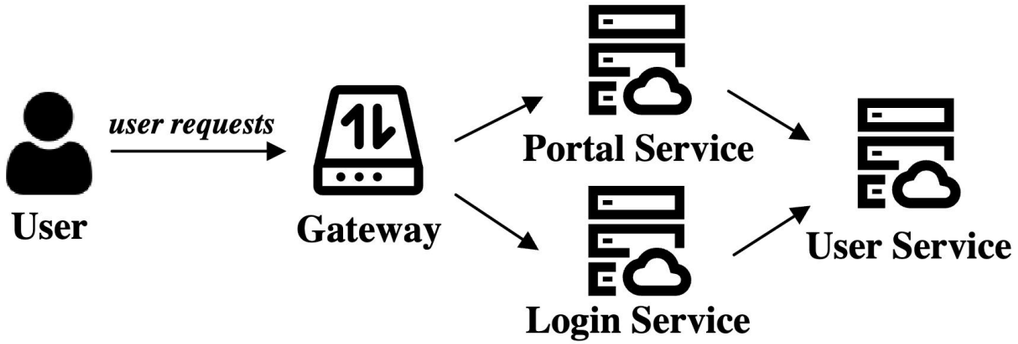


Figure 1: An example that illustrates the typical architecture of a microservice application.

Microservices: Inter-service Communication

- Lightweight **network communication** mechanism (e.g., REST, gRPC) that connect services and pass data

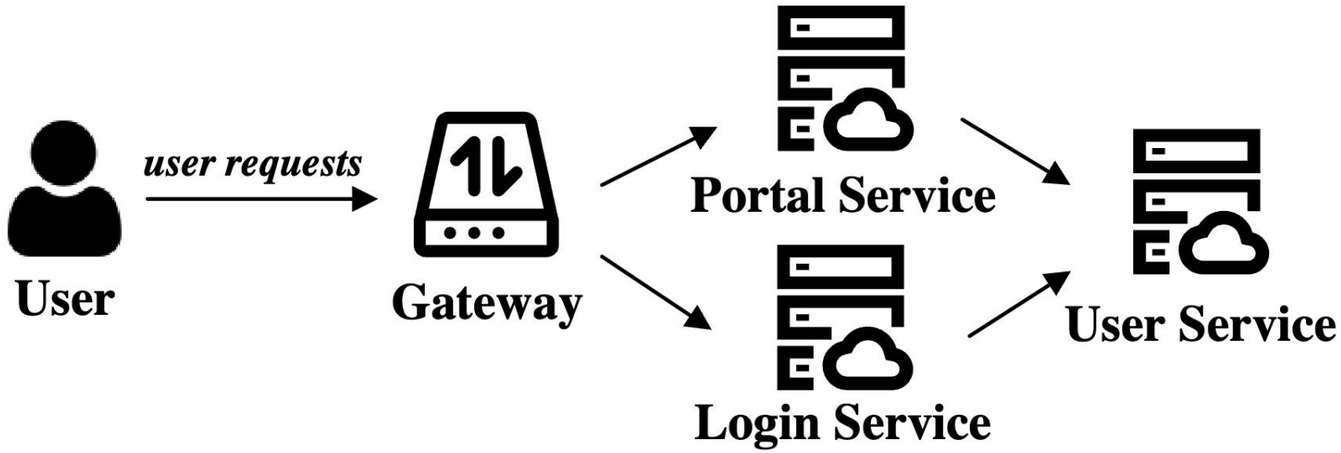


Figure 1: An example that illustrates the typical architecture of a microservice application.

Agenda

- Warm-up & Industry Context
- The Attack Surfaces in Microservices
- Real Case Study
- How MScan Works
- Evaluation
- Conclusion & Takeaways

Taint-style Vulnerabilities in Microservice App

- Intra-service Vulnerability
- happens within a single microservice

```
   // Portal Service (can be accessed)
1  @Path(value = "/portal/query")
2  public User query(String id) {
     ...
3    String query = (new ScriptEngineManager()).eval(id);
4    return select(query);
5  }
```

Taint-style Vulnerabilities in Microservice App

- Inter-service Vulnerability
- involves Inter-service communication


```
// Portal Service (can be accessed)
1  @Path(value = "/portal/query")
2  public User query(String id) {
3    String op = "query";
4    KafkaProducer kafkaProducer =
                new KafkaProducer(String.format("user/%s", op));
5    kafkaProducer.send(id);
     ...
6  }

// User Service (can NOT be accessed)
7  @KafkaListener(topics="user/query")
8  public User queryTask() {
     …
9    String id = kafkaConsumer.poll();
10   String query = (new ScriptEngineManager()).eval(id);
11   return select(query);
12 }
```

Agenda

- Warm-up & Industry Context
- The Attack Surfaces in Microservices
- Real Case Study
- How MScan Works
- Evaluation
- Conclusion & Takeaways

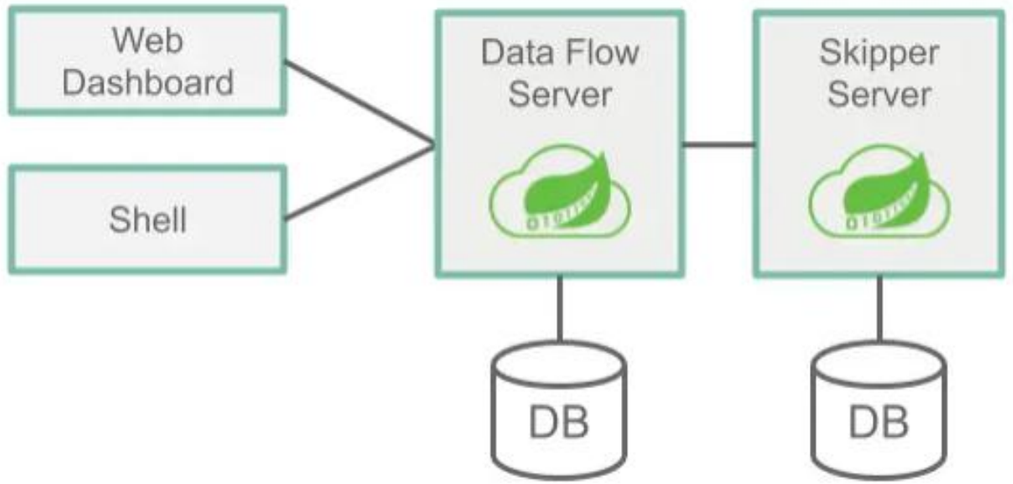
Real Case: Spring Cloud Dataflow

- A cloud dataflow platform under Spring Projects

Spring Security Advisories

CVE-2024-22263: Arbitrary File Write Vulnerability in Spring Cloud Data Flow

HIGH | MAY 23, 2024 | CVE-2024-22263



Real Case: Spring Cloud Dataflow

- Entry: Stream Rest Service
- Edge: RestTemplate
- Service: Package Rest Service
- Sink: Files.write

```
1  @RequestMapping("/streams/deployments")
2  Response deploy(Map<String, String> properties) {
3      this.deployStream.upload(properties);
4  }

5  public Object upload(UploadRequest uploadRequest) {
       ...
6      String url = String.format("%s/%s", baseUri, "upload");
7      Object response = restTemplate.exchange(url, entity);
8  }
```

a) Code snippet in DataFlow Service

```
9  @RequestMapping("/api/package/upload")
10 Metadata upload(UploadRequest uploadRequest) {
11     return packageService.uploadFile(uploadRequest);
12 }

13 public Metadata uploadFile(UploadRequest req) {
       …
14     Path file = Paths.get(uploadDir + req.getName());
15     Files.write(req.getFileAsBytes(), file);
16 }
```

b) Code snippet in Skipper Service

Agenda

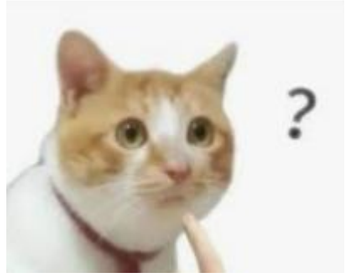
- Warm-up & Industry Context
- The Attack Surfaces in Microservices
- Real Case Study
- How MScan Works
- Evaluation
- Conclusion & Takeaways

Challenges

- Hidden entry points due to gateway routing rules

```
// Allow access
portal-route:
    path: /portal/**
    service: portal
    filters: AddHeader=X-Portal
```

```
// Deny access
user-route:
    path: /user/**
    service: user
    filter: SetResponseStatus=403
```



Challenges

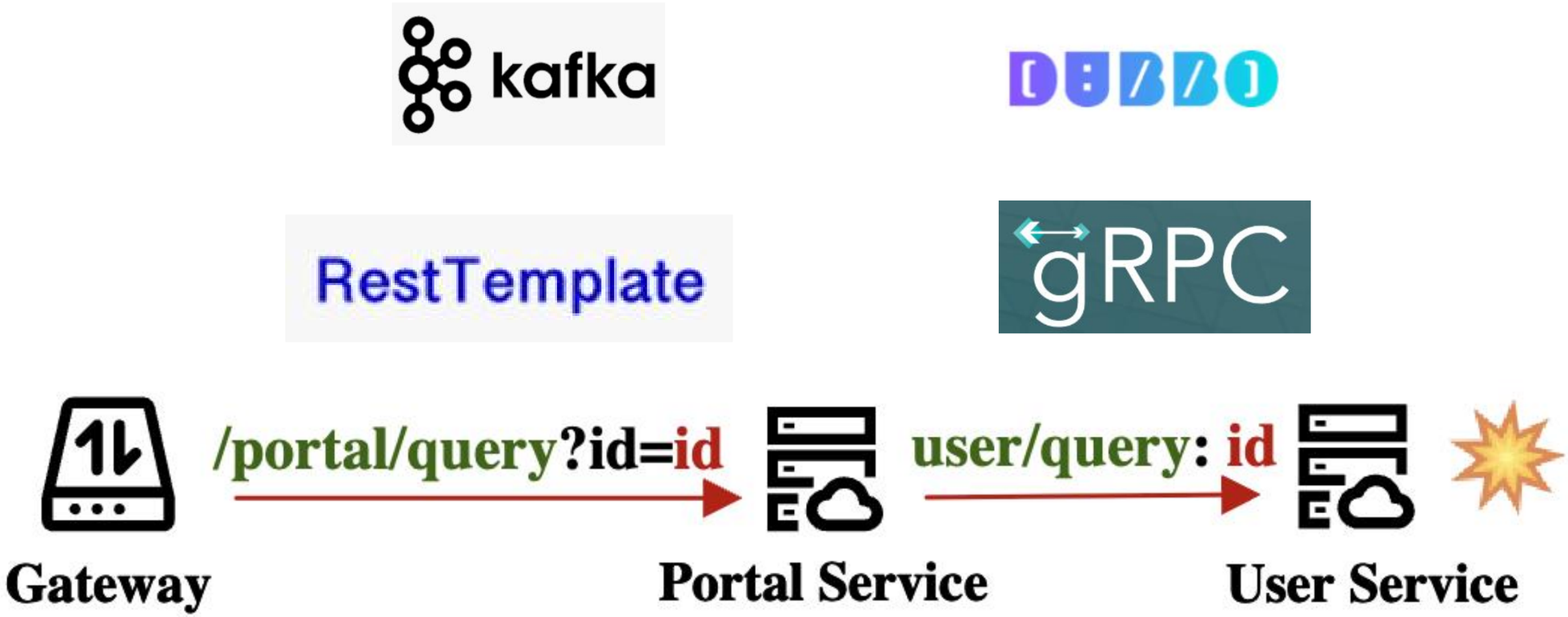
- Hidden entry points due to gateway routing rules
- Not all methods are user-accessible
- Gateways control access with flexible, unstructured configs

```
1  // Allow access
2  portal-route:
3      path: /portal/**
4      service: portal
5      filters: AddHeader=X-Portal
```

```
1  // Deny access
2  user-route:
3      path: /user/**
4      service: user
5      filter: SetResponseStatus=403
```

Challenges

- Cross-service data flow is hard to track



Challenges

- Cross-service data flow is hard to track
- Services communicate via many diverse mechanisms (gRPC, Message Queue), making taint tracking non-trivial

Challenges

- Long call chains break traditional context-sensitive analysis
- Deep call stacks across multi services cause context-sensitive analysis to timeout or run out of memory

Mscan Overview

- LLM-based entry identification and distance-guided taint analysis in Mscan

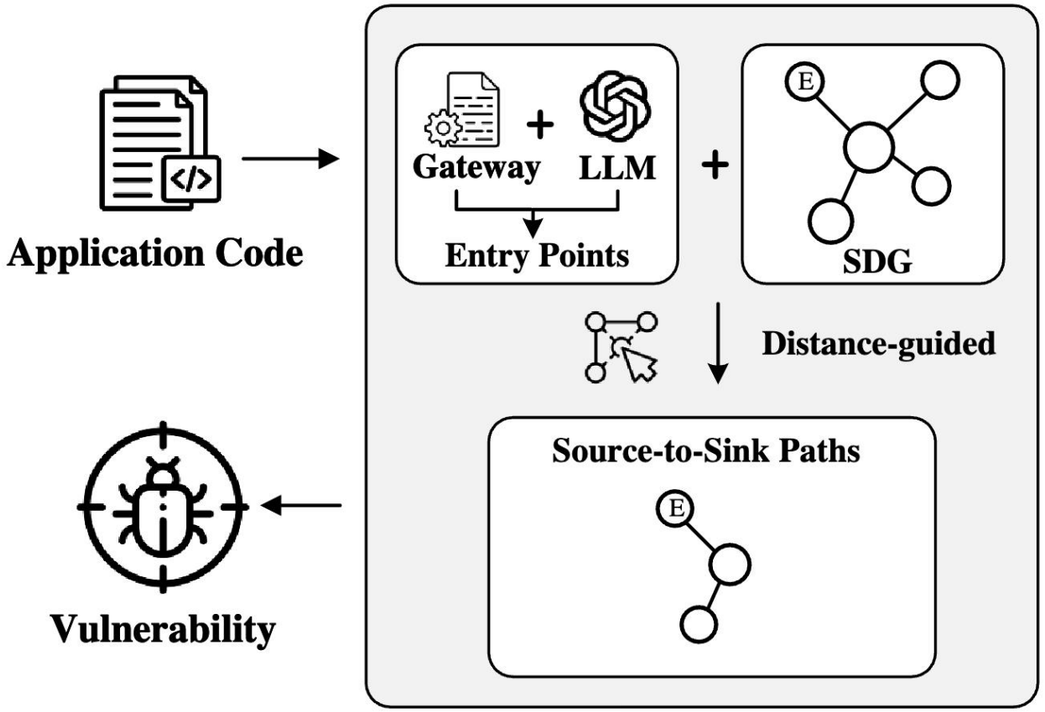


Figure 5: The Architecture of MScan.

Stage I: Entry Point Identification

- LLM-assisted Routing Rule Extraction
- User-accessible Entry Point Identification

```
   // Allow access                          // Deny access
1  portal-route:                           user-route:
2      path: /portal/**                        path: /user/**
3      service: portal                         service: user
4      filters: AddHeader=X-Portal             filter: SetResponseStatus=403

5  api-route:                              admin-route:
6      path: /api/**                           path: /admin/**
7      service: api                            service: admin
8      filters: PrefixPath=/v1                 filter: AddResp=X-Admin, Denied
```

Stage I: Entry Point Identification

Task Description

system

You are a gateway routing rule reader. Read the following routing rules of a microservice application and list all that forward requests. You must follow these rules:
1. You must respond with a JSON list of strings, where each string represents a routing path to a microservice.
2. You need to retain the regex content in the rules as-is.
3. You must not include any other information in your response.
4. By default, assume that the rules will forward requests.

Actual query

user

```
routes:
  portal-route:
    path: /portal/**
    service: portal
  util-route:
    path: /util/**
    filter: deny
    service: util
```

K Shots

user

```
routes:
  - id: add-route
    predicates:
      - Path=/add/**
    filters:
      - AddRequestHeader=X-Request-red
  - id: log-route
    predicates:
      - Path=/log/**
    filters:
      - Status=403
```

assistant

["/add/**"]

...

LLM Response

assistant

["/portal/**"]

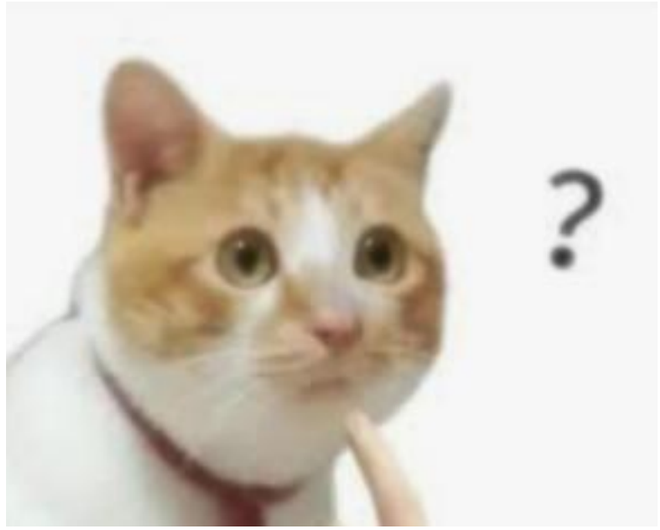
Stage II: Construct Service Dependence Graph

- Identify Communication APIs
- Too many APIs

Framework / Lib	Type	APIs
OpenFeign	Sync	@FeignClient
RestTemplate	Sync	RestTemplate.get RestTemplate.post RestTemplate.exchange
gRPC	Sync	*ImplBase.* *BlockingStub.*
JDK Native	Sync	URL.openConnection HttpClient.send
Apache HttpClient	Sync	HttpClient.execute
Hutool-http	Sync	HttpUtil.get HttpUtil.post
Dubbo	Sync	@DubboReference @DubboService
Kafa	Async	KafkaProducer.send KafkaConsumer.poll
RabbitMQ	Async	Channel.basicPublish Channel.basicConsume
Redis	Async	Jedis.get Jedis.set
MQTT	Async	MqttClient.publish MqttClient.subscribe

Stage II: Construct Service Dependence Graph

- Identify Communication APIs



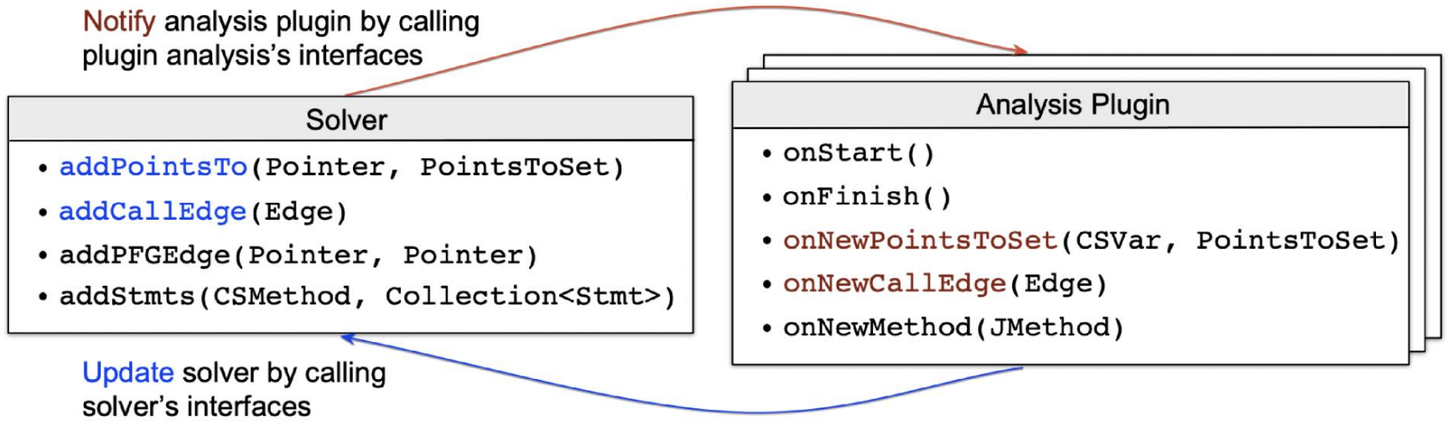
- Use plugin system to handle all

Framework / Lib	Type	APIs
OpenFeign	Sync	@FeignClient
RestTemplate	Sync	RestTemplate.get RestTemplate.post RestTemplate.exchange
gRPC	Sync	*ImplBase.* *BlockingStub.*
JDK Native	Sync	URL.openConnection HttpClient.send
Apache HttpClient	Sync	HttpClient.execute
Hutool-http	Sync	HttpUtil.get HttpUtil.post
Dubbo	Sync	@DubboReference @DubboService
Kafa	Async	KafkaProducer.send KafkaConsumer.poll
RabbitMQ	Async	Channel.basicPublish Channel.basicConsume
Redis	Async	Jedis.get Jedis.set
MQTT	Async	MqttClient.publish MqttClient.subscribe

Stage II: Construct Service Dependence Graph

- “Tai-e introduces a novel analysis plugin system to easily develop and integrate new analysis (that interacts with pointer analysis) like taint analysis and exception analysis, etc. ”

Tai-e



Stage II: Construct Service Dependence Graph

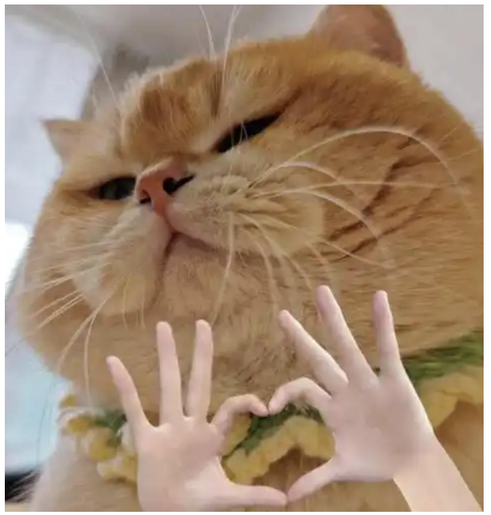
- Use plugin system to handle all
- OpenFeign plugin, gRPC plugin, RestTemplate plugin…

kafka

DUBBO

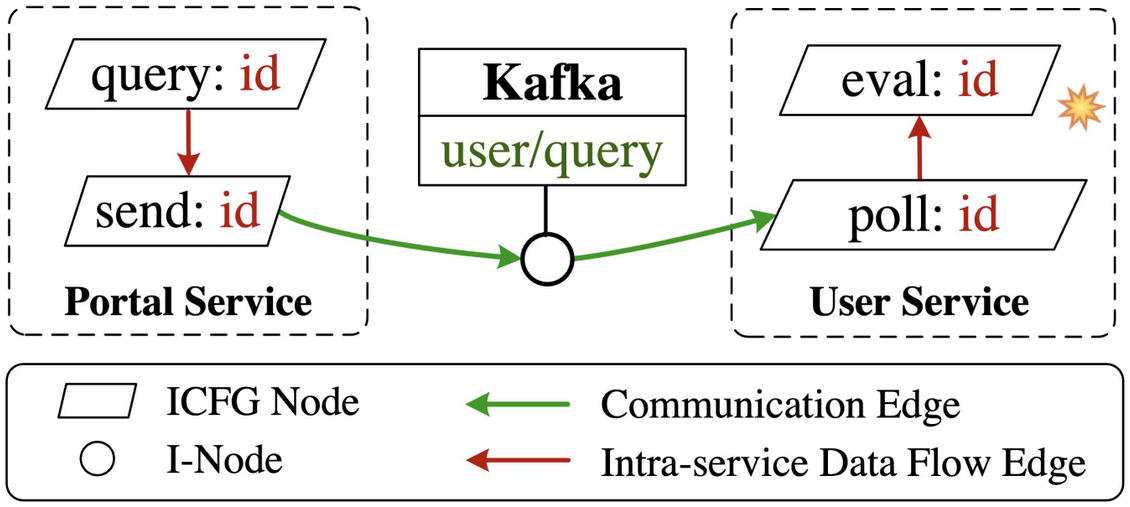
RestTemplate

gRPC



Stage II: Construct Service Dependence Graph

- Resolve Identifier Nodes
- Link Communication Edges
- Connect Inter-service Data Flows




```
// Portal Service (can be accessed)
1  @Path(value = "/portal/query")
2  public User query(String id) {
3     String op = "query";
4     KafkaProducer kafkaProducer =
                new KafkaProducer(String.format("user/%s", op));
5     kafkaProducer.send(id);
      ...
6  }

// User Service (can NOT be accessed)
7  @KafkaListener(topics="user/query")
8  public User queryTask() {
      ...
9     String id = kafkaConsumer.poll();
10    String query = (new ScriptEngineManager()).eval(id);
11    return select(query);
12 }
```

Stage III: Vulnerability Detection

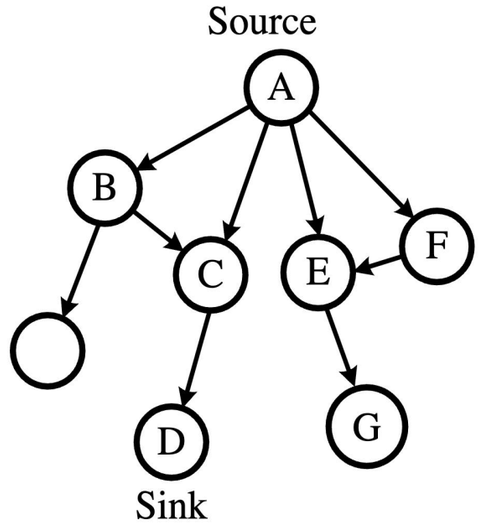
- Identify Taint Source
- Match Taint Sinks
- Check for Missing Sanitization
- Selective Context-sensitive Taint Tracking via SDG

Identify Taint Source Sink and Sanitizer

- **Taint source:** Parameters of user-accessible entry points identified from gateway rules
- **Taint sink:** Security-sensitive operations like SQL queries, file writes, SSRF requests, etc.
- **Taint sanitizer:** If tainted data reaches a sink without proper sanitization → report as vulnerability.

Selective Context-Sensitive Taint Analysis

- What is Context Sensitivity in Taint Analysis?
- Same method, different callsites = different analysis!
- For example, *D()* is called in **4 different contexts**:
- A→B→C→D, A→C→D, B→C→D, C→D



(a) An Example of Call Graph

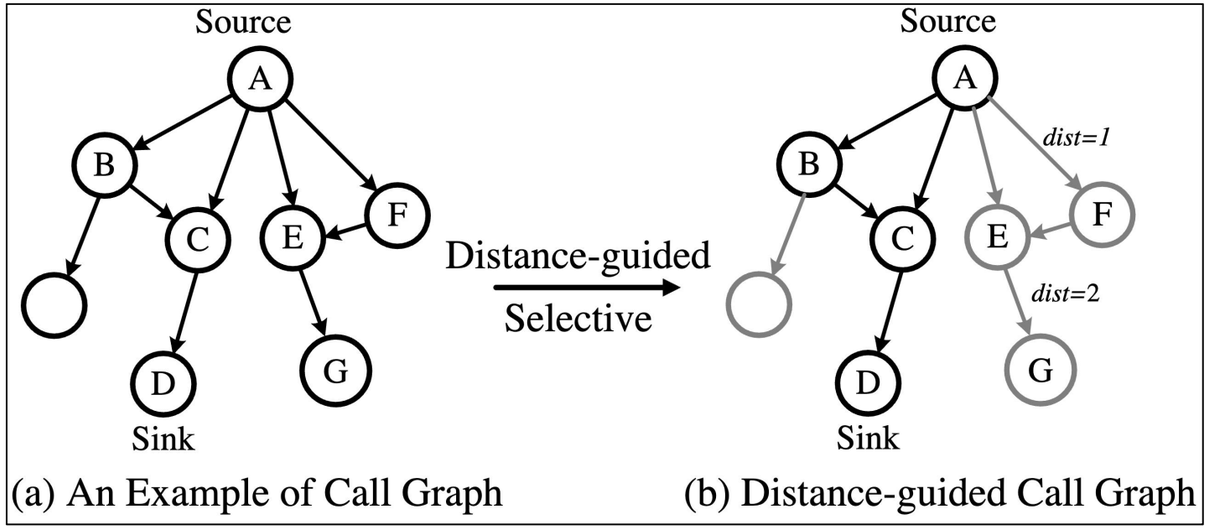
Selective Context-Sensitive Taint Analysis

- Why Full Context Sensitivity Fails in Microservices?
- Long call chains often span multiple services due to inter-service data flows and complex interactions
- Full context tracking generates excessive context objects → high memory, slow analysis, even OOM

Selective Context-Sensitive Taint Analysis

- Goal: Balance Accuracy and Overhead → Precise Yet Scalable Context-sensitive Analysis
- Our Idea: Distance-guided Strategy

$$S(n) = \begin{cases} S_{\mathrm{m}} & n \in DN \\ \max\left(1, \frac{S_{\mathrm{m}}}{\mathrm{dist}(n, DN)^2 \cdot K}\right) & n \in UN \end{cases}$$



(a) An Example of Call Graph (b) Distance-guided Call Graph

Agenda

- Warm-up & Industry Context
- The Attack Surfaces in Microservices
- Real Case Study
- How MScan Works
- Evaluation
- Conclusions & Takeaways

Evaluation Setup: Implementation

- Built on top of Tai-e, a state-of-the-art pointer analysis engine
- ~7K lines of Java code, supports 8 types of vulnerabilities
- SQLi, SSRF, XXE, AFW, code/command injection, etc.

Evaluation Setup: Dataset

- 25 open-source microservice applications, all with 1K+ GitHub stars
- Cover diverse domains: e-commerce, file services, code hosting, etc.
- 5 industrial applications from a world-leading fintech company
- Real-world scale, with complex cross-service logic

Evaluation Result: Effectiveness

- MScan detected **59** 0-day vulns

Vulnerability Type	TP	FP	Prec(%)
Intra-service	27	12	69.23%
Inter-service	32	11	74.42%
Total	59	23	71.95%

Evaluation Result: Baseline

- MScan detected **59** 0-day vulns
- CodeQL detected **23** vulns, missed **36**

Baselines	TP	FP	FN	Prec(%)	Recall(%)
CodeQL	23	35	36	39.66%	38.98%
MScan	59	23	0	71.95%	100.00%

Ablation Study

- **NoEntryDet**: Disables entry point filtering → uses all entry methods
- **NoSDG**: Removes inter-service communication edges
- **MScan-CS**: Uses full context sensitivity (no distance-guided strategy)
- **MScan-CS-2call**: Uses 2-call bounded context sensitivity (from Tai-e)

Ablation Study

Baselines	TP	FP	FN	Prec(%)	Recall(%)
MScan-NoEntry	59	89	0	39.86%	100%
MScan-NoSDG	27	12	32	69.23%	45.76%
MScan-CS	29	11	30	72.50%	49.15%
MScan-CS-2call	59	251	0	19.03%	100.00%
MScan	59	23	0	71.95%	100.00%

Case Study: Site Where

- A famous IoT platform
- Vuln: SQL Injection
- Entry: Device Rest Portal
- Edge: gRPC
- Service: Device Event Service
- Sink: InfluxDB.query

```
1  @Path("/alternate/{alternateId}")
2  public Event getEventByAltId(String alternateId) {
3     return getDeviceEventById(alternateId)
4  }

5  public Event getDeviceEventById(String alternateId) {
6     EventGrpc.EventStub stub = EventGrpc.newStub();
7     return stub.getDeviceEventById(alternateId);
8  }
```

a) Code snippet in WebRest Service

```
9  public class Event extends EventGrpc.EventImplBase {
10    public Event getDeviceEventById(Request request) {
11       return getEvent(request.getAltId());
12    }
13 }

14 public static IDeviceEvent getEvent(String altId) {
15    String query =
            "select * from events where altid='" + altId + "'";
16    return getInflux().query(query);
17 }
```

b) Code snippet in Event Service

Case Study: Yudao Cloud

- A famous cloud platform
- Vuln: SQL Injection
- Entry: File Rest Portal
- Edge: OpenFeign
- Service: File Rest Service
- Sink: FileUtil.writeBytes

```
1  @RequestMapping("/upload-material")
2  public Result upload(Material req) {
3      return materialService.createFile(req.getFile());
4  }

5  @FeignClient
6  public interface FileApi {
7      @Target("/file/create")
8      String createFile(@RequestBody File file);
9  }
```

a) Code snippet in Portal Service

```
10 @RequestMapping("/file/create")
11 public String fileHandler(FileDto file) {
12     return fileService.uploadFile(file);
13 }

14 public String uploadFile(FileDto file) {
15     String filePath = getFilePath(file.getPath());
16     FileUtil.writeBytes(file.getBytes(), filePath);
17     return formatFileUrl(filePath);
18 }
```

b) Code snippet in File Service

Case Study: Mogu Blog

- A famous blog system
- Vuln: Server-Side Request Forgery
- Entry: Wechat Rest Portal
- Edge: OpenFeign
- Serive: Picture Rest Service
- Sink: URL.<init>

```
1  @PostMapping("/wechatCheck")
2  public String index(HttpServletRequest request) {
       ...
3      return pictureClient.uploadPicsByUrl(fileVO);
4  }

5  @FeignClient("mogu-picture")
6  public interface PictureFeignClient {
7      @Target(value = "/uploadPicsByUrl")
8      String uploadPicsByUrl(FileVO fileVO);
9  }
```

a) Code snippet in Web Service

```
10 @PostMapping("/uploadPicsByUrl")
11 public String uploadPictureByUrl(FileVO fileVO) {
12     return fileService.uploadPictureByUrl(fileVO);
13 }

14 public String uploadPictureByUrl(FileVO fileVO) {
       ...
15     URL url = new URL(fileVO.getUrl());
16 }
```

b) Code snippet in Picture Service

Agenda

- Warm-up & Industry Context
- The Attack Surfaces in Microservices
- Real Case Study
- How MScan Works
- Evaluation
- Conclusion & Takeaways

Conclusion & Takeaways

- Attendees will know the current state and key challenges of detecting taint-style vulnerabilities in microservice apps.
- Attendees will understand how Mscan works and why it detects taint-style vulnerabilities in microservice apps efficiently and precisely.
- Attendees will learn how to optimize taint analysis when adapting it to a cross service system.