Issue 16, 2024

https://doi.org/10.23939/sisn2024.16.069

УДК 004.056:004.354.6:657

METHODS AND MEANS OF ANALYZING APPLICATION SECURITY VIA DISTRIBUTED TRACING

Oleh Faizulin[1], Mariia Nazarkevych[2]

[1, 2] Lviv Polytechnic National University,
Information Systems and Networks Department, Lviv, Ukraine,
[1] E-mail: oleh.r.faizulin@lpnu.ua, ORCID: 0000-0001-5781-0600
[2] E-mail: mariia.a.nazarkevych@lpnu.ua, ORCID: 0000-0002-6528-9867

The article describes methods and means of digital security that are utilizing distributed tracing to detect, investigate, and prevent security incidents. The described methods and means are applicable to solutions of any scale – from large enterprises to pet projects; of any domain – healthcare, banking, government, retail, etc. The article takes a comprehensive approach to digital security including identification, alerting, prevention, investigation, and audit of existing security incidents. Described approaches to application security via tracing are focused on general purpose applications, but they can be extended to cover a domain specific use-case. All Approaches are production tested and utilized in existing distributed IT systems in one way or another, however certain examples and use-cases are intentionally simplified for the demonstration purposes and ease of understanding. Nevertheless, it must be understood that methods and means described in the article complement existing security practices and cannot replace all of them, however they may improve overall security of the system by decreasing incident detection time, decreasing resources and efforts needed to investigate breaches or passing a security audit.

Keywords: Security, distributed tracing, behavior analysis, alerting, automated scaling, distributed IT systems, metrics, logging, observability, APM, auditing.

Introduction

Modern world is full of challenges which often require non-standard, proactive approaches to security. Recent events, such as the attack on Kyivstar (Dec 2023, Ukraine) demonstrated how crucial cybersecurity is for stable functioning of basic activities such as communications, healthcare, payments, etc.

IT systems of any scale, from pet projects to large enterprises must be secured by default. Ignoring security may lead to different consequences, manageable losses, or severe financial and reputational consequences.

Any IT system that is exposed to global network, either directly or indirectly must be protected and follow at least very basic security best practices. Depending on the importance and adoption of application, criticality, usage of certain data kinds – various security regulations must be followed. Modern enterprises are investing large amounts of money to ensure security, but still, incidents happen.

Problem statement

Security investigations are always stressful and time-consuming. Unfortunately, many enterprises as well as regular users tend to ignore security or apply minimal effort needed. Once security incidents such

as breach, leak, unauthorized data modification occur the importance of security instantly skyrockets, and significant efforts are dedicated to the investigation. By applying the methods and means described in the article any distributed it system can shorten investigation times, easily pass security audits, recreate event sequences that led to a specific security event. Additionally, the article addresses the problem of loosely coupled context, where individual log records or other security related efforts can't be matched.

Analysis of the recent research and publications

In the book "Distributed tracing in practice: Instrumenting, analyzing, and debugging micro-services". The authors present an in-depth exploration of distributed tracing, covering its implementation, data collection, and analysis. The authors focus on practical aspects of using distributed tracing in microservices, offering best practices for instrumentation, managing overhead, and utilizing tracing to enhance performance and troubleshoot issues. The book is aimed at practitioners looking to gain actionable insights from their distributed systems and improve operational visibility.

In the paper "Privacy-risk detection in microservices composition using distributed tracing" 0 the authors discuss the use of distributed tracing to detect privacy risks in microservices compositions. They highlight how tracing data can be used to identify and mitigate potential privacy issues in complex microservice environments. The approach involves analyzing trace data to monitor and flag privacy-related anomalies, ensuring compliance with privacy standards in dynamic and distributed systems.

In the research "Detecting anomalies in microservices with execution trace comparison. Future Generation Computer Systems" the authors focus on anomaly detection in microservices by comparing execution traces. The authors propose a method that leverages trace data to identify deviations from normal behavior, which can indicate potential security or operational issues. The methodology includes collecting and analyzing trace data to detect unusual patterns that may signify underlying problems in the microservice architecture.

In the paper "Localizing and explaining faults in microservices using distributed tracing". The authors explore the use of distributed tracing for fault localization and explanation in microservices. They present techniques for using trace data to pinpoint the source of faults and provide explanations for these faults, helping developers to quickly understand and address issues in their distributed applications. The approach enhances the troubleshooting process by offering detailed insights into the flow of requests across microservices.

In the study "Detecting Cyber Security Attacks against a Microservices Application using Distributed Tracing" the authors investigate the use of distributed tracing for detecting cybersecurity attacks on microservice applications. They describe methods for leveraging trace data to identify suspicious activities and potential security breaches. By analyzing the traces, the approach aims to detect anomalies that could indicate attacks, providing a proactive measure for enhancing the security of microservice-based systems.

In the article "Detection of microservice-based software anomalies based on OpenTracing in cloud" 0, Khanahmadi et al. explore methods for identifying anomalies in cloud-based microservices by leveraging OpenTracing, a popular distributed tracing framework. The study focuses on detecting abnormal behavior in microservices, which can help prevent issues such as performance degradation, security threats, and system failures.

In the article "Security in Microservices Architectures" 0, Mateus-Coelho, Cruz-Cunha, and Ferreira explore the various security challenges and solutions associated with microservices-based architectures. The authors analyze how microservices, due to their distributed nature and independent deployment, introduce new vulnerabilities that traditional monolithic architectures do not face. Additionally, the paper discusses the importance of DevSecOps, which integrates security practices into the development and operations processes, ensuring that security is considered throughout the software lifecycle. The authors

propose various tools and techniques to address these challenges, providing both theoretical and practical perspectives on securing microservices architectures in modern IT environments.

In the article “Anomalous Distributed Traffic: Detecting Cyber Security Attacks Amongst Microservices Using Graph Convolutional Networks” 0, Jacob et al. investigate a novel approach for detecting cybersecurity attacks in microservice architectures using Graph Convolutional Networks (GCNs). The authors propose a method that models communication patterns between microservices as a graph and uses GCNs to detect anomalous traffic patterns that could indicate malicious activities.

In the article “Building Secure Microservices-Based Applications Using Service-Mesh Architecture” 0, Chandramouli and Butcher focus on the security challenges and solutions in microservices architecture, particularly through the implementation of service mesh. This National Institute of Standards and Technology (NIST) publication offers a comprehensive framework for securing microservices-based applications by integrating security controls within the service mesh architecture.

In the article “Adaptive Observability for Forensic-Ready Microservice Systems” 0, Monteiro, Yu, Zisman, and Nuseibeh propose a framework for enhancing the observability of microservice systems with a focus on forensic readiness. The authors address the challenge of making distributed systems more capable of supporting forensic investigations, particularly in cases of security breaches or operational failures.

In the article “Scalable Compositional Static Taint Analysis for Sensitive Data Tracing on Industrial Micro-Services” 0, Zhong et al. propose a method to trace sensitive data in industrial microservices using a scalable static taint analysis technique. The authors focus on addressing the challenges of sensitive data leakage in large-scale microservice architectures, particularly in industries where the security of personal and sensitive information is paramount.

In the article “Design, Monitoring, and Testing of Microservices Systems: The Practitioners’ Perspective” 0, Waseem et al. provide insights into how industry professionals approach the development, monitoring, and testing of microservices architectures. Based on extensive surveys and interviews with practitioners, the authors examine the challenges and best practices encountered in real-world microservices implementations.

In the article “Transparent Tracing System on gRPC Based Microservice Applications Running on Kubernetes” 0, Perdanaputra and Kistijantoro present a system designed to implement transparent distributed tracing for gRPC-based microservices operating within Kubernetes environments. The authors focus on addressing the challenges of monitoring microservices, particularly when it comes to tracing inter-service communications in large-scale distributed systems.

In the article “Evaluation of the Effectiveness of Different Image Skeletonization Methods in Biometric Security Systems” 0, Nazarkevych et al. analyze various image skeletonization techniques to determine their suitability and effectiveness in biometric security applications. Skeletonization, a key process in biometric recognition systems, involves reducing images to their essential structures, allowing for efficient pattern recognition while preserving important features.

In the article “Methods of Protection Document Formed from Latent Element Located by Fractals” 0, Medykovskyy, Lipinski, Troyan, and Nazarkevych explore innovative methods for enhancing document security using fractal-based techniques. The authors propose a novel approach to embedding latent elements within documents to protect them from forgery and unauthorized duplication. These latent elements, when designed using fractal patterns, offer a highly secure and difficult-to-replicate method for verifying the authenticity of documents.

In the paper “Unsupervised Detection of Microservice Trace Anomalies through Service-Level Deep Bayesian Networks” 0, Liu, Xu, Ouyang, Jiao, Chen, Zhang, and Pei propose an advanced approach to identifying anomalies in microservice architectures using deep Bayesian networks. The authors introduce a novel unsupervised method that leverages probabilistic modeling to detect anomalies without relying on labeled training data.

In the paper “Microservice Security: A Systematic Literature Review” 0, Berardi, Giallorenzo, Mauro, Melis, Montesi, and Prandini provide a comprehensive review of the current state of research on microservice security. Published in PeerJ Computer Science, this study aims to consolidate existing knowledge and identify gaps in the field of securing microservice architectures.

In the paper “Visualizing Microservice Architecture in the Dynamic Perspective: A Systematic Mapping Study” 0, Gortney, Harris, Cerny, Al Maruf, Bures, Taibi, and Tisnovsky explore methods for visualizing microservice architectures with a focus on their dynamic aspects. Published in IEEE Access, this study systematically maps out the various approaches to visualizing the evolving nature of microservice-based systems.

In the paper “Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis” 0, Luo, Xu, Lu, Ye, Xu, Zhang, and Xu investigate the dependencies and performance characteristics of microservices using trace data from Alibaba. Presented at the ACM Symposium on Cloud Computing, this study provides insights into the behavior and performance of microservice architectures based on empirical data from a major technology company.

In the paper “Trace-Based Microservice Anomaly Detection through Deep Learning” 0, Bai and Zhang explore an advanced approach for detecting anomalies in microservice architectures using deep learning techniques. Presented at the Second International Conference on Electronic Information Engineering, Big Data, and Computer Technology (EIBDCT 2023), this study focuses on leveraging deep learning models to analyze microservice traces and identify potential anomalies.

These, and other articles collectively illustrate the versatile applications of distributed tracing in improving the reliability, security, and performance of microservice architectures. They highlight its critical role in modern distributed systems, from troubleshooting and performance optimization to privacy risk detection and cybersecurity.

Article goals

1. Describe methods and means of digital security that utilize distributed tracing to detect, investigate, and prevent security incidents.
2. Demonstrate the applicability of these methods and means to solutions of any scale, from large enterprises to small projects, and across various domains, including healthcare, banking, government, and retail.
3. Provide a comprehensive approach to digital security that includes identification, alerting, prevention, investigation, and auditing of security incidents.
4. Focus on application security through tracing, with an emphasis on general-purpose applications, while also highlighting the potential for extending these approaches to domain-specific use cases.
5. Present production-tested approaches that are utilized in existing distributed IT systems, with simplified examples and use-cases for clarity and ease of understanding.

Emphasize that these methods and means complement existing security practices, aiming to improve overall system security by reducing incident detection time and decreasing the resources and efforts required for investigating breaches or passing a security audit.

Methods and Means of Analyzing Application Security via Distributed Tracing

Distributed Tracing Overview

Distributed tracing and traces describe a process, typically and end-to-end transaction workflow made in a web service or other IT system. It provides visibility into services and components of the IT system, focusing on complete transparency and visibility. By recording and visualizing transactions, it allows operations team to analyze and compare user workflows, detect abnormal behavior, etc. As a requests travels through the network, from frontend to back–end to database or to other services – it’s being linked on each level, recorded, and stored for future analysis.

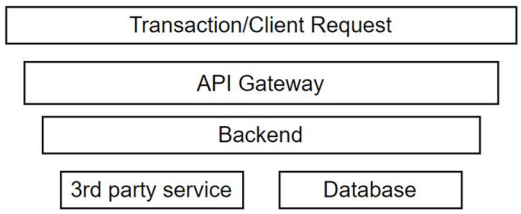

Fig. 1. Example of component view in tracing

Tracing Vocabulary

In order proceed, it's needed to be familiar with tracing vocabulary below:

- TraceId – This is an id that's assigned to a single request, job, or action. Something like each unique user-initiated web request will have its own traceId.
- SpanId – Tracks a unit of work. Think of a request that consists of multiple steps. Each step could have its own spanId and be tracked individually. By default, any application flow will start with the same TraceId and SpanId.
- Tag – a key value pair that can be added to a particular unit of work.
- Trace Context – a collection of standardized headers that allow distributed tracers to communicate without dropping context information.

Tracing History

Dapper, a large-scale distributed tracing system was by Google, 2010. Two years later, Twitter open-sourced Zipkin, their tool for distributed application performance tuning. Zipkin was the first fully open-sourced tracing solution for the distributed IT systems. In 2015, Uber introduced Jaeger, their alternative to Zipkin.

In 2016 Ben Sigelman, founder of Lightstep, wrote a blog post called Toward Turnkey Distributed Tracing, which described Open Tracing as a standard. It's also often REFERENCESd as Open Tracing Manifesto. Open Tracing allowed developers of applications, libraries and components and frameworks to instrument their code in a unified manner without binding themselves to a specific solution. The goal was to solve the standardization problem.

Later in 2016, The Cloud Native Computing Foundation (CNCF) accepted Open Tracing as one of their projects and Open Tracing 1.0 was released. Later in the same year Jaeger joined the Open Tracing project.

In 2019, World Wide Web Consortium (W3C) was proposed to standardize tracing context specification. Same year, Open Tracing and Open Telemetry projects merged into a single project under the umbrella of CNCF.

10 years later, tracing grew from a single paper to a rich ecosystem of components available for any layer and technology. At the same time, tracing moved from just tracing to observability, which includes logging and metrics as well.

Tracing in 2024

Over the years a rich ecosystem of commercial and open-source software was built to cover practically all the possible needs of distributed applications, observability and beyond. So far, there are two kinds of solutions: "all in one" and "do one thing and do it well."

The table below demonstrates a subset of widely adopted commercial and open-source solutions available on the market.

Table 1

Observability solutions

Name	Commercial/Open Source	Capabilities
Instana	Commercial	Tracing, metrics, logs, alerts
Datadog	Commercial	Tracing, metrics, logs, alerts
Jaeger	Open Source	Tracing
Prometheus	Open Source	Metrics, alerts
ELK Stack	Open Source	Logs, alerts

As it's visible from the table, commercial solutions mostly try to cover all observability components and be all-in-one solutions. In contrast, open-source solutions typically target one thing and can be extended via a set of plugins. For example, it's possible to integrate logging and tracing into Prometheus, but the usability of such integration is at least questionable.

How Tracing Works

The key concept of distributed tracing is the ability to transfer tracing metadata across all the layers within the system, including backend services, proxies, databases, etc.

The whole request (business transaction) is covered by a single trace id. Subsequent calls or sub-transactions are covered by span ids. Spans can be nested inside another span.

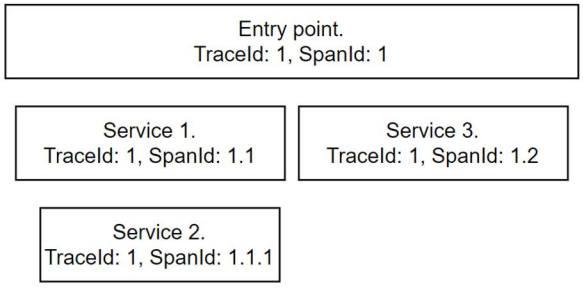

Fig. 2. TraceId and SpanId propagation

Each component reports spans to the collector, which records in in the storage. Later, a UI system can be used to analyze traces. For the sake of simplicity, for the paper Jaeger is used both as visualization tool, trace collector and storage. In the real-world enterprise systems collectors are standalone, UI components are standalone, and traces are stored in some trace storage, for instance Elasticsearch. Additionally, each trace can record tags – a key/value pairs of information that allow to store custom information within the trace.

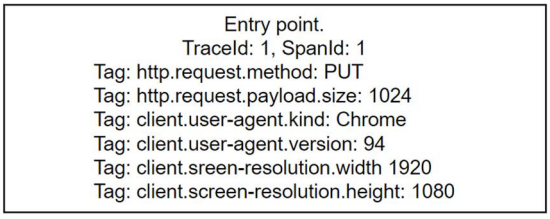

Fig. 3. Adding custom tags to the trace record

Tracing and logging integration

There is a reason why tracing and logging are often REFERENCESd together under umbrella of observability. Typically, in a distributed system logs are written by each application standalone. Ideally, they are collected to a certain centralized log storage. For example, Elasticsearch. However, there are several critical issues that logging on its own can't address:

- It's not possible to understand what caused certain errors or events.
- It's not possible to link request/business transaction to the log record.

At first look, these issues can be considered as minor, but taking a broader view – it's often not possible to understand what chain of events led to a specific scenario. The logging solution that is designed to solve the issues above is Message Diagnostic Context (MDC). The idea behind MDC is simple – allow adding additional metadata to every single log record. Thanks to MDC, the typical log record integrated with tracing includes both TraceId and SpanId, so it's making it clear to understand what transaction caused specific log record.

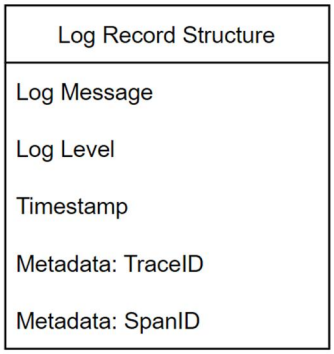

Fig. 4. Log Record Structure

Tracing implementation variations

Despite being standardized by W3C, the in-practice tracing standards differ. W3C suggests using traceparent and tracestate headers, where Zipking and compatible implementations suggest using B3 headers. In a nutshell, there is minor to none difference on what header set to use. Trace components typically can handle both formats.

Collecting data via tracing

A typical, out of the box tracing solution collects certain data. Referring HTTP as an example, a bare minimum of data that is collected contains request URI, request method, response status code. Additionally, if an exception occurs the system will mark a trace as failed, so it's possible to see what component misbehave.

At the same time, traces can be enriched both automatically and/or manually with extra data. Thanks to its tagging functionality, it's possible to add as much data as is needed for a particular business case. Most enterprise frameworks provide required functionality out of the box, but at the same time it's always possible to implement a custom solution. For future References Java, Spring Boot, and Lombok will be used to showcase examples.

Collecting HTTP Request Data

By accessing request context (HttpServletRequest) and Tracer context implementations, it's possible to enrich current span with custom request information, for instance the image below demonstrates code sample that implement logging of user agent and remote address into trace context:

```java
@Component
@RequiredArgsConstructor
public class HttpRequestTracingSample
        extends OncePerRequestFilter {

    private final Tracer tracer;

    new *
    @Override
    protected void doFilterInternal(
            HttpServletRequest request,
            HttpServletResponse response,
            FilterChain chain)
            throws ServletException, IOException {

        var span = tracer.currentSpan();
        span.tag("client.address", request.getRemoteAddr());
        span.tag("client.user-agent", request.getHeader(USER_AGENT));

        chain.doFilter(request, response);
    }
}
```

Fig. 5. Adding HTTP request context as a trace information

Moreover, it's besides tacking request metadata it's even possible to store request/response payload itself, for instance on S3 bucket. After storing the payload, it's possible to link request and/or response payload to the trace context.

```java
@Component
@RequiredArgsConstructor
public class PayloadStoringRequestSample
        extends OncePerRequestFilter {

    private final Tracer tracer;

    new *
    @Override
    protected void doFilterInternal
            (HttpServletRequest request,
             HttpServletResponse response,
             FilterChain filterChain) throws ServletException, IOException {

        var requestPayloadId = UUID.randomUUID().toString();
        Path requestStoragePath = Paths.get( first: "storage-dir", requestPayloadId);
        Files.copy(request.getInputStream(), requestStoragePath);
        tracer.currentSpan().tag("http.request.payload.id", requestPayloadId);

        // custom implementation of HttpServletRequest
        request = new CustomPayloadSupportHttpServletRequest(
                request,
                Files.newInputStream(requestStoragePath)
        );

        filterChain.doFilter(request, response);
    }
}
```

Fig. 6. Implementation of payload storage for HTTP Request

Code-level tag recording

Besides HTTP data, it's possible to record method data either manually or automatically. Manual recording allows more flexibility, however, requires more effort from the engineering team. Automated recording, in contrast, can be added via aspect-oriented programming (AOP), but it may lack some details. The best approach is to mix both, for instance in the following way:

```
@Component
@RequiredArgsConstructor
public class MethodParameterRecordingSample {

    private final Tracer tracer;

    no usages    new *
    @NewSpan
    public int recordingSample(
            @SpanTag(key = "a") int a,
            @SpanTag(key = "b") int b){
        var result =  a + b;
        tracer.currentSpan().tag("result", result);
        return result;
    }
}
```

Fig. 7. Creating new span, adding parameters as tags well as method result

Tracing of non-http communication

Besides HTTP there are plenty of tools and protocols that require tracing. Certain of them, such as gRPC and Kafka are very widely used in the distributed systems. Fortunately, all major protocols and platforms support metadata, where header information is stored. In result, there is flawless integration between HTTP and all other major protocols: gRPC, AMQP (message brokers), MQTT (message broker for IOT) and others. Moreover, there is no restriction that child span must be executed during the parent execution. This makes tracing applicable in scenarios where scheduled jobs are created and executed later, or simply messages are being processed asynchronously.

```
new *
@NewSpan
public void send(){
  var producerRecord = new ProducerRecord<String, String>
      ( topic: "demoTopic",  value: "MyMessage");
  var context = tracer.currentSpan().context();
  var headers = producerRecord.headers();
  headers.add("traceId", context.traceId().getBytes());
  headers.add("spanId", context.spanId().getBytes());
  kafkaTemplate.send(producerRecord);
}
```

Fig. 8. Adding custom headers in Kafka / Spring Boot

```
no usages  new *
@KafkaListener(topics = "demoTopic")
public void listen(
    @Header(name = "traceId") String traceId,
    @Header(name = "spanId") String spanId
    ){
  // process the message
}
```

Fig. 9. Processing custom Kafka headers in Spring Boot

Applying tracing to execution flow.

Let's create a simple geo-weather application that will be audited for security. The components used are:

- Application gateway
- Weather service
- Location service

Additionally, two clients for the weather service are in place – mobile device and web site. For simplicity, we assumed that application is already secured by OAuth2, so the part related to authentication and authorization is excluded.

Application can be presented in the following way as a component diagram:

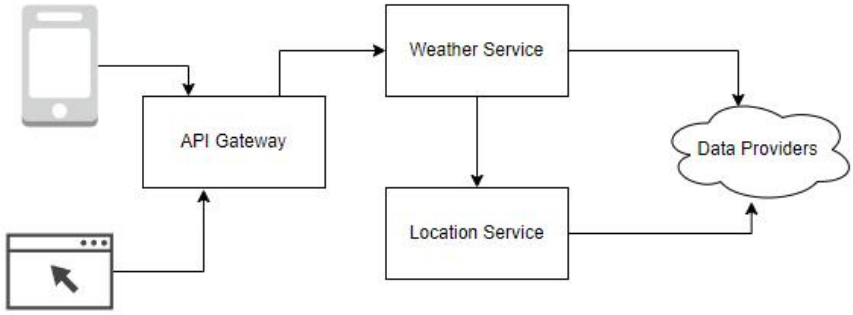

Fig. 10. Application Architecture

The use-cases that are going to be analyzed by using the sample architecture:

- Authentication and Authorization Tracking
- Anomaly detection and Incident Response
- Forensics and investigation
- Microservices security monitoring
- Data integrity verification
- Compliance monitoring
- Attacks protection and prevention
- Security context tracking

Authentication and Authorization tracking

Let's start with something simple, but important. Assuming JWT authentication is in place a client must send token via "Authorization: Bearer …" header. Application should respond 401 Unauthorized for such requests.

```
@Override
protected void doFilterInternal(
    HttpServletRequest req,
    HttpServletResponse resp,
    FilterChain chain) throws ServletException, IOException {

  var header = req.getHeader(HttpHeaders.AUTHORIZATION);
  if(header == null || !header.startsWith("Bearer ")){
    resp.setStatus(HttpStatus.UNAUTHORIZED.value());
    tracer.currentSpan().tag("error", "true");
    tracer.currentSpan().tag("http.remote.addr", req.getRemoteAddr());
    return;
  }

  chain.doFilter(req, resp);
}
```

Fig. 11. Web filter to forbid requests without bearer token

Now, it is possible to find such requests via Jaeger UI (or any other analytics tool) by applying filter expressions. Let's apply "status=401" expression:

Fig. 12. Applying filter to see all unauthorized requests

A similar approach can be used to track insufficient access permissions. For example, let's assume that user was authorized, but doesn't have corresponding permissions to invoke location-aware services. In such cases, a natural response would be 403 Forbidden.

```
var key = Keys.hmacShaKeyFor("secret".getBytes());
var parser = Jwts.parser().verifyWith(key).build();
var token = (DefaultJws<DefaultClaims>) parser
    .parse(header.substring( beginIndex: 7));
var roles = token.getBody().getSubject().split( regex: ",");
if (!Arrays.asList(roles).contains("location")) {
  resp.setStatus(HttpStatus.FORBIDDEN.value());
  resp.setStatus(HttpStatus.UNAUTHORIZED.value());
  tracer.currentSpan().tag("error", "true");
  tracer.currentSpan().tag("reason", "Missing 'location' permission");
  tracer.currentSpan().tag("token", header.substring( beginIndex: 7));
  return;
}
```

Fig. 13. Using JWT Token parser to validate permissions

The token itself and explanation of 403 are being added to the current trace, so it's possible to see and understand the reason of error directly via analytics toolset.

Anomaly detection and Incident Response

Let's assume that somebody wants to use weather website to get the data in automated fashion and it's not intended to be used in the following way. Certain indirect signs of such behavior could be:

- Way too big amount of request for a certain user/token instance
- Way too big number of requests from a certain remote address.
- Non-standard user agent sent by a client.
- Usage of non-supported or non-typical parameters, for instance checking some US city if the site is targeted only for EU market.

```
span.tag("http.request.headers.user-agent",
    req.getHeader(HttpHeaders.USER_AGENT));
span.tag("http.remote.addr", req.getRemoteAddr());
req.getParameterMap().forEach((k, v) -> {
  span.tag("http.request.params." + k,
      String.join( delimiter: ", ", v));
});
```

Fig. 14. Tracking remote address, user agent and HTTP parameters

By analyzing traces, security teams can react and add certain security rules, like deny access from the certain IP ranges. It must be noted that scenarios like this must be analyzed with advanced tooling like Elasticsearch/Kibana or all-in-one tooling like Instana. Jaeger uses logfmt format which is limited to key/value pairs. In contrast, all the tools mentioned above allow to do and/or, in/not in, contains and other types of queries. Moreover, visualization of results is also possible.

Forensics and Investigations

Let's consider a use-case, where user database was compromised, and certain sensitive data leaked. In such case in such case tracing allows to re-construct full end-2-end flow of the attacker. Let's imagine, that leak become publicly available, and the security team must take actions to understand the following:

- What data has been leaked?
- Reconstruction and timeline of the attack?
- What actions mut be taken to prevent such incidents in future?

It's possible to answer the first and second bullet by simply analyzing traces. Let's refer to Fig. 6. By storing responses at the certain storage, it's possible to exactly see what data has been collected by the attacker. At the same time, trace id will point to the full request/response sequence that allowed the attacker to download the sensitive data. By analyzing the request structure, it's possible to understand how the attacker behaved, when the breach was first used and if there are any other leaks besides the reported one.

Speaking of reconstruction of an event sequence, integration of tracing and logging opens extra possibilities what are not possible without tracing. For instance, all log records are enriched with trace id and span id attributes, what allows them to be linked to the spans. At the same time, each log record contains a timestamp attribute, so it's possible to understand in a multi-node distributed system in what order events took place.

```
{
  "@timestamp": "2024-02-05T19:28:50.4185106+02:00",
  "traceId": "d9603553506ca2633cabe642ae5916d1",
  "spanId": "c4357db323c599ae",
  "message": "Requested weather in current location",
  "logger_name": "com.cpits.demo.WeatherController",
  "level": "INFO"
}
```

Fig. 15. Log record from Weather Service enriched with traceId and spanId attributes

Microservices security monitoring

By observing the structure of the traces, it's possible to detect anomalies that may indicate service misuse or breach. Referring Fig. 10, application architecture, it's clearly visible that location service is not linked to API Gateway and should be invoked only via Weather service.

Therefore, the following abnormal cases can be detected via tracing:

- Location service is invoked via API Gateway
- Location service is invoked without any parent trace context.
- Weather service is invoked without any trace context.

The first use-case indicates a security flaw, where location service is exposed via API Gateway and used by some consumers.

The second and third use-cases are more complex. They indicate that there are network requests that invoke some services in a manner that system should not. It is a subject for investigation, case it may be possible that attacker gained access to internal application network.

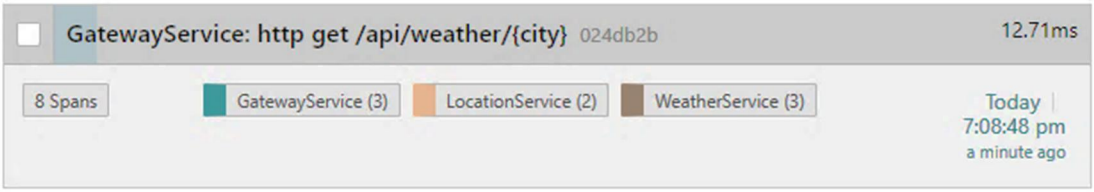

Fig. 16. Expected components invoked when location service is invoked

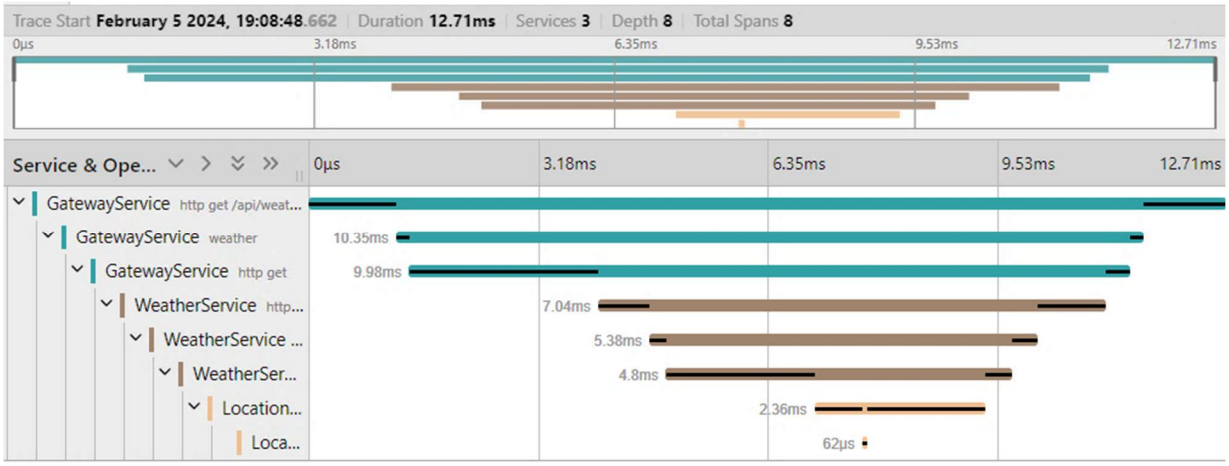

Fig. 17. Expected execution path outlook when location service is invoked

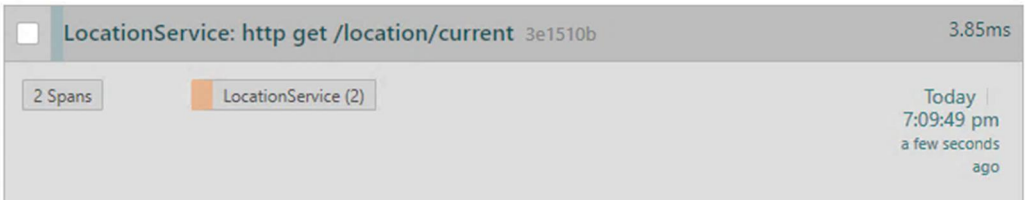

Fig. 18. Location service is invoked without parent context (API Gateway and Weather Service)

Compliance monitoring and auditing

Let's imagine a use-case, where a distributed application must match regulated environments, for instance it should flow GDPR (General Data Protection Regulation) and CCPA (California Consumer

Privacy Act). Additionally, some other requirements, for instance HIPPA (Health Insurance Portability and Accountability Act) can apply.

In such a case, application is subject to strict guidelines for the protection of confidentiality, integrity, and availability of information. In such cases, the compliance team initiates set up a continuous monitoring process, which is focused on monitoring and tracking activities related to personal (patient in case of HIPPA) data, access controls, and communication within any 3rd party services. Such monitoring is employed to trace the flow of user data. It answers the questions on who and which services access the data, how they transformed that data, did they pass the data to another system, etc.

The same approach applies to compliance auditing. Traces are being leveraged to generate reports and conduct audits.

Attack protection and prevention

Let's consider a typical example of a DDoS (Distributed Denial of Service) attack. Real-time monitoring of traces allows to implement mechanisms that will throttle or block malicious traffic based on trace patterns ensuring the availability of services. Additionally, distributed tracing integrates with Circuit Breaker pattern, what allows to see which services were affected by DDoS and how the system reacted to a given attack pattern.

At the same time, it's not limited to DDoS only. Other use cases would be bruteforce attacks, XSS (Cross-Site scripting) and CSRF (Cross-Site Request Forgery) detection and prevention.

Security context tracking

In distributed applications, each microservice or software component may have its own security measures. Distributed tracing allows security teams to ensure that security controls are being applied to all services what helps to main system security overall. Additionally, it allows to detect security flaws or incidents including layers on what they occurred. Let's assume that user is allowed to query weather by city but is not allowed to query weather by location. Thanks to tracing, it's possible to understand what exactly request failed, which user was authenticated, which roles user had at the given moment of time.

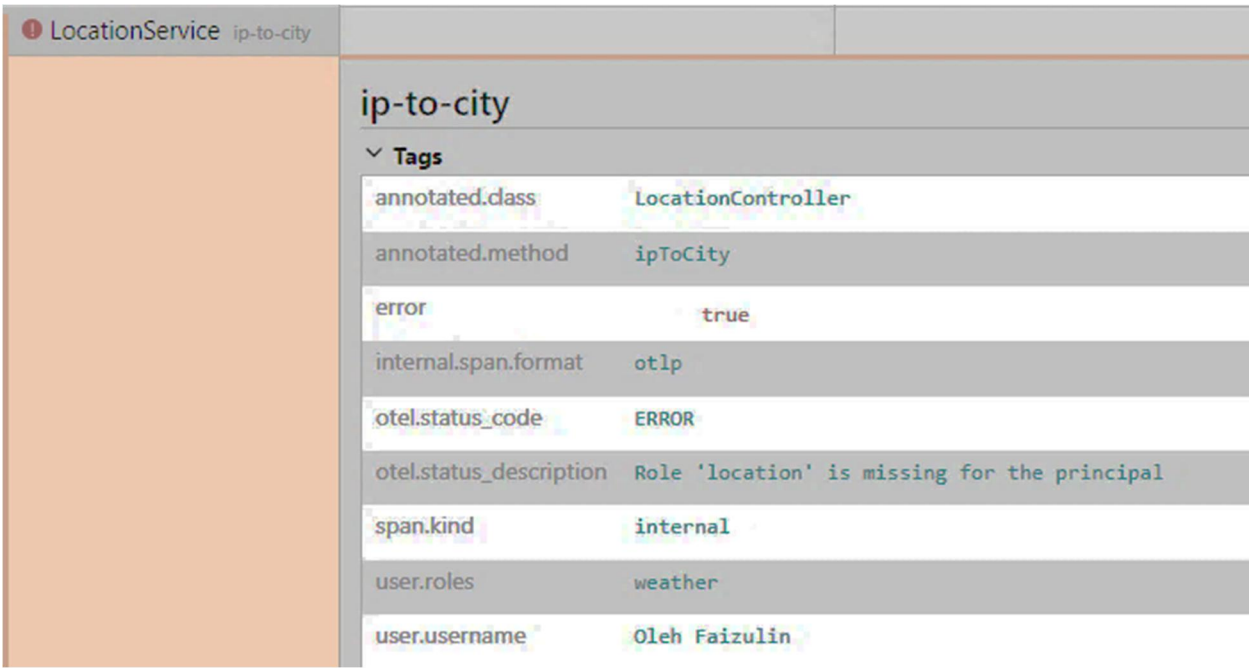

Fig. 19. Location service exposing user roles and username as tags

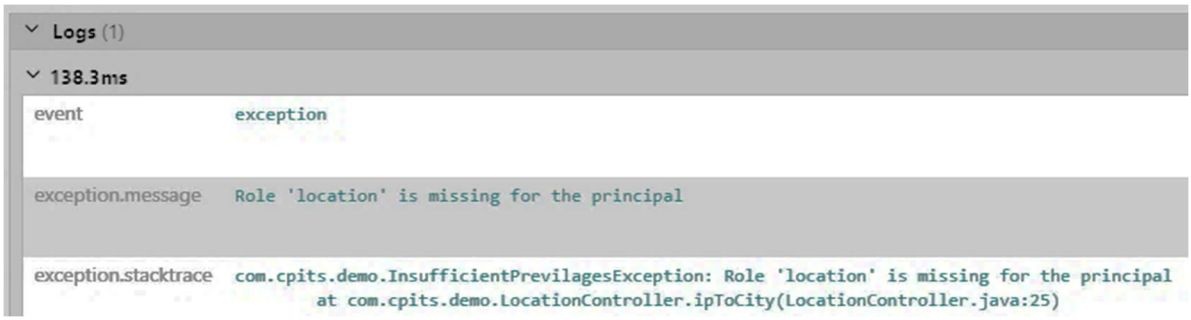

Fig. 20. Location service is attaching log record to the given span including the error details

Tooling overview

Tracing information, as well as any other information needs to be carefully analyzed and processed both in real time and during the audit sessions. Strictly speaking, tracing information without powerful tools and query mechanisms top of tracing and logging data is not very useful. The same applies to real time alert mechanisms. When working with traces, typically there are the following components in place:

- Collector, which accepts traces from the systems and stores them.
- Trace storage.
- User UI or any other analysis tool that is applied to top of stored data.

While collectors are mostly the same, the only difference between them that they are using different formats and protocols, the storage system and UI are essential for receiving value from collected traces.

Let's take an overview of how different tool categories can be used in a security context.

Standalone tracing systems

The first category, standalone tracing systems, is the simplest and the least powerful in traces analysis. Such systems excel in finding application bottlenecks, but that's it. Search capabilities are very limited and advanced query options are missing. In a security context they are barely useful and are applicable only for the simplest use cases. An example of such systems would be Jaeger and Zipkin.

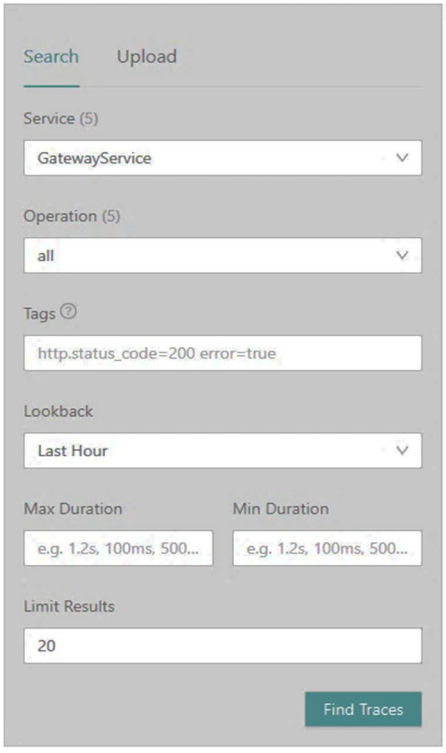

Fig. 21. Jaeger search capabilities

Commercial APM Tools

Despite the name, APM (Application Performance Monitoring) tools are often used to analyze application behavior and raise alerts. An example of such tools is Instana, Datadog, New Relic, etc. All such tools allow to configure alerting and get tracing insights. At the same time there are couple of severe drawbacks. First one – it's not necessary that all custom tracing information be available out of the box. In practice, extra configuration is required to support highly customizable tracing and often it's not an easy task to perform without vendor. Typically, this leads to extra cost and vendor lock-in.

Prometheus (Open-source tooling)

A great example of open-source software developed to provide APM functionality are Prometheus and Grafana. Prometheus is an open-source monitoring and alerting toolkit designed for reliability and scalability. Developed by CNCF, Prometheus became the de-facto standard for monitoring containerized applications and distributed application.

One of the key features of Prometheus is a full, multi-dimensional data model with key-value pairs where its time series are uniquely identified. This data model allows efficient querying and analysis of the data via PromQL (Prometheus Query Language). PromQL enables users to perform complex queries, aggregations and transformations on the collected data facilitating real-time monitoring and alerting. Prometheus is often used with Grafana, a popular open-source dashboarding and visualization platform. Grafana allows users to create custom dashboards using Prometheus data, providing a rich and interactive visualization experience. In addition to PromQL, Grafana introduces Tempo (yet another distributed tracing backend) and TraceQL (a query language specifically designed for distributed tracing in Grafana Tempo 2.0), what further facilitates distributed tracing analysis.

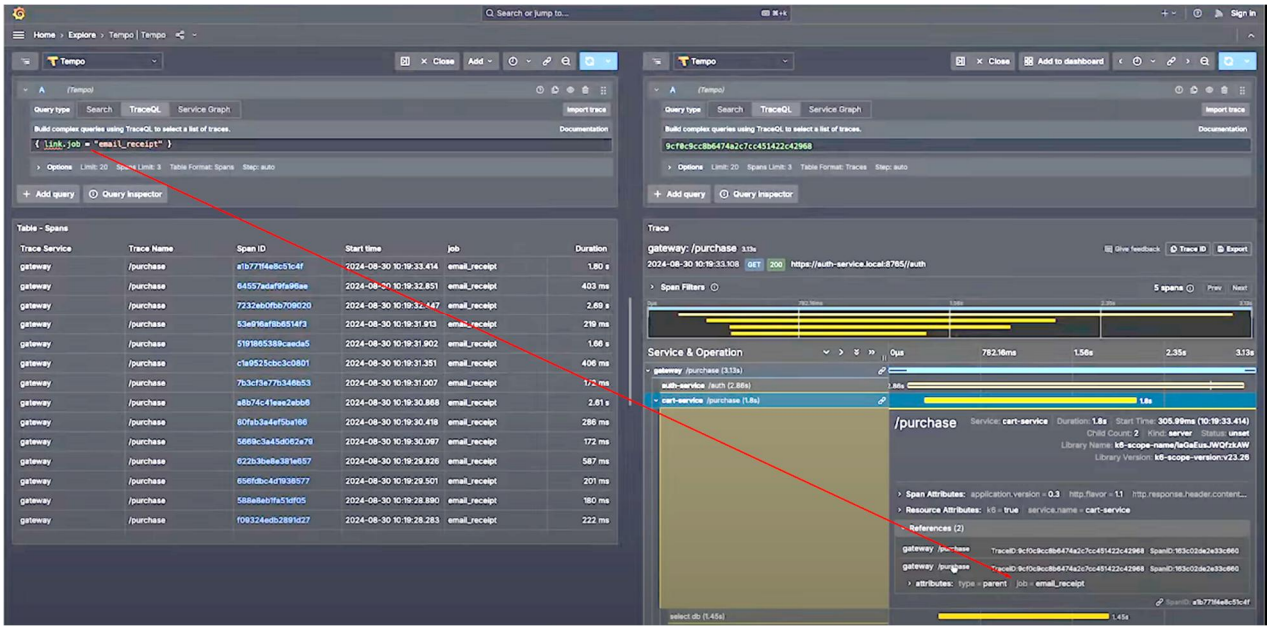

Fig. 22. Grafana, Tempo and TraceQL

Alertmanager, an alerting component of Prometheus is responsible for alerting functionality. It allows flexible alerting rules configuration including deduplication, grouping, and routing alerts to external channels (e-mail, Slack, SMS, etc.).

Elasticsearch / OpenSearch (Open-source tooling)

Despite being a full-text search engine, Elasticsearch/OpenSearch are a popular choice for saving tracing information. A set of characteristics makes the mentioned search engines an excellent choice for trace storage.

First one, near real-time indexing allows new traces to be indexed and processed almost immediately after they become available.

The second characteristic is schema-free json document structure, what allows to pre-configure individual indexes with specific patterns for trace storage.

Finally, a very powerful Query DSL that allows users to perform queries virtually of any complexity. It includes support for full-text search, filtering, aggregations providing rich capabilities on querying the data.

Kibana (OpenSearch Dashboards in case of OpenSearch) are typically used in conjunction with Elasticsearch. It is an open-source data visualization platform, which allows interactive search and analytics, including interactive exploration of the data. Like Prometheus, Kibana has built-in alerting functionality.

Production usage

Traces are one of the key observability features that must be applied to production applications either in form of commercial APM or Open-Source one. This means that typically, in enterprise grade systems data is already available. This means that applying trace-based security comes with low extra cost and usually is a matter of investment of development effort and with reasonable extra cause. Of course, applicability heavily depends on application domain and profile. Data collected by tracing can vary from megabytes to terabytes, depending on system load. Usually, some sort of sampling is done to reduce the amount of data stored.

Unfortunately, it's hard to measure a direct value of distributed tracing versus simple logging or metrics, instead tracing, metrics and logging is used in conjunction under umbrella of application observability. Moreover, as with any security tool it requires a tight integration with infrastructure components at all layers – in other cases it's not possible to apply any real-time security.

The approach for tracing-based security is not new, but, unfortunately, is rather rare. The main obstacle here is the time and effort needed to be invested in applying security queries and rules, inter-system communication, events processing. Due to the required efforts, it's often being skipped for non-mission critical systems but applied in healthcare and other security-critical domains.

In practice, if distributed tracing is implemented and properly enriched with additional metadata, it's a solution that is being used in incident investigations such as security breaches, exception analysis, application performance issues investigations. While it doesn't replace specialized tools like security scanners or application profiling tools it is very good in narrowing down the scope of the given investigation.

Conclusions

A tracing-based security approach is a very powerful tool, but it comes with a cost. Most likely, for non-mission critical application applying such approach would be overkill due to the amount of effort needed to build the infrastructure. At the same time, in a modern enterprise applications described approach is efficient and powerful tool that comes virtually at no cost. The difference between use-cases lies in an area of existing infrastructure – some form of tracing is standard to enterprises and distributed architectures because it's the only way how multi-team multi-project solutions can analyze their performance in an efficient manner. Depending on the domain, such an approach may be the only applicable one, because it allows literally to replay the sequences of events, what is not possible with other tools.

Rich ecosystem of free and commercial tools allows to setup monitoring for every scale and quality, from small websites to global enterprises. Tools like Zipkin and Jaeger can be configured in minutes to provide minimal tracing information thanks to built-in collectors, storage, and instrumentation libraries. In contrast, Enterprise APMs are costly, but they provide big amounts of application insights without any

serious development effort and are mostly a drop-in to existing applications. Their integration with tools like Kubernetes makes things easier and even more transparent for developers.

Security events coverable by trace analysis are virtually unlimited and depend on effort applied in identifying and/or automating the data processing pipelines, application domain and security regulations. The article did a review of the most typical use-cases applicable to applications of small and large scale, but at the same time there are way more use-cases not described.

Applications of mission-critical domains such as healthcare, banking, government are subject to increased security and risks, so they are major candidates for applying described approaches. A non-obvious benefit of replaying event sequence is beneficial for such applications even beyond the security – for instance it can be used to investigate user interactions with their bank accounts, so support team can decide in a support request or contradicting support cases.

Tracing based security approach doesn't compete with other approaches, methods and means of security but rather complements them. Yes, it can replace certain defaults like access logs, but it doesn't replace any of the security practices that have to be applied to applications. Developer teams are still responsible for keeping applications with up-to-date components to prevent vulnerabilities, implementing best practices of OWASP to prevent security breaches, use best practices to avoid vulnerabilities such as SQL injection, XSS, CSRF, container vulnerability scanning, etc. It does not replace infrastructure capable of protecting from DDoS attacks but can be a data source/signal for such infrastructure.

REFERENCES

1. Parker, A., Spoonhower, D., Mace, J., Sigelman, B., Isaacs, R. (2020). *Distributed tracing in practice: Instrumenting, analyzing, and debugging microservices*. O'Reilly Media.

2. Gorige, D., Al-Masri, E., Kanzhelev, S., Fattah, H. (2020, October). Privacy-risk detection in microservices composition using distributed tracing. *2020 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)*, 250–253. Ieee.

3. Meng, L., Ji, F., Sun, Y., Wang, T. (2021). Detecting anomalies in microservices with execution trace comparison. *Future Generation Computer Systems*, *116*, 291–301.

4. Rios, J., Jha, S., Shwartz, L. (2022, July). Localizing and explaining faults in microservices using distributed tracing. *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, 489–499. IEEE.

5. Jacob, S., Qiao, Y., Lee, B. (2021). Detecting Cyber Security Attacks against a Microservices Application using Distributed Tracing. *ICISSP*, 588–595.

6. Khanahmadi, M., Shameli□Sendi, A., Jabbarifar, M., Fournier, Q., Dagenais, M. (2023). Detection of microservice□based software anomalies based on OpenTracing in cloud. *Software: Practice and Experience*, *53*(8), 1681–1699.

7. Mateus-Coelho, N., Cruz-Cunha, M., & Ferreira, L. G. (2021). Security in microservices architectures. *Procedia Computer Science*, *181*, 1225–1236.

8. Jacob, S., Qiao, Y., Ye, Y., Lee, B. (2022). Anomalous distributed traffic: Detecting cyber security attacks amongst microservices using graph convolutional networks. *Computers & Security*, *118*, 102728.

9. Chandramouli, R., Butcher, Z. (2020). Building secure microservices-based applications using service-mesh architecture. *NIST Special Publication*, *800*, 204A.

10. Monteiro, D., Yu, Y., Zisman, A., Nuseibeh, B. (2023). Adaptive observability for forensic-ready microservice systems. *IEEE Transactions on Services Computing*.

11. Zhong, Z., Liu, J., Wu, D., Di, P., Sui, Y., Liu, A. X., Lui, J. C. (2023, May). Scalable compositional static taint analysis for sensitive data tracing on industrial micro-services. *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice* (*ICSE-SEIP*), 110–121. IEEE.

12. Waseem, M., Liang, P., Shahin, M., Di Salle, A., Márquez, G. (2021). Design, monitoring, and testing of microservices systems: The practitioners' perspective. *Journal of Systems and Software*, *182*, 111061.

13. Perdanaputra, A., Kistijantoro, A. I. (2020, September). Transparent tracing system on grpc based microservice applications running on kubernetes. *2020 7th International Conference on Advance Informatics: Concepts, Theory and Applications* (*ICAICTA*), 1–5. IEEE.

14. Nazarkevych, M., Dmytruk, S., Hrytsyk, V., Vozna, O., Kuza, A., Shevchuk, O., Sheketa, V. (2021). Evaluation of the effectiveness of different image skeletonization methods in biometric security systems. *International Journal of Sensors Wireless Communications and Control*, *11*(5), 542–552.

15. Medykovskyy, M., Lipinski, P., Troyan, O., Nazarkevych, M. (2015, September). Methods of protection document formed from latent element located by fractals. *2015 Xth International Scientific and Technical Conference "Computer Sciences and Information Technologies"* (*CSIT*), 70–72. IEEE.

16. Liu, P., Xu, H., Ouyang, Q., Jiao, R., Chen, Z., Zhang, S., Pei, D. (2020, October). Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks. *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, 48–58. IEEE.

17. Berardi, D., Giallorenzo, S., Mauro, J., Melis, A., Montesi, F., Prandini, M. (2022). Microservice security: a systematic literature review. *PeerJ Computer Science*, *8*, 779.

18. Gortney, M. E., Harris, P. E., Cerny, T., Al Maruf, A., Bures, M., Taibi, D., Tisnovsky, P. (2022). Visualizing microservice architecture in the dynamic perspective: A systematic mapping study. *IEEE Access*, *10*, 119999–120012.

19. Luo, S., Xu, H., Lu, C., Ye, K., Xu, G., Zhang, L., Xu, C. (2021, November). Characterizing microservice dependency and performance: Alibaba trace analysis. *Proceedings of the ACM Symposium on Cloud Computing*, 412–426.

20. Bai, L., Zhang, C. (2023, May). Trace-based microservice anomaly detection through deep learning. *Second International Conference on Electronic Information Engineering, Big Data, and Computer Technology (EIBDCT 2023)*, *12642*, 697–701. SPIE.

МЕТОДИ ТА ЗАСОБИ АНАЛІЗУ БЕЗПЕКИ ІНФОРМАЦІЙНИХ СИСТЕМ ІЗ ВИКОРИСТАННЯМ РОЗПОДІЛЕНОГО ТРАСУВАННЯ

Олег Файзулін[1], Марія Назаркевич[2],

[1, 2] Національний університет "Львівська політехніка",
кафедра інформаційних систем та мереж, Львів, Україна,
[1] E-mail: oleh.r.faizulin@lpnu.ua, ORCID: 0000-0001-5781-0600
[2] E-mail: mariia.a.nazarkevych@lpnu.ua, ORCID: 0000-0002-6528-9867

Стаття описує методи та засоби цифрової безпеки, що використовують розподілене трасування для виявлення, розслідування та запобігання інцидентам. Описані методи та засоби застосовують до рішень будь-якого масштабу (від великих підприємств до невеликих проєктів) та будь-якої галузі (охорона здоров'я, банківська справа, урядові установи, роздрібна торгівля тощо). У статті застосовано комплексний підхід до вирішення питань цифрової безпеки, охоплено процеси ідентифікації, оповіщення, запобігання, розслідування та аудиту наявних інцидентів. Описані підходи до безпеки програмного забезпечення через трасування фокусуються на інформаційних системах загального призначення, проте їх можна адаптувати для специфічних галузевих випадків. Усі підходи випробувані у виробництві в умовах і використовуються в наявних розподілених IT-системах тим чи тим способом, однак деякі приклади та випадки використання навмисно спрощені для демонстраційних цілей та простоти розуміння. Зауважимо, що методи та засоби, описані в статті, доповнюють наявні практики безпеки і не можуть повністю їх замінити, проте можуть покращити загальну безпеку системи, скорочуючи час виявлення інцидентів, зменшуючи ресурси та зусилля, необхідні для розслідування порушень або проходження аудиту безпеки.

Ключові слова: безпека, розподілене трасування, аналіз поведінки, оповіщення, автоматичне масштабування, розподілені IT-системи, метрики, логування, спостережуваність, управління продуктивністю програм, аудит.