Aalto University
School of Science

Vulnerability Diagnostics for Machine Learning Systems in Edge-Cloud Continuum

Hong-Tri Nguyen
Department of Computer Science
hong-tri.nguyen@aalto.fi

CS-E4660 Advanced Topics in Software Systems, Fall 2024
October 24, 2024

Outline

Recalls from previous Hand-ons

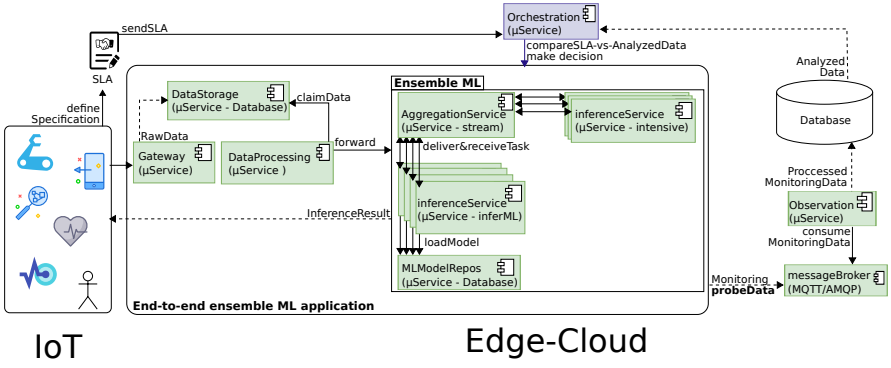

Figure: Minh-Tri's work on software system for an ML application

Vulnerability in ML systems

Data Security and Privacy

- Data Leakage: Sensitive data can be exposed
- Unauthorized Access: Ineffective access controls can lead to unauthorized access

Inter-Service Communication

- Network Security: Data transmitted between microservices can be intercepted
- Man-in-the-Middle (MitM) Attacks: Without secure communication channels, attackers can hijack the communication between services

Model Security

API Security

- Injection Attacks: APIs may be vulnerable to various forms of injection attacks

Configuration and Deployment

OWASP Machine Learning Security Top Ten[1]

- ML01:2023 Input Manipulation Attack
- ML02:2023 Data Poisoning Attack
- ML03:2023 Model Inversion Attack
- ML04:2023 Membership Inference Attack
- ML05:2023 Model Theft
- ML06:2023 AI Supply Chain Attacks
- ML07:2023 Transfer Learning Attack
- ML08:2023 Model Skewing
- ML09:2023 Output Integrity Attack
- ML10:2023 Model Poisoning

[1] https://owasp.org/www-project-machine-learning-security-top-10/

OWASP Machine Learning Security Top Ten[2]

- ML01:2023 Input Manipulation Attack
- ML02:2023 Data Poisoning Attack
- ML03:2023 Model Inversion Attack
- ML04:2023 Membership Inference Attack
- ML05:2023 Model Theft
- ML06:2023 AI Supply Chain Attacks[1]
- ML07:2023 Transfer Learning Attack
- ML08:2023 Model Skewing
- ML09:2023 Output Integrity Attack
- ML10:2023 Model Poisoning

[1] https://thehackernews.com/2024/08/researchers-identify-over-20-supply.html
[2] https://owasp.org/www-project-machine-learning-security-top-10/

Keypoint?

Hold on

Keypoint?

Hold on

What is the main concern here?

Keypoint?

Hold on

The flow of data, isn't it ???

Provenance

Def [1]

Data provenance refers to the capability to trace the lineage of data, including **WHERE** it originated from, and **HOW** it was processed and stored. This allows for a deep understanding of how and where the data is processed and stored, revealing spots along its path that could be susceptible to leaks.

AI system based on provenance

Protecting the Intellectual Property of DNN models [2]

- Provenance of Training (PoT) for verifying DNN model ownership without accessing the training dataset or affecting model performance.
- This method requires technique from intermediate checkpoints during model training, allowing the owner to verify the model chain coherence.

Transparency & Consequences in Dataset Collection [3]

- The issue of transparency in dataset collection and the need for data provenance and licensing tools to facilitate data usage in pretraining and fine-tuning.
- Lack of transparency can lead to decline in understanding training data, data leakage between training and testing datasets.

Auditing the Provenance of System Intrusions

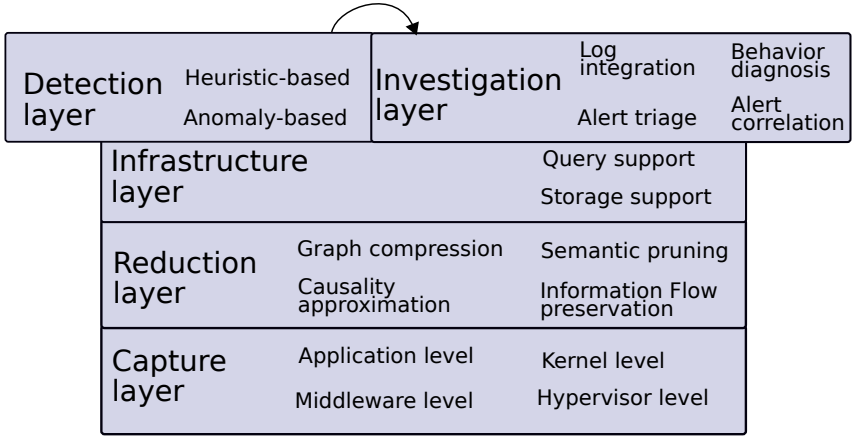

Figure: Provenance-based system auditing model [4]

Logs and metrics

Dealing with Bugs aka ?

What the developer must do?

- Logs can reveal when, where, and what caused an unexpected event or error.
- Use logs to record important system events, transactions, system failures, etc.

Logs and metrics

Is that enough?

Logs and metrics

Is that enough?

Actually, nope, and why?

Logs and metrics

Figure: RPC call graph at Uber [5] - How to analyze it? - Deep something?

Logs and metrics

Challenges from logging techniques

- When systems are complex, correlating logs from **many** different components to understand a single request can be challenging.
- Log verbosity: Too much data can be as unhelpful as too little.
- Log management: Ensuring logs are kept safe and replaced regularly to not fill up storage systems.

What is distributed tracing?

Definitions [6]

Distributed tracing is a method to monitor applications, especially based on microservices architecture, by chronicling and tracking or logging end-to-end requests when they flow through various services or components.

What is distributed tracing?

Definitions [6]

Distributed tracing is a method to monitor applications, especially based on microservices architecture, by chronicling and tracking or logging end-to-end requests when they flow through various services or components.

Trace

A trace is a record of a specific request, capturing a set of spans, events, and annotations with timestamps and ordering from every machine that the request traverses. As a Directed Acyclic Graph (DAG) formed by spans

What is distributed tracing?

Definitions [6]

Distributed tracing is a method to monitor applications, especially based on microservices architecture, by chronicling and tracking or logging end-to-end requests when they flow through various services or components.

Trace

A trace is a record of a specific request, capturing a set of spans, events, and annotations with timestamps and ordering from every machine that the request traverses. As a Directed Acyclic Graph (DAG) formed by spans

Span

A span, constituting a unit of work in a trace tree. Each edge in the trace tree represents the causal relationship among spans. A span model follows (timestamp, operation) pairs

Benefits for systems?

Benefits

- Helps to observe and understand the performance of a system
- Provides a clear, end-to-end view of how a request travels through different services and components
- Captures timing and contextual data, allowing for easier **diagnosis** and optimization.

Benefits for systems?

What are diagnoses?

Benefits for systems?

Diagnosis

- **Tail latency.** Once recognizing the peak latency, localizing is not enough, instead, it is required to identify more detail in service, code paths, and condition which cause that latency [7, 8, 9]
- **Error diagnosis.** With the existing of system failures, identifying root causes of errors is crucial; however, rare combinations of factors in a complex system generate problems whose symptoms are far from the root causes [10].
- **Temporal provenance.** In a high large scale complex system, a request perhaps interact among each others, which points the need in following the temporal provenance to determine subsequent traces of related requests through shared components [11].

Trace analysis

- Service Dependency Graphs
- Root Cause Localization
- Performance Bottleneck Identification from SLO
- Anomaly Detection

in Provenance

Def [1]

Data provenance refers to the capability to trace the lineage of data, including **WHERE** it originated from, and **HOW** it was processed and stored. This allows for a deep understanding of how and where the data is processed and stored, revealing spots along its path that could be susceptible to leaks.

1. Distributed tracing provides us with the processing lineage of the data, showing where it originated, where it moved over time, and how it was modified
2. Provenance, built on this tracing information, becomes a powerful tool to track and monitor data handling, process optimization, fault detection, and security enhancements in distributed systems.

in Root Cause Analysis [12]

An incident signifies any event that disrupts regular service operations or degrades SLA. Upon such incidents arise, a root cause analysis is undertaken to identify the underlying issue responsible for the disturbance

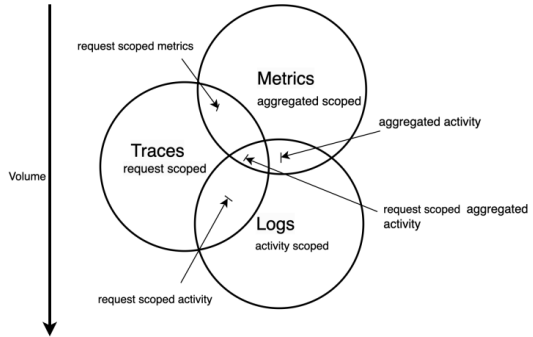

Figure: Metric-trace-log in root cause analysis

Take-home messages

Many vulnerabilities in ML systems

Pros Scalability, decoupling deployment

Cons Complexity for management/maintenance for security

- Diagnostics in performance and vulnerabilities via monitoring system data
- Challenges with interaction among μservices

Tracing as a solution

Pros Trace requests traversing any components

Cons Overhead and costs

Sol Head-based and tail-based samplings

All are perfect?

What do you think?

All are perfect?

Overhead and scalability

- Large-scale systems such as Google and Facebook [13, 14] generate billions of traces daily, with each trace approximately several **megabytes** in size, leading to substantial overhead.
- Fragmenting an entire trace can result in losing end-to-end visibility and thereby reducing the quality of the trace. To maintain trace coherence, it's necessary to process the granularity of an entire trace.
- Sampling provides a potential solution to the overhead issue. It's a coherent technique for omitting unnecessary traces while preserving essential information.

Sampling as a solution

- Many requests cause significant overhead and costs due to trace data generation. This calls for effective sampling techniques
- Sampling reduces data trace storage requirements. Cutting-edge distributed tracing systems currently collect **0.001% trace samples** [13, 14, 15].

Sampling techniques

Head-based a naive set of techniques for sampling, aim to randomly generate trace data. Upon the awareness of an incoming request, head sampling makes a decision to generate trace data

Tail-based tail sampling captures trace data for all requests and then decides to keep only useful traces and ignore redundant common-case traces. Most current works are based on ML or AI

Head-based sampling

- Head sampling is a random sampling technique used in distributed tracing
- The system begins generating trace data **once it detects an incoming request**
- Used by major companies like Google [13] and Facebook [14], as well as open-source projects like Jaeger and Zipkin

\+ Straightforward and less resources or overhead

\- As **a uniform random selection technique**, it often results in common-case execution paths and may contain redundant information

Tail-based sampling

+ See a full picture of tracing from all requests
- Challenge is defining **(1) what constitute useful traces** and decisions on tracing have to be **(2) fast and scalable**

- The variability in trace data can make it hard to distinguish between useful and redundant traces.
- Two types of information from trace data.
  - Structural information includes details such as call path, span depth, etc.
  - Temporal information pertains to the span duration.
- The tail-based sampling process can be divided into three phases:
  1. Encoding: Converting the raw trace data into a machine format
  2. Transform: Manipulating the encoded data to maximize the usefulness
  3. Sampling: Selecting a subset of the transformed traces for further processing

Trade-off

Overhead Reduction and Edge-Cases

- The head-sampling technique, as a uniformly random sampling of traces, often sacrifices useful information for overhead reduction.
- With an unbiased approach to sampling decisions for any incoming request, head sampling effectively reduces overhead.
- However, by applying a threshold or probability for trace decisions, this technique can limit necessary data for isolating an error's root cause.
- This issue is intensified in cases of temporal provenance, where traces are connected or have causal relationships.

Trade-off

Overhead Remain and Overheads Scalability

- Edge-case symptoms are directly recorded in trace data, making the retention of all trace data valuable for subsequent analysis and procedures.
- To decrease the overhead, tail-sampling techniques make use of span attributes or metrics targeting specific range of symptoms and edge-cases.
- However, since traces are generated for each incoming request, this can affect application performance in terms of latency and throughput.

Trade-off

Scalability Issues

- With the ingestion of trace data, substantial network bandwidth is required, potentially leading to bottlenecks in trace transmission.
- This can create scalability and robustness issues in trace coherence.
- Hence, significant investment in the backend trace collector is required to efficiently handle trace data, such as horizontal scaling.

Thank you

hong-tri.nguyen@aalto.fi

References I

[1] P. Buneman, S. Khanna, and T. Wang-Chiew, "Why and where: A characterization of data provenance," in *Database Theory — ICDT 2001* (J. Van den Bussche and V. Vianu, eds.), (Berlin, Heidelberg), pp. 316–330, Springer Berlin Heidelberg, 2001.

[2] Y. Liu, K. Li, Z. Liu, B. Wen, K. Xu, W. Wang, W. Zhao, and Q. Li, "Provenance of training without training data: Towards privacy-preserving dnn model ownership verification," in *Proceedings of the ACM Web Conference 2023*, WWW '23, (New York, NY, USA), p. 1980–1990, Association for Computing Machinery, 2023.

[3] S. Longpre, R. Mahari, A. Chen, N. Obeng-Marnu, D. Sileo, W. Brannon, N. Muennighoff, N. Khazam, J. Kabbara, K. Perisetla, X. Wu, E. Shippole, K. Bollacker, T. Wu, L. Villa, S. Pentland, and S. Hooker, "The data provenance initiative: A large scale audit of dataset licensing attribution in ai," 2023.

References II

[4] M. A. Inam, Y. Chen, A. Goyal, J. Liu, J. Mink, N. Michael, S. Gaur, A. Bates, and W. U. Hassan, "Sok: History is a vast early warning system: Auditing the provenance of system intrusions," in *2023 IEEE Symposium on Security and Privacy (SP)*, pp. 2620–2638, 2023.

[5] Z. Zhang, M. K. Ramanathan, P. Raj, A. Parwal, T. Sherwood, and M. Chabbi, "CRISP: Critical path analysis of Large-Scale microservice architectures," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, (Carlsbad, CA), pp. 655–672, USENIX Association, July 2022.

[6] Y. Shkuro, *Mastering Distributed Tracing: Analyzing performance in microservices and complex systems*.
Packt Publishing Ltd, 2019.

References III

[7] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[8] J. Li, N. K. Sharma, D. R. K. Ports, and S. D. Gribble, "Tales of the tail: Hardware, os, and application-level sources of tail latency," in *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, (New York, NY, USA), p. 1–14, Association for Computing Machinery, 2014.

[9] P. A. Misra, M. F. Borge, I. n. Goiri, A. R. Lebeck, W. Zwaenepoel, and R. Bianchini, "Managing tail latency in datacenter-scale file systems under production constraints," in *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19, (New York, NY, USA), Association for Computing Machinery, 2019.

References IV

[10] L. Zhang, Z. Xie, V. Anand, Y. Vigfusson, and J. Mace, "The benefit of hindsight: Tracing Edge-Cases in distributed systems," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, (Boston, MA), pp. 321–339, USENIX Association, Apr. 2023.

[11] Y. Wu, A. Chen, and L. T. X. Phan, "Zeno: Diagnosing performance problems with temporal provenance," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, (Boston, MA), pp. 395–420, USENIX Association, Feb. 2019.

[12] T. Wang and G. Qi, "A comprehensive survey on root cause analysis in (micro) services: Methodologies, challenges, and trends," 2024.

References V

[13] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag, "Dapper, a large-scale distributed systems tracing infrastructure," 2010.

[14] J. Kaldor, J. Mace, M. Bejda, E. Gao, W. Kuropatwa, J. O'Neill, K. W. Ong, B. Schaller, P. Shan, B. Viscomi, V. Venkataraman, K. Veeraraghavan, and Y. J. Song, "Canopy: An end-to-end performance tracing and analysis system," in *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, (New York, NY, USA), p. 34–50, Association for Computing Machinery, 2017.

[15] P. Las-Casas, G. Papakerashvili, V. Anand, and J. Mace, "Sifter: Scalable sampling for distributed traces, without feature engineering," in *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '19, (New York, NY, USA), p. 312–324, Association for Computing Machinery, 2019.

Hands-on

Aim

- Show

Software system for ML applications

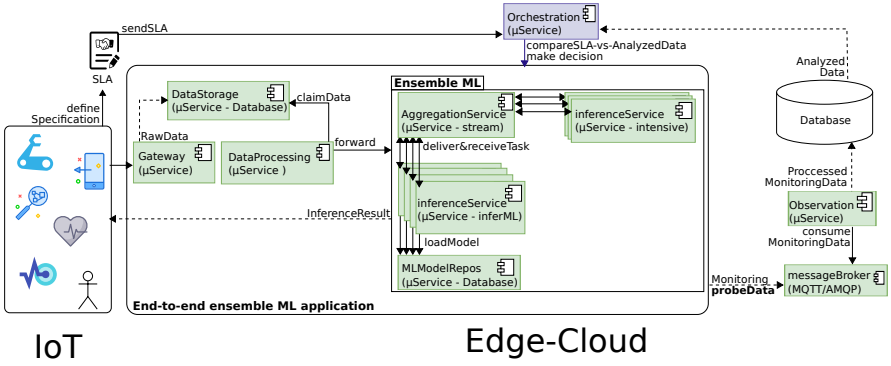

Figure: Minh-Tri's work on software system for an ML application

Distributed tracing system

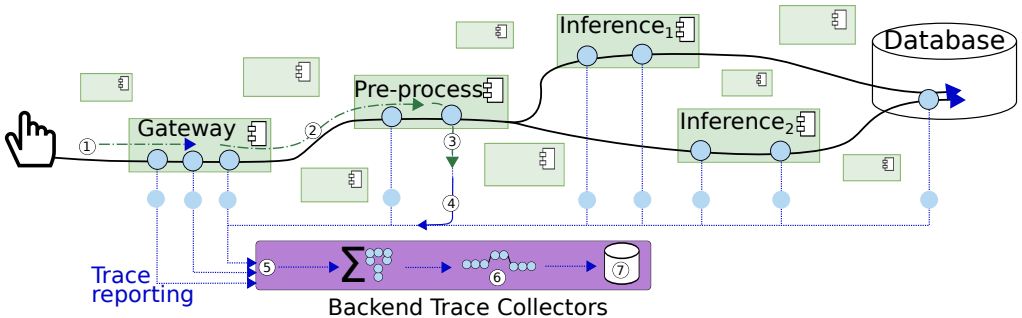

Figure: A request traverses system processes: (1) assigning a unique *traceID*, (2) propagating *traceID* and *sampled flag*, (3) annotating with *traceId*, (4) transmitting trace data, (5) backend receives, (6) processing, (7) storing

Example

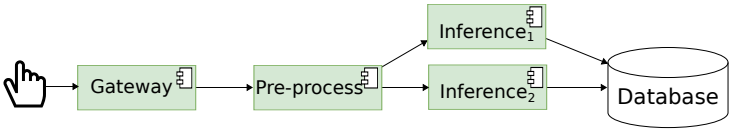

Figure: The relation between span and trace: the timeline is from left to right

Example

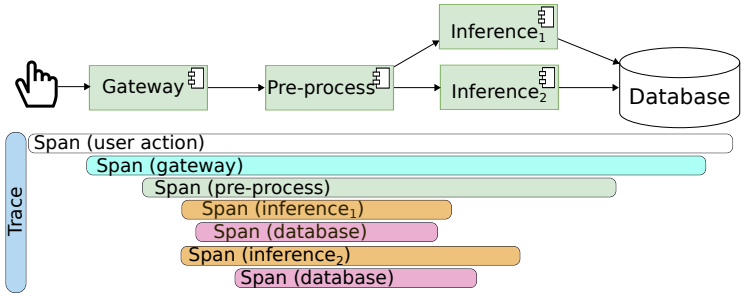

Figure: The relation between span and trace: the timeline is from left to right

Example

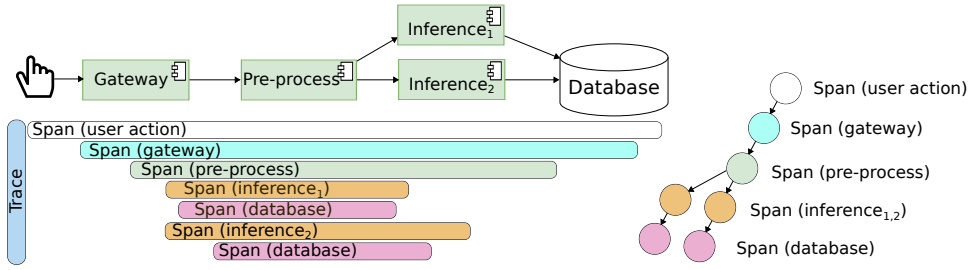

Figure: The relation between span and trace: the timeline is from left to right