# Policy Zones: Purpose Limitation at Scale using Information Flow Control

Abhishek Tiwari

Citation: *A. Tiwari*, "Policy Zones: Purpose Limitation at Scale using Information Flow Control", Abhishek Tiwari, 2024.
doi:10.59350/dg140-43703

Published on: December 11, 2024

At the heart of privacy lies the principle of purpose limitation, dictating that data should only be processed for explicitly stated purposes. This principle presents a considerable challenge, especially for organisations operating at the scale of Meta, which handles vast amounts of data from billions of users. This article examines Policy Zones, a core component of Meta's Privacy Aware Infrastructure (PAI), designed to address the complexities of purpose limitation at scale. This review is based on notes from a recent 2024 USENIX Conference on Privacy Engineering Practice and Respect(PEPR'24) presentation by Rituraj Kirti and Diana Marsala from Meta (see [1]).

## Purpose Limitation

The principle of purpose limitation, enshrined in Article 5(1)(b) of the General Data Protection Regulation (GDPR), emphasizes that personal data must only be collected for specified, explicit, and legitimate purposes. This principle ensures that data controllers define clear objectives for data processing at the point of collection, thereby promoting transparency and trust. Moreover, data should not be processed in ways that are incompatible with these initial purposes.

## Point Checking

Traditional approaches to purpose limitation often rely on "point checking" controls, essentially checkpoints within the code or data access mechanisms that verify if the intended use aligns with the stipulated purpose.

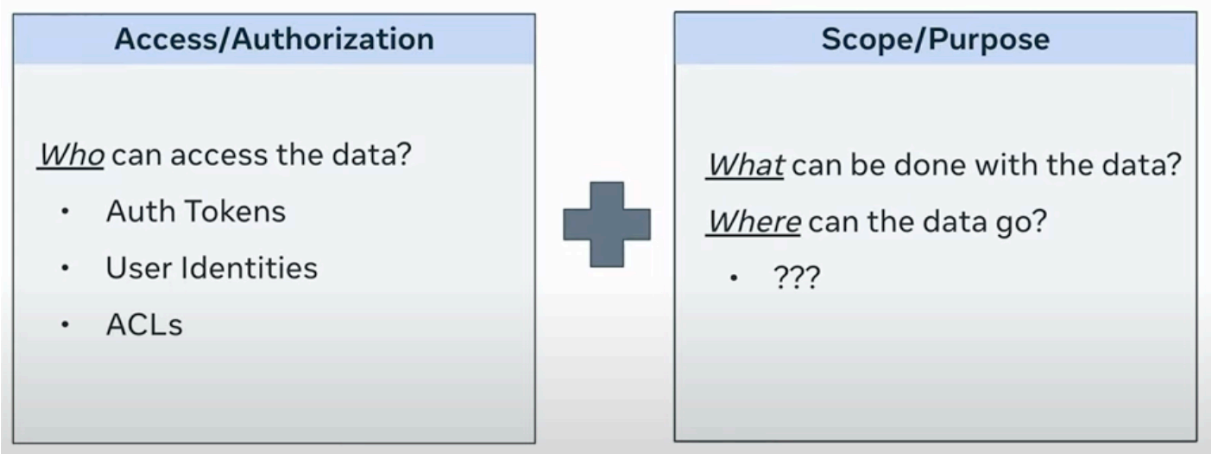


**Figure 1:** Limited prior arts in industry for doing purpose limitation at scale

Point checking controls has proven increasingly brittle in modern systems. While conceptually straightforward, point checking encounters scalability issues when applied to complex, ever-evolving systems like Meta's. In a recent article, we covered at Meta's massive microservices

architecture which encompasses more than 18,500 active services (see [2]). As codebases evolve and grow, maintaining these point checks becomes exponentially more complex. Each code modification requires careful auditing to ensure these controls remain valid, creating a maintenance burden that grows with the system. It's like trying to maintain hundreds of security checkpoints, each with its own unique ruleset that needs constant updating.

The challenge becomes even more pronounced when dealing with data access controls. Traditional ACL mechanisms require physically separating data into distinct assets to maintain purpose-specific boundaries. This is akin to storing different types of sensitive documents in separate physical vaults, each with its own access list. While this might work for smaller systems, it becomes unwieldy when scaled up.

Furthermore, enforcing purpose limitation through access control mechanisms often requires physical separation of data based on purpose. This separation, while effective at a smaller scale, introduces complexities and limitations when dealing with a sprawling infrastructure like Meta's, where data serving different purposes may be processed by shared code. This predicament necessitates a more robust and adaptive solution, leading to the development of Policy Zones.

## Data Lineage

The effectiveness of point checking can be enhanced by incorporating data flow intelligence. This advanced approach tracks data throughout its journey across systems, creating a comprehensive map of how information moves and transforms.

Think of data flow tracking like a sophisticated GPS system for your data. Using a combination of powerful techniques - static code analysis to examine how code handles data, strategic logging points to monitor data movement, and post-query processing to understand data transformations - we can build a complete picture of data's journey through our systems.

This tracking creates what we call a "data lineage" graph - essentially a family tree for your data. Like a river system with its tributaries and distributaries, data lineage maps show how data flows from its sources to its various destinations (sinks). Each connection in this graph represents a relationship where data moves or transforms, providing crucial context about how information propagates through the system.

By leveraging this data lineage information, we can make smarter decisions about permissions. Rather than applying controls blindly at individual checkpoints, we can understand the full context of data movement and apply permissions that make sense within the broader data ecosystem. It's the difference between having security guards at random checkpoints versus having a comprehensive understanding of all possible paths through a facility.

However, while combining point checking with data lineage represents an improvement over point checking alone, it still faces scaling and operational challenges. Teams must still audit numerous individual assets, and the complexity of maintaining accurate lineage information grows exponentially with system size.

## Information flow control

Secure information flow, or information flow control (IFC), refers to mechanisms designed to ensure that data within a system is transmitted or processed according to predefined security and privacy policies (see [3] and [4]). It aims to prevent unauthorized dissemination or leakage of sensitive information while allowing permissible data exchanges to occur seamlessly. This concept is particularly critical in systems where confidentiality, integrity, and regulatory compliance are paramount, such as government, finance, and healthcare domains.

At its core, IFC enforces policies like non-interference, ensuring that high-security-level data cannot influence low-security-level outputs. Techniques such as static analysis, dynamic enforcement, and hybrid methods are used to monitor or restrict how information propagates through software systems or networks. IFC models often categorise data into distinct security levels or compartments and rely on rules derived from models like Bell-LaPadula for confidentiality or Biba for integrity.

## Policy Zones

Policy Zones mark a departure from traditional methods by embracing the IFC model. Instead of relying on intermittent checks or post-hoc audits, Policy Zones operate in real time, meticulously controlling how data is accessed, processed, and transferred throughout its lifecycle. This paradigm shift offers a more dynamic and comprehensive approach to purpose limitation, effectively addressing the shortcomings of point checking and data lineage-based controls.

## The Mechanics of Policy Zones

The functionality of Policy Zones can be best understood by examining its layered structure. At the foundation lies the concept of data annotation. Data assets, which could range from database entries to request parameters, are tagged with metadata labels that encapsulate the purpose limitation requirements associated with that data. These annotations act as identifiers, informing the system about the permitted uses of the data.

The example of banana data, consistently referenced in the sources, serves as a useful illustration. Imagine a requirement stipulating that banana data can be used for making smoothies and fruit baskets but not for banana bread.

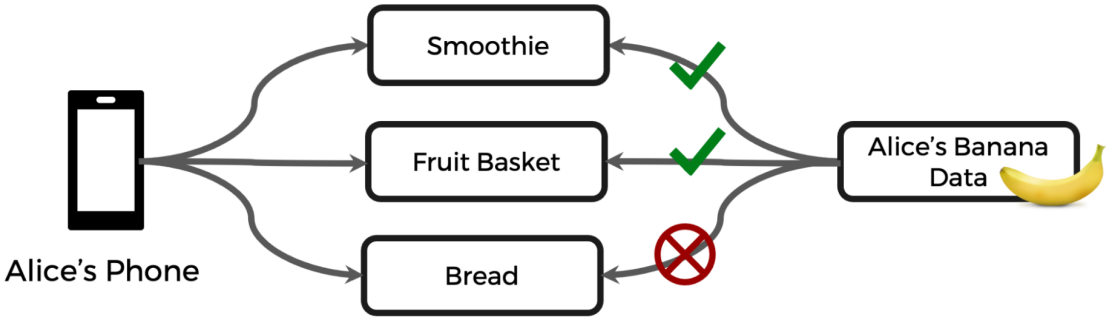


**Figure 2:** Limiting purpose of banana data. Image credits Meta.

To implement this, developers would annotate any data asset containing banana information with a label, such as `BANANA_DATA`. This label effectively embeds the purpose limitation rule within the data itself.

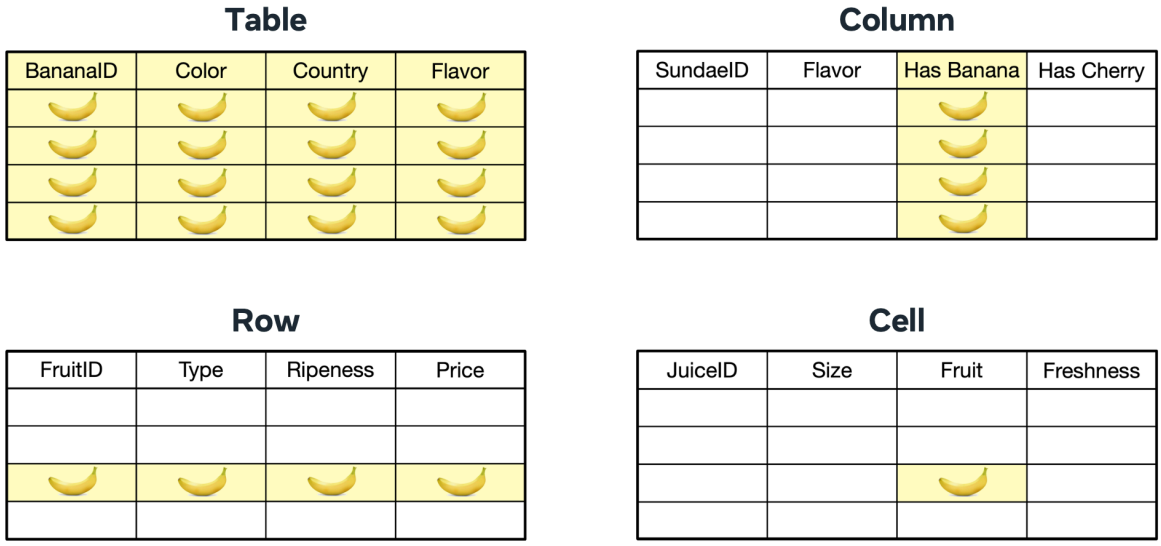


**Figure 3:** Annotation is associated with the purpose limitation requirement as a set of data flow rules that enable systems to understand the allowed purposes for the data

The second layer involves runtime enforcement. As annotated data traverses through the system, Policy Zones actively monitors the flow, verifying if each processing step adheres to the stipulated purpose limitations. This enforcement mechanism is deeply integrated within Meta's diverse systems, including function-based systems like web frontends and backend services, as well as batch-processing systems responsible for data warehousing and AI workloads. See following illustrations for how Pol-

icy Zones works for the function-based systems, while the same logic applies to the batch-processing systems as well.

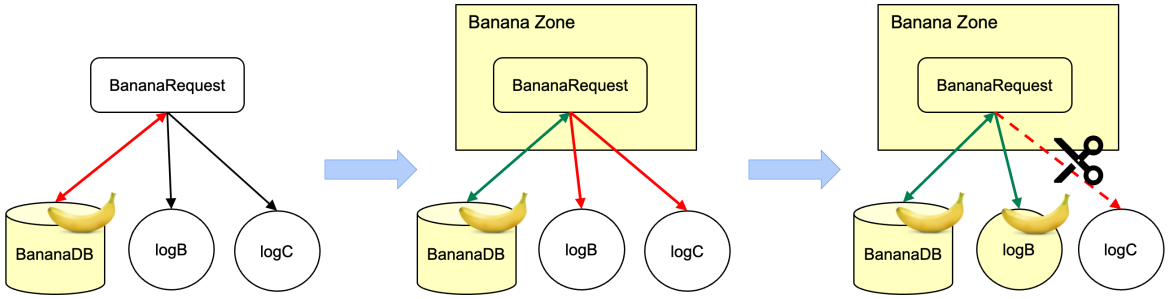


**Figure 4:** When BananaRequest loads data from BananaDB, it creates a violation due to unclear intent. After annotating BananaRequest with BANANA_DATA label, Policy Zones creates a Banana Zone. The system monitors data flows, identifies violations to logB and logC, and resolves them by annotating logB and removing logC. Once remediated, enforcement mode prevents unauthorized data flows. Image credits Meta.

When data tagged with the BANANA_DATA annotation is processed, Policy Zones springs into action, evaluating if the operation aligns with the permitted uses, namely smoothie or fruit basket creation. Any attempt to utilise this data for purposes outside this scope, such as baking banana bread, would be flagged as a violation.

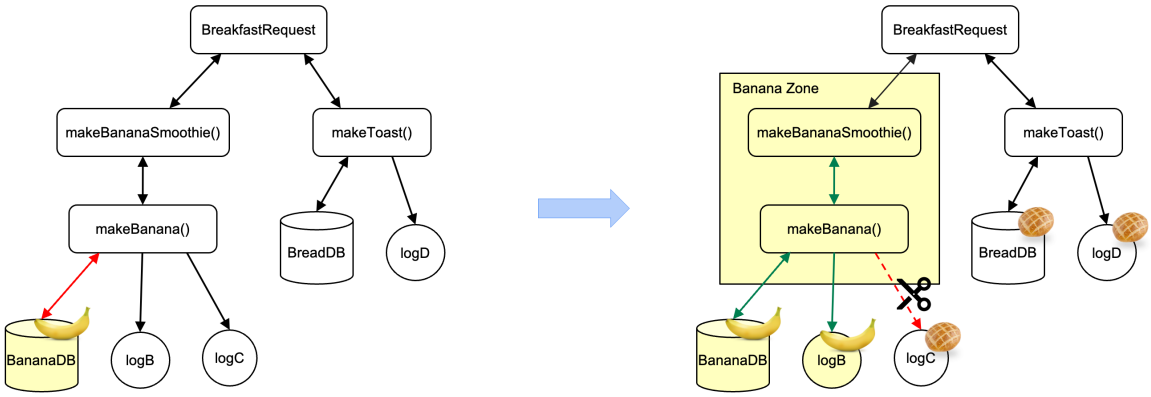


**Figure 5:** When BreakfastRequest's makeBananaSmoothie() function calls makeBanana(), we face a data flow violation as makeBanana() returns banana data. To resolve this, we create a Banana Zone starting from makeBananaSmoothie() that encompasses all functions in its call chain. Image credits Meta.

## Zone Creation and Violation Remediation

The process of data flow monitoring by Policy Zones often involves the creation of "zones". A zone essentially demarcates a section of code or a data processing pipeline where specific purpose limitations are in effect. For instance, a "Banana Zone" might encompass all functions and data flows involved in processing banana data for smoothies and fruit baskets. Policy Zones meticulously track the movement of data within and across these zones, ensuring no data escapes the confines of its designated purpose.

The identification of a data flow violation triggers a remediation process. The sources outline three primary remediation options. The first involves annotating the sink asset if the data flow is deemed safe. For instance, if banana data is used to generate an aggregate statistic about fruit consumption, where individual banana data is no longer identifiable, the sink asset receiving this statistic can be annotated with the `BANANA_DATA` label, signifying that the data flow is permissible.

The second option involves blocking data access and code execution if the intended use violates the purpose limitation. This action effectively halts the flow of data, preventing it from being misused. In the banana example, any attempt to access banana data within a function designed for bread making would be blocked, ensuring that the data remains within the bounds of its intended purpose.

The third option, termed "reclassified flow", is applied when the data flow does not actually utilise or propagate the restricted data. In such cases, the data flow is annotated to indicate its harmless nature. For instance, a function might receive input containing both banana and apple data but only utilises the apple data. This flow, while technically receiving banana data, does not process or propagate it further and hence can be reclassified as permissible.

## Policy Zone Manager (PZM)

The scale and complexity of Meta's infrastructure necessitates a robust management framework to effectively implement and oversee Policy Zones. This need is addressed by the Policy Zone Manager (PZM), a suite of UX tools designed to empower requirement owners, the individuals responsible for translating policy requirements into technical implementations, to navigate the complexities of purpose limitation.

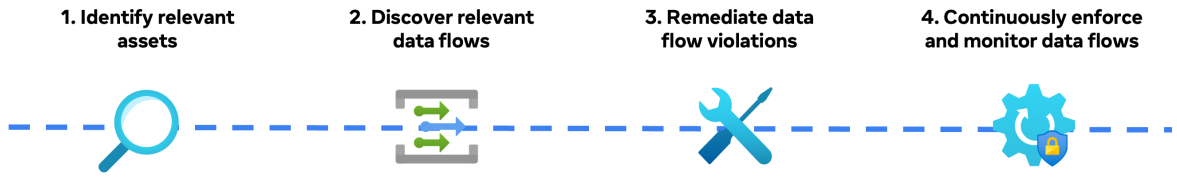


**Figure 6:** Policy Zone Manager (PZM) workflow to satisfy their purpose limitation needs in existing systems. Image credits Meta.

The workflow facilitated by PZM can be broadly categorised into four steps. The first step involves identifying relevant assets. This stage entails pinpointing the entry points of the data in question. For the banana data example, this could involve scrutinising product features that collect banana-related information, such as user preferences or purchase history. This identification process often involves manual code inspection supplemented by automated techniques, such as machine learning-based classifiers, which can scan vast codebases for patterns indicating the presence of the target data.

Once the source assets are identified, the next step is to discover relevant data flows. This task is greatly aided by data lineage tools, which provide a map of how data flows from its source to various sinks within the system. By tracing the lineage of annotated source assets, requirement owners can gain a comprehensive view of where the data travels and identify potential points of violation.

The third step, remediating data flow violations, involves applying the three remediation options discussed earlier. This stage requires careful consideration of each data flow and the context of its use. The decision to annotate, block, or reclassify must align with the overarching purpose limitation policy.

The final step is continuously enforcing and monitoring data flows. After remediating identified violations, Policy Zones can be switched from a “logging mode”, where violations are flagged but not blocked, to an “enforcement mode”. In enforcement mode, Policy Zones actively prevent any unauthorised data flows, ensuring continuous compliance with the purpose limitation policy. This enforcement is further augmented by continuous monitoring mechanisms that track the effectiveness of the implemented controls and alert administrators to any potential issues or newly emerging data flows.

## Lessons from the Trenches

Meta’s extensive experience deploying Policy Zones across its diverse systems has yielded valuable insights into the practical considerations of building and implementing a robust purpose limitation framework. One key takeaway is the importance of starting with a specific end-to-end use case. Focusing on a concrete problem, such as limiting the use of banana data, helps refine the design and

implementation of Policy Zones in a practical context. This targeted approach allows for iterative development, addressing challenges and refining the system based on real-world feedback.

Another crucial lesson is the need to streamline integration complexity. Meta's sprawling infrastructure comprises a multitude of systems, each with its own intricacies. To seamlessly integrate Policy Zones across this diverse landscape, Meta invested heavily in developing reliable, computationally efficient, and widely applicable libraries in various programming languages. These libraries provide a standardised interface for interacting with Policy Zones, easing the integration burden for developers working across different systems.

Early investment in computational and developer efficiency is another key factor contributing to the success of Policy Zones. Initial iterations of the system suffered from overly complex annotation APIs and computationally expensive data flow checking mechanisms, hindering developer productivity and scalability. Meta addressed these issues by simplifying policy representation and evaluation, leveraging language-level features for efficient context propagation, and optimising policy annotation structures. These efforts resulted in significant improvements in both computational efficiency and developer experience.

Meta also recognised the need for simplified and independent annotations to scale purpose limitation enforcement across a wide range of requirements. Initial attempts to use a monolithic annotation API to encode complex data flow rules proved unwieldy, particularly when dealing with data subject to multiple, potentially conflicting requirements. To alleviate this issue, Meta adopted a more modular approach, decoupling data from specific requirements and utilising separate data flow rules for different purposes. This separation simplifies annotation management, improves developer workflow, and enhances the system's ability to handle evolving requirements.

The importance of tooling cannot be overstated, as highlighted by Meta's experience developing PZM. Early implementations of Policy Zones relied heavily on manual processes, placing a significant burden on developers and increasing the risk of errors. The development of PZM addressed this bottleneck by providing a suite of tools that automate key tasks, such as data flow discovery and violation remediation. These tools, coupled with built-in rules and classifiers, guide developers through the implementation process, ensuring consistency and reducing the potential for human error.

## Conclusion

By transitioning from a static, point-checking-based approach to a dynamic, real-time information flow control model, Meta has created a system that offers a more scalable solution. While Policy Zones marks a significant advancement in the field, Meta acknowledges that this is just the beginning of their journey towards robust and sustainable privacy protection.

## References

[1] R. Kirti and D. Marsala, “Approaches and Challenges to Purpose Limitation across Diverse Data Uses,” 2024. Available: https://www.usenix.org/conference/pepr24/presentation/kirti

[2] A. Tiwari, “Microservice architecture of Meta,” 2024, *Abhishek Tiwari*. doi: 10.59350/7x9hc-t2q45.

[3] A. Sabelfeld and A. C. Myers, “Language-based Information-Flow Security,” *IEEE Journal on Selected Areas in Communications*, 2003, doi: 10.1109/JSAC.2002.806121.

[4] D. E. Denning, “A Lattice Model of Secure Information Flow,” *Communications of the ACM*, 1976, doi: 10.1145/360051.360056.