# Faceted Data: Protecting Systems from Data Leakage

# Introduction

Faceted data is an approach that helps protect systems from data leakage by offering precise control over information flow (Schmitz et al., 2016). It allows for selective disclosure of sensitive data based on the security context.

# Advantages of Faceted Data

* Fine-grained control: Faceted values provide flexibility in defining and enforcing security policies, allowing for dynamic adaptation to changing security requirements.
* Selective disclosure: Faceted data enables the disclosure of sensitive information only to authorized entities, preventing unauthorized access and leakage.
* Security context awareness: Faceted values carry contextual information about the security level of data, ensuring proper handling and access control.
* Policy enforcement: Faceted data supports the enforcement of security policies during runtime, reducing reliance on static analysis and providing more dynamic protection.

# Disadvantages of Faceted Data

* Performance overhead: The use of faceted data can introduce computational overhead due to the need for managing different views and performing security checks.
* Complexity: Handling and reasoning about multiple views and security contexts can add complexity to the system design and development process.

## Applying Faceted Data in Python

## Define Faceted Value Class

Create a class to represent faceted values, encapsulating the underlying data and security contexts.

*class FacetedValue:*

*def \_\_init\_\_(self, value, security\_contexts):*

*self.value = value*

*self.security\_contexts = security\_contexts*

## Implement Security Policy Functions

Define functions to enforce security policies based on the security context of faceted values. These functions should handle the logic for selective disclosure and access control.

*def enforce\_policy(value, current\_context):*

*if current\_context in value.security\_contexts:*

*return value.value*

*else:*

*return None # Access denied*

*def join\_faceted\_values(value1, value2):*

*# Perform faceted value join operation based on the security contexts*

*# and combine the underlying values*

*...*

## Incorporate Faceted Values in Data Handling

Use faceted values in operations involving sensitive data, ensuring proper security checks and enforcement of policies.

*def handle\_sensitive\_data(data, security\_context):*

*faceted\_value = FacetedValue(data, [security\_context])*

*result = enforce\_policy(faceted\_value, security\_context)*

*if result is not None:*

*# Process the data*

*...*

*else:*

*# Access denied, handle accordingly*

*...*

## Utilize Faceted Values for Secure Information Flow:

Apply faceted values to control the information flow within the system, allowing selective disclosure and access control based on security contexts.

*def process\_data(data, security\_context):*

*faceted\_value = FacetedValue(data, [security\_context])*

*result = enforce\_policy(faceted\_value, security\_context)*

*if result is not None:*

*# Perform operations with the processed data*

*...*

*else:*

*# Access denied, handle accordingly*

*...*

# Conclusion

Faceted data provides a powerful mechanism for protecting systems from data leakage by offering fine-grained control over information flow. By incorporating faceted values and implementing proper security policies, systems can ensure selective disclosure and access control, enhancing overall security and privacy.

# References

Schmitz, G., Torrini, P., & De Cristofaro, E. (2016). Faceted Data: It’s More than Security!. Proceedings on Privacy Enhancing Technologies, 2016(4), 151-167.