Analyzing Cyclic Data Flow Diagrams Regarding Information Security

Benjamin Arp
benjamin.arp@student.kit.edu
Karlsruhe Institute of Technology (KIT)

Nils Niehues
nils.niehues@kit.edu
Karlsruhe Institute of Technology (KIT)

Tom Hüller
tom.hueller@student.kit.edu
Karlsruhe Institute of Technology (KIT)

Felix Schwickerath
felix.schwickerath@student.kit.edu
Karlsruhe Institute of Technology (KIT)

Nicolas Boltz
nicolas.boltz@kit.edu
Karlsruhe Institute of Technology (KIT)

Sebastian Hahner
sebastian.hahner@kit.edu
Karlsruhe Institute of Technology (KIT)

Abstract

Data flow diagrams are commonly used in system design to represent data processing and exchange. They are valuable in security analysis due to their applicability in assessing information security-related properties like confidentiality. However, many existing tools for data flow analysis are limited by the assumption that data flows form acyclic graphs, which inhibits the analysis of cyclic data flows, common in real-world software systems. This paper addresses this gap by implementing a novel method to resolve cycles in data flow diagrams while preserving their semantics regarding information security. We validate our method, ensuring it is accurate, lucid and preserves information security-related behavior.

1 Introduction

Data Flow Diagrams (DFDs) are a fundamental format used in system design and security assessments [3]. DFDs depict how data flows through a system, providing a well-known structure that can be extended to support automated information security analysis [4, 5]. However, existing tools for Data Flow Analysis (DFA), like the approach of Boltz et al. [5], are designed to work on Directed Acyclic Graphs (DAGs), which fail when dealing with data flow loops.

The ability to model cyclic DFDs is essential in scenarios where a system's behavior is non-linear, involving repetitive data processing or iterations. This paper explores different types of cycles that can occur in DFDs and outlines resolution strategies. We introduce enhancements to the DFA framework proposed by Boltz et al. [5] that transforms cyclic DFDs into acyclic Transpose Flow Graphs (TFGs), to extend its capabilities to accommodate cyclic DFDs. We discuss the resulting capabilities of the enhanced DFA and validate it based on a dataset of DFDs derived from open-source micro-service projects. Our findings show that this transformation preserves the critical security behavior of systems and improves the performance and coverage of existing analysis frameworks.

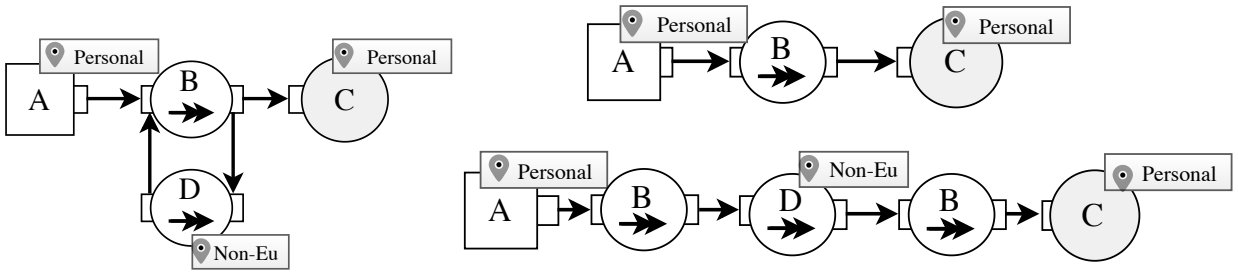
2 Background

Our approach in this paper builds upon the unified data flow diagram metamodel of Boltz et al. [5], a widely used and maintained framework for DFA. In the center of their notation is the representation of *behavior* and *labels*. Labels are shown as boxes attached to a node in Figure 1. They can either be defined as a label of a *node* or as a label of data flowing between nodes. The behavior of a node defines which labels flow from one node to the next via the connecting data flow. A simple example is the forward behavior, shown by $\twoheadrightarrow$ in Figure 1. A node with the forward behavior passes on all labels that flow into it to the next node.

Simplified, the DFA approach of Boltz et al. [5] uses a depth-first search, starting at sink nodes of the DFD, following against the flow of data until all possible source nodes that flow into the sink have been reached. Each individual flow of data in the DFD is represented as a TFG, on which the following analysis steps operate. This results in the limitation that the analysis requires *well-defined* TFGs that have clear sources and sinks [5]. In DFDs with cycles, it is not always possible to identify definite source and sink nodes, requiring solutions to resolve or deal with cycles. To examine confidentiality-related security properties, Boltz et al. [5] employs recursive propagation of each label within the TFG, facilitating the querying of constraints. This method assumes that the DFD does not exhibit oscillating behavior, as such behavior would prevent the determination of a clear termination point.

3 Approaches to Cycle Resolution

We first need to understand how cycles arise, to then resolve cycles in DFDs. We define cycles in a DFD as two or more data flows with cyclic dependencies, such



(a) Data flow diagram with cycle

(b) Cycle-free transposed flow graphs

Figure 1: Example of a simple DFD containing a one cycle with no further interactions. And the two resulting TFGs achieved by our cycle resolving.

as between *node B* and *node D* in Figure 1a. Here, the flow from *B to D* forwards labels from B to D, which in turn forwards its labels from *D to B* and so on.

Through our analysis, we identified two dimensions that characterize a cycle. The first dimension pertains to the position in the DFD, whether the cycle is the *source*, *sink* or an *embedded* part within the DFD. For instance, a DFD composed of a single loop is source and sink simultaneously. The second dimension is the interaction outside of a cycle with the rest of the flow, which may consist of multiple other cycles. The DFD in Figure 1a represents the simplest form: a *standalone cycle*, which is embedded in an otherwise cycle-free graph. When multiple cycles exist within a DFD, we categorize them into parallel, sequential, and shared interactions. *Parallel Cycles* refer to multiple cycles that coexist simultaneously, running parallel to each other on different branches. Although these cycles have no direct interaction, they may influence each other when analyzing their behavior and label propagation. *Sequential Cycles* occur when two or more cycles exist in sequence without shared nodes between them, interacting with the flow of the previous cycle. When two or more cycles share common nodes, we define this as a *shared interaction*. These shared interactions vary in complexity, from sharing a single node to more intricate configurations, such as nested or overlapping cycles. A single DFD can encompass multiple types of cycles, increasing the complexity of the graph and making the analysis more computationally demanding.

In our study, we consider three strategies to resolve cycles efficiently: fully removing the cycles in the DFD before the analysis, resolving the cycles during the transformation of the DFD into the internal TFG representation of the analysis, and modifying the DFA algorithm and concepts to accept cyclic flow graphs.

Removing the cycles in the DFD requires an extra preprocessing step dedicated to cycle resolution. For this, we could imagine collapsing a loop into a Strongly Connected Component (SCC) [1], or alternatively, decomposing the loop into a finite or infinite amount of sub-graphs, as proposed by [2, 4]. While collapsing the loop into an SCC is computationally efficient, it leaves questions about how to analyze the node and data labels of the flows within the cycle. Resolving the loop into sub-graphs is computationally more demanding but allows for acyclic analysis.

A similar resolution approach can be used to enhance the DFD to TFG conversion step of the analysis of Boltz et al. [5]. This approach allows us to keep the input DFD as modeled and requires only slight changes to the conversion and resulting internal artifacts of the analysis.

Finally, we also briefly considered modifying the DFA algorithm to process cyclic flows. However, due to the existing complexity of the algorithm and its initial design for processing acyclic TFGs [4, 5], we chose not to pursue this variant in detail.

4 Enhancing the Data Flow Analysis

To resolve the cycles during the transformation of the DFD into TFGs, we rely on the assumption of well-definedness, as described in Section 2, and the following considerations. First, we treat loops as optional flows rather than mandatory ones, allowing the system to function without necessarily entering these loops. This approach is essential due to the implications of the *Halting problem*—after completing one iteration of the loop, we would otherwise be compelled to re-enter the loop, as it would be required by the node's behavior. However, disregarding a required flow in this manner could result in unexpected constraint violations. Additionally, we assume that a subset of the TFG is sufficient for conducting cyclic DFA. We, however, acknowledge that specific interactions may be too complex to compute and may require human intuition and intervention.

Building on these assumptions, we extended the depth-first search algorithm proposed by Boltz et al. [5] to handle additional cases where new TFGs are generated without a cycle. The following algorithm outlines the process of resolving cycles in DFDs:

```
Algorithm: Resolve cycles in DFD
Input: DFD with possible cyclic flows
Output: Set of acyclic TFGs
1. Initialize empty set of visited nodes
2. Perform TFG search
   a. Perform depth-first search
   b. If cycle is detected through revisiting a node:
      i. Identify the whole cycle
      ii. Iterate ONCE before exiting the cycle
      iii. Generate corresponding TFGs
      (with and without the cycle)
   c. Continue search to analyze the remaining DFD
3. Return the set of acyclic TFGs
```

This approach enables the resolution of various cycle types, enhancing the flexibility of the analysis. The table below highlights our capability to handle different cycle interactions across various positions in the DFD:

Interaction\Position	**Source**	**Embedded**	**Sink**
Standalone	✓	✓	-
Parallel	✓	✓	-
Sequential	✓	✓	-
Shared	∼	∼	-

Table 1: Resulting capabilities: ✓ indicates exact representation, ∼ indicates good estimation based on a subset, and − indicates the case is not supported due to undefined behavior.

In summary, this approach provides an effective approximation for most cycle types, though complex cases involving shared nodes may require further refinement and human intervention. Performance remains robust in under a second, with efficient handling of common cycle patterns, although cases involving sink nodes present limitations due to their undefined behavior in the context of this framework.

5 Validation

We establish three key validation goals to ensure that our approach reflects the original system's intended functionality and security properties without compromising its integrity. **Behavior-preservation** stipulates that the transformation process must preserve the semantics of the original flow graph or, if necessary, approximate them accurately. **Completeness** requires all acyclic paths in the original flow graph to be present in the transformed flow graph. **Lucidity** mandates that the transformed flow graph should not contain any paths not present in the original flow graph.

We validate our approach with the dataset by Schneider et al. [6], which includes 97 well-defined cyclic DFDs representing various cycle types (excluding sink cycles), as a base for evaluating the behavior-preservation of our approach. To assess whether our approach maintains information security behavior, we formalized constraints described by Schneider et al. [6] to test for the absence or presence of security concerns. The results demonstrated that resolving the cycles does not introduce false negatives, ensuring that all existing security issues are still identified. To validate this, we compared the analysis results against the vulnerabilities documented in the dataset and did a manual review of the models to ensure that the approach accurately captured all relevant security properties. However, by approximating the data flow in more complex cycles, we observed negligible false positives and achieved a precision of 0.989 in accurately preserving the security behavior. The algorithm is designed to ensure both completeness and lucidity. Completeness is conditionally ensured by the assumption that the found subset of TFG is sufficient to preserve the model's behavior, which is the primary objective in maintaining the integrity of the analysis. This assumption, together with the completeness of the approach, is therefore validated by the high precision achieved in behavior preservation. Lucidity is inherently guaranteed as we only replicate existing flows, thereby preventing the introduction of any new flows.

A potential external threat is that the set of 97 models used for evaluation may not fully represent the vast range of real-world systems. However, this risk is mitigated by the diverse backgrounds of the microservice applications and security rules used. To enhance reliability, the algorithm is available online at `DataFlowAnalysis.org`. Limitations of the approach include incomplete support for sink cycles and highly complex nested cycles, which may require manual intervention for accurate analysis.

6 Conclusion

In this paper, we presented our approach to resolving cycles in DFDs. We identified 12 types of cycles that can be categorized into two dimensions: position and interaction. To address these cyclic flows, we introduced an additional step in the conversion process from DFDs to TFGs, enabling us to represent cyclic flows as a finite set of acyclic TFGs. Our validation shows that this approach effectively preserves information security behavior while minimizing false positives.

During our work, we identified several points for future research, like incorporating cyclic sink resolving techniques, which would allow our method to handle sink cycles. Additionally, integrating artificial intelligence into the cycle resolution process could solve the starting point problem we face with sink cycles or improve estimations.

Acknowledgements

This publication is partially based on the research project SofDCar (19S21002), which is funded by the German Federal Ministry for Economic Affairs and Climate Action. This work was also supported by funding from the topic Engineering Secure Systems of the Helmholtz Association (HGF) and by KASTEL Security Research Labs, the BMBF (German Federal Ministry of Education and Research) grant number 16KISA086 (ANYMOS), and the NextGenerationEU project by the European Union (EU).

References

[1] R. Tarjan. "Depth-First Search and Linear Graph Algorithms". In: *SIAM Journal on Computing* 1.2 (1972).

[2] R. Kramer, R. Gupta, and M. Soffa. "The combining DAG: a technique for parallel data flow analysis". In: *TPDS* 5.8 (1994).

[3] K. Bernsmed et al. "Adopting threat modelling in agile software development projects". In: *JSS* (2022).

[4] S. Seifermann. "Architectural Data Flow Analysis for Detecting Violations of Confidentiality Requirements". Dissertation. KIT, 2022.

[5] N. Boltz et al. "An Extensible Framework for Architecture-Based Data Flow Analysis for Information Security". In: *ECSA*. Springer. 2023.

[6] S. Schneider et al. "microSecEnD: A Dataset of Security-Enriched Dataflow Diagrams for Microservice Applications". In: *MSR*. 2023.