B. Arp, N. Niehues, T. Hüller, F. Schwickerath, N. Boltz, and S. Hahner, "Analyzing Cyclic Data Flow Diagrams Regarding Information Security", 16th Symposium on Software Performance (SSP), Softwaretechnik-Trends, Gesellschaft für Informatik e.V., 2024, https://dl.gi.de/handle/20.500.12116/45545

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 further introduce enhancements to the DFA framework proposed by [5], 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 microservice projects.

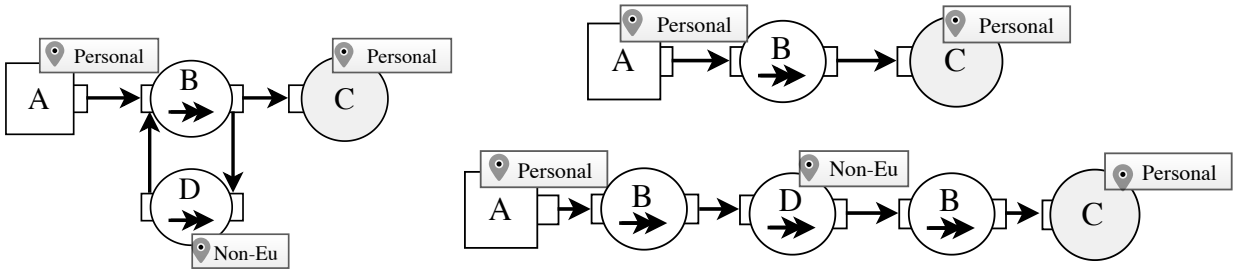
2 Current Data Flow Analysis

Our approach in this paper builds upon the unified data flow diagram metamodel of [5]. In the center of the 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 Transpose Flow Graphs (TFGs), 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.

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 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. In addition to their own cyclic interactions, cycles interact with the rest of the flow. During our work, we identified 12 possible types of cyclic interaction. These interactions define how cycles connect with other elements of the DFD. For clarity, we use



(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.

the term *in-node* to refer to a node that has an incoming flow from outside the loop, and *out-node* to refer to a node that allows outflows to exit the loop.

The DFD in Figure 1a represents the simplest type, a **Single Cycle** in an otherwise cycle-free graph. If we have more than one cycle, we can have *direct* and *indirect* interactions between the cycles. **Sequential Cycles** occur when two or more cycles exist in sequence without shared nodes between them. **Parallel Cycles** refer to multiple cycles that exist simultaneously, running parallel to each other or on different branches. While these three types of cycles have no direct interaction, they might interact when looking at the behavior and labels. In contrast, **Branching Cycles**, which occur when one cycle branches off from another, creates a direct interaction where one cycle depends on the iteration of the loop.

When taking a look at cycles that share nodes, we discovered three distinct types: **Shared Node Cycles**, where two or more cycles share a common node, with the in-node of one cycle serving as the out-node of another. **Nested Cycles** where a cycle exists within another cycle, forming a nested loop structure. **Partially Overlapping Cycles** which involve two or more cycles that share multiple, but not all, nodes.

Lastly, we identified two types, that interfere with the structure of the DFD: **Starting Cycles** exist without an in-node, causing the DFD to have no direct data flow source. **Terminal Cycles** lack an out-node, causing the DFD to have no direct sink of the data flow. The complexity and nature of each type of cycle and interaction present unique challenges.

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 pre-processing 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 unresolved 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. However, how the result can be used as input for the DFA is unclear and might involve a high level of effort to correctly post-process the analysis results so that they are comprehensible and align with the original DFD.

A similar resolution approach can also 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, meaning the system can operate without entering these loops. 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 follow the depth-first search algorithm of the analysis of Boltz et al. [5] and introduce additional cases in which new TFGs are created. For the simple cases, like shown in Figure 1a that contain one cycle in an otherwise cycle-free graph, we extract two TFGs: one excluding the loop altogether and another including a single it-

eration. Figure 1b shows the two resulting TFGs for the loop shown in Figure 1a.

For more complex loops and interactions, we iterate over every sub-graph of the loop with the depth-first search. Once a cyclic flow is identified, we pinpoint the specific data flow responsible for causing the loop by tracing the flow until we encounter nodes in the cycle that have already been revisited. This flow is then excluded from the further analysis, and we continue our search. If we do not find further flows, we create a new source node, approximating a source. This approach aligns with the principles outlined by [4].

With this approach, we can provide exact representations of DFDs with Single Cycles, Sequential Cycles, Parallel Cycles, or Branching Cycles. With the approximation of source nodes, we effectively handle Starting Cycles and Shared Node Cycles. While it is possible to generally analyze DFDs with Nested Cycles and Partially Overlapping Cycles, the results might be faulty due to their general complexity, requiring human interaction beforehand. However, we are not able to correctly estimate sink nodes in DFDs that contain Terminal Cycles, resulting in not well-defined DFDs that can not be analyzed.

Our approach's additional general limitation is cycles containing oscillating data flows. This means that labels that are passed along the data flow by the behavior of the nodes vary from one iteration of a loop to the next [4], resulting in an infinite number of TFGs or TFGs of infinite length as a loop can never be clearly identified without extensive additional effort.

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. **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. **Behavior-preservation** stipulates that the transformation process must preserve the semantics of the original flow graph or, if necessary, approximate them accurately.

We use the dataset by Schneider et al. [6], which includes 97 well-defined cyclic DFDs representing various cycle types (excluding terminal cycles), as a base for our evaluation. To assess whether our approach maintains information security behavior, we developed constraints 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. However, by approximating the data flow in more complex cycles, we observed negligible false positives with a precision of 0.989.

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 three groups: standalone Cycles, interacting Cycles and TFG-disrupting cycles. To address these cyclic flows, we introduced an additional step in the conversion process from DFDs to TFGs, enabling us to represent the flow as a 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 Terminal Cycles. Additionally, integrating artificial intelligence into the cycle resolution process could solve the starting point problem we face with Terminal 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.