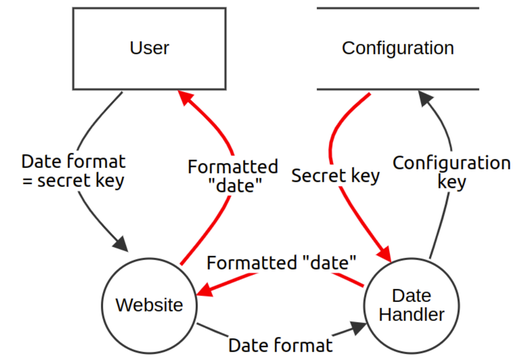
Accurate Architectural Threat Elicitation From Source Code Through Hybrid Information Flow Analysis

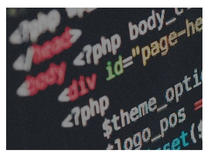
Bernd Gruner, Secure Software Engineering Group, DLR Institute of Data Science

Why to Trace Information Flows?

Software processes a vast amount of sensitive data, such as passwords, certificates, or configurations. However, **tracing information flows within complex programs poses challenges** but could help to identify and mitigate threats. On the right-hand side, an information flow graph of a Django application with a known vulnerability [1] is depicted. In this scenario, an unintended information flow occurs between sensitive content from the configuration file and the user, caused by the malicious date format. **Existing approaches using fuzzing, taint analysis, or symbolic verification do not address such threats.**

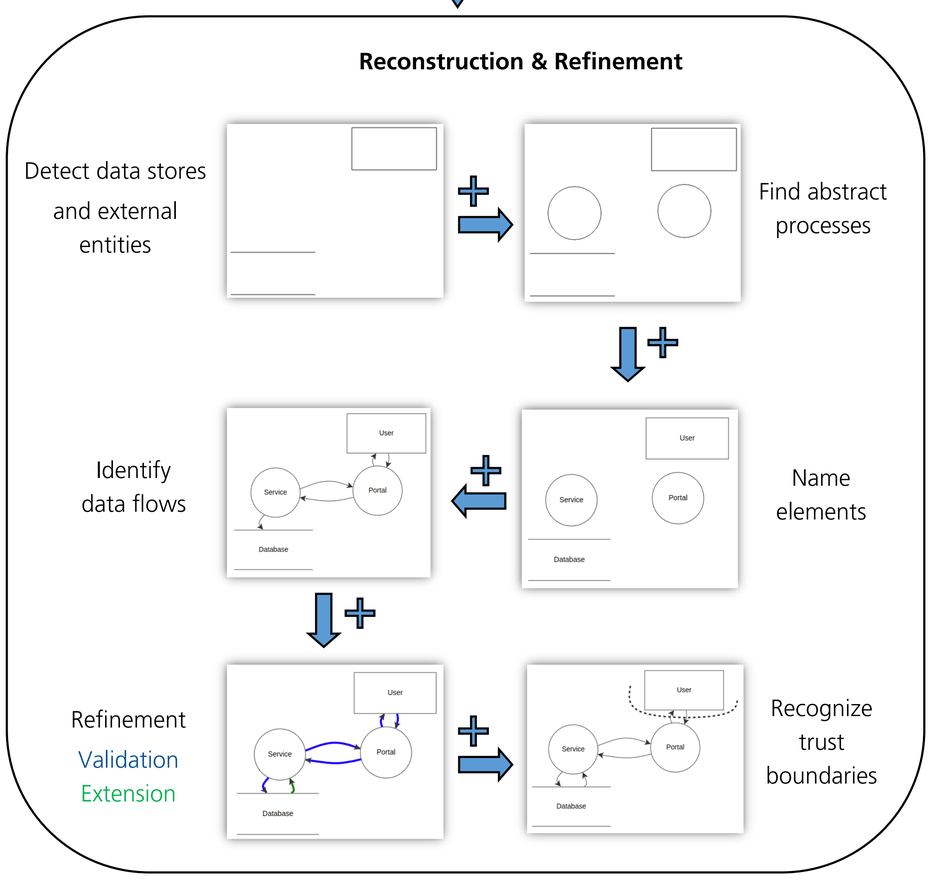


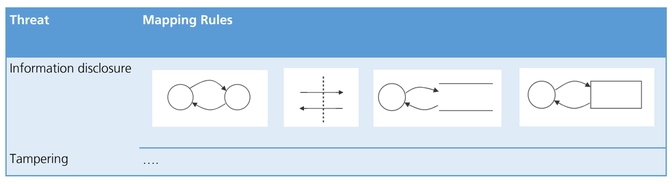
Malicious Information Flow in a Django Web Application

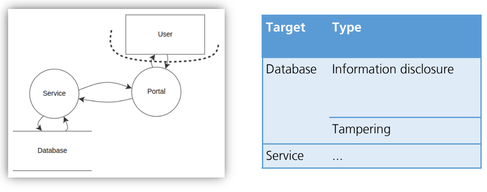


Software Project

Code + Artifacts







Evaluation

I investigate the following **research questions** (RQ):

1. How many and which elements of the information flow graph can be reconstructed and correctly assembled?

2. Can FlowFuzz effectively identify information flows and what is its efficiency?

3. Considering the reconstructed and refined information flow graph, what is the nature and number of threats elicited?

For evaluation, I create a **dataset comprising ten open-source repositories** with information flow graphs and threat models, manually supplementing any missing artifacts.

Information Flow Graph Reconstruction & Threat Elicitation

I propose an approach for **reconstructing and refining information flow graphs** from the source code of a software project to be used as input for threat elicitation.

Reconstructing Information Flow Graphs

I have divided the extensive reconstruction task into subtasks (see left side). Most of the challenges will be addressed through **static analysis**, including detecting external entities, data stores, trust boundaries, and information flows. Additionally, I will incorporate **clustering techniques** to identify abstract processes and **natural language processing-based methods** to name the graph elements.

Refinement with Information Flow Fuzzing

Information flow fuzzing is an approach I introduce to **steer a fuzzer toward identifying information flows** between a source (input) and a sink (output). It is used to validate the statically discovered information flows and to uncover missed ones. My implementation is named **FlowFuzz** [5] and functions with any coverage-guided fuzzer.

Oracle

For each input from the fuzzer, the program is executed in the original state R, and subsequently, the **secret undergoes a controlled and isolated mutation** before a second execution R' of the program (steps 1 & 2). An **alteration in the sink** after the mutation of the secret data signifies the **presence of an information flow** (steps 5 & 6).

Guidance

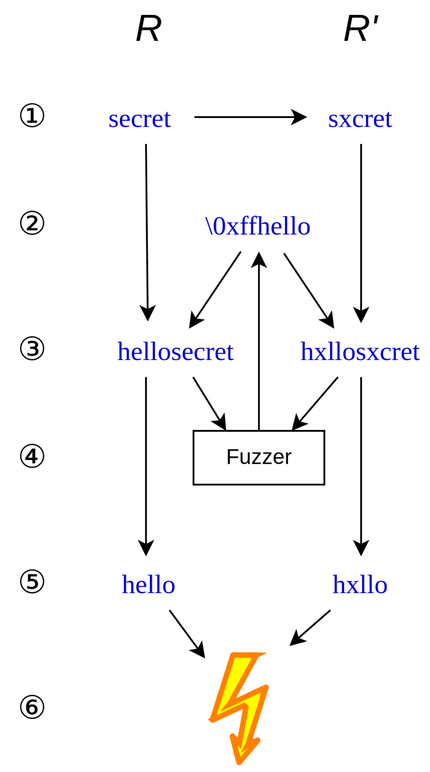
Moreover, I present a guidance strategy to explore information flows more effectively. In this strategy, the **fuzzer** not only strives to maximize coverage but also **focuses on inducing changes in program state** between the two consecutive runs (step 4).

Django Example

FlowFuzz is capable of tracing information flows in the above-provided Django web application example. In this process, the chosen input is the date format, and the configuration file undergoes mutation. If the input matches our secret key in the configuration file the outputs can look as follows:

- R: "002023-05-05T00:00:00Fri, 5 May 2023 00:00:00 +000031"
- R': "00x2023-05-05T00:00:00Fri, 5 May 2023 00:00:00 +000031"

This change in the outputs is reported by FlowFuzz as information flow.



Functioning of FlowFuzz

Mapping Threats to Information Flow Graph Elements

The reconstructed and validated information flow graph will be used to **elicit threats**, for example, insecure information flows or unencrypted data stores. I will develop an **automated, rule-based system** by building upon previous research [2] using threat mapping rules from the threat analysis approaches Linddun [3] and Stride [4].

References

[1] NVD „CVE-2015-8213 Detail" www.nvd.nist.gov/vuln/detail/CVE-2015-8213 Retrieved 2024-04-04.

[2] Tuma, Katja, et al. "Automating the early detection of security design flaws." Proceedings of the 23rd ACM/IEEE International Conference on Model Driven. Engineering Languages and Systems. 2020.

[3] Deng, Mina, et al. "A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements." Requirements Engineering 16.1 (2011).

[4] Shostack, Adam. Threat modeling: Designing for security. John Wiley & Sons, 2014.

[5] Under review as a registered report at ACM-TOSEM.

DLR Institute of Data Science

Funded by the European Union NextGenerationEU

SPONSORED BY THE Federal Ministry of Education and Research

Pre-Print

LinkedIn

Supervisors:
Andreas Zeller (CISPA)
Clemens-Alexander Brust (DLR)

Bernd Gruner
Secure Software Engineering Group
DLR-Institute of Data Science
bernd.gruner@dlr.de