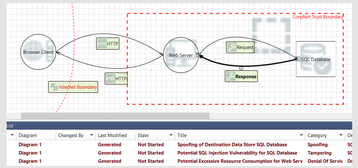
Machine Learning Applications in Secure Software Engineering

B.-E. Bouhlal, B. Gruner, T. Sonnekalb, C.-A. Brust | Secure Software Engineering Group „SST“ | DLR Institute of Data Science

Software Architecture Reconstruction

Many security-related vulnerabilities arise already during the software architecture and design stage, even before the implementation phase begins. Automated **architectural threat modeling** makes it possible to identify, assess, prioritize, and mitigate threats. Due to agile development and changing requirements, the software architecture changes constantly, and requires regular threat model updates. Architectural reconstruction can generate an abstract representation of the source code as input for threat modeling. It is crucial to utilize the source code to assess the actual risks because there are often discrepancies between the actual and planned architectures.



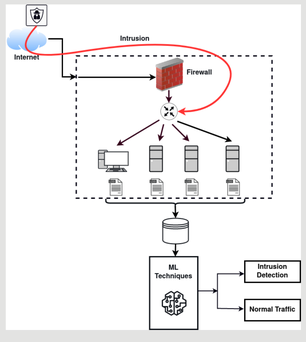
In architecture reconstruction, the **data flow diagram** (DFD) is often laboriously created manually or semi-automatically. Some promising approaches attempt to create the DFD using machine learning. The challenge is that there are no benchmark datasets to train or evaluate the methods. Due to the lack of data, unsupervised clustering methods that group source code elements based on similarities are mainly applied. An open challenge is to transform the resulting output into a complete DFD that can be used as input for threat modeling. This requires mapping the detected clusters to a component type in the DFD (e.g. database) and adding elements, such as trust boundaries.

Infrastructure Health Monitoring

Security incidents are increasing over time, which requires developing innovative protection methods. We are using ML techniques to build **intrusion detection** systems able to identify intrusions (cyberattacks) that a conventional protection system (e.g. firewall) would have difficulties identifying.
The trained model identifies signatures of previously classified malicious activities and also detects **new abnormal traffic** that may be an indication of criminal activities (unknown attack).

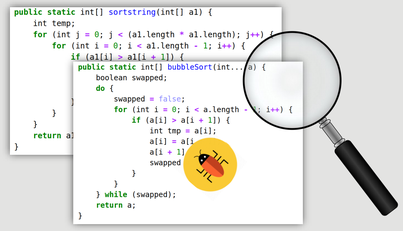
Input data:
- Network-based and host-based log files are gathered and analyzed to build a training dataset.

Challenges:
- Building models with high detection performance and minimal false alarm rates.
- Effective identification of zero-day attacks.



Clone Bug Detection

Imagine copying a piece of code from StackOverflow that initially does its job, but only later turns out to contain an error. In the meantime, however, the code snippet has been used in several places in the project. The task of clone bug detection is to find the same bugs across the entire software project. Often these bugs are **semantic code clones**, i.e. code snippets that have the same function but differ greatly in their notation.
We frame clone bug detection as a binary classification task. For this purpose, we use the Juliet Test Suite dataset, which we have prepared to contain code pairs that are both clones and bugs or vulnerabilities. As a benchmark, we compare ourselves to classical codeclone detection methods as well as vulnerability detection methods.



Analyzed programming languages: C, Java

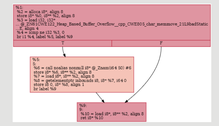
Challenges:
- find a good representation for smenatic clones
- expand the task from classification to information retrieval

Architectures:
- Graph Convolutional Networks
- Transformer Networks
  CodeBERT, GraphCodeBERT, CodeT5, ...

Vulnerability Detection

With Control Flow Graphs + GCN

- A **control flow graph** (CFG) represents all possible execution orders of uninterruptible code blocks.
- Suitable for modeling certain vulnerabilities:
  - Insufficient control flow management,
  - business logic errors, and
  - behavioral problems.



- A **graph convolutional network** (GCN) can process CFGs directly.
- Can distinguish non-vulnerable and three classes of vulnerabilities @87.5% accuracy.

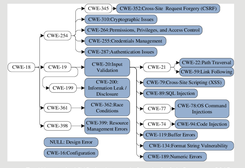
With Assembly Language and Transformers

```
!1c383:
push rbp
mov rbp,rsp
sub rsp,0x10
mov DWORD PTR [rbp-0x4],0x0
mov DWORD PTR [rbp-0x4],0x0
mov eax,DWORD PTR [rbp-0x4]
sub eax,0x1
mov DWORD PTR [rbp-0x8],eax
mov eax,DWORD PTR [rbp-0x8]
mov edi,eax
call 1c1f8
leave
ret
```

- **Augmented assembly language** is a rich text representation of programs.
- We use **large language models** to detect 91 types of vulnerabilities in the Juliet benchmarkt suite.
- CodeBERT+AAL: 96.9% accuracy.

With Semantic Knowledge Integration

- Classification tasks typically consider classes mutually exclusive and don’t consider any relations between them.
- Some classes are more similar than others.
- There is a **hierarchy** of vulnerability types in the common weakness enumeration (CWE).
- **Knowledge integration** of the hierarchy into a model can improve results.
- Model does not have to learn relations from data.
- Heavily relies on correctness.



Tao Wen et al., ASVC: An Automatic Security [...], J. Commun., 2015.

Security is important throughout the development lifecycle – how can ML help?

Design
- Software Architecture Reconstruction

Implementation
- Clone Bug Detection
- Vulnerability Detection
- Software Quality Model

Operation
- Integrated Modular Avionics Monitoring
- Infrastructure Health Monitoring

Integrated Modular Avionics

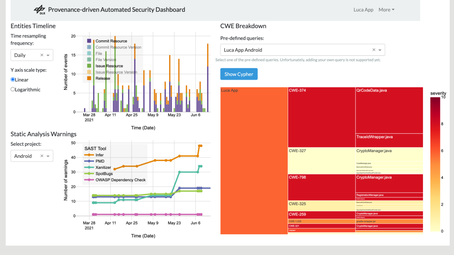
Aerospace application require considerable verification efforts for safety reasons. Ideally, safety-crititcal programs are verified and proven to perform as intended at design time. This is not always possible for sufficiently complex code. Hence, **runtime verification** is an accepted alternative. However, verifying formal specification satisfaction in real-time is resource-intensive and might still not cover rich applications with AI-based decision making or computer vision components.

Because it is not always possible to specific operating conditions formally and in a measurable way, we propose a runtime monitoring approach based on **anomaly detection** and a human-in-the-loop.

We apply this approach in the RESILIENZ project where we help develop an integrated modular avionics demonstrator. Our **health monitor** offers a continuous estimate of the expected performance and quality of software components.

Software Quality Model

Code metrics can be used to give developers feedback on code that has already been written and to point out areas in need of improvement. A combination of code metrics can be summarized as a model to draw conclusions about the security and software quality of a project.
The aim of this project is to find out the aspects that make up software quality and to identify the appropriate metrics to calculate a **general quality score** that can be integrated into our developer dashboard.



Alternatives to and Challenges of ML in SSE

<u>Challenges</u>
- Machine learning tools and models are not constructed with **software engineering data types** in mind:
  - Source code, binaries, process artifacts (issues, commits…)
  - Documentation
- Benchmarks are scarce and are often built in ways that **go against ML best practices**. We frequently observe:
  - Label inaccuracy, data leakage
  - Inappropriate measures, sampling bias
- Security-relevant tools are typically applied in the context of a **risk management** framework. ML-based solutions carry novel hard to calculate risks which reduces acceptance.

<u>Alternatives</u>
- Static and dynamic analysis **security testing** based on rules or „fuzzing“.
- **Security by design**: careful planning and security consideration during design can alleviate the need for later security testing efforts somewhat.

s.DLR.de/sse @DLRdatascience

DLR