[Micro]services: threat, challenge, or opportunity for sound static program analysis?

Pietro Ferrara[1]

[1] Università Ca' Foscari di Venezia, Italy

✉ pietro.ferrara@unive.it

Abstract. Microservices represent a unique opportunity to revamp sound static analysis and its application to industrial software. However, several formal, technical, implementation, and cultural challenges must be addressed. In this talk, we will briefly introduce static program analysis and its application to industrial software during the last few decades, and then we will discuss the threats, challenges, and opportunities of its application to the microservices world.

1 Introduction

Static program analysis (SPA) [8] consists of checking properties on programs without executing them. The approach is dual w.r.t. dynamic program analysis (e.g., testing), where the program is executed and the property is checked on a concrete trace of execution. Over the last decades, several analyzers have been formalized and developed. An appealing property for SPA is to be sound, that is, if the analyzer states that the property is satisfied by the program, then all the possible executions of the programs satisfy that. However, if SPA states that the property is not satisfied, then this might be either a real issue or a so-called false alarm.

To make the reasoning more concrete, consider the following program:

```
float foo(int val) {
  if(val!=0)
    return 1/val;
  else return 0;
}
```

We want to check if this program can raise a division by zero error. If we statically reason on this program in a sound way, we can abstract the value of variable val in many different values: with an interval [a..b] (tracking that val must be greater or equal than a, and less or equal then b), with its sign (e.g., tracking if it is positive, zero, or negative), or with the values it cannot assume (that is, with a set of excluded values). With the first two abstractions, we would get a false alarm: in both cases, the condition `val!=0` does not add any knowledge to the static analysis (since `val` could have any value at the beginning of the method), and therefore both analyses would raise a false alarm. Instead, the third abstraction would track that inside the `then` branch `val` cannot be zero, therefore proving that no execution leads to a division by zero error.

Because of the halting problem, it is not possible to compute a sound and complete approximation for any property on any program. Therefore, abstraction is needed. Abstract interpretation

[2, 3] is a general framework that allows one to formalize and prove the soundness of abstractions. Other approaches, such as model checking and deductive verification via theorem proving, have been widely applied to this field [1]. Developing a sound static analyzer based on these theories requires quite some effort, and in particular to

1. Mathematically formalize the analysis
2. Prove formally the soundness of the analysis
3. Implement the analysis of a full programming language

Because of this effort, the practical and industrial application of these techniques has been mostly focused on specific sectors (e.g., avionics and automotive) where a single bug might have catastrophic results. However, security application vulnerabilities have been rather appealing in this context. Microservices appear to be yet another environment where sound SPA might play a relevant role, given the pervasiveness of their adoption. In this paper, we discuss what threats, challenges, and opportunities microservices arise to sound SPA.

2 Sound SPA in industry

First of all, we start by analyzing what industrial impact sound SPA had in the last few decades.

Since the 90s, several sound analyzers have been applied in the industry. An initial main focus was on safety-critical software, such as automobile or aerospace software. In such a context, a single bug (and in particular, run-time error) might lead to catastrophic effects such as the loss of human lives. Therefore, achieving a run-time error-free software was a main concern, several standards of those fields imposed them, and several sound static analyzers, such as ASTREE [4] and Mathworks Polyspace [9] were commercialized. Several industrial standards and certifications (such as DO-178C for the avionic software certification[7]) impose the adoption of formal methods to prove the absence of bugs.

Unfortunately, sound static program analysis has found little industrial impact on the application, Web, mobile, and cloud software since the 2000s, even if some notable examples exist (such as the Julia static analyzer [10]). In such a context, the presence of a bug is not perceived as such a big deal, since a run-time error would cause at most the crash of a computer system, but it would not kill people or have effects on the physical world. In addition, the variety of programming languages, frameworks, and libraries adopted in this context poses a hard challenge to sound SPA: each programming language requires a new analyzer, the run-time behavior of each framework requires manually specifying and implementing it, and libraries often require to manually synthesize their behavior (otherwise the analysis would easily need to deal with 100KLOCs if not MLOCs, and it is particularly challenging to scale at this level).

However, things changed over time: cybersecurity threats and the identification of software vulnerabilities (such as various forms of injections and cross-site scripting) combined with a wider and wider application of those programs (that moved from a purely desktop-based world to a Web/cloud world where more and more critical services, such as e-banking, are provided) pushed again the need of proving the absence of bugs (or software vulnerabilities).

[1] If on the one hand, the purpose of this talk is not to give an overview of SPA techniques, on the other hand, we will be available to discuss in details those techniques if it will be of interest for the audience

For instance, among the thousands of bugs and vulnerabilities classified by CVE every year[2], the Equifax data breach[3] in 2017 is one of the most famous. This breach, caused by a vulnerability in Apache Struts, caused the leak of sensitive information of about 150 million US citizens, and losses in the order of hundreds of millions of dollars to the enterprises. Over the years, many other vulnerabilities caused high economic damage to enterprises once exploited by hackers.

Because of this situation, nowadays more and more tools based on formal methods (for instance, the aforementioned Julia[10], Zoncolan and Infer at Meta[6]) are appearing on the market and they are adopted by industries.

3 Microservices and sound SPA

The advent of first service-based applications, and then microservices and serverless architectures, changed the scenario again. From a static analysis standpoint, we oversimplify all those architectures as software composed of many independent units (where each unit can be deployed separately) that communicate (synchronously or asynchronously) exchanging objects. In addition, those units are micro (aka, fine-grained or minimal) in the sense that they deal with only one specific aspect (aka, domain) of the overall application.

While such a definition is rather naive (they might communicate through REST APIs or message brokers, they might exchange information through JSON, XML, or binary format, etc...), it already comprises all the things that matter from a static analysis standpoint. From now on we will improperly refer to these architectures as microservices, but we want to underline that such a term encompasses a larger software base for our purposes.

3.1 The threats

Each microservice implements a small part of the system, and how different microservices communicate is hard to infer statically since it might depend on external configuration files, dynamic computation of string values (URLs or names of topics), etc. In addition, each microservice could be implemented in any available technological stack, and the same system might encompass different technological stacks. The most popular technological stacks rely on JavaScript (such as MEAN[4] and MERN[5]), Python (such as FastAPI[6] and Flask[7]), or Java (such as Spring[8]). Other dozens of less popular technological stacks are nowadays applied at an industrial scale.

3.2 The challenges

Each threat is also a challenge. Abstract interpretation is believed to be a theory so generic that it can be applied to any form of abstraction[5], and therefore to any programming language or technological stack. We are ready to defend this idea with very strong arguments against any possible objections. However, generality comes to a price, as we underline at the end of Section 1. The main challenge will

[2] https://cve.mitre.org/
[3] https://en.wikipedia.org/wiki/2017_Equifax_data_breach
[4] https://meanjs.org/
[5] https://mern.js.org/
[6] https://fastapi.tiangolo.com/
[7] https://flask.palletsprojects.com/
[8] https://spring.io/

be to deal with all those threats on the existing variety of technological stacks. On the other hand, the minimal size of microservices (in terms of implemented functionalities, and therefore lines of code) opens the door to the application of extremely precise sound SPA. Precise and sound SPAs usually do not scale up to software above 10KLOCs[9].

In addition, we believe that the current dichotomy between the research field of static analysis and the one of (empirical) software engineering/architecture poses a further challenge. It seems that over the last 20 years, the first community mostly focused on theoretical studies with prototype implementations, while the second one mostly focused on strong and (almost) industrial implementations with very little (if any) formalization. This led to a scientific environment where the researchers in static analysis tags as not scientific the user study with few dozens of participants published by software engineering researchers, and the latter researchers that tag as Greek feta[10] the formalization of the first community, stating that they are useless and not understandable. Making these two worlds communicate again is quite challenging, and it is the main reason for this talk.

3.3 The opportunities

Microservices are nowadays pervasive. With the IoT revolution, the dichotomy between safety critical systems and cloud software went away, since nowadays a lot of those systems (such as cars, airplanes, and industrial plants) are connected to the Internet and communicate with microservices[1]. We believe this is a huge opportunity for sound static program analysis to target: it is nowadays pretty clear that such an environment will need deep debugging methods, and only sound SPA can prove the absence of bugs. However, the challenges to solve are huge, and there will be a lot of effort (in terms of people involved in projects, funding, and conceptual work) to fill the gap.

4 Conclusion

In this paper, we briefly introduced sound static program analysis (SPA), illustrating its main industrial applications over the last decades, and then we illustrated what threats, challenges, and opportunities microservices arise for sound SPA. We believe that the state-of-the-art of the different technologies and scientific communities suggests the following takeout messages:

1. Sound static program analysis has been experiencing little industrial interest during the last 2 decades
2. Microservices will probably be an extremely appealing environment to flourish again sound SPA, but the challenges to solve are huge!
3. These challenges need the collaboration of the scientific communities in sound SPA and software engineering

As already underlined in the introduction, the main advantage of sound SPA w.r.t. other techniques such as testing and syntactic SPA is that it can approximate all possible executions. This is

[9] Notable exceptions such as ASTREE[4] exist. However, they often target a specific codebase and they have been optimized to scale up on such software

[10] We take this term by a colleague that recently gave us this illumination, but he/she also said that this was not his invention and the copyright it belongs to somebody else. Therefore, we prefer to keep anonymous. If anybody wants to publicly state that this was his invention, we would be more than happy to cite him. Otherwise, we can further push this idea, since we believe it perfectly summarizes the current status of our research fields

fundamental to proving the absence of bugs or software vulnerabilities. We believe that this might be the key feature for the potentially wide adoption of sound SPA to microservices for two main reasons:

- microservices offer more and more critical services that must comply with high privacy and security standards, and
- software vulnerabilities are usually exposed only by specific input and configurations.

Therefore, tools proving the absence of bugs might be particularly appealing even if they raise false alarms that need to be manually investigated and discarded.

Acknowledgments: This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU, project iNEST-Interconnected NordEst Innovation Ecosystem funded by PNRR (Mission 4.2, Investment 1.5) NextGeneration EU – Project ID: ECS 00000043, and project SPIN-2021 "Static Analysis for Data Scientists" funded by Ca' Foscari University.

References

[1] Björn Butzin, Frank Golatowski, and Dirk Timmermann. Microservices approach for the internet of things. In *Proceedings of ETFA '16'*. IEEE, 2016.

[2] P. Cousot and R. Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proceedings of POPL '77*. ACM, 1977.

[3] P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Proceedings of the 6th Symposium on Principles of Programming Languages (POPL)*, pages 269–282. ACM, 1979.

[4] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The ASTRÉE analyzer. In *Proceedings of ESOP '05*. Springer-Verlag, 2005.

[5] Patrick Cousot and Radhia Cousot. Abstract interpretation: past, present and future. In Thomas A. Henzinger and Dale Miller, editors, *Proceedings of LICS '14*. ACM, 2014.

[6] Dino Distefano, Manuel Fähndrich, Francesco Logozzo, and Peter W. O'Hearn. Scaling static analyses at facebook. *Commun. ACM*, 62(8):62–70, 2019.

[7] Gabriella Gigante and Domenico Pascarella. Formal methods in avionic software certification: the do-178c perspective. In *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. Springer, 2012.

[8] Anders Møller and Michael I Schwartzbach. Static program analysis. *https://users-cs.au.dk/amoeller/spa/spa.pdf*, 2023.

[9] Mathworks Polyspace. https://www.mathworks.com/products/polyspace.html.

[10] Fausto Spoto. The julia static analyzer for java. In *Proceedings of SAS '16*. Springer, 2016.