SCIENCES & BIOENGINEERING SCIENCES

The Research Group
**Software Languages Lab**

has the honor to invite you to the public defence of the PhD thesis of

# Maarten Vandercammen

to obtain the degree of Doctor of Sciences

Title of the PhD thesis:

**Inter-Process Concolic Testing of Full-stack JavaScript Web Applications**

Promotor: **Prof. dr. Coen De Roover**

The defence will take place on

**Friday, October 13, 2023 at 16h in auditorium D.0.07**

The defence can be followed via a livestream. Please send an email to Maarten.Vandercammen@vub.be for more information.

**Members of the jury**

Prof. dr. Viviane Jonckers (VUB, chair)
Prof. dr. Joeri De Koster (VUB, secretary)
Prof. dr. Beat Signer (VUB)
Prof. dr. Kris Steenhaut (VUB)
Prof. dr. Cristian Cadar (Imperial College London, UK)
Prof. dr. Xavier Devroey (Université de Namur)

## Curriculum vitae

Maarten Vandercammen obtained his Bachelor degree in Computer Science in 2013 and his Master degree in Computer Science in 2015 at the Vrije Universiteit Brussel, both summa cum laude. He started his PhD at the Software Languages Lab (SOFT) under the supervision of Prof. Dr. Coen De Roover. He currently contributes to the CyberSecurity ICON project "APAX". His research interests lie in automated testing of web applications.

Maarten coauthored two international journal papers (both as first author), one international conference paper (as second author), and six international workshop papers (of which five as first author). His work was presented at international conferences and workshops. He supervised eight Bachelor theses and four Master theses.

## Abstract of the PhD research


Web applications are becoming increasingly prevalent. Example applications include collaborative text editors and drawing applications. We define *full-stack JavaScript web applications* as web applications of which both the client and the server have been implemented in JavaScript.

Several automated testing approaches have been proposed to verify the correctness of sequential, non-distributed applications. Prominent among these is *concolic testing*, which systematically explores all execution paths through the program by collecting symbolic constraints over the program inputs.

We transpose concolic testing to the domain of full-stack JavaScript web applications, which gives rise to several challenges unique to these systems. For example, both client and server processes are event-driven, so concolic testers for these applications must craft elaborate event sequences to explore some parts of the program. Furthermore, because of the interconnected nature of these processes, the execution of one process may affect that of another in unexpected ways.

We propose an approach to concolic testing for these types of applications that addresses these challenges. This approach relies on performing *inter-process* testing of the application, which tests all instances of the client and server processes while observing their communication. Inter-process testing hence preserves information flow between processes, thereby increasing precision and preventing false positive errors. Inter-process testers stand in contrast to *intra-process* testers, of which the execution paths do not cross the boundary of a process. We implement inter-process testing in a novel concolic tester called *StackFul*.

StackFul also considers the event-driven nature of full-stack JavaScript web applications. Event-driven code gives rise to the problem of *state explosion*, where an exponential number of states are created while testing the application. To solve this problem, we introduce a novel form of *state merging* for concolic testing of event-driven applications that reduces the number of states by merging together similar states.

We evaluate StackFul on eight real-world applications. We measure to what extent StackFul is capable of i) covering execution paths, ii) finding errors on the server of these applications, and iii) discerning high-priority from low-priority server errors. Furthermore, we evaluate the impact of incorporating state merging. We show that state merging almost always requires fewer test runs to achieve higher code coverage.