RH RQ

Bringing great research ideas into open source communities

Tomáš Černý

The Baylor University professor talks cross-cultural exchange, microservice evolution, and quality assurance

+

Can streaming data and machine learning build better communities?

Lessons from an upstream hypervisor fuzzer

Verification of a Linux distribution

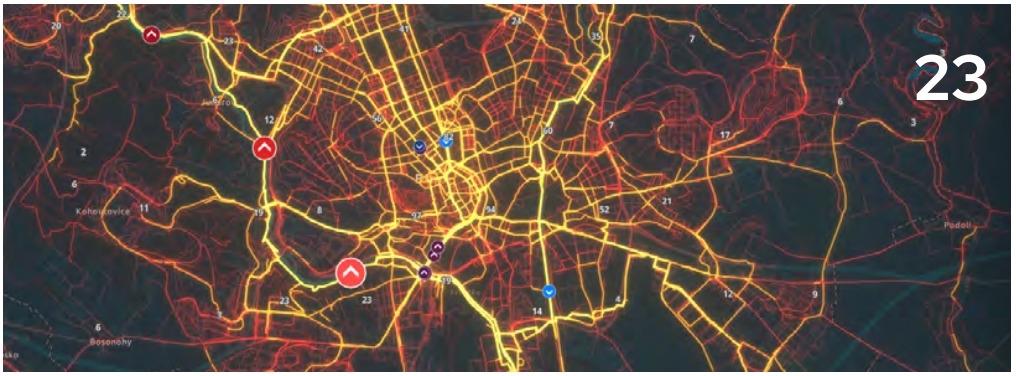
Red Hat Research Quarterly
Volume 4:2 | August 2022 | ISSN 2691-5251

RESEARCH QUARTERLY

VOLUME 4:2

Red Hat

Table of Contents



Departments

Features

RESEARCH QUARTERLY

VOLUME 4:2

Red Hat

From the director

About the Author
Hugh Brock is the Research Director for Red Hat, coordinating Red Hat research and collaboration with universities, governments, and industry worldwide. A Red Hatter since 2002, Hugh brings intimate knowledge of the complex relationship between upstream projects and shippable products to the task of finding research to bring into the open source world.

Working fuzzier, not harder

by Hugh Brock

I first met Boston University PhD student Alex Bulekov at Red Hat's Boston office in the fall of 2018. At the time, I had very little idea of what a "fuzzer" was, let alone why building a better one would be a useful and noteworthy thing. (In case you, too, are ignorant on this topic, I recommend you read Alex's article, "Applying lessons from our upstream hypervisor fuzzer to improve kernel fuzzing," to find out more.) Alex, along with his Red Hat mentor Bandan Das, gave me a complete and patient education on the topic of fuzzers and their benefits. His project then was to overcome the difficulties of making a fuzzer that could successfully fuzz DMA inputs to a process—in this case, the QEMU virtual machine management tool. Today, he has not only succeeded with his initial goal, he has moved on to building a generic fuzzing framework that may relieve the need to write complex and precise descriptions of exactly how to fuzz specific inputs to the Linux kernel. If his project succeeds, it will be a significant contribution to the security and testability of the kernel, and I would count it a major achievement both for Alex and for Red Hat Research.

Our focus on testing continues with our interview with long-time Red Hat Research partner Tomáš Černý. Currently a professor at Baylor University, Tomáš began working with Red Hat while he was still at ČVUT in Prague, in the area of automated test development done by probing the interfaces presented by microservices. As with Bulekov's fuzzing framework, Tomáš's interest is in relieving the burden of manual test authoring from the developer while ensuring greater test coverage and hence higher quality. Tomáš is also our foremost cultural exchange ambassador at Red Hat Research: every year, he brings ten or so Baylor undergraduates to Prague and Brno to work with Red Hat engineers and researchers at Czech universities. The students rave about the experience, and no wonder: they get to meet great scientists and engineers from a unique culture while sampling what is, in my opinion, the best beer in the world.

Frequent readers will be aware that 2022 saw the beginning of Red Hat Research's expanded $4 million annual research agreement with Boston University. Last issue, we featured the largest single grant awarded from our 2021 RFP, the AI for Cloud Ops project. In this issue, we feature a similarly ambitious effort that also touches AI but is otherwise completely different: the Smarta Byar ("Smart Villages," in English) project to build a complete digital twin of a village in southern Sweden called Veberöd. The project aims to collect data on everything from bus movements to barbeque smoke to see what kind of applications can be built with such a large, continuous flow of information. One of the first challenges, of course, will be familiar to every data scientist: how do you improve the quality of the incoming data while reducing the quantity to a manageable level? Event-driven algorithms are one possibility, but there are others; have a look at Red Hatter Jim Craig's overview of the project to learn more.

As I write this, the whole Red Hat Research team is frantically at work preparing for our first real in-person event since January of 2020: Red Hat Research Day is coming to Brno, Czech Republic, on September 15 of this year! We are more than a little excited about the ability to again meet with our university and industry partners in person and learn firsthand about where research with open source impact will go next. We hope you will be able to join us there— keep an eye on research.redhat.com for details.

RESEARCH QUARTERLY

VOLUME 4:2

Red Hat

News

Linux now includes a real-time analysis toolset

Daniel Bristot de Oliveira's research in real-time systems led to the inclusion of the RTLA in Linux 5.17.

by Shaun Strohmer

Red Hat Research's Dr. Daniel Bristot de Oliveira presented the Real-Time Linux Analysis Toolset (RTLA) at the Red Hat Open Source Summit held June 21-24, 2022, virtually and in Austin, TX, USA.

The result of several years of research, the RTLA was one of the most significant changes in the release of Linux 5.17. Over the last decade, Daniel has been exploring methods to improve the analysis of the real-time properties of Linux. His research explores the tracing features of Linux to derive fine-grained properties of the kernel, overcoming a known limitation of the usage of black-box testing by developers while increasing confidence in Linux usage on safety-critical real-time systems.

While the black-box method provides an overview of the system, it fails to provide a root-cause analysis for unexpected values. Developers must use kernel trace features to debug these cases, requiring extensive knowledge about the system and fastidious tracing setup and breakdown. The RTLA is a user-space tool that addresses these challenges. Part of the Linux kernel toolset, RTLA provides a benchmark-like interface for in-kernel tracers that extract meaningful information about the timing capabilities of the system by simulating typical real-time workloads. RTLA also facilitates root-cause analysis of timing failures via tracing, automating the most common setup while enabling advanced tracing features in a single tool.

Such analysis tools are becoming essential to the development of Linux as a whole as PREEMPT_RT becomes an integral part of the operating system, meaning that more new users—or busy developers—will have to dedicate time to the analysis of Linux as a real-time operating system (RTOS). RTLA gives them an easy-to-use solution that doesn't require detailed knowledge of the system.

The initial implementation was substantial enough to place Daniel among the top 20 contributors of the 5.17 kernel release, and further developments are expected. RTLA was designed with extensibility in mind. It is expected to be extended with other theoretical analysis tools, serving as the starting point for researchers and practitioners to develop tracing-based analysis tools.

Daniel was also a keynote speaker at the 17th Workshop on Virtualization in High-Performance Cloud Computing, held in

About the Author
Shaun Strohmer is the editor of *Red Hat Research Quarterly*. She has worked as a writer and editor in academic publishing for over twenty years, and since 2014 she has focused on software development, cybersecurity, and computer science.

Daniel's research was featured in a series of articles for RHRQ: "A threat model for the real-time Linux kernel" (2:3), "Efficient runtime verification for the Linux kernel" (2:4), "Demystifying real-time Linux scheduling latency" (3:1).

Hamburg, Germany, from May 29-June 2, 2022. His paper "Operating system noise in the Linux kernel," co-authored with Daniel Casini and Tommaso Cucinotta, was recently accepted for publication by *IEEE Transactions on Computers* for a special issue on real-time systems.

About Daniel

Daniel Bristot de Oliveira has a joint PhD in Automation Engineering from Universidade Federal de Santa Catarina (Brazil) and Embedded Real-Time systems from Scuola Superiore Sant'Anna (Italy). Currently, he is a Senior Principal Software Engineer at Red Hat, working on developing the real-time features of the Linux kernel. Daniel helps in the maintenance of real-time related tracers and toolings for the Linux kernel and the SCHED_DEADLINE. He is an affiliate researcher at the Retis Lab and researches real-time and formal methods. He is an active member of the real-time academic community, participating in the technical program committee of academic conferences, such as the Real-Time Systems Symposium, the Real-Time Technology and Applications Symposium, and the Euromicro Conference on Real-Time Systems.

RESEARCH QUARTERLY

VOLUME 4:2

Red Hat

News

Undergraduate research projects advance the Red Hat Collaboratory's educational mission

by Shaun Strohmer

The Red Hat Collaboratory at Boston University is supporting select undergraduate student research projects during Summer 2022, in keeping with its mission of advancing education in open source technologies. So far, six projects have been chosen to receive funding and supervision from BU computer engineering professors active in their own Collaboratory projects, with more expected.

The award recipients were selected from a competitive pool of applications in areas including cloud computing, systems engineering infrastructure, and security:

- Nengneng Yu, "**Fine-grained, automated security detection via semi-supervised learning,**" in collaboration with PhD student Yajie Zhou, supervised by Professor Alan Liu. The project aims to detect and recover attack stories behind security incidents using natural language processing techniques.
- Julia Hua, "**Network-accelerated in-memory key-value store live migration,**" in collaboration with PhD student Zeying Zhu, supervised by Professor Alan Liu. The project aims to develop an efficient open source software migration system to meet demands for high throughput and low latency in the cloud.
- Quan Pham, "**Real-time quality assurance,**" supervised by Professor Gianluca Stringhini. The project aims to develop plugins for the Jupyter Notebook environment that will analyze code and provide real-time feedback on software in development.
- Ethan Klein, "**Red Hat unikernel Secrecy project,**" supervised by Professor Orran Krieger. The project aims to develop a unikernel implementation of Secrecy, a platform enabling parties to perform shared computations on private data without sharing their actual data.
- Xiteng Yao, "**Practical programming of FPGAs with open source tools: test-code generation and guided search through supervised learning,**" supervised by Professor Martin Herbordt. This project aims to improve an open source programming tool flow known as LLVM using reinforcement learning.
- Shun Zhang, "**D-COLLECTIVE: democratized data collection and collaborative training for extreme-scale autonomous systems,**" supervised by Professor Eshed Ohn-Bar. This project aims to develop technologies to allow everyday people to participate in data collection and model training for autonomous systems, specifically self-driving cars.

Student award recipient Xiteng Yao heard about the Red Hat Collaboratory from Professor Herbordt, who has worked on multiple projects as part of the Collaboratory. "I was excited to see so many interesting projects in the Red Hat Collaboratory," Yao said. "It's interesting to work

in a team with academic and industry members; it is an excellent opportunity to learn from different types of people." Open source research was another appealing aspect of working with the Collaboratory: "What's exciting about open source research is that your work can be reused by numerous developers around the world. I'd like to develop a robust solution to a real-world problem that could help others."

BU professor Ari Trachtenberg was instrumental in developing the program and says that undergraduate researchers are essential to the long-term viability of the Collaboratory's mission. "Not only can these students make valuable contributions in their own right," Trachtenberg said, "but they will also become the graduate students, engineers, and professors that will drive the computer systems community going forward. It's important that these students experience practical research mentorship to preserve the cultural aspects of modern systems research, such as properly establishing, controlling, and analyzing large-scale tests; determining convincing evidence of improvement; and understanding the subtleties involved with reading or writing a technical systems paper."

Systems research typically requires significant manual and infrastructural investments from a wide variety of groups over a sustained period, making it out of reach for most undergraduate researchers. The Red Hat Collaboratory and its various partners, such as the MOC Alliance, give researchers a ready-made experimental environment and a collection of interested colleagues. This setting provides the students and their mentors the opportunity and the means to try out new systems ideas without the heavy lift of a national, multi-partner grant. "I think the most important feature of this program is that it has enabled computer systems research that may otherwise not have occurred," Trachtenberg said.

Undergraduate research projects are generally projects that can be completed within a shorter time frame, such as a four-month summer session, by a junior researcher with limited experience. Some of the projects are also extensions of projects that the Collaboratory is already funding, such as projects relating to unikernel Linux, practical programming of FPGAs, and security detection.

"I think the most important feature of this program is that it has enabled computer systems research that may otherwise not have occurred."

—Prof. Ari Trachtenberg

Publication highlights

August 2022

Red Hat Research collaborates with universities and government agencies to produce papers that bring open source contributions along with them. This is a sampling of recent publications and conference presentations; to see more visit the publications page on the Red Hat Research website.

"**Beating the I/O bottleneck: a case for log-structured virtual disks,**" Mohammad Hossein Hajkazemi (Northeastern University), Vojtech Aschenbrenner (EPFL, Switzerland), Mania Abdi (Northeastern University), Emine Ugur Kaynar (Boston University), Amini Mossayebzadeh (Boston University), Orran Kreiger (Boston University), Peter Desnoyers (Northeastern University). The paper was published in *EuroSys 2022: Proceedings of the 17th European conference on computer systems*.

"**A Closer look at Intel Resource Director Technology (RDT),**" Parul Sohal (Boston University), Michael Bechtel (University of Kansas), Renato Mancuso (Boston University), Heechul Yun (University of Kansas), Orran Krieger (Boston University). The paper was published in *RTNS 2022: Proceedings of the 30th international conferences on real-time networks and systems*.

"**Design and analysis of microworlds and puzzles for block-based programming,**" Radeck Pelánek (Masaryk University, Czech Republic) and Prof. Tomáš Effenberger (Masaryk). The paper was published in *Computer Science Education* 32:1 (2022).

"**DLACEP: a deep-learning-based framework for approximate complex event processing,**" Adar Amir (Technion—Israel Institute of Technology), Ilya Kolchinsky (Research Supervisor, Red Hat), Assaf Schuster (Technion). The paper was selected for presentation at the 2022 SIGMOD conference, held June 12-17 in Philadelphia, PA, USA.

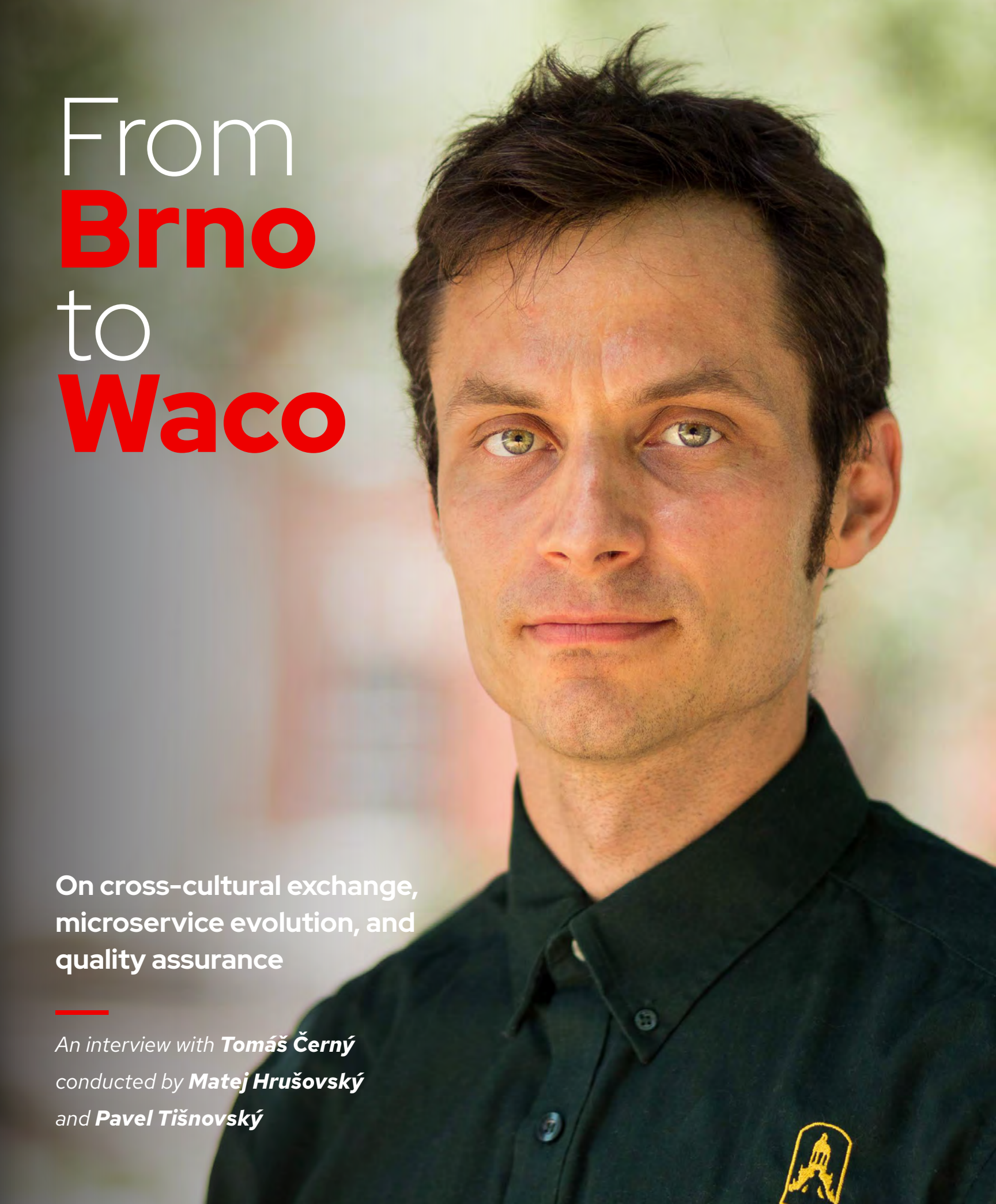
"**HYPERSONIC: a hybrid parallelization approach for scalable complex event processing,**" Maor Yankovitch (Technion), Ilya Kolchinsky (Research Supervisor, Red Hat), Assaf Schuster (Technion). The paper was selected for presentation at the 2022 SIGMOD conference, held June 12-17 in Philadelphia, PA, USA.

"**Operating system noise in the Linux kernel,**" Daniel Bristot de Oliveira (Senior Principal Software Engineer, Red Hat), Daniel Casini (Sant'Anna School of Advanced Studies, Italy), Tommaso Cuinotta (Sant'Anna School of Advanced Studies). The paper is forthcoming in *IEEE Transactions on Computers*.

"**Profile-driven memory bandwidth management for accelerators and CPUs in QoS-enabled platforms,**" Parul Sohal (Boston University), Rohan Tabish (University of Illinois at Urbana-Champaign), Ulrich Drepper (Distinguished Engineer, Red Hat), Renato Mancuso (Boston University). The paper was published in *Real-time systems* (2022).

"**RTLA: finding the source of OS noise on Linux,**" Daniel Bristot de Oliveira (Senior Principal Software Engineer, Red Hat). Daniel's keynote speech was delivered at the 2022 Workshop on Virtualization in High-Performance Cloud Computing (VHPC), held June 2 in Hamburg, Germany.

From **Brno** to **Waco**



On cross-cultural exchange, microservice evolution, and quality assurance

An interview with ***Tomáš Černý*** *conducted by* ***Matej Hrušovský*** *and* ***Pavel Tišnovský***

Red Hat

Interview

RHRQ asked Brno research manager Matej Hrušovský and Red Hat engineer Pavel Tišnovský to talk with long-time collaborator Tomáš Černý, a native of the Czech Republic now teaching at Baylor University in Waco, Texas. Prof. Černý was in Brno recently as part of his highly successful student research initiative, which brings Baylor students to the Czech Republic to work with university researchers and Red Hat engineers, a project funded in part by the National Science Foundation. They discuss this project and his research on developing static analysis tools for microservices.

Matej Hrušovský: Let's begin by introducing you. Tell us about your position at Baylor University.

Tomáš Černý: This is my sixth year at Baylor. It's the hardest, because I have my tenure review. I am also teaching software engineering, and I'm active in software engineering research and the software engineering doctoral program. Currently, we are looking into code analysis of microservices and cloud-native systems.

Baylor is one of the best schools in Texas. It is an interesting school of about 20,000 students that recently joined the R1 tier (*a US classification meaning "very high research activity"–Ed.*), so we have a lot of funds coming into research. We're a small computer science department; we have less than 20 faculty members. Our graduate program has grown significantly in the last few years, and we now offer an online master's program. The research spans from software engineering, of course, to machine learning, data science, and cybersecurity, so there are significant investments in those areas.

Matej Hrušovský: Before Baylor, you were at the Czech Technical University (Prague) in the faculty of electrical engineering. Can you tell us a bit more about your history, like how we met and how your first cooperation with Red Hat started?

Tomáš Černý: Some of my students at CTU reached out to me and said, "They teach such cool classes in Brno with Red Hat. We would really like to have those classes as well." So it was a student initiative. I reached out to Jiří Pechanec (*an engineer in Brno-Ed.*), who was teaching that course, and asked, "Would it be possible to do something similar in Prague?" Then the wheels started to spin, we initiated multiple classes, and it was amazing. The students really loved it. Then we started doing other things: the grant we got with colleagues in TAČR (Technology Agency of the Czech Republic) and the Red Hat Lab in Prague. But about that time, there was a new position open at Baylor in software engineering.

And since I got my master's from Baylor University, I reached out, and they were as excited as I was. So I ended up here, but the collaboration with Red Hat continued, and I'm extremely happy about that. At this point, we have supported nearly 30 students who were able to do research with Red Hat over the past four years, and they've had a life-changing experience. Anytime I reach out to one of those students, they are responsive, they are excited, and they don't regret doing research with us.

Matej Hrušovský: I remember when you moved to Baylor, you immediately reached out and wanted to collaborate with Red Hat. At the same time, I'm sure Red Hat was not your only opportunity for industrial cooperation. Is there a reason you chose Red Hat?

Tomáš Černý: I think your culture is unique—I don't think any other company has that. Presenting open source software to funding agencies and universities, versus proprietary closed software, makes a significant difference. Open source

research creates a direct path for the student to make a contribution. If they're working on research and their contribution is to a company product, no one can ever see that. Especially for students who are maturing and learning how the industry works, this is a fantastic experience no other company can offer.

Matej Hrušovský: Eventually, you were even able to get a National Science Foundation Grant in cooperation with Red Hat as an industrial partner. Can you tell us how that began?

Tomáš Černý: I still have warm feelings for the Czech Republic and for Red Hat as well. So I thought, let's take the students from Texas to the Czech Republic to do research. We reached out to Red Hat, and now we are in the fourth year of this project. We have 40 publications out of this collaboration, which is a giant number, and I have broadened my current research around all that. I'm really amazed by what has happened.

Matej Hrušovský: What was the first experience like? I know you spent one month in Prague at CTU and the second month at Red Hat in Brno.

Tomáš Černý: It was great. When we were in Prague, one student said, "Every building here looks important." You can imagine how the buildings differ in Texas. You rarely have a building more than 100 years old. The students were very impressed. Then, of course, they learned to do research, and their research was very successful. Very quickly, we got papers published based on prototyping and addressing practical problems.

Baylor students socialize in Brno. From left: Stephanie Alvord, Schaeffer Duncan, Karel Frajtak (CTU), Denton Wood, Tomáš Černý , Egle Uljas, Boba Mannova (CTU), Hugh Brock (Red Hat), Michael Coffey, Asher Snavely, Andrew Walker, Jan Svacina, Jonathan Simmons

Of course, there were some challenges in getting used to research, and we had to learn so much the first year. Now when students come with zero experience, we know exactly where to start. The work must be precise; we have to do a sufficient survey of the literature and the prototypes and directions for a given research problem. We also need to describe the problem very well.

Now, after COVID, we want to collaborate with Masaryk University (MU) as well. There are many universities in the Czech Republic, but MU is a great partner of Red Hat. We could offer a double-degree program with the university where students would stay two years in the Czech Republic and two years in the United States or vice versa. They would have two mentors, and they could research practical topics that Red Hat could transfer to practice. Especially being a software engineer, where research is not the core of our work, that's exactly where Red Hat fits in very well.

Matej Hrušovský: What has it been like to combine the academic side at CTU with the applied side at Red Hat?

Tomáš Černý: Having the perspective of academia plus an industrial partner is very important, especially for inexperienced students. They might question whether these things are purely academic or also practical. Would the students do the research without this program? The difficulty is that they don't know how some simple things work, and we can teach them that. They also don't have the motivation without the industry perspective, so they might be thinking, "Is this even necessary? Is this even

The campus of Baylor University in Waco, TX

useful?" When they see someone from an industry who is actually interested in the topic, they don't question anymore.

Matej Hrušovský: Do you have any particular projects already planned out for the students? Are there any technologies in mind? And I would very much love it if Pavel, our engineer, chimed in with his more technical questions at this point.

Tomáš Černý: This year, the topics are related to architectural visualization and description languages. One very exciting thing is architectural degradation, or architectural decay. We build systems that work very well, but over time they suddenly stop working as anticipated. How do we know what went wrong? This is a very challenging question. Maybe you've heard about technical debt. We make small contributions to a project by offering new features, but we are building a debt by implementing it quickly. Eventually, we have to pay off this debt with interest when something slows down, breaks down, or is inefficient concerning maintenance.

The question we want to answer this year is not necessarily how we can solve this, but how we can visualize the changes in the system. If we can see the system from both the dynamic and the static perspective as it is now, is it very different from the previous version? Can we highlight the changes between the versions?

There are three aspects to this problem. There's the architectural description language that maintains what should be preserved within the system when it evolves. There's the visualization of the dynamic and static perspectives of a system. I know that Pavel's interest is in seeing how the data flows through a log pipeline. Finally, my interest is how we can visualize that the system has changed and observe the impact of the change. If something changed recently, can we observe the impact on other dependent systems? We also want to trace the consequent changes, just like we do with a Git tree.

Pavel Tišnovský: So we as DevOps engineers need to comprehend what's going on because the world of architectures based on microservices is very, very large. From my experience, almost nobody from the team understands all the consequences when something gets changed. It's vital to us to say, "When we change this configuration, this path, or how the data flows, what will be affected by it?"

Tomáš Černý: Yes, and this is typically done by dynamic analysis and by tracing. Dynamic analysis is certainly a great direction that everyone has taken so far, but when you apply dynamic analysis, you apply it when the system is in production. Isn't it too late if your customer is testing something you should be aware of?

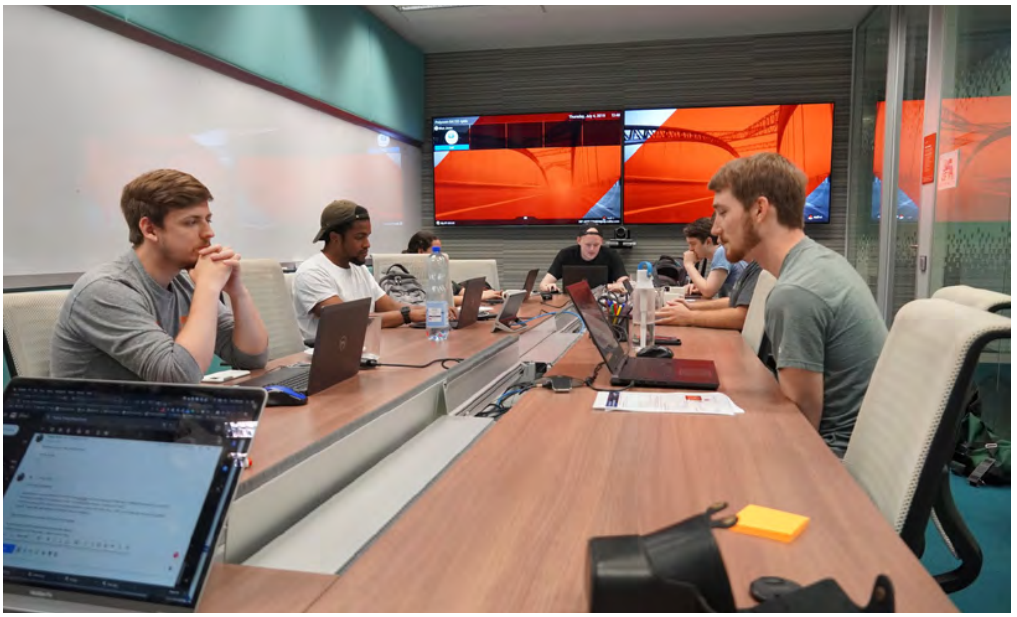
So, in our long-term project goals, we are also looking into static analysis. This is challenging because you have systems coming from Python, Go, Java, and C++, and you have to combine the knowledge within them. We address this by using intermediate representation. There is one great approach that does a similar thing, LLVM, but the problem is that it's a compiled-level intermediate representation. We want to recognize at a high level the components being used in the system. We cannot compile the code because we would lose this information. Still, we would like to have this intermediate representation of the component level from different

languages. If you look at microservices, they do recognize components. Who would build microservices without components, and why?

If a component is an element we recognize, and the intermediate representation model of the system is based on components, then we don't care about the languages. We can start to reason about those systems, whether it's architectural reasoning, security reasoning, privacy reasoning, and so on. Yes, we face many challenges, but there are also many things already answered from that perspective.

Pavel Tišnovský: Right now, it seems the world is moving towards microservices. My feeling is that this trend arose from the pains of monolithic architectures. But is there any strong scientific background on it? For example, we have some scientific background for compilers. There is some scientific background for functional programming or object-oriented programming. Would it be possible to have something for the microservices, mesh-based world?

Tomáš Černý: You are asking a great question. Right now, microservices in cloud-native systems are very much driven by the industry. Industry is much faster in implementing things than academia. In the past, we were building monolithic systems, and we have been very comfortable with that. But with microservices, we have to think in a decentralized mindset, so we lose the system-centric view. We need to understand the holistic system perspective so we can establish some reasoning on whether an aspect of the system has the right quality or not.



Visiting Baylor students working in the Red Hat offices in Brno, CZ

Over the years, we have established evaluation criteria that allow users of monolithic systems to say, "Is this the right approach to build a system?" For a distributed system, we have to develop new metrics and patterns, and we need the mechanisms to detect these. If suddenly we are operating at multiple codebases, the problem is significantly harder. Can we do this with dynamic analysis? Currently, we can detect cyclic dependency, coupling or structural coupling, and similar measures. But if we have access to the code and get the entire perspective before something gets published, we certainly have much stronger quality assurance.

The development process with decentralized systems is complicated because you have independent teams. They make their own decisions, and something that doesn't cause a problem in their microservice might have a significant impact on the overall system. Right now, we deploy it and wait for DevOps to see it and say, "Something went wrong. We have to fix it." That's not the right approach. The most efficient approach is to get feedback immediately: "We have committed something that will have some implications."

Right now, those tools are not there. We need those tools because we would like to have the comfort of building decentralized systems with a monolithic-like view, and there is nothing like that at this point. To get there, we need to determine what a system-centric view of a decentralized system looks like, so we can assess the implications and impacts of

Red Hat offices in Brno, CZ

a codebase change within one microservice on the overall system.

Pavel Tišnovský: What about security? Will it be possible in the future to have some theory on how to make systems that are secure? Or rather, to have some patterns or rules, not just about what not to make, but how to make the system reliable and secure enough?

Tomáš Černý: First of all, there are three perspectives. The systems have to be deployed somewhere. We have the operating systems and the containers, and there is research on how easily, if one container is compromised, it can spread to the other containers. Then we can look at the security within the implemented system. Since it's decentralized, we have custom roles for role-based security in one system part and different ones in another part. Do we enforce it in a similar way across the holistic system? Right now, we depend on code analysis done manually, and no one is going to check this regularly, right? The system evolves very quickly, and one of your microservices can end up having a restriction, but another completely avoids the restriction. You don't even know what data you are sending out; maybe those data are private.

Then there is an entirely different area of research: secure by design. But are developers experts in security? No. Are they going to understand the policies and the guidelines for security? No. So what do they end up with? They end up with whatever they have been doing until now. We need a tool that will indicate, "Hey, something is wrong." The problem again is the heterogeneity of those systems.

We need to determine what a system-centric view of a decentralized system looks like, so we can assess the implications and impacts of a codebase change within one microservice on the overall system.

About the Authors

Matej Hrušovský has been with Red Hat for more than nine years, seven of which have been spent managing the university program in EMEA. Aside from attracting new talent mainly from universities and schools, the core of Matej's job is to find and put the right people from Red Hat and academia in the same room together.

About the Authors

Pavel Tišnovský is famous for his in-depth articles on various technical topics for the Czech Linux magazine root.cz. He taught computer graphics at Brno Technical University and worked as a C, C++, and Java developer in various companies before he joined Red Hat as a quality assurance engineer in the OpenJDK team. Now he works as a software developer and tech lead using Python and Go programming languages to develop microservices for OpenShift Insights. He also teaches professional Java and Go training.

Baylor students visit the Red Hat office. From left: Samuel Kim, Luka Lelovic, Michael Mathews, Mia Gortney, Garrett Parker, Patrick Harris, Kate Burgess, Dante Hart, with Pavel Tišnovský (Red Hat)

So for all these reasons, we need new tools that will tell us what our system looks like. Are those tools going to be based on dynamic analysis with tracing because it's comfortable and easy to add? I don't know. I think we should look for static analysis to answer those questions. Are there any tools that could do that? No, but I believe there should be, because look how many problems it would solve.

Matej Hrušovský: I think you've answered all our questions. Is there anything that you wish we had asked that we didn't?

Tomáš Černý: No, but I'll add that I'm excited to work with you guys! There is a great opportunity to look into the static analysis we are currently researching. I talked at a Europe RIG meeting about static analysis used for microservices, which I hope motivates others to get their hands on it and contribute to it.

So if you are an academician, reach out. If you are a technical person, reach out. You can either test our tools or share with us the challenges you are facing but don't have the resources to solve, and we can develop a prototype. We cannot promise a fully working tool, but a prototype proof of concept could solve some of your problems. Then it could have the momentum of the community to spin off a community-based open source project.

Matej Hrušovský: That is the hope of everything Red Hat Research is doing: connecting the right people and creating opportunities for people to work together on interesting research that may be useful elsewhere. Thank you very much for finding the time to talk with us.


MAKING THE CLOUD LESS, WELL, CLOUDY.
Join the MOC Alliance, as we create the world's first open cloud.
The Mass Open Cloud Alliance (MOC Alliance) is a collaboration of industry, the open-source community, and research IT staff and system researchers from academic institutions across the Northeast that is creating a production cloud for researchers. Of course, a collaboration is only as good as its collaborators.
So, we invite you to check out the partners at massopen.cloud and join us to create tomorrow's open cloud.
To speak with someone, please call +1 617-353-4118 or email contact@massopen.cloud.
MOC ALLIANCE
Hosted at
Boston University Rafik B. Hariri Institute for Computing and Computational Science & Engineering
BOSTON UNIVERSITY

RESEARCH QUARTERLY

VOLUME 4:2

Feature

About the Author
Jim Craig is a product manager in Red Hat's global public sector team. He helps define solutions to customer challenges using Red Hat technology and services as well as services from the partner community.

Can streaming data and machine learning build better communities?

An open source powered smart village project underway at the Red Hat Collaboratory may have the potential to change the world—or at least a town near you.

by Jim Craig

For as long as I can remember—and after almost 40 years in the IT industry, that's quite a while now—every year for the last 20 years or so has been "the year of the smart city." It's like when groupware was new and promised the paperless office, prompting the amusing response, "We are more likely to see the paperless toilet before the paperless office."

Fortunately, only one of these predictions has recently come to pass, and there are now numerous examples of smart cities, from Malaga to Montevideo. These smart cities are increasingly driven by open source, open standards, open data, and open APIs that, when combined to create a platform, make the design and deployment so much easier and quicker.

That is the foundation of a project now in process, "Creating a global open research platform to better understand social sustainability using data from a real-life smart village." In December 2021, the project was one of the recipients of the inaugural Red Hat Collaboratory Research Incubation Awards, awarded to projects that take advantage of the collaboration between Boston University and Red Hat Research. But this story begins much earlier.

PLATFORMS

Early in my tenure with Red Hat, I was introduced to FIWARE, a not-for-profit foundation based in Berlin, Germany, that had developed an open source platform for smart solutions. Originally funded by investment from the European Union, and now through membership fees and engaging in public sector funded projects, FIWARE developed a context broker able to ingest data in many different formats, from many different sources. After normalization, this data is available for developers to build applications to deliver value to citizens, city officials, and their partners and stakeholders.

With over 400 members at the time of writing, FIWARE has curated a framework of partners able to deliver data across a range of domains, from cities to utilities, manufacturing, and agrifood, with more being added all the time. Bringing this data together can have a powerful impact. For example, a street lamp and a bus have two very different contexts, yet if the street lamp "knows" where the bus is, it can switch on as the

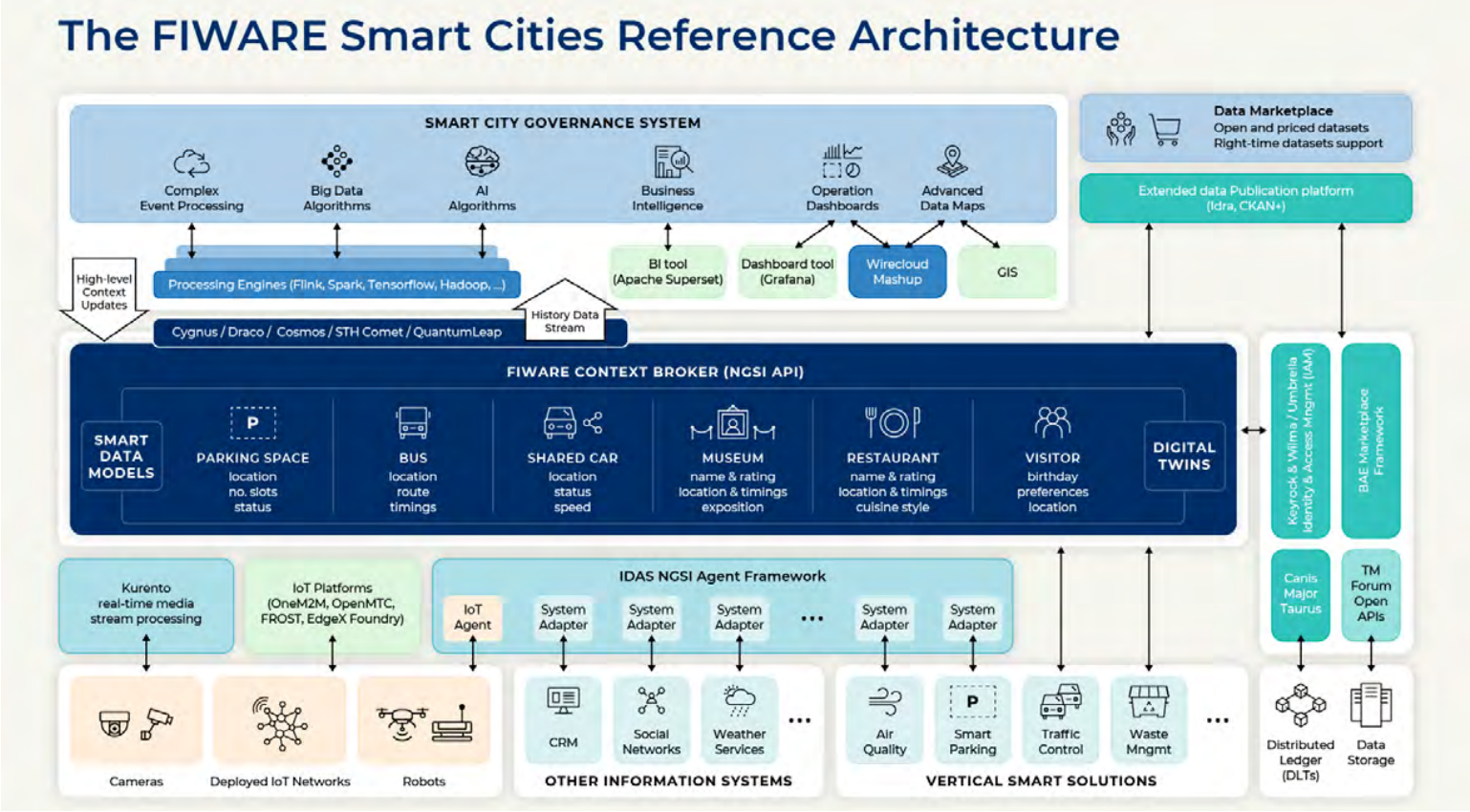


Figure 1: *An example of the possible data collection and analytics sources for a smart village using the FIWARE context broker. Image source: FIWARE*

bus arrives, allowing passengers to see better as they alight. The lamppost could provide direction information, local news and events, and a WiFi hotspot, remaining illuminated until all the passengers have left its coverage zone, switching off to save energy.

Figure 1 indicates the extensibility of the FIWARE architecture.

SMARTA BYAR AND SENSATIVE

During a webinar on smart cities hosted by Red Hat with OpenForum Europe in March 2021, Peter Geršak, State Secretary for Public Administration of the Government of Slovenia, said something that really struck a chord. Mr. Geršak pointed out that Slovenia has only one large city; it is a mostly rural country. He wanted to ensure that rural communities would benefit from these new developments and therefore used the term smart communities, which I like and have since adopted. After all, communities of all sizes can be more sustainable and efficient for their citizens, not just cities.

About a week later, completely by coincidence, Jan Malmgren contacted me with an idea to scale up the work he has been doing in Veberöd, Sweden, called Smarta Byar, which translates to *smart villages* in English. The village of Veberöd is in the Lund municipality in southern Sweden; it has a population of just over 5,000 people and is expected to almost double in the next few years. Malmgren, a resident of Veberöd since his childhood, decided to use technology to improve the lives of his fellow villagers.

Armed with a drone camera, Malmgren took around 50,000 aerial photographs of the village. The local university at Lund used

Figure 2: *Veberöd high-level digital twin. Image source: Jan Malmgren*

Figure 3: *Veberöd 3D digital twin. Image source: Jan Malmgren*

a supercomputer to stitch them together (**Figure 2**) and create a 3D model of the village (**Figure 3**).

To take his idea to the next step, Malmgren needed some Internet of Things (IoT) devices, a platform to capture and make sense of the data, and—most importantly—some smart people to make it all work. Enter local FIWARE partner Sensative, based in Lund. Sensative developed Yggio (pronounced *ig-yu*), a digitalization infrastructure management system (DiMS). Yggio is an open, unifying, and massively scalable IoT platform.

At the time of writing, the Unity game engine is the platform for the digital twin, which is being replaced by a platform from another Swedish organization, Smart Visualizer. (Smart Visualizer is based on Unity.)

EARLY USE CASES

With its rural location, the Veberöd area is frequented by summertime walkers who also like to barbecue. Using humidity and temperature-monitoring sensors, the Veberöd municipality can determine which barbecue areas are in use. Veberöd also lies in the primary agricultural region of Sweden. To ensure that pumps delivering water to cattle troughs are working, humidity sensors were installed and connected to Sensative's Yggio platform to detect any lack of water.

In collaboration with Telia, another IoT operator in the region, and through NB-IoT, bikes are tracked in the village. If someone moves a bike without the owner's phone being geolocated nearby, the system sends an alarm to the owner and the police. The bike is also GPS-tracked so that it can be more easily recovered. School gates are monitored for whether they are open or closed to help with child safety. Finally, water quality and use are monitored in collaboration with the local water utility company VA Syd.

SCALABILITY THROUGH DATA SCIENCE

Almost immediately after meeting Jan Malmgren, I shared the project idea

with my colleague Alexandra Machado, who leads Red Hat's Social Innovation Program. The Social Innovation Program works with not-for-profits to help them realize a benefit to their operations using Red Hat open source technology. Past engagements have included the World Health Organization, Greenpeace, UNICEF, and Curriki. Machado suggested I approach the Red Hat Research team and submit a proposal for a Collaboratory Research Incubation Award with Boston University. Adding Boston University (BU) to the ongoing collaboration between Red Hat and Smarta Byar has provided a real injection of energy, intellectual rigor, and a focus on new use cases.

Distinguished Professor of Engineering at BU Christos G. Cassandras, our Principal Investigator (PI), helped with our successful bid for the incubation award. Cassandras is also head of the Division of Systems Engineering and Professor of Electrical and Computer Engineering at the Center for Information and Systems Engineering (CISE) at BU, and he brought with him a talented team of academics and students to work on the next phase of the project, including co-PIs Vasiliki Kalavri, John Liagouris, and Mayank Varia. I asked Cassandras how the project would scale up and out and what part data science plays in scaling.

Professor Cassandras explained: "There are two aspects of scalability that apply to our project and the question of how we can take what we develop from a small village like Veberöd to large mega-cities worldwide. First, there's the software platform side of scalability. Can we

transfer a smart city app in Veberöd to a large city? I believe this is what the Red Hat OpenShift Container Platform can accomplish. For example, suppose our primary use case at the end of this project were an adaptive traffic light control (TLC) app for the main—and possibly only—major intersection in Veberöd. This would automatically adapt green-red cycle times based on the traffic sensed, including the arrival of pedestrians.

"Now, suppose we want the same app to scale up to a big city intersection with multiple traffic flows and much higher traffic volumes. If our software platform is properly designed, any tweaking we do in Veberöd (where we can continue to study the app's operation with the luxury of less traffic) can be readily transferred to the big city traffic light even though its operating conditions are seemingly very different.

"The other aspect is the system side of scalability. This aspect is a bit more subtle, but one I can better explain in terms of data science, and potentially one with broader implications for modern technology. Take the same example of an adaptive TLC app first developed in Veberöd's lone main intersection. In a large smart city, we first want to scale our ability for adaptive TLC to multiple intersections interconnected in a large network. The ability to do that would allow traffic lights to communicate with each other—and with smart vehicles—and potentially create green waves, or cascading green lights that keep traffic flowing continuously.

"As a result, vehicles seldom stop, thus drastically reducing congestion, energy consumption, and, in the case of fossil-fueled vehicles, toxic gas emissions. The difficulty with this is the curse of dimensionality, a term coined by Richard Bellman, the inventor of the dynamic programming optimization algorithm. In other words, the complexity of extending

what works in one node of a network to multiple nodes increases exponentially with the size of the network. Things just don't scale using one particular approach, but there is an alternative.

"However, the curse of dimensionality starts with the premise that the algorithms we develop with data science are based on data collected through the traditional time-driven paradigm: With every tick of some underlying clock, the algorithm is updated. This approach is extremely computationally expensive and often unnecessary, since the specific information an algorithm needs to operate may not have changed from one clock tick to the next. Not to mention that if the algorithm operates in an energy-constrained wireless environment, communication with every clock tick wastes battery life (especially crucial for IoT devices!). And, from a security standpoint, every clock tick becomes an opportunity for data to be jammed or stolen.

"Fortunately, we now have the new event-driven paradigm at our disposal: the algorithm is based on events of interest to its specific operation, so it is triggered only when relevant data are detected. In this mode of operation, computations are fewer, communication is drastically reduced, and security is tighter. There are fewer instances of information exchange, and while clock ticks are predictable to a malicious agent, events are generally not predictable. In the adaptive TLC example, the algorithm that adjusts green-red cycles does not need to be updated in a time-driven fashion. The only events that matter are the IoT sensors' detections of vehicles or pedestrians—and, of course, the light-changing events, which we control directly.

"How does this affect scalability as defined above, extending from one node in a network to multiple potentially interacting nodes? If a data-driven algorithm operates in an event-driven manner, it scales with the size of the events involved, not the size of the network. Going back to the TLC example: in a single intersection, the TLC algorithm is triggered only when one of a small number of possible events occurs. Therefore, as nodes are added, the additional TLC algorithms also depend on a small number of events that occur at the associated intersection.

In this mode of operation, computations are fewer, communication is drastically reduced, and security is tighter.

"If we want to extend our TLC app from a single intersection in Veberöd to hundreds of intersections in Stockholm or Boston, this effort is scalable—that is, it increases linearly—with the number of events at each intersection. If it scaled with the number of nodes in the network, the effort would increase exponentially."

At this stage of the project, there are so many interesting and innovative ideas flying around, it's like being a kid in a candy store. One of these ideas is what Malmgren has termed social sustainability: checking on the health and mental well-being of his fellow villagers using a simple app combined with machine learning and AI models to understand if there is a correlation between water consumption, happiness, and well-being.

Or could it be speeding cars, light pollution, or air quality impacting the well-being of the villagers? Another exciting example discussed recently is reducing or possibly eliminating cars from the village through the use of on-demand, autonomous electric vehicles.

Whatever use case(s) we arrive at, there is confidence it will have been determined with the highest level of citizen engagement, intellectual rigor, and eye to the future. Malmgren is very clear in his goal: He would like this smart village platform to go global, being adapted, tweaked, and tuned for different environments and use cases. Stay tuned to Red Hat Research to see what happens next!

See Robert Spal's article "How open data standards make Brno a better city" on the next page for an example of smart community concepts in action.

RESEARCH QUARTERLY

VOLUME 4:2

Feature

How open data standards make Brno a better city

by Robert Spal

About the Author
Robert Spal is a GIS Specialist in the data and analytics department of the city of Brno. He manages the technical and functional development of the data.brno.cz urban datastore to ensure that the content, functionality, operation, and user experience meet the needs of its users. He is also responsible for data acquisition, processing, and in-depth spatial analysis. When not working, Robert likes to swim and travel and is an enthusiastic cyclist.

Brno, Czech Republic, is home to the world's largest Red Hat technology center, and it was the birthplace of the university-industry relationship model that became Red Hat Research. Here's how the smart city concept has been implemented in one of our hometowns. The article stems from a presentation at DevConf.cz 2022.—Ed.

To flourish in an ever-changing world, cities need to increase their use of urban data. Municipalities all across the globe are following this trend, and the city of Brno is no exception. Geographic information systems (GIS) and spatial data help form the backbone of our city's decision making and policy planning. Additionally, our urban open data platform allows the city and the general public to use the data. Without these components, we would not be able to deliver the high-quality services and great quality of life Brno is famous for.

Our city collects and maintains millions of data entries each day. They vary widely, ranging from greenery to traffic data to population data. Most of the data collected have a spatial dimension and are best maintained in the GIS, which we use throughout the urban environment. The city itself, city companies, city districts, and all publicly funded entities use a single GIS. This allows for efficient data sharing across the organizations and robust utilization of the data for both day-to-day management and strategic planning and policymaking.

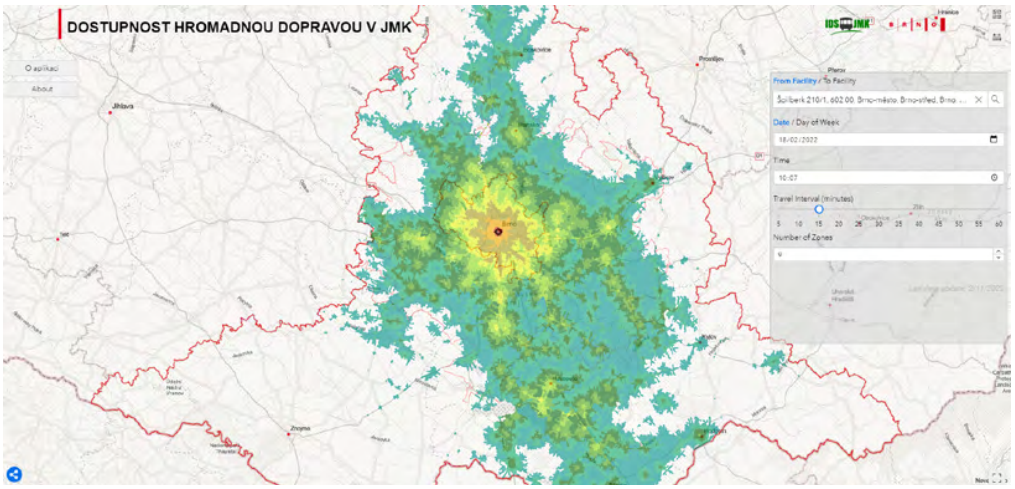
On top of the GIS sits the urban open data platform data.brno.cz, which serves as an entry point for anyone from the public to download or incorporate city data into their projects or systems. As part of promoting transparency and openness, we have also begun to emphasize publishing data using open data standards.

> Open data is one of the fundamental pillars of our smart city concept.

This means that data is published in a machine-readable format, free of license restrictions, and easily accessible. Most of them are continuously updated, and all have complete metadata. We provide around 150 datasets this way.

TRANSIT

One of the use cases we are most proud of is an app that creates accessibility models of our region by public transit using general transit feed specification (GTFS) data. The application is used primarily by traffic planners to identify



In this visualization of public transit accessibility, each color represents a 15-minute zone from the city center by public transit.

A map visualizing bike traffic intensity data through different colors also shows the number of bike accidents.

how different parts of a city or region are doing in terms of accessibility. They can detect areas with better or worse accessibility by public transit and get a quick and easy comparison with other areas. This insight is then used in planning to ensure better accessibility, connectivity, and efficiency for the whole system.

However, the application is also accessible to the general public. It can be used, for example, to select locations when looking for new housing so that it is possible to get to work or school in a specific time interval. The application can also serve private companies who need to better understand the transportation context of a location and compare the connectivity of different parts of the city or region.

CLIMATE CHANGE

Another app built with the help of our data is the map of photovoltaic potential, which visualizes how much sunlight falls across the whole city. The detailed map allows citizens to identify their rooftops' potential for installing photovoltaic panels.

Brno has a comprehensive plan to build solar panels on city-owned buildings, and the app allows the environmental department to identify and assess where they would deliver the best returns.

Together with a thermal map used for better tree planting, it is one of the supporting tools we use to combat climate change.

RECYCLING AND WASTE MANAGEMENT

We have also developed a walking accessibility model that visualizes how long it takes to get to the nearest container for recyclables. The model contains three zones (for one, two, or three minutes of walking) and is constructed for each type of recyclable, such as paper or plastic. The model is updated daily, allowing our colleagues in the environmental department to assess the efficiency of the complete network and propose changes. Our goal is to get to a state where no one need walk more than three minutes to get to the needed container; later, we would like to reduce that to two minutes.

TRAFFIC

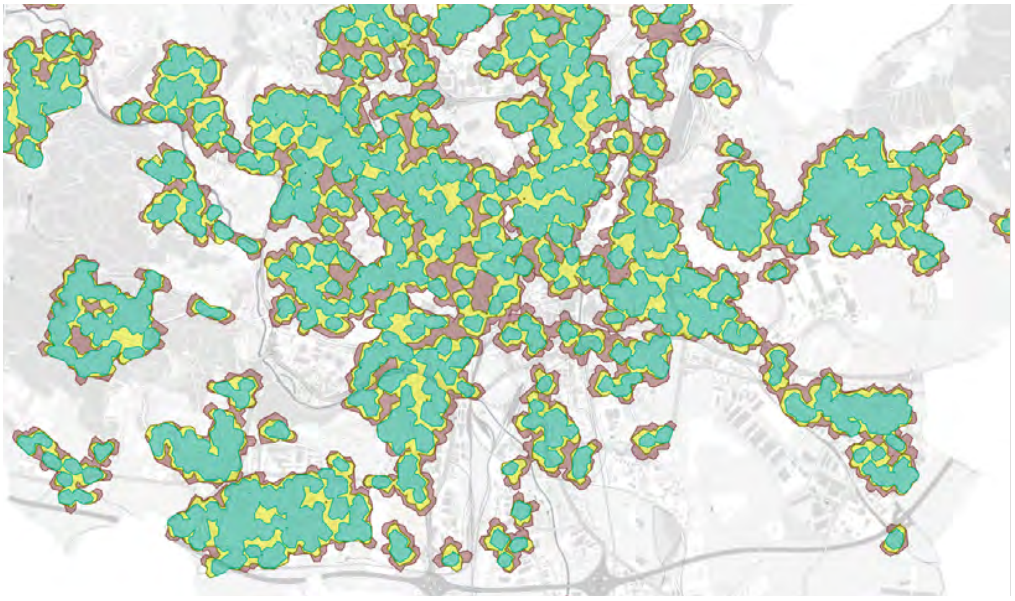
Whether for bikes, cars, or any other type of vehicle, we collect traffic data and use it extensively. For example,

every major intersection in our city collects the number of cars in real time. This way, we know what will happen if we close some streets and can prevent or at least mitigate traffic jams. The model is enhanced by Waze near-real-time data feeds, including current road events or traffic speeds, which we get as a partner of the Waze for Cities program. We also collect the public transit positional data, which, for example, allows transit vehicles to get preferential access at traffic lights. We expect heavy utilization of this data by the public, as we've received many requests.

Our transportation department also collects the bike traffic intensity data from several sources. We have bike traffic intensity sensors on the major cycling thoroughfares and get the data from a "Bike to work" event, where thousands of people record their daily trips to work for the entire month of May. The Strava app provides recreational biking data, and we also survey the streets every two years. All this biking data is then used to plan new cycle lanes and paths and assess the current ones. Most of this data can be found in the bike traffic intensity data app.

Parking data is a vital part of traffic data. We maintain huge swaths of parking data, ranging from specific allocated parking spots through parking meters to real-time car park capacity data. The data is used in policy planning for the regulated parking system that covers almost the whole of the city. We recently opened the data to the general public, and several companies already utilize it in their navigational apps or systems.

In the very near future, we plan to publish the positional data of our garbage trucks so that navigation companies such as Waze or Garmin can display them in their systems. This will help drivers save time by avoiding streets with traffic jams caused by garbage trucks, as many of our streets are exceptionally long and narrow.



Each color change represents a one-minute walking distance to the nearest container for recyclables, from green (one minute) to red (three minutes).

PUBLIC DATA ACCESS

These are only some of the ways data is used to make Brno a healthier, happier city. There is a wide range of valuable applications that can be built on top of city data to improve the environments we live in. Publishing those data using high-quality open data standards can further their impact and will eventually introduce new tools or apps that make the lives of citizens easier and more sustainable.

Although the examples given here reflect the Brno region at a very local level, all of the results are replicable in other cities as well since we use only worldwide standards and software solutions. The best part is that if you would like to test and try out some of your ideas and your city doesn't provide the data you are looking for, you can always try ours! Everything discussed above is published in an open format on data.brno.cz, and it is also in English.

RESEARCH QUARTERLY

VOLUME 4:2

Feature

About the Author

Alexander Bulekov is a Computer Engineering PhD Candidate at Boston University and an intern at Red Hat Research. Alex is advised by Professor Manuel Egele and is interested in systems security topics including fuzzing, web security, and OS/Virtualization security.

Applying lessons from our upstream hypervisor fuzzer to improve kernel fuzzing

Could a grammarless approach increase its effectiveness?

by Alexander Bulekov and Bandan Das

Low-level systems such as Linux kernels and hypervisors form the foundation of cloud systems today. The virtual machines (VMs) provided by hypervisors are attractive targets for attackers. Bugs in hypervisors create the risk of an attacker in a malicious VM, compromising the isolation guarantees provided by the hypervisor, the underlying system. and the neighboring VMs. Similarly, OS kernels continue to be targets for attackers, as they are often tasked with managing access to hardware and enforcing separation of privileges between processes. A kernel compromised in a VM can place the entire physical system at risk, since an attacker with a VM kernel has complete access to the hypervisor's virtual-device attack surface.

Fuzz testing has recently become popular as a proactive measure to identify bugs in software before they are exploited. Fuzz testing (or fuzzing) is an automated technique for generating software input and detecting bugs in its execution. Fuzzing has been applied to both hypervisors and kernels; however, the state-of-the-art techniques require extensive manually written grammars to fuzz each individual interface. These grammars create a potential scalability problem, as hypervisors and kernels grow rapidly and outpace manual efforts to create descriptions.

In RHRQ 2:1, we introduced Morphuzz, a technique to fuzz hypervisors ("Fuzzing hypervisor virtual devices"). Since then, our Morphuzz conference paper has been accepted and will appear at Usenix Security '22. We fully upstreamed Morphuzz into the QEMU (Quick EMUlator) hypervisor and found dozens of bugs. Additionally, we leveraged the lessons learned while developing Morphuzz to create a novel technique for fuzzing OS kernels. This article will describe the improvements we made to Morphuzz that made it successful upstream. We will also briefly delve into how we are applying Morphuzz' concepts to kernel fuzzing.

MORPHUZZ UPSTREAM

The Morphuzz fuzzer permits fuzzing virtual devices across all three major input/output (I/O) interfaces—port I/O (PIO), memory-mapped I/O (MMIO), and direct memory access (DMA)—without any specific harnesses or grammars for individual virtual devices. One of Morphuzz' key benefits is its ability to transparently fuzz complex DMA-based data structures by reshaping the input space. Typically, DMA transfers are performed asynchronously by the device, making DMA fuzzing difficult for a fuzzer operating from the perspective of a CPU. However, Morphuzz applies hooks to the hypervisor to transform DMA into a synchronous operation. This allows

the fuzzer to populate data accessed by DMA on demand, just before it is read by the device.

Continuous fuzzing has become an integral part of the software development process for many projects, as it can rapidly catch bugs before they make it into releases. Thus once our original prototype implementation of Morphuzz for QEMU was successful, we were enthusiastic about integrating it into upstream QEMU to enable continuous open source fuzzing.

During the upstreaming process, we developed:

- Documentation for using Morphuzz
- Scripts to fuzz QEMU on the OSS-Fuzz platform, which provides resources to fuzz security-sensitive open source projects continuously
- Tools to unbend and minimize crashes detected by Morphuzz, as well as a script to convert crashes into copy-pasteable reproducers and upstreamable QEMU test cases

The tools, in particular, are important for end users. At its core, Morphuzz uses a simple opcode interpreter to convert inputs received from the fuzzing engine (libFuzzer) into sequences of I/O operations. When Morphuzz finds a crash, it is easily reproduced by simply providing the crashing input to the Morphuzz opcode interpreter. However, QEMU is maintained by close to 200 developers, most of whom may not be familiar with the fuzzing infrastructure. Simply providing binary Morphuzz crashes creates additional work for the developer, who will need to build QEMU with fuzzing support and understand Morphuzz' essential inner workings. For our upstream integration, it was necessary to provide a mechanism for converting Morphuzz crashes into standard QEMU QTest test cases that developers are familiar with.

QEMU features a simple facility called QTest for creating virtual-device test cases. QTest allows developers to easily create readable unit tests for virtual devices. Morphuzz unbends each crash into a standalone QTest reproducer. It then replays the crashing input through the opcode interpreter and logs the resulting linear sequence of MMIO/PIO and DMA-related device I/O commands in the order they were issued. Since real VMs do not populate DMA buffers on demand, Morphuzz annotates all I/O commands used to fulfill DMA accesses in the log with a prefix. Then, Morphuzz simply rearranges the logged I/O commands so that each command filling a DMA request precedes the direct PIO/MMIO command that triggered it.

The result is a linear QTest API trace, which can be piped into a standard QEMU process to reproduce the crash. We also provide a facility to minimize these QTest traces by removing operations (or parts of operations) unnecessary to reproduce a crash. As a result, we can often reduce several-megabyte-sized QTest traces into several hundred bytes. The QTest trace can be sent to virtual-device developers along with the command line used to specify the connected virtual devices, as a simple, self-contained, straightforward way to reproduce crashes.

We also provide a lightweight tool that can convert Morphuzz' automatically generated QTest traces into C code that can be committed upstream for regression testing new changes. OSS-Fuzz is continuously running Morphuzz for over 40 virtual device configurations. Due to Morphuzz' generic design, it usually takes just seconds to add support for fuzzing additional devices. Combining OSS-Fuzz reports with the infrastructure for creating reproducers has led to over 100 bugs reported and 15+ CVEs (common vulnerabilities and exposures) to date.

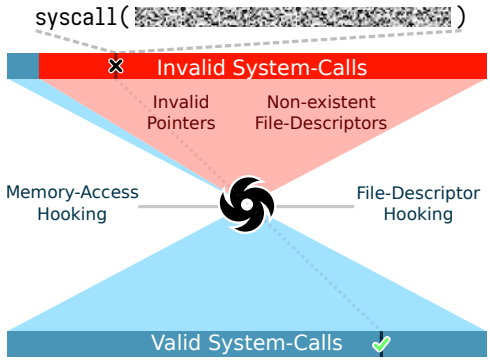
FUZZING THE LINUX KERNEL

The Linux kernel continues to serve as one of the most security-critical building blocks in modern computing. The kernel's fundamental role in managing resources and enforcing isolation between applications makes it an attractive target for attackers. Recognizing

About the Author

Bandan Das is an engineer in the Virtualization group at Red Hat. He spends most of his time on KVM, QEMU, and, more recently, containers. His research interests are in the areas of systems performance and hardware partitioning. As part of Red Hat Research, he is involved in the fuzzing project, teaching, and mentoring.

One of Morphuzz' key benefits is its ability to transparently fuzz complex DMA-based data structures by reshaping the input space.

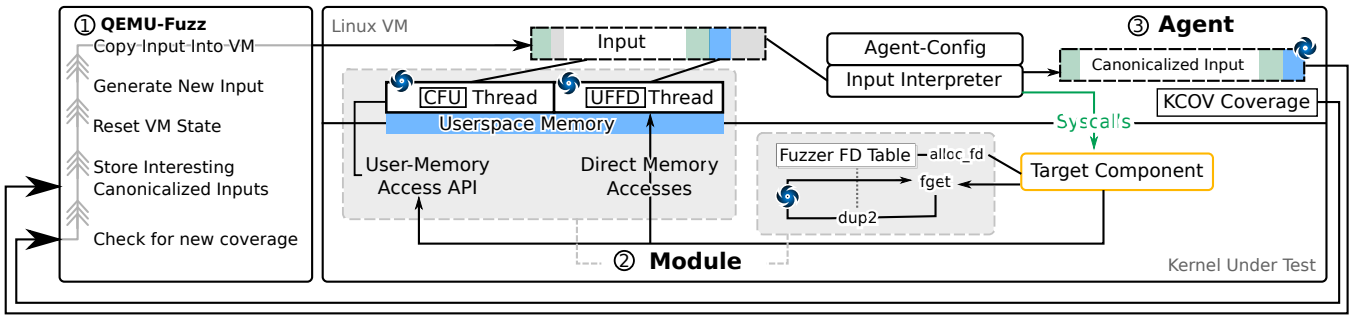


By default, randomly generated system-calls cannot reach deep kernel-code paths, as the arguments quickly trigger error conditions. Our fuzzer avoids this by reshaping the memory and file-descriptor input spaces.

the critical nature of OS security, many fuzzers have targeted OS kernels. Most OS fuzzers focus on the critical system-call interface, which enables user-space applications to request services from the kernel. The state-of-the-art kernel fuzzer, Syzkaller, has reported thousands of Linux kernel bugs. Thanks to its accessibility and automation, Syzkaller has become an integral component of the kernel-development process. However, Syzkaller relies heavily on manually written descriptions for every kernel interface. For example, it features 2,013 lines of hand-written descriptions of the kernel's KVM interface (used to provide access to hardware-accelerated virtualization). Syzkaller contains tens of thousands of interface descriptions, with many interfaces still missing support.

While developing Morphuzz, we noticed key similarities between the virtual-device input space on hypervisors and the system-call input space exposed by kernels. Hypervisors can asynchronously read DMA inputs from any location in guest memory. Likewise, kernels can access data in user-space process memory while handling system calls. Thus, we identified that Morphuzz' model for fuzzing virtual devices could likely be replicated in the kernel. However, the Linux kernel is a significantly more complex target than most hypervisors. While QEMU has a sizeable 2 million lines of code, Linux has close to 30 million lines. Additionally, unlike hypervisor fuzzing, kernel fuzzing has been an active area of development for decades.

A simple grammarless fuzzer could pass fuzzer-provided integers as arguments to system calls. However, such a fuzzer is quickly rendered useless by the effectively boundless system-call input space induced by pointer and file-descriptor arguments. Fuzzers such as Syzkaller require annotations of struct types, flag fields, enumerations (enums), and constants passed as system-call arguments. Our fuzzer is based on the core insight that instead of relying on extensive system-call descriptions, the system-call input space can be reshaped



Overview of our kernel fuzzer featuring a combination of snapshot-fuzzing, kernel-hooking, and test-case execution components.

(much like Morphuzz' reshaping of the DMA space) to make it conducive to fuzzing. That is, we leverage the APIs used by kernel code to handle system calls to reduce the input spaces associated with memory and files. By reshaping the input space, we can essentially rely on a general-purpose fuzzer (e.g., libFuzzer) to produce complex system-call behaviors. This technique eliminates the need for detailed descriptions and harnesses for individual system calls.

Harnessing the kernel is more difficult than harnessing QEMU (a user-space application). As such, we relied on emerging full VM snapshotting to execute kernel fuzzer test cases in individual VMs, which are rapidly restored from a snapshot after each input. As we mentioned above, file descriptors pose a challenge for kernel fuzzing. File descriptors are passed through system calls as simple integers. A typical process only has a handful of files open. As a result, the semi-random mutations of a fuzzer are highly unlikely to generate an integer system-call argument associated with a valid (i.e., open) file. Even if the fuzzer guesses a valid file-descriptor integer, it would simultaneously need to pick a system call and arguments that are valid for that type of file.

Component	Syzkaller		Grammarless Fuzzer	
	Edge Count	Syzlang LoC	Edge Count	Conf g LoC
bpf	3623	864	3242 (89.48%)	1 (0.12%)
video4linux	595	381	540 (90.76%)	4 (1.05%)
rdma	545	1474	503 (92.29%)	5 (0.34%)
binder	339	272	342 (100.88%)	6 (2.21%)
cdrom	136	351	139 (102.21%)	5 (1.42%)
kvm	7802	891	7986 (102.36%)	7 (0.79%)
vhost_net	218	157	225 (103.21%)	9 (5.73%)
drm	1832	745	1962 (107.10%)	7 (0.94%)
io_uring	960	343	1115 (116.15%)	6 (1.75%)
tty	1408	381	1845 (131.04%)	9 (2.36%)
Average			103.55%	1.67%

Our grammarless approach achieves coverage that is competitive with Syzkaller, at a small fraction of the configuration cost.

Our fuzzer reshapes the file-descriptor space by hooking the internal APIs used by the Linux kernel to associated file-descriptor numbers with the underlying resources. Then, using the dup2 system call, we associate the fuzzer-provided integers with valid files. This ensures that any fuzzer-provided value interpreted as a file descriptor by the kernel is mapped to a valid open file. Our strategy for user-space memory accesses is similar; we hook the major APIs used by system calls to copy data from user space. However, the kernel provides additional mechanisms to access data

The kernel's fundamental role in managing resources and enforcing isolation between applications makes it an attractive target for attackers.

in user space, some of which cannot be hooked through a centralized API. To work around this, we applied the Linux kernel's **`userfaultfd`** feature to detect access attempts to user-space memory at the MMU level.

Thus we have two layers of hooks for user-space memory: the centralized API hooks and the **`userfaultfd`** hooks. Similar to Morphuzz, at its core, our kernel fuzzer simply interprets binary inputs from a general-purpose fuzzing engine (libFuzzer) into sequences of system calls. Hooks transparently allow the fuzzer to populate file descriptors and complex data structures just in time for the kernel to access them.

We found that even though our kernel fuzzer does not feature detailed system-call descriptions, it achieves competitive coverage compared to Syzkaller. Furthermore, we found new issues in code already covered by Syzkaller, highlighting that manually written descriptions can often underfit or overfit the interface in question. Our prototype implementation requires only a few lines to add support for fuzzing additional kernel interfaces (1.7% as many lines as Syzkaller's descriptions) while achieving 103.5% of Syzkaller's coverage.

Currently, our grammarless solution is completely separate from the kernel's existing extensive fuzzing infrastructure (based mainly on Syzkaller). Syzkaller benefits from years of contributions by dozens of developers. As such, we hope to integrate our grammarless methods into existing kernel fuzzing code, which provides extensive facilities for creating reproducers, identifying commits that introduced bugs, and detecting regressions. Furthermore, Syzkaller can simultaneously fuzz an entire kernel, while each instance of our grammarless fuzzer focuses on an individual kernel component. Since bugs often lie at the intersection of a complex combination of kernel features, we are actively investigating ways our grammarless approach can be improved to fuzz all system calls without negatively impacting fuzzing performance. We are also working towards integrating a VM-snapshot-based fuzzer into upstream QEMU (this is the subject of a Google Summer of Code 2022 Project).

The Linux kernel is growing rapidly, and security efforts need to advance just as quickly. We found that it is possible to decrease the reliance of kernel fuzzers on manually written descriptions while maintaining competitive coverage and bug-finding performance. We are excited to see the potential of grammarless approaches when combined with modern fuzzing techniques, and we are actively investigating ways to upstream parts of our kernel fuzzer so that it may serve the Linux community for years to come.

When this issue went to press, the 31st USENIX Security Symposium had not yet occurred. Visit usenix.com/conferences to find and view media from this event.

THE UNIVERSAL AI SYSTEM FOR HIGHER EDUCATION AND RESEARCH

NVIDIA DGX A100

Higher education and research institutions are the pioneers of innovation, entrusted to train future academics, faculty, and researchers on emerging technologies like AI, data analytics, scientific simulation, and visualization. These technologies require powerful compute infrastructure, enabling the fastest time to scientific exploration and insights. NVIDIA® DGX™ A100 unifies all workloads with top performance, simplifies infrastructure deployment, delivers cost savings, and equips the next generation with a powerful, state-of-the art GPU infrastructure.

Learn More About **DGX** @ nvda.ws/dgx-pod
Learn More About **DGX on OpenShift** @ nvda.ws/dgx-openshift

RESEARCH QUARTERLY

VOLUME 4:2

Feature

About the Author

Kamil Dudka joined Red Hat as an intern in 2008 while finishing his master's degree in Intelligent Systems at Brno University of Technology. In 2009, he started to work on a formal verification tool named Predator, which has won several gold medals in the International Competition on Software Verification (SV-COMP). Since 2011, Kamil has been developing open source tools for fully automatic static analysis of RPM packages. In 2019 he joined the AUFOVER project, where he worked on the automation of formal verification in open source Linux distributions.

Verification of a Linux distribution

While research on formal verification continues, fully automatic dynamic analysis of RPM packages is now available for Fedora users.

by Kamil Dudka

In 2019, Red Hat joined the AUFOVER (Automation of Formal Verification) project, which focused on fully automatic detection of bugs in complex software products based on formal verification. The project was driven by Honeywell and supported by the Technology Agency of the Czech Republic. At that time, Masaryk University and Brno University of Technology were developing formal verification tools primarily intended for research. The industrial partners Honeywell and Red Hat were working to make the research tools practically useful and integrate them into their business workflow.

Red Hat collaborated primarily with Masaryk University because they were developing the tools we were interested in, namely Divine, which uses explicit-state model checking, and Symbiotic, which combines instrumentation, slicing, and symbolic execution. Our task was to integrate the formal verification tools into our environment and use them to formally verify the software that we distribute in our Linux distributions (Fedora and Red Hat Enterprise Linux). Thanks to the AUFOVER project, these tools are now available in Fedora through a simple user interface.

Our Linux distributions consist of RPM packages. The RPM package manager is used to install and remove software components of the system. We took advantage of this packaging system and built the automation of formal verification on top of it. In addition to RPM packages that can be installed on the target system, there are so-called source RPM packages, which contain source code and machine-readable metadata describing how to build the software from source.

The source RPM packages can be processed by tools like rpmbuild and mock in a fully automatic way to produce binary RPM packages. We have developed RPM-based automation of formal verification that takes advantage of the machine-readable metadata as well as the existing tools for automatic builds of RPM packages. We run the tools in a modified run-time environment, however.

The formal verification tools Symbiotic and Divine perform formal verification of C programs. The C language ecosystem does not come with any native framework for a fully automatic build from source code. The presence of the metadata in source RPM packages was crucial for the success of the solution we developed. On the other hand, the RPM packaging was initially developed for the build and installation of programs. The inventors of the RPM packaging format did not take fully automatic formal verification of

programs into account. To succeed with our task, we had to overcome a few technical obstacles.

The tools Symbiotic and Divine operate on the intermediate code of the LLVM compiler (so-called bitcode) rather than binary code for a specific architecture. However, we need to build the architecture-specific code, too, in order not to break the native build process. The binary files created by the compiler are often executed during the build, and the results they produce have a significant impact on the further build process. To address this problem, we modified the build environment such that the intermediate code is captured during the build and embedded directly into the resulting binaries. The intermediate code is later extracted from the binaries when the formal verification tools need it.

Another problem we needed to solve is how to automatically trigger the formal verification tools on the captured intermediate code. The RPM packaging format does not include any metadata describing which binaries should be given which parameters upon execution. The situation is even worse with RPM packages that install dynamically linked libraries only. The dynamically linked libraries cannot be executed on their own. They need to be linked to binary executables at run time.

Luckily, modern RPM packages contain test suites that run during the build. The entry point for these tests is a shell script that may, in theory, do anything. In addition to executing the programs produced by the build of an RPM package, the script may use external testing frameworks and other programs installed on the system.

Our goal was to trigger formal verification tools only for the programs produced by the build of the given RPM package. To achieve this goal, we used an innovative solution, csexec, based on wrapping the system dynamic linker. We also experimented with an alternative solution, ldpwrap, based on instrumentation of the `main()` function, as the commonly used entry point of C programs.

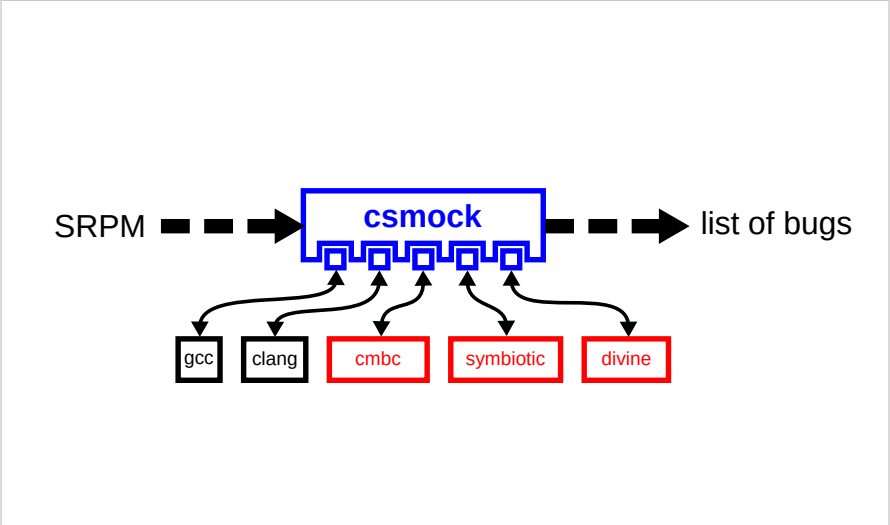


Figure 1. *Integration of formal verification tools as csmock plugins*

In both cases, the formal verification tool automatically launched upon each execution of a binary produced by the build of an RPM package, and the verification tool was given the original command-line arguments passed by the testing framework. In some cases, RPM packages contain a single test that executes multiple binaries in parallel, possibly communicating with each other, for example, through a pipe. Moreover, the tests themselves are often executed in parallel to deliver results faster. In such cases, the automatically triggered formal verification tools also run in parallel. Their results are being captured while the tools are running and subsequently serialized in a single file that contains all relevant results of the formal verification tools.

Thanks to the csmock tool, which we originally developed for static analysis of RPM packages, we were able to hide all the technical details mentioned above and offer our solution through a simple user interface. We have developed experimental csmock plugins (see **Figure 1**) for Symbiotic and Divine. Fedora users can just install a csmock plugin, run csmock with the plugin enabled, and wait for the results. Of

For an earlier report on AUFOVER, see "Automated Formal Verification," RHRQ 2:2 (August 2020).

course, csmock does not remove any limitations of the verification tools, like memory and time complexity or annotation requirements. Still, csmock makes it much easier to evaluate formal verification tools on RPM packages and, consequently, on a Linux distribution.

The formal verification did not finish successfully in the vast majority of RPM packages we tried. Instead, we optimized the automation to deliver at least partial results in a predictable amount of time. This is achieved by specifying a timeout for a single run of a formal verification tool. By default, the timeout is set to 30 seconds. For more complex RPM packages with higher test coverage, we needed to decrease the timeout value to get the results in a reasonable time.

Our most expensive experiment was the formal verification of coreutils-8.32-31.fc35 with Symbiotic, where the formal verification job took approximately eight hours on a machine with 16 CPU cores, even though the timeout was set to only eight seconds. This was primarily caused by the fact that, during the formal verification of a single RPM package, Symbiotic was triggered 24,470 times in total. To increase the search depth, we would need to increase the timeout value—at the cost of waiting longer for the results—or use more powerful hardware. As initially expected, we hit the obstacle of the excessive time complexity of formal verification.

All the interesting results of our experiments with formal verification of RPM packages are publicly available on GitHub, including an evaluation framework that can be used to reproduce our results on any Fedora system with sufficient performance. I would also like to highlight that Symbiotic was successful at the 11th Competition on Software Verification (SV-COMP 2022), winning three gold medals and one bronze. This success confirms that we selected the right tools to experiment with.

While the formal verification tools are still under research, the tooling we developed for automation is already practically useful. Thanks to our dynamic linker wrapper (csexec), we could extend covscan, our internal service for static analysis of RPM packages, to support fully automatic dynamic analysis with the tools Valgrind and strace.

Based on our experiments with bigger software projects, the developers of Symbiotic decided to work further on improving the tool so that it handles more practical use cases. The automation we developed will help us to monitor progress on this effort.

RESEARCH QUARTERLY

VOLUME 4:2

Red Hat

Column

Matchmaking for engineers: how we learned to bring research and industry together in a way that works

by Ilya Kolchinsky

About the Author
Ilya Kolchinsky is a research scientist and research supervisor with Red Hat Research and Technion—Israel Institute of Technology. He has a PhD and BSc in computer science, both from the Technion. Ilya's research interests span a wide range of topics in big data processing, such as distributed event-based systems, data stream mining, and applications of AI and machine learning in stream processing engines.

As a research supervisor, one of my most important tasks is finding a good fit between engineers with a problem and academicians who can collaborate with them on a project to explore some aspect of that problem.

Not every engineering problem is a good match for research—and not all professors are interested in topics immediately relevant to industry. Finding the right people and building a bridge between them is a process that has taken time to develop.

STARTING UP AS A RESEARCH SUPERVISOR

When I began work at the Red Hat Research site in Israel, we did not have someone in a research supervisor role. Although this role existed at other sites, every region has different relationships among academic, industry, government, and related entities; we needed to create our own processes.

Idan Levi (Research Director of Red Hat Research in Israel) and I knew that the research supervisor would function as the person between the academic and the industry sides of a project. What we needed to determine through experience was all the different elements of bringing engineers and academics together. For example, when a professor speaks to someone from industry, they may be using the same words but mean very different things. For successful industry-academia collaboration, you need someone who can translate. But the work goes well beyond that. In fact, it starts before a project even exists and then continues all the way to successful publication and upstreaming.

THE LIFECYCLE OF A RESEARCH PROJECT

My first responsibility is to find unsolved problems: is there a challenge an engineer or team is facing that could be solved with science? For example, is there a problem that an engineer is trying to overcome with a lot of compute resources that an efficient algorithm could address instead so the cost of resources and run time is not so high? My background (a PhD in Computer Science from the Technion University in Haifa, Israel, and a history of publishing) is very helpful here, because a research supervisor needs to understand both how an engineer thinks and what constitutes a good research problem.

Then I need to identify the relevant party in academia who would be a good fit for creating a collaborative project. To do this, I'm working all the time on expanding our network of academic connections, finding researchers who work in the technical fields relevant to Red Hat Research. I build a list of problem-solving engineers on the industry side and a list of researchers on the academic side, and then I can do the magic of matching the right people together. And that's all before the project begins.

At that point, I bring the parties together for a conversation, or sometimes several conversations, and help mediate to ensure that we can establish a research-worthy challenge and a common goal.

> Typically, academics and industry have different priorities: professors want to publish a paper, and engineers are thinking about products. At Red Hat Research, we have a unique position that helps us bring the two groups together.

More and more academics are becoming aware of the advantages of open research and want to contribute to open source projects. Red Hat is easy to collaborate with: we don't want to claim any intellectual property (IP), so we encourage researchers to publish freely. However, we also implement solutions in a practical way by upstreaming them as high-quality code. It does take some discussion to get the problem defined in a way that all can agree on, but it's essential to do that in advance so the project stays on track.

After the collaborators shake hands and start work, the research supervisor's role combines advanced project management and participating in the actual research, for example, by attending brainstorming sessions or supervising students on the technical aspects of the project. It's my responsibility to monitor the project for the entire lifecycle and make sure that it doesn't get frozen at some point. If a problem arises, I'm in charge of solving it. Here again, the research supervisor needs to see things from multiple perspectives. I need to understand the technical aspects of the project and academic process, and if someone is not satisfied, I need to understand why so I can resolve it and make sure everyone is happy with the results at the end of the day.

When the project is finished, I help see that it gets upstreamed. At some point, we will promote the innovation in hopes it will become part of a product that can benefit the masses, but that's not our role. Our goal is to push the innovation's boundaries.

CREATING HIGH-QUALITY PROJECTS

Because this way of working is still relatively new for us, I can't point to a project that completed the full cycle—yet. However, a good example of the process comes from an engineering team based partially in Israel that is working on low-level network visualization. In this case, we were approached by the team, which is the reverse of how it usually happens. This team wanted to begin a research project but didn't have the academic connections to do so.

I connected them with a professor from one of the universities we work with, and they have been in regular, active brainstorming sessions every two weeks or so for about the past five months. As of this writing, they have not entirely determined a final proposal, but that in itself is quite instructive. Although it has been difficult for them to find common ground and define a project that satisfies the needs of both sides, they really liked each other from the very beginning and are eager to work together. That suggests to me that they are a good match, so the project they eventually create will be something worthwhile.

That's one of the most valuable things about the research supervisor role. Having a person to make these matches and facilitate these conversations leads to higher quality projects. And they are high-value projects because they are based on an understanding of where the industry is going, what the leading research topics are, and where the potential for overlap is.

After taking the time to develop our processes for research supervision at Red Hat Israel, we are also sharing them. For example, in Brno at Red Hat Czech, Martin Ukrop has begun working as a research supervisor using what we've learned. This model has been successful here and in Brno, and I hope we can keep spreading this success to many locations over the coming years.

RESEARCH QUARTERLY

VOLUME 4:2

Research project updates

Each quarter, *Red Hat Research Quarterly* highlights new and ongoing research collaborations from around the world.

This quarter we highlight collaborative projects in Europe at Karlstad University (Karlstad, SE), Masaryk University (Brno, CZ), and the University of Zurich.

PROJECT: Interpersonal conflicts in code review

ACADEMIC INVESTIGATORS:
Prof. Alberto Bacchelli and Pavlina Wurzel Gonçalves (University of Zurich)

RED HAT INVESTIGATORS:
Cali Dolfi and Martin Ukrop

Code reviews benefit from generating alternative ideas and obtaining diverse feedback. However, this makes code review a place in which communication is difficult and interpersonal conflicts can arise. These can influence the quality of team cooperation in both positive and negative ways.

The Zurich Empirical Software Engineering Team (ZEST) from the University of Zurich has recently started cooperating with Red Hat to investigate interpersonal conflicts in code review. After a successful qualitative study on the topic, the group has contacted Red Hat Research to conduct a more broad quantitative study focusing on selected Red Hat's open source projects. The research will compare objective data from the publicly available repository metadata with the self-reported experience of Red Hatters and other community developers. If you are interested in gaining more insights into your project's code review health and code review conflicts, contact Martin Ukrop at mukrop@redhat.com.

Contact academic@redhat.com for more information on any project described here, or explore more research projects at research.redhat.com.

PROJECT: Vega project

ACADEMIC INVESTIGATOR:
Gabriel Szasz (Masaryk University)

RED HAT INVESTIGATORS:
Nikolaos Moraitis, Zdeněk Švécar, and Filip Hubík

Our curiosity constantly pushes technology towards higher goals, unveiling the secrets of distant realms far beyond our imagination. Astrophysicists at the Department of Theoretical Physics and Astrophysics (DTPA) at the Faculty of Science, Masaryk University, are among world-leading experts in stellar astrophysics. In this field, researchers tend to neglect the effects of stellar rotation to avoid the complexity of the problem, claiming these effects are too subtle to play an important role. The research group led by Ernst Paunzen has already collected some intriguing evidence that rotation significantly impacts the parameters and evolution of stars. They now need to prove their findings on a larger scale to convince a broad scientific community.

To achieve this goal, DTPA has started to build the new model atmosphere grid for the rotating main-sequence stars. In the end, this new model grid will help prove the case; at the same time, it will give astrophysicists a powerful online tool to study the effects of rotation on the parameters of stars and their evolution. This work has the potential to redefine what we know about the stars in our area of the universe.

Calculating such an extensive model grid is a highly demanding effort requiring high-performance computing to be feasible within a human lifetime. Conventional proprietary HPC solutions turned out to be too inflexible to fit the project requirements. The Vega Project team at Red Hat came up with an idea to harness the power of Kubernetes to provide the next-generation open source tool for high-performance computing. The project is well past the design phase, and the team is already working on the proof-of-concept implementation using the Red Hat OpenShift Container Platform.

The team's March 2022 presentation for Red Hat Research Days can be found on the Vega Project's research webpage at research.redhat.com/blog/research_project/vega_project.

PROJECT: Building the next generation of programmable networking—powered by Linux

ACADEMIC INVESTIGATORS:
Prof. Anna Brunstrom and Dr. Per Hurtig (Karlstad University)

RED HAT INVESTIGATOR:
Toke Høiland-Jørgensen

The eXpress Data Path (XDP) is a high-performance programmable data plane that runs in line with the regular data path in the Linux kernel. Powered by BPF, this technology allows flexible high-performance programmable networking to function in concert with the regular networking stack. This research project aims to explore how this capability can be leveraged to turn Linux into a first-class platform for next-generation programmable networking.

Thus far, the project has focused on two areas of inquiry: designing a programmable queueing interface for XDP, and using XDP and BPF for always-on passive network monitoring and analysis. For the queueing track, we have developed a programmable packet scheduling extension for the XDP framework, leveraging schemes for programmable queues recently proposed in the literature. This extension allows programmers to define their packet schedulers using BPF while benefiting from the XDP fast data path. This work aims to add an API for packet queueing in XDP to the upstream Linux kernel and use this to implement various queueing algorithms for XDP.

For the passive network monitoring track, we are investigating using the programmable network capabilities of BPF to enable an always-on passive network monitoring application, based on the existing "passive ping" (pping) utility. This can be used to extract latency measurements from any point in the network without injecting any traffic into the network, making it very lightweight and thus enabling continuous monitoring of latency in network flows.

PROJECT: Security and safety of Linux systems in a BPF-powered hybrid user space/kernel world

ACADEMIC INVESTIGATORS:
Prof. Anna Brunstrom and Dr. Tobias Pulls (Karlstad University)

RED HAT INVESTIGATOR:
Toke Høiland-Jørgensen

With the introduction of BPF into the Linux kernel, we are seeing a sea change in the traditional application model. With BPF, it is now possible to execute parts of the application logic in kernel space, leading to a novel hybrid userspace/kernel model. This exciting development brings with it many opportunities, but also some challenges, especially in the area of security. The new hybrid application model presented by BPF means that the kernel/userspace barrier is no longer the demarcation point between applications and (trusted) kernel code, affecting threat modeling for both applications and the kernel.

This project aims to look broadly at the various security issues related to BPF, the Linux kernel, and hybrid applications, hopefully improving the overall trust in the technology. The project is still in an early exploratory stage, but possible directions include:

- **Defining** a coherent framework for threat modeling of BPF applications, as no such model exists today
- **Exploring** the limitations of the existing in-kernel BPF verifier, in an effort to build more confidence in its capabilities
- **Analyzing** kernel resource constraint mechanisms and memory safety issues in relation to BPF
- **Exploring** the use of cryptographic signatures for improving the security of BPF programs