Serge Kosyrev

Curriculum vitae

✍ Zelenograd, Moscow (RU)
☎ +7 (905) 5380951
✉ kosyrev.serge protonmail com
ϟ deepfire.github.io

Role sought

⇒ Technologist
⇒ Architect / team lead
⇒ Senior software engineer

System building

⇒ a SoC-targeted system software development and assurance toolchain (IEEE 1149.1)
⇒ two transpilers (one used in critical production environment)
⇒ a package manager and a proto-CI system for a diverse environment (personal project)
⇒ a hypervisor-based secure endpoint (prototype); the hypervisor was written from scratch

High-level declarative design & programming

⇒ pure, typed, functional: *to support program reasoning*, refactoring and assurance
  • Haskell (*expressive* higher-kinded & dependent types, reactive (FRP), lazy evaluation)
⇒ metaprogramming: expanding ability to express solutions to very complex problems
  • Common Lisp (an *extensible* compiler)

Program semantics, compilation and analysis

⇒ written two transpilers, an assembler/disassembler and a control flow analysis tool
⇒ had a passing interest in flow analysis (CFA/DFA) of higher-order programming languages

Mainstream

⇒ mid-level POSIX programming
  • debugging sockets, threads, profiling, zero-copy (going fast), IPC, conventional GUI (gtk2)
⇒ low-level hardware programming
  • debugging C, x86-64, MIPS, Linux kernel, bare-metal hypervisor with `printf()`, gdb, JTAG

Miscellaneous

⇒ entry-level OpenGL and shaders
⇒ TeX / LaTeX / TikZ, some Web (front/back)
⇒ expert-level Linux administration & trouble-shooting
⇒ knowledge visualisation and interaction systems
  • *this has been my long-time fascination*

Education

2000-2001 **Engineering institute**
National Research University of Electronic Technology, Moscow

2002-2006 **Business school**
Institute of International Business Education, Moscow

Public projects

2017 **undisclosed project**
a knowledge representation and visualisation tool. Don't feel like talking about it yet

2017 **reflex-glfw**
a library facilitating use of Reflex FRP with OpenGL/GLFW

2017 **Barrelfish OS contributions**
Nix-based build environment reproducibility (merged upstream), stack traces (work in progress)

2015-ongoing **Contributions to Nixpkgs**
packaging software I need for the NixOS Linux distribution/package repository

2015 **Ruin**
a heterogenous, declarative build system: when your build is too twisted with conventional tools

2014-2016 **weld, youtrack, org-magit-review, some unreleased)**
tools for git and project management

2013 **cl-org-mode**
a suite of parsers/serialisers for org-mode

2011-2013 **partus**
a transpiler of a subset of Common Lisp to Python3

2008-2011 **executor, gittage, desire**
a suite of libraries culminating in a git-based distributed software delivery and automated testing system (that never really took off)

Work experience

September 2014 - now (2 years 5 months)
Positive Technologies
Department of virtualisation, head

Leading development of a hypervisor-based secure endpoint prototype:

⇒ Managing a diverse team of up to 13 members, mostly researchy-kind of people

⇒ Leading the design and architecture effort

- security architecture, interdomain communication
- facilitation of consensus in a heavily democratically-slanted context
- too much conflict management

⇒ Implementation all across the board: hypervisor, userspace and tooling

⇒ Organised further infrastructure development: build system, testing automation & continuous integration

- three build systems, one culminating in an open source project (building a deliverable package including hypervisor, kernel drivers, OS services and userspace is a non-trivial task): Ruin
- guiding deployment of Nix and Docker as means for reproducible builds in a precisely specified environment

⇒ Resource allocation and planning, hiring

⇒ Talking to sales people

⇒ Making presentations for external consumption

⇒ Developed an administrative process, to facilitate staged, planned materialisation of a high-level project vision. Implementation of this process was ultimately unsuccessful

⇒ Personal decision to end the project

Research direction:

⇒ Organised research into Intel Management Engine: threats, deactivation methods. This research culminated in a deactivation tool and a conference talk.

⇒ Organised a research survey on the kernels suitable as basis for the next product iteration.

⇒ Produced a preliminary design of a next-generation hypervisor-based secure endpoint system based of the Barrelfish OS.

⇒ Produced a research survey on the state-of-art in security kernels:

- origin of security kernels
- fundamental problem of security policy enforceability
- separation kernels
- state of art in verified kernels

January 2013 - August 2014 (1 year 8 months)
Positive Technologies
Department of virtualisation, team lead

Spear-headed development of a hypervisor-based endpoint prototype for consumer x86-64-based hardware:
...think consumer-friendly Qubes OS

⇒ overall architecture

⇒ build system & testing automation

⇒ general implementation

- memory management
- interdomain communication

⇒ code repository maintenance

⇒ managing a growing team

January 2012 - December 2012 (1 year)
Positive Technologies
Department of advanced development, Senior Developer / Analyst

⇒ Supported further deployment of the new system, through applying first-hand experience of developing a couple of forensics analysis modules within the new framework:

- a fast regex on steroids
- analysis of the Windows eventlog event streams and correlation heuristics for suspicious patterns

⇒ Analysis of usage practices and shortcomings of in-house knowledge base development infrastructure.

⇒ Early research on the viability of a secure endpoint based on a virtualisation-enforced isolation. Transformation of the management's high-level concept of such an endpoint into a technical vision.

October 2010 - December 2011 (1 year 2 months)

Positive Technologies

Senior developer

⇒ Produced a detailed (HyperSpec-style) reference specification for semantics of an in-house ad-hoc dataflow language (including relevant parts of its runtime system) used to capture domain-specific knowledge used by the flagship company product.

⇒ In collaboration with in-house domain experts, captured the design requirements for a next generation of the dataflow language.

⇒ Designed alternate, Python3-based syntax & semantics for the dataflow language. Implemented a runtime system for these semantics.

⇒ Designed and implemented a transpiler (inter-language compiler) (in Common Lisp) from the original ad-hoc dataflow language to the new Python semantics. The transpiler included a measure of simple static analysis and helped catching a number of bugs in the knowledge base.

⇒ Built an online compilation service, to facilitate smooth transition of the constantly evolving knowledge database.

⇒ Oversaw a successful transition of the entire knowledge base from the old system to the new language & runtime.

October 2003 - September 2010 (7 years)

Elvees

Developer

⇒ Maintenance of a Linux kernel port to the in-house Elvees Multicore series of SoCs. Linux kernel driver development (NICs, custom protocol serial interlink, DSP access device).

⇒ Development and maintainership of a pre-existing JTAG access toolstack used to facilitate both chip validation (in-house engineers) and software development (both in-house and external). The toolstack consisted of a portable (Windows, Linux) low-level JTAG TAP access library, a portable console-based debugger and a Windows IDE plugin.

⇒ Developed a series of binary analysis tools for the Multicore platform:

- a library for declaratively-specified assemblers and disassemblers, and its mips32 instance: assem. Attempts of its extension to x86-64 ultimately failed.
- a library for declaratively-specified parsers: bin-type
- a declaratively-specified ELF parser: cl-io-elf
- a MIPS binary analysis library and application used to employ flow-sensitive analysis to search application binaries for instruction traces with particular properties, that were found to be problematic on certain company CPUs: turing

  in-house version of the tool included a McCLIM-based GUI, facilitating interactive search and visualisation of problematic subsequences in the basic block graph

⇒ Developed an alternate JTAG toolstack, that was ultimately abandoned:

- a library for declarative description of register format/sets as well as devices and their hierarchies. Pro: a single, human-readable piece of text facilitating both register accessor code generation, validation and documentation purposes. The library supported partial validation of device / register / field / value usage correctness at compile-time: bitmop
- extensions and a port of a Common Lisp GDB stub library by Julian Stecklina: gdb-remote
- a tool for high-speed flashing of JTAG target devices, based on a combination of host-target bulk transfer and a code generator producing a platform/flash-chip-specific flashing routine on the target
- a programmable debugger substrate, based on the above: common-db
- a toolchain, facilitating automation of Linux kernel debugging experience, based on above

⇒ Consumer-ready packaging of the high-speed flashing tool: console UI, documentation and support request servicing.

⇒ Developed a customization in the GCC code generator to work around an FPU bug in a version of company CPU product

⇒ Helped to identify several CPU bugs: timing-sensitive cache/TLB interaction, bus access anomalies

Keywords

Languages	Haskell, Common Lisp, Python, C99
Haskell	type-driven design, FRP, FFI, making DSLs with Template Haskell, higher-kinded types, existentials, light dependent types (type families, GADTs, data kinds), exploiting laziness, shell-like programming; dabbling with: STM, free (and freer) monads; excited about: linear types, dependent types
Common Lisp	DSL design, macros, deep exploitation of staging, monadic parsing, FFI, GUI, low-level hardware access
Virtualization	x86-64 platform, VMx, EPT, VT-d, interrupt virtualisation, PCI device passthrough, CPU takeover, firmware hooking
OS kernels	Linux, Barrelfish
Bare metal	x86-64, mips32
Security	separation kernels, security policy enforceability, security modeling, attack surface analysis
Analysis	transpilers (language-to-language compilers), passing interest in control flow/data flow analysis (aka CFA/DFA)
High load	whole-system performance analysis, data path analysis, zero-copy programming, bcc, perf, strace, gprof, sar, iotop, blktrace, vmstat, slabtop, tcpdump
Dev tools	ghc, sbcl, gcc, clang, valgrind et al., GDB, VOGL, make, shell, git, emacs, intero, slime, git-svn, Nix
Devops	Travis CI, phabricator, gerrit, gitlab, github, NixOS, docker, personal projects
OS admin	NixOS, Debian, Fedora, CentOS, Nix, selinux, systemd, postgresql, qmail, tinydns, iptables, OpenVZ, docker, OpenVPN

Keywords

Management	org mode, Taskjuggler, YouTrack, yEd, VUE
Typesetting	TeX, LaTeX, TikZ
UX (GUI)	any haskell FRP GUI library I'll likely be comfortable with, McCLIM, gtk; likely can do Qt without much of a problem
OpenGL	legacy GL with display lists, LambdaCube3d (purely functional GPU pipeline programming), VOGL (Valve OpenGL debugger); generally find interactive visualisation fascinating
Web	html, css, jquery; excited about: GHCJS, WebAssembly, TypeScript, Elm
Research	extended experience reading scientific publications on a variety of topics: programming language theory, type theory, compiler internals, hypervisor & OS implementation, vulnerability exploitation, computer security in general

Relocation

Possible, but not before Q3/Q4 2017.

Communication skills

Russian	Native speaker
English	Oral: fair – Written: good
French	Stale, barely functional, but used to have good pronounciation : -)

Non-work stuff I enjoy

⇒ running

⇒ fasting

⇒ mountains of all kinds: hiking, alpinism, rock climbing (in past, sadly..)