Guarding Serverless Applications with Kalium

Deepak Sirone Jegan, *University of Wisconsin-Madison;*
Liang Wang, *Princeton University;* Siddhant Bhagat, *Microsoft;*
Michael Swift, *University of Wisconsin-Madison*

https://www.usenix.org/conference/usenixsecurity23/presentation/jegan

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

USENIX'23 Artifact Appendix: Guarding Serverless Applications with Kalium

Deepak Sirone Jegan
University of Wisconsin-Madison
dsirone@cs.wisc.edu

Liang Wang
Princeton University
lw19@princeton.edu

Siddhant Bhagat
Microsoft
sbhagat3@wisc.edu

Michael Swift
University of Wisconsin-Madison
swift@cs.wisc.edu

A Artifact Appendix

This artifact appendix is meant to be a self-contained document which describes a roadmap for the evaluation of Kalium.

A.1 Abstract

As an emerging application paradigm, serverless computing attracts attention from more and more adversaries. Unfortunately, security tools for conventional web applications cannot be easily ported to serverless computing due to its distributed nature, and existing serverless security solutions focus on enforcing user specified information flow policies which are unable to detect the manipulation of the order of functions in application control flow paths. In this paper, we present *Kalium*, an extensible security framework that leverages local function state and global application state to enforce control-flow integrity (CFI) in serverless applications. We evaluate the performance overhead and security of Kalium using realistic open-source applications; our results show that Kalium mitigates several classes of attacks with relatively low performance overhead and outperforms the state-of-the-art serverless information flow protection systems.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Kubernetes installation requires root privileges on all the Kubernetes nodes and certain ports need to be exposed for the OpenFaas installation. The Kalium controller node requires a TLS issued by Let's Encrypt. All the nodes need to be able to communicate with each other through a DNS name (possibly public to the Internet).

A.2.2 How to access

The artifact including its sub-repositories can be found here: https://github.com/multifacet/kalium_artifact/tree/83110fcfd091d9f8bd164007b1570742e0ad107c

A.2.3 Hardware dependencies

Our testbed uses machines with an Intel Xeon E5-2630 2.40GHz CPU and 64 GB RAM on CloudLab. Each machine is connected to a star topology LAN network with a speed of 25 Gbps. The Kalium controller runs on a separate identical node outside the LAN but on the same datacenter.

Any machines of comparable specifications, connected in a LAN of uniform speed maybe used for the Kubernetes nodes. The Kalium controller should run on the same datacenter to minimize noise as opposed to running it elsewhere on the internet.

All the machines need to be addressable from each other with a hostname. The Kalium controller doubles up as an image server that needs a certificate issued by Let's Encrypt, this mandates that the hostname of the Kalium controller should be visible to the internet.

A.2.4 Software dependencies

All the Kubernetes nodes and the Kalium controller have been testing using Ubuntu 18.04 LTS. We highly recommend using Ubuntu 18.04 LTS for all machines.

The build machine needs docker installed for the build process. Please install docker as per the build machine's distro's instructions https://docs.docker.com/engine/install/. Please do not install docker on any of the Kubernetes nodes or the Kalium controller.

A.2.5 Benchmarks

All the required benchmarks and data is packaged in the artifact.

A.3 Set-Up

[Mandatory] This section should include all the installation and configuration steps required to prepare the environment to be used for the evaluation of your artifact.

Provision 5 machines for running Kubernetes and 1 machine for the Kalium controller as specified in A.2.3 and A.2.4. A possible configuration is:

- node0: Kubernetes Worker
- node1: Kubernetes Worker
- node2: Kubernetes Worker
- node3: Kubernetes Worker
- node4: Kubernetes Controller
- node5: Kalium Controller

Root access is assumed on all the Kubernetes node as well as the Kalium controller node. Kubernetes requires port 6443 for the service API exposed on all the nodes while OpenFaas needs port 31112 for the gateway. The Kalium Controller listens at port 5000 and needs to be exposed to all the Kubernetes nodes. The Image Server listens at port 4443 and needs to be exposed to all the Kubernetes nodes.

Provision a machine for building the artifact binaries. This system should *not* be one of the Kubernetes nodes or the Kalium controller. Clone the artifact repository into the build machine using `git clone https://github.com/multifacet/kalium_artifact && cd kalium_artifact && git submodule update -init -recursive && git checkout 83110fcfd091d9f8bd164007b1570742e0ad107c`.

Please follow `kalium-benchmarks/README.md` to obtain a TLS certificate issued by Let's Encrypt for the Kalium Controller node.

A.3.1 Installation

Please follow the steps in README.md except "Running Benchmarks". By this time, the build machine should have a `build/bin` folder that contains the various artifact binaries, Kubernetes and OpenFaas should be setup in the cluster.

A.3.2 Basic Test

Please refer to README.md for detailed steps to do a basic test.

A.4 Evaluation workflow

A.4.1 Major Claims

The main claims validated by this artifact are related to the main claim of Kalium achieving comparable performance to Valve and Trapeze and being a usable solution due to its low system call overhead. As noted in Section 5.1.1, the semi-automated policy generation from existing applications is not a major contribution of the paper as a lot of the analysis was done manually. Our paper mainly focuses on defining control flow integrity for serverless applications, its challenges and enforcing the same with low overhead.

(C1): *Kalium achieves comparable performance as the state of the art information flow systems Valve and Trapeze. This is proven by the relative latency overhead experiment described in Section 7.4 of the paper whose results are illustrated in Figure 8.*

(C2): *Kalium achieves tolerable per system call overhead of the order of a few milliseconds in the worst case. This is proven by the per-syscall measurements in Section 7.4*

A.4.2 Experiments

All the experiments have been described in detail in the README.md file provided with the kalium-benchmarks sub repository. We omit repeating all the steps here for brevity.

(E1): *Valve Benchmarks: Run the Valve Benchmarks with stock gVisor and Kalium to generate Figure 8 in the paper. The figure shows the relative overheads of Kalium, Valve and Trapeze with respect to stock gVisor baseline. This validates claim C1*

How to: Run steps 1-3 in kalium-benchmarks/README.md

Preparation: Run steps 1-2 kalium-benchmarks/README.md

Execution: Run step 3 in kalium-benchmarks/README.md

Results: Run step 5 in kalium-benchmarks/README.md to generate Figure 8. The graph should show that Kalium has comparable overhead as Valve and Trapeze.

(E2): *Per Syscall Overhead: Run a microbenchmark function to generate per system-call overheads in Kalium. This validates claim C2*

How to: Run steps 4 and 7 in kalium-benchmarks/README.md

Preparation: Run steps 1-2 kalium-benchmarks/README.md only if it has not been run yet

Execution: Run step 4 in kalium-benchmarks/README.md

Results: Run step 7 in kalium-benchmarks/README.md to print out the per-syscall (SendMsg and Write) overheads which include (i) parsing TLS records (ii) TLS record cache lookup (iii) Event construction time (iv) Guard Local Graph Lookup (v) Controller Query Total Time and (vi) Total Syscall Overheads. The overheads should be comparable to that in Section 7.4 in the paper. The total syscall overhead

should be of the order of a few milliseconds in the worst case.