Valentin **Manès**

Contact
valentinmanes@outlook.fr

Profile
Software Engineer

Website
jiliac.com

Links
@Jilyac
Jiliac
valentinmanes

Programming
Go, **C/C++**
Java, Python, julia

Languages
French: Mother Tongue
English: Near Native
Spanish: Intermediate
Korean: Basic

Interests
Card Games
Languages
Books
Travel

Experience

2020 **PacketAI** Paris, France

PacketAI aims to *predict* cloud infrastructure incidents and identify their root cause. I was developing the agent, a software running on client hosts collecting events and metrics, and sending them to PacketAI servers via **Kafka** pipes. I was also in charge of the microservices, developed in **Go**, treating this data.

- PacketAI product is based on the **ELK** stack: The **Beats** to produce data and **Logstash** to transform and forward it to ElasticSearch.
- Many tools or test environments are deployed with `docker-compose`. I was involved in the development of the **CI/CD** pipelines of our Go projects on GitLab.
- Communication between microservices using **REST APIs**.
- **Scrum** method used based on Trello and GitLab.
- **Mentored** the integration of an intern to the team.

2016-19 **Cyber Security Research Center - KAIST** Daejeon, South Korea

I first worked on developing a kernel hardening solution by limiting the kernel attack surface. Then, I reoriented myself towards Automatic Software Testing (also called fuzzing). Fuzzers repeatedly run a program with generated inputs with the intent of finding misbehavior in softwares.

- At CSRC collaboration was done using **Slack** and **Gitlab**. Most notably our survey involved seven members. All contributions were made via merge requests.
- Experiments setup in **Docker** containers to be reproducible and scalable to multiple servers. Command-line tools are invaluable: `htop`, `grep`, `find` etc...
- Ankou (described below), I started as an investigation on the usage of machine learning techniques to improve fuzzers bug finding ability. For this, standard python libraries were used: **Keras**, **TensorFlow**, **Numpy**, **Pandas**. The two parts of the project, in Go and in Python, were communicating via **RabbitMQ**.

Education

2015-16 **KAIST - Exchange** Daejeon, South Korea

KAIST was a very different studying environment than I was used to: more centered around research. In particular, I focused on kernel hardening techniques and software security.

2013-16 **Telecom ParisTech - Master's degree** Paris, France

Telecom ParisTech is one of France's top five graduate science schools (*grandes écoles*), and is considered the leading French school in Information and Communication Technology. I specialized in Information Security.

2011-13 **Lakanal - Preparatory School** Sceaux, France

2006-11 **Lycée Franco-Méxicain** Mexico City, Mexico

Publications

2020 **Boosting Fuzzer Efficiency: An Information Theoretic Perspective**
Foundations of Software Engineering (Second Author)
Code: github.com/llvm/llvm-project/commit/e2e38fca
Entropic is an information-theoretic power schedule implemented based on LibFuzzer. It boosts performance by changing weights assigned to the seeds in the corpus. Seeds revealing more ”information’’ are assigned a higher weight. Entropic has been independently evaluated by a team at Google and invited for integration into mainline LibFuzzer @ LLVM (C++ code base), whereupon Entropic was subject to a substantial code reviewing process.

2020 **Ankou: Guiding Grey-box Fuzzing towards Combinatorial Difference**
International Conference on Software Engineering
Code: github.com/SoftSec-KAIST/ankou
Grey-box fuzzing search process is not expressive enough because it does not take *combinations* of software features into account. We propose a way to account for combinations. However, it is too computationally expensive, thus we reduce the dimensionality of the problem via a modified version of the Principal Component Analysis. This was a large engineering project: 15K lines of Go.

2019 **The Art, Science, and Engineering of Fuzzing: A Survey**
IEEE Transaction on Software Engineering
Companion website: fuzzing-survey.org
This survey presents a unified, general-purpose model. By identifying the key algorithmic stages of fuzzers, we could effectively summarize the literature.

2018 **Domain Isolated Kernel**
Elsevier Computer & Security
Code: github.com/Jiliac/DIKernel
Kernel extensions (i.e. drivers) are the weakest kernel part security-wise. DIKernel isolates extensions by lowering their memory access permission and their execution privilege. We keep our solution convenient for both the end-users, by ensuring a low-performance cost, and developers, by not requiring any change in the code of extensions. DIKernel was implemented on top of Linux 4.13 kernel with 1.5K lines of C.