Jean-Yves Marion: How to recover self-modifying codes?
Most of malware are self-modifying programs. A run is a sequence of code waves where each wave is generated by the previous one. As a result, a static analysis fails to find the payload of a malware because it is hidden in one wave or split in several waves. In order to recover a self-modifying code, the first goal is to reconstruct the set of waves and the relationships between waves. Then, the second goal is to find the code in a given wave, that is to disassemble a wave. This task is difficult with X86 and it becomes very arduous when the code is obfuscated.

Sergio Maffeis: Formal, executable semantics of web languages: JavaScript and PHP
In order to be of any practical significance, rigorous analyses of security properties of web applications must be grounded on precise and comprehensive formal models of the technologies involved. In this talk, I will share my experience in developing the formal semantics of JavaScript and PHP. In particular, I will discuss some of the challenges encountered in handling pre-existing systems complexity, developing and using semantics at scale, testing, analyzing, and proving properties of *real* programming languages.
Roberto Giacobazzi: BinJuice Automata
We introduce the notion of BinJuice Automata as a specialization of symbolic automata with semantic representations of the input/output relations of basic-blocks in control-flow graphs of binary code. It is known that Symbolic Automata extend classical automata by using symbolic alphabets instead of finite ones. In our case the symbolic representation is extracted as an abstract interpretation of a sequential and contiguous fragment of a disassembled binary code. We apply BinJuice automata to malware analysis.

Federico Maggi
The spread of malicious or otherwise unwanted Android apps (a.k.a., Android malware) has been rising since late 2010. This includes, for instance, ad-aggressive apps, information stealers, premium dialers or texters, and variations of the above. The driving motivation for the malicious developer is direct or indirect financial gain. As of today, according to vendors and researchers, hundreds of thousands Android malware samples are found in alternative and, sometimes, official markets. We lack quantitative data about how much revenue the malicious developers and distributors can make. In parallel, hundreds of paid and free security apps for Android (a.k.a., Android anti-virus or anti-malware apps) landed on the market. In this talk, I will glance the basics of the Android security model, to understand how malware violates it in order to perform malicious actions. Moreover, we will understand why this security model, not really changed in KitKat, still prevent security software to properly monitor an Android device. As a consequence, we will see how these limitations allow malware developers or distributors to evade security checks in simple ways. Then, I will give an overview of the most famous malware families known since 2010, and present the AndroTotal data-collection project that we started early year at Politecnico di Milano, to provide an open repository of malware for research. I will conclude the presentation discussing, hopefully interactively, the controversial aspect of actual infection rate vs. predicted infection rate vs. amount of Android malware spotted in the wild.

Marco Cova: Hunting web-based malware
Web-based malware is malicious web pages or other web-distributed content designed to exploit vulnerabilities in the systems used by their users and to take control of their devices. Web-based malware has been evolving at a fast pace in the last few years. New techniques are continuously introduced to foil analysis, prevent the extraction of malicious behavior, and impede its identification. For example, cybercriminals routinely tweak their malicious web content to create new and more effective variants (for example, by
incorporating exploits targeting newly-discovered vulnerabilities) or to evade commonly-used defensive tools. In addition, the malware is increasingly becoming more stealthy and environment-aware. In this talk, I will describe our research on finding, detecting, and analyzing web-based malware. We will focus on the techniques we have used to efficiently identify malicious content online, to analyze it at large scale, and to detect attempts to bypass analysis environments.

Lorenzo Cavallaro: CopperDroid
Today mobile devices and their application marketplaces drive the entire economy of the mobile landscape. For instance, Android platforms alone have produced staggering revenues exceeding 9 billion USD, which unfortunately attracts cybercriminals with malware now hitting the Android markets at an alarmingly rising pace. To better understand this slew of threats, in this talk I present CopperDroid, an automatic VMI-based dynamic analysis system to reconstruct the behavior of Android malware. Based on the key observation that all interesting behaviors are eventually expressed through system calls, CopperDroid presents a novel unified analysis able to capture both low-level OS-specific and high-level Android-specific behaviors. To this end, CopperDroid presents an automatic system call-centric analysis that faithfully reconstructs events of interests, including IPC and RPC interactions and complex Android objects, to describe the behavior of Android malware regardless of whether it is initiated from Java or native code execution. CopperDroid's analysis generates detailed behavioral profiles that abstract a large stream of low-level---sometimes uninteresting---events into concise high-level semantics, which are well-suited to provide effective insights. Extensive evaluation on more than 2,900 Android malware samples, show that CopperDroid faithfully describes OS- and Android-specific behaviors and, through the use of a simple yet effective app stimulation technique, successfully triggers and discloses additional behaviors on more than 60% (on average) of the analyzed malware samples, qualitatively improving code coverage of dynamic-based analyses.