SAD: building software moving targets to prevent massive attacks on distributed applications

Context and scope of the project

Don't reinvent the wheel! All software developers adopt this motto and massively reuse code all over the software stack. For example, the Wordpress content management system is massively reused to build web sites worldwide: it is used by 24% of the 500000 most popular web sites [1], according to our observations in the Spring of 2014 [1].

High gains: From the very infancy of software engineering software reuse has been an essential practice to handle the complexity of the software applications that surround us.

High risks: Yet, reuse in the era of the Internet has a darker side: the exact same malware can be used to infect the exact clones of a software component, which are deployed on millions of interconnected devices This phenomenon of "software monoculture" was coined more than a decade ago to highlight the risks of using a handful of operating systems and databases [11]. In the recent months, we have seen spectacular exploitations of such vulnerabilities to create botnets of video cameras or to distribute ransomware at large.

The main objective of this SAD project is to build a platform to **automatically synthesize massive quantities of variants of client-server software components, to create a moving target against the large scale exploitation of vulnerabilities**.

Novelty with respect to state of the art and other projects

Our recent survey [2] shows that most of automatic software diversification occurs at the operating system (OS) level. The seminal work of Forrest [3] established the foundations of OS diversity for security and protection purposes. Subsequent main works include address space layout randomization (ASLR) [8], NOP insertion [4] and moving target defenses [5], which introduce temporal diversity (switch from one variant to another at runtime). If these works do not identify the concept of rigidity, they address very similar concerns in OSs. For example, before ASLR, system libraries were always loaded at the same address in memory, providing a precious piece of information for code injection. Nowadays, ASLR protects all major OSs.

Our project will advance software diversification beyond assembly code, to face the emergence of applicative monoculture. This requires handling richer languages (e.g., applications are built with many languages, while diversity in OS is all based on X86) and investigating new forms of code transformations. Today, there exist few transformations that diversify application code. These works include Rinard et al.'s loop perforation [10], Schulte's novel observations about mutational robustness [7], and my work on multi-level diversification of web applications [1]. Our project will contribute to the area of application-code diversification, with a strong emphasis on the improvement of safety, security and adaptivity of applications.

[1] According to `http://www.alexa.com/topsites`

Expected novelty

- Radically new methods and language support to increase the diversity of software applications that run on a large number of interconnected and heterogeneous devices
- Tool support to accompany the software developers in the construction and maintenance of a large number of software variants
- The development and deployment of real-world case studies to demonstrate the increased level of safety and security of software equipped with the ability to diversify.

Missions

Diversity measurement The quantification of software diversity and its impact on security is a key challenge in this project. The student will be in charge of defining and testing metrics to measure diversity.

Experiments The definition and scientific assessment of diversification technologies heavily rely on empirical investigations. The student will be in charge of setting up benchmarks and sound experiments.

Software development The DiverSE team currently develops several components that will be at the core of the moving target framework [2] [3]. The student will leverage these techniques to build an integrated framework for automatic software diversification and reconfiguration.

References

[1] Simon Allier, Olivier Barais, Benoit Baudry, Johann Bourcier, Erwan Daubert, Franck Fleurey, Martin Monperrus, Hui Song, and Maxime Tricoire. Multi-tier diversification in web-based software applications. *IEEE Software*, 32(1):83–90, Jan 2015.

[2] Benoit Baudry and Martin Monperrus. The Multiple Facets of Software Diversity: Recent Developments in Year 2000 and Beyond. Technical report, 2015.

[3] Stephanie Forrest, Anil Somayaji, and David Ackley. Building diverse computer systems. In *Proc. of HotOS*, pages 67–72, 1997.

[4] Andrei Homescu, Stefan Brunthaler, Per Larsen, and Michael Franz. Librando: transparent code randomization for just-in-time compilers. In *Proc. of CCS'13*, pages 993–1004, 2013.

[5] Hamed Okhravi, Thomas Hobson, David Bigelow, and William Streilein. Finding focus in the blur of moving-target techniques. *IEEE Security & Privacy*, 12(2):16–26, Mar 2014.

[6] Martin Rinard. Obtaining and reasoning about good enough software. In *Proc. of DAC*, pages 930–935, 2012.

[7] Eric Schulte, Zachary Fry, Ethan Fast, Westley Weimer, and Stephanie Forrest. Software mutational robustness. *Genetic Programming and Evolvable Machines*, pages 1–32, 2013.

[8] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh. On the effectiveness of address-space randomization. In *Proc. of CCS*, pages 298–307, 2004.

[9] Mary Shaw. Self-healing: softening precision to avoid brittleness: position paper for woss'02: workshop on self-healing systems. In *Proceedings of the first workshop on Self-healing systems*, pages 111–114. ACM, 2002.

[10] Stelios Sidiroglou-Douskos, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. Managing performance vs. accuracy trade-offs with loop perforation. In *Proc. of ESEC/FSE*, pages 124–134, 2011.

[2] `https://github.com/DIVERSIFY-project/sosiefier`

[3] `http://kevoree.org/`

[11] Mark Stamp. Risks of monoculture. *Comm. of the ACM*, 47(3):120, 2004.

[12] Gerald Jay Sussman. Building robust systems an essay. *Citeseer*, 2007.