## C35 Fuzzing Memory Forensics Frameworks

*Arian Dokht Shahmirza, BSc*, Louisiana State University, Baton Rouge, LA 70808; Aisha Ali-Gombe, PhD, Towson University, Towson, MD 21252; Andrew Case, MS, Louisiana State University, Baton Rouge, LA; Golden G. Richard III, PhD, Louisiana State University, Baton Rouge, LA 70808*

**Learning Overview:** Fuzz testing memory forensics frameworks can reveal important vulnerabilities that are crucial in the effort to create robust forensics tools. After attending this presentation, attendees will be familiar with fuzzing techniques that are applicable to testing memory forensics frameworks, the differences between popular memory forensics frameworks and other tools, and the impacts of said differences on adopting the correct fuzzing techniques. Furthermore, due to the fact that memory forensics frameworks utilize relatively large memory images, fuzz testing these frameworks requires optimizations both in fuzzing and in parallelizing the tester. Attendees will learn additional distribution techniques and how significantly they affect the efficiency of fuzz testing.

**Impact on the Forensic Science Community:** This presentation will impact the forensics science community by discussing an open-source fuzzing tool that can test memory forensics frameworks and uncover issues that may potentially lead to inaccurate forensic results by causing the memory forensics tools to crash or behave unexpectedly. Consequently, the frameworks will become more robust by solving these issues, and the quality and efficiency of memory forensics as a whole will be improved.

Memory forensics is on a path to automation and its tools need to be more robust. Volatility is a popular framework used regularly in investigations, and in order to properly implement automated analysis, the ability to discover critical vulnerabilities that might arise must be included. Fuzz testing is a technique in which the program is intentionally fed flawed data to discover whether it enters an unexpected state. Due to the nature of these images, their mutations are tailored to the frameworks. The memory forensics frameworks are written in Python and are provided with one image file. The preferred fuzzing technique in terms of performance and efficiency is intercepting system calls using LD_PRELOAD. This also provides the fuzzer the portability and flexibility to test other programs.

Fuzzing involves thousands of mutations on a single file and testing various plugins of the memory forensics framework on those mutations. This requires copious amounts of computing time and energy. Because of this, the tests were efficiently distributed using HPX. HPX is a general purpose C++ runtime system for parallel and distributed applications of any scale, and it has the capability of being adjusted to parallelize computation-heavy applications such as fuzzer.[1] HPX futures, for example, provide the possibility to set up work to be performed, fire it off, and wait for it to be finished. HPX takes care of creating the threads, moving the work across node boundaries, and making sure the calling thread suspends when it wants the value from a future that is still executing, provided that for any function wrapped with a future, any parameters of those functions are serializable.

The results of this research are presented in two aspects: the performance of the fuzzer and how efficiently it can test a memory forensics framework, and its findings during the tests. These findings will demonstrate how effective the fuzzer can be in memory forensics and in helping improve the frameworks, hence, making forensics more reliable and efficient. This fuzzer can be useful for other researchers to test other memory forensics frameworks and to help eliminate defects from them.

**Reference(s):**

1. http://stellar.cct.lsu.edu/projects/hpx/.

**Memory Forensics, High Performance Computing, Fuzzing**