Review “AEG”

Name: Wenhui Zhang

Paper Name：

Avgerinos, T., Cha, S. K., Rebert, A., Schwartz, E. J., Woo, M., & Brumley, D. (2014). Automatic exploit generation. *Communications of the ACM*, *57*(2), 74-84.

Contribution:

This paper proposes a new technique for end-to-end automatic exploit generation (AEG) on real world programs. This technique is specifically for control flow hijack attacks.

Motivation:

This paper is trying to demonstrate automatic exploit generation. That is, given a program, we could find bugs and demonstrate exploitability in an automatic way. There are some challenges regarding to automatic exploit generation. For example, there is a bug like “length(input) > sizeof(ifr_name)” in the program. Maybe it is introduced by ifr_name overflowing the return addresses on the stack. Static analysis will tell that this is an exploitable bug, due to one line of code. However, in reality, it might not be exploitable, and prohibited due to boundary checks later on. The traditional method is slow.

Related works:

Traditional symbolic execution covers all paths, which is slow to find bugs, such as KLEE [Cadar’08]. It generate inputs achieving high code coverage, however AGE focuses on exploitable paths. There are some hand made tools [Medeiros et al, Toorcon’07], however there is no experiments or code mentioned in this paper. Brumley et al. [Oakland’08] does an automatic patches based exploit generation, it requires patch to point out bug and problem. Heelan et al. [MS Thesis’09] does automatic generation of control flow hijacking exploits, however it requires input that triggers exploitable bug, 1 real example is mentioned.

Methodology:

This paper firstly does vulnerability discovery using symbolic execution on source code. It then conduct exploit generation based on dynamic binary analysis. This work only explore buggy paths by conducting precondition symbolic execution, thus it is a fast approach. It does a precondition check before moving in depth to the paths. The length is selected through lightweight static analysis. It uses the size of the largest statically allocated buffer. After finding the buggy path, it uses dynamic binary analysis to test exploitability of buggy path. In dynamic binary analysis part, it uses satisfiability modulo theories (SMT) solver for generating the input combinations.

Results:

This method allows AEG to fully automatically detect 10 of the 16 exploits. They also compare the performance between real humans and AEG. For the same set of bubby programs, it takes 4 hours for students in software security class with exploit generation experiences to generate iwconfig exploit. They analyzed 14 applications or 3 hours and generated 16 working control flow hijack exploits. While it takes several minutes in general for AGE to deliver the exploits. AEG is used in smpCTF, and it takes down questions in less than 10 mins.

Take away:

Given exploitable bugs, AGE still could not guarantee that it will find exploits.