Vulnerability Discovery and Triage Automation Training

Richard Johnson

Overview

This class is designed to introduce students to the best tools and technology available for automating vulnerability discovery and crash triage with a focus on delivering a practical approach to applying this technology in real deployments at any scale.

Through an applied understanding of introductory program analysis and binary translation, techniques for finding various bug classes and methods for improved crash debugging will be discussed. We will analyze the properties of memory corruption from integer overflows, uninitialized variables, use-after-free and look at applying tools such as compiler plugins, binary instrumentation frameworks, memory debuggers, and fuzzers to discovering each one.

Next take a deep dive into fuzzing, covering all aspects of this practical approach to finding bugs. As the most approachable and versatile of the available tools, the student will apply various fuzzing techniques to several real-world pieces of software. Students will learn strategies for analyzing attack surface, writing grammars, and generating effective corpus. We will explore in detail the latest innovations such as harnessing code coverage for guided evolutionary fuzzing and symbolic reasoning for concolic fuzzing.

We approach crash analysis through the lens of scriptable debuggers and program analysis. We will once again look at properties of how memory corruption manifests in a crashing condition. We will apply tools like reverse debugging and memory debugging scripts to assist in interactively diagnosing root cause of crashes. Then we will leverage the power of dynamic taint tracking and graph slicing to help isolate the path of user controlled input in the program and identify the exact input bytes influencing a crash. Lastly, we will look at possible ways to aid in determining severity of a vulnerability.

This class will focus on x86/x64 architecture and target Windows and Linux environments, however some discussion regarding applications to ARM and mobile platforms will also be included and all of the concepts if not the direct tools will be useful in other environments.

Who Should Attend

This class is meant for professional developers or security researchers looking to add an automation component to their software security analysis. Students wanting to learn a programmatic and tool driven approach to analyzing software vulnerabilities and crash triage will benefit from this course.

Key Learning Objectives

- Learn an effective strategy for using the latest tools & technology to discover vulnerabilities
- Understand applications of static analysis for bug hunting
- Learn how to decompose programs and analyze them with powerful frameworks
- Learn how to write basic clang-analyzer plugins
- Introduction to intermediate languages for program analysis
- Introduction to graph search, transformation, and slicing
- Leverage dynamic binary translation for efficient tracing and deep program inspection
- Master the latest fuzzing techniques and strategies for file and network fuzzing
- Learn grammar fuzzing, evolutionary fuzzing, in-memory fuzzing, and symbolic fuzzing
- Best practices for corpus generation, fuzzer deployment, and targeting
- Apply powerful techniques like taint analysis and graph slicing towards crash analysis

Prerequisite Knowledge

Students should be prepared to tackle challenging and diverse subject matter and be comfortable writing functions in in C/C++ and python to complete exercises involving completing plugins for the discussed platforms. Attendees should have basic experience with debugging native x86/x64 memory corruption vulnerabilities on Linux or Windows.

Hardware / Software Requirements

Students should have the latest VMware Player, Workstation, or Fusion working on their machine

Agenda

Day 1: Program Analysis

Morning:

Strategies for automating vulnerability discovery
- Analyze properties of memory corruption vulnerabilities
- Understand capabilities and applications of program analysis

Programmatic analysis of C/C++ source code
- Introduction to advanced pattern matching and AST search
- Introduction to using dataflow analysis for bug detection
- Introduction to writing clang-analyzer plugins
- Exercises: writing clang analyzer plugins

Afternoon:

Dynamic memory analysis for blackbox bug hunting

- Effectively instrument Linux and Windows with binary translation
- Introduction to Valgrind and Dr. Memory
- Code coverage with PIN, DynamoRIO, and DynInst
- Introduction to dynamic taint analysis with BAP
- Exercises: code coverage, taint analysis, and property checks

Day 2: Fuzzing & Triage I

Morning:

- Attack surface analysis for blackbox vulnerability research
  - Enumerating trust boundaries
  - Enumerating inputs
  - Leveraging graph reachability analysis
  - Exercises: reachability analysis
- Best practices for generational and mutational fuzzing
  - Triggering vulnerabilities through fuzzing
  - Effective mutation engines
  - Effective corpus generation
  - Creating protocol and file format grammars
  - Crash detection
  - Exercises: generational fuzzing

Afternoon:

- Evolutionary fuzzing
  - Code coverage driven fuzzing
  - Modifying targets for optimal evolutionary fuzzing
  - Best practices for high performance
  - Exercises: evolutionary fuzzing binary and source targets
- Interactive and scripted crash debugging
  - Root cause identification for memory corruption bug classes
  - Memory debuggers and code coverage
  - Debugger plugins and scripting APIs
  - Reverse debugging
  - Exercises: debugger scripting

Day 3: Fuzzing & Triage II

Morning:

- Advanced fuzzing
  - Fuzzing kernels and other architectures with QEMU
  - Hybrid fuzzing with concolic execution
  - Browser and language interpreter fuzzing
  - Exercises: concolic, browser, and interpreter fuzzing

Afternoon:

Advanced crash analysis

- Dynamic taint analysis and graph slicing
- Forward symbolic execution for exploitability analysis
- Exercises: debugging with taint slicing and symbolic execution