Practical Fuzzing: A Hands-On Learning Experience for Uncovering Vulnerabilities on Linux and Windows Platforms

Introduction

Fuzzing is a powerful technique for identifying vulnerabilities in software. This hands-on training will cover the theory and practical aspects of fuzzing, including coverage-guided fuzzing, basic blocks and binary instrumentation, corpus collection and minimization, target selection, crash triage and root cause analysis, and real-life CVE analysis. Attendees will have the opportunity to practice fuzzing on Linux and windows platforms and apply the concepts and techniques learned in the training to fuzz real world software.

We will also see how can we customize WinAFL to add new features and suit our needs.

This training is suitable for attendees with a basic understanding of software development and testing and is beginner friendly.

This training will start from user mode fuzzing and later on covers topics like linux kernel fuzzing and firmware fuzzing etc.

In this training, attendees will also learn about the different types of vulnerabilities that can be found through fuzzing, including buffer overflows, heap overflows, integer overflows, use-after-free errors, and out-of-bounds read/write errors. We will discuss the underlying causes and potential impacts of these vulnerabilities, as well as how to identify and address them through fuzzing. In addition to coverage-guided fuzzing, we will also introduce other types of fuzzer, such as dumb fuzzers and mutation fuzzers, and discuss their benefits and limitations. Attendees will also learn how to use tools such as GDB, WinDBG and Crashwalk to debug and analyse crashes, and to perform root cause analysis to identify the underlying cause of vulnerabilities. We will also see how to debug crashes using time travel debugging on windows, measure code coverage, collect corpus and minimise them.

About the trainer

Hardik Shah (@hardik05) is an experienced cyber security professional with 19+ years of experience in the computer security industry. Currently works as a Principal Security Researcher at Vehere where he is responsible for analysing latest threats, detecting them and product improvements. In the past he has worked with various security companies like Sophos, McAfee, and Symantec, where he has built research teams from ground zero, managed various critical cyber threats to provide protection to customers, implemented various product features and has mentored many people.

Hardik is also known for his skills in fuzzing and vulnerability discovery and analysis. He has discovered 50+ vulnerabilities in Microsoft and various open source software. He had conducted workshops at various industry leading cyber security conferences such as Defcon, Bsides, RSA dark arts, and many others. Hardik enjoys analysing latest threats and figuring out ways to protect customers from them.

Day wise Training Agenda

Day 1

- Introduction
- Different types of vulnerabilities
    - Buffer overflow
    - heap overflow
    - integer overflow
    - use after free
    - out of bound read/Write
    - This will cover some real life vulnerability example as well.
- Hands on: Manually identifying the vulnerabilities in sample C code.
- What is fuzzing?
    - Fuzzing Process
    - Different types of fuzzer
    - dumb fuzzer
        - Example - radmasa
    - mutation fuzzer
        - Example - sulley
    - coverage guided fuzzer.
        - Examples - AFL, WinAFL, AFL++, libfuzzer, Honggfuzz
- Basic blocks and code coverage
- Binary instrumentation
- Corpus Managment
    - Corpus collection
    - Corpus minimization
- What is AFL and AFL++?
    - How does it works?
    - Fork server Vs persistent mode
    - How to write harness for persistent mode
    - Fuzzing Strategies
- Different Sanitizers
    - ASAN
    - UBSAN
    - MSAN
- Using AFL++
    - How to compile and install AFL++
    - How to compile Simple C program with AFL++
    - Various compilation options for AFL++
        - AFL_HARDEN, AFL_USE_ASAN,
        - AFL_DONT_OPTIMIZE etc.
    - Fuzzing Simple C program using AFL++
    - Using persistent mode to improve fuzzing speed
    - Using shmem mode to improve fuzzing speed
    - Fuzzing in Qemu Mode
    - Fuzzing Different Arch Binaries with Qemu
    - Using dictionaries to fuzz
    - Using CMPLog Feature to fuzz
    - Rewriting binaries with e9afl and fuzzing them with AFL++

- Fuzzing network binaries with AFL++

Day 2

- Recap of what we learned at day 1
- Root cause analysis and debugging using GDB
  - Debugging crashes using GDB
  - Finding root cause
  - Crash triaging using Crashwalk
  - How to install crashwalk
  - Using Cwtriage,Cwdump
  - How to use it to do automated crash triaging
- Fuzzing real world programs
  - Fuzzing TCPDump
    - Getting source code and dependencies
    - Compiling with AFL++
    - Collecting Corpus
    - Minimising Corpus
    - Fuzzing the program
    - Looking at issues found through fuzzing
  - Fuzzing libtiff
    - Getting source code and dependencies
    - Compiling with AFL++
    - Collecting Corpus
    - Minimising Corpus
    - Fuzzing the program
    - Looking at issues found through fuzzing
  - Fuzzing ImageMagick
    - Getting source code and dependencies
    - Compiling with AFL++
    - Collecting Corpus
    - Minimising Corpus
    - Fuzzing the program
    - Looking at issues found through fuzzing
  - Fuzzing FFMpeg
    - Getting source code and dependencies
    - Compiling with AFL++
    - Collecting Corpus
    - Minimising Corpus
    - Fuzzing the program
    - Looking at issues found through fuzzing
  - Fuzzing libEMF
    - Getting source code and dependencies
    - Compiling with AFL++
    - Collecting Corpus
    - Minimising Corpus
    - Fuzzing the program
    - Looking at issues found through fuzzing
  - Fuzzing libGD

- Getting source code and dependencies
- Compiling with AFL++
- Collecting Corpus
- Minimising Corpus
- Fuzzing the program
- Looking at issues found through fuzzing

- OSS-Fuzz introduction
  - How to set it up locally
  - How to build docker images and fuzzers
  - How to fuzz various Open Source Software with OSS-Fuzz

- Introduction to Firmware Fuzzing
  - How to extract firmware
  - How to Fuzz Software
- Introduction to Linux Kernel Fuzzing
  - How to setup syzkaller on system
  - Various tools syz-manger etc., dir structure, config files.
  - How to configure and build linux kernel for fuzzing
  - How to replicate old issue with corpus
  - How to fuzz a simple kernel module, write syzkaller descriptions, enabling/disabling syscalls
  - How to compile kernel and syzkaller with custom module and definitions
  - How to fuzz using syzkaller
  - How to replicate crashes
  - How to debug crashes with gdb and qemu

Day 3

- Fuzzing open source vs close source programs
- How does winafl works?
- Instrumentation with DynamoRIO
- Fuzzing Strategies
- Using WinAFL
  - Hands on: Compile Sample C program using Visual Studio.
  - Hands on: Find Offset of Fuzz Function.
  - Hands on: Run Winafl in debug mode to check everything is working fine.
  - Hands on: Fuzz using WinAFL.
  - Hands on: Analyzing the crashes and finding root cause.
- Fuzzing real world programs.
  - Write a harness/test program to read/parse MDB files.
  - Write a harness to fuzz EMF files and fuzzing it using winafl.
  - Write a harness to fuzz LNK files and fuzzing it using winafl.
- Checking code coverage using IDA, dynamoRIO, lighthouse
- WinAFL Customization
  - Modifying WinAFL to use Shared Memory Mode
  - Modifying WinAFL to get error code for the crashes
  - Adding Mopt Mutators, Power Schedulers
- Jackalope Fuzzer
- Debugging using windbg
  - Time Travel Debugging
- Question and Answers