Hacking 101
Stack Smashing

Pre-talk Fundamentals

Arjun Gopalakrishna

Fundamentals

What you need to know before the talk

Compiling & Linking

Chipset ISAs

Assembly Code

Stack (Layout, PUSH, POP)

Registers

Fuzzing

Debugging

Compiling & Linking

The recipe that converts ingredients to food

- Modern applications are written in a high-level language which cannot be understood by a computer.

- A computer only understands machine language (binary).

- How do we go from “English” source code to Binary machine language?

- A compiler converts source code to ‘object’ files in machine language, but this is not executable.

- A linker bundles the ‘object’ files into an executable that can be run on the computer

Chipset ISAs

All Mediterranean restaurants serve falafel.

- ISA = Instruction Set Architecture
- A single ISA (x86) specifies how code on all processors (Intel, AMD) for that architecture will run.
- This enables a cheaper Intel processor to run the same code as a high-end AMD processor. They are part of the same ISA, and hence compatible.
- To be compatible, they use the same ‘ingredients’ (registers, data-types etc.)

Assembly Language

Almost machine language, but not quite

- Computers only understand machine language, but that is very hard to write.
- Assembly language is the closest relative, and allows a developer to specify steps in very minute operations.
  - E.g. Move this number there. Add that to this other number.
- Assembly language depends on the processor being used.
  - The code to add two numbers for an Intel CPU is different than for ARM.

Stack (Layout, PUSH, POP)

A stack of plates.

- When you want to save ‘something’, you need to put it ‘somewhere’, potentially with other things with some organization.
- A Stack is a data-type that is a collection of elements that only allows access using 2 operations:
  - Put something on the top of the stack (PUSH)
  - Take something off the top of the stack (POP)

Registers

Cash money goes in a cash register

- Registers are specific locations to store information.
- x86 has 8 general-purpose registers.
    - The register we care about the most is the Instruction Pointer.
- The Instruction Pointer contains the memory location of the next instruction to be executed in code.
    - It essentially tells the computer to “do this thing next”.

Fuzzing

Ever tried vodka-battered onion rings?

- Most software is written with a ‘golden path’ in mind - this is what the user is expected to do.
    - What happens if there are deviations from this ‘golden path’?
- Fuzzing is the process of passing in ‘unexpected’ input to a system to see how it is handled.
    - We will ‘fuzz’ the application to see how it handles arbitrarily long inputs.
    - Based on how this is handled, we can find issues in implementation.

Debugging

Don’t you wish you could read people’s minds?

- When software is running, you usually only see and interact with the user interface.
  - What is happening ‘under the hood’/‘behind the scenes’?
- Debugging is a way of being able to get insight into code execution, by looking at the code being executed and being able to influence its execution.
  - Debugging helps in understanding a system when it is running, and is part of ‘dynamic analysis’.

Session Abstract

Exploiting an Out of Bounds Memory Write

You will leverage knowledge of the IA32 ISA to iteratively develop a working exploit that will result in Remote Code Execution in Windows, taking advantage of an Out of Bounds Memory bug.

Starting with fuzzed input, we will debug a running process to identify an OOB Write bug and develop a working exploit that will lead to arbitrary code execution on the PC.

You will leave the session with a deeper appreciation of hacking and exploit development.