Binary Code Analysis

Decomposition of an ELF binary file

Petr Holášek <pholasek@kerio.com>
Kerio Control team, Kerio Technologies

Lab exercises

- 5 x 1,5 hours
- odd Mondays 09:00-10:50, even Tuesdays 08:00-09:50 [FIT VUT, Q305]
- attendance is recommended
- will help you with classified homeworks

Useful sources

- Ljubuncic, I.: Linux Kernel Crash Book, 2011
- Drake, C., Brown, K.: Panic! UNIX System Crash Dump Analysis, Prentice Hall, 1995.
- Hofmann, F.: The Solaris Operating System on x86 Platforms, Crashdump Analysis, Operating System Internals, 2005
- Intel Corporation: Intel 64 and IA-32 Architectures Software Developer Manuals, 2015
- Matz, M., Hubicka, J., Mitchell, M.: System V Application Binary Interface, AMD64 Architecture Processor Supplement, 2013
- The OSDev wiki (http://wiki.osdev.org)
- ```
  man objdump, gcc, crash, ...
  ```
  ```

Syllabus of lab exercises

1. Decomposition of an ELF binary file, decoding its sections, and code disassembling.
2. Decomposition of an ELF binary file, decoding its sections, and code disassembling.
3. Using the crash tool on Linux.
4. Crash dump analysis of a Linux system on the AMD64 architecture.
5. Crash dump analysis of a Linux system on the AMD64 architecture.
6. Crash dump analysis of a Linux system on the AMD64 architecture.

Q305 lab environment

- RHEV-M managed VMs instantiated from prepared templates
- VMs include the base of Fedora distribution
- There is no graphic environment (why?)
- System should include all necessary tools

Decomposition of an ELF binary file, decoding its sections, and code disassembling

- Create simple object file
- ```
  vim hello.c
  ```
- ```
  #include

  int main(int argc, char **argv)
  {
          printf("Hello world\n");
          return 0;
  }
  ```
- ```
  gcc -O0 -Wall -c -o hello.o hello.c
  ```
  ```

Decomposition of an ELF binary file, decoding its sections, and code disassembling

- Create simple object file
- ```
  $ vim hello.c
  ```
- ```
  #include

  int main(int argc, char **argv)
  {
          printf("Hello world\n");
          return 0;
  }
  ```
- ```
  $ gcc -O0 -Wall -c -o hello.o hello.c
  ```
- ```
  $ gdb hello.o
  ```
- What content you can find on stack before call to printf()?

Decomposition of an ELF binary file, decoding its sections, and code disassembling

- List ELF file header:
- ```
  $ readelf -h hello.o
  ```
- What does REL type mean?
- What is magic?
- List ELF section headers:
- ```
  $ readelf -S hello.o
  ```
- Wher you can find "Hello world\n" string?
- ```
  $ readelf -x .rodata test.o
  ```
  ```

Decomposition of an ELF binary file, decoding its sections, and code disassembling

- Try to add your custom section my_section to ELF file containing some string:
- Hint - use gcc variable attribute.
- ```
  __attribute__((section(xxx)))
  ```
- ```
  const char my_section[] __attribute__((section(".my_section"))) = "Very cool stuff!";
  ```
- Dump the section as printable string.
- ```
  $ readelf -p .my_section test.o
  ```
  ```

Linking

- Process of merging sections from source object files into resulting executable segments
- ```
  $ gcc hello.o -o hello
  ```
- ```
  $ readelf -l test
  ```
- How did we linked printf() call?
- List all symbols.
- ```
  $ nm test
  ```
- Try to write linking script - section .text put at 0xdeadbeef and add 0x1000 gap after it
- ```
  $ gcc -c hello.c
  ```
- ```
  $ ld -o hello_mod -T custom.lds hello.o
  ```
  ```

Dynamic linker

- Show all needed dynamic libraries for the program
- ```
  $ readelf -d hello
  ```
- Is there another way how to do it?
- ```
  $ ldd test
  ```

Homework 1

- Topic: ELF decomposition and analysis.
- In your inboxes in ~1 week!