Software Protection
Lab: software analysis for reverse engineering

Leonardo Regano

leonardo.regano@unica.it

PhD Course on Software Security and Protection

University Of Cagliari, Italy

Software analysis for reverse engineering

- multiple tools available, usually with multiple functionalities
  - disassemblers
  - decompilers
  - deobfuscators
  - graph generation (CFG,CG,DDG,…)
- free tools
  - radare2
  - Ghidra
- commercial tools
  - IDA Pro
  - ollydbg
  - BinaryNinja (free web version)
  - CodeSurfer

Part of this presentation is based on the slides presented by Prof. Cataldo Basile in the Security Verification and Testing course at Politecnico di Torino.

Executable binaries

- executable files include much information, not just the code to be executed
  - data
  - code
  - additional management information
    - memory allocation
    - symbols
- executable formats describe how executables are structured
  - Executable and Linkable Format (ELF)
    - Linux
  - Portable Executable (PE)
    - Windows

ELF format

- program header table
  - how to create the memory image ◦ segments
- symbol table
  - introduces all the debugging symbols
    - one line per entry
  - info about dynamic linking, relocation, ...
- dynamic linking
  - info for the runtime loading of the shared library
    - more overhead than static linking
    - dynamic links resolved when needed
- different types of sections (RO, W, data, code, etc.)

Tools for executable inspection/tampering

- reading elf and assembly
  - objdump
  - readelf
- modify elf files
  - elfutils
    - elfdiff
    - elfedit
    - elfpatch

objdump and readelf

- command-line programs for displaying various information about object files
  - work for ELF files
    - disassembler to view an executable in assembly form
  - show sections and structure

objdump	readelf	description
-p	-e	header info
-h	-S	sections’ headers
-x		all headers (with the symbol table)
-t	-s	symbol table
-d		shows the assembly of the binary
-g		debug symbols
	-n	print notes
	-d	dynamic sections
-M intel|att		most common assembly formats

Global Offset Table and Procedure Linkage Table

- .got, .plt, and got.plt section used for dynamic linking
  - allow lazy binding
  - allow Position Independent Execution (PIE)
- different entities involved
  - a GOT entry provides direct access to the absolute address of a symbol
  - the PLT redirects position-independent function calls to absolute locations
  - the dynamic linker determines the absolute addresses of the destinations and stores them in the GOT
- when a new shared function is called...
  - 1) call *shared_func*
  - 2) look into the PLT for the *shared_func@plt* symbol
    - an indirect JMP to a GOT entry where to find the absolute address
  - 3) the first time the function is called
    - the GOT entry contains the address of a function (in GOT.PLT) that calls the dynamic linker
    - it will resolve the address and save it in the GOT
  - 4) after that, the GOT entry will contain the absolute address of the *shared_func* function

Binaries: state-of-the-art

- variable-sized instruction sets (Intel x86, CISC)
  - don’t know instruction boundaries for stripped binaries
  - desynchronize the parsing of instructions
- instruction sets are densely used
  - almost all the bytes are valid opcodes
- instructions may overlap
  - same bytes executed multiple times
    - each time being interpreted as belonging to a different instruction
  - interpretation may start at different places
    - JMP instructions
- data may be embedded in the code
  - separating data and code is undecidable!
  - e.g., jump tables into the code section
  - obfuscators extensively use this

Binaries: state-of-the-art

- indirect jumps/calls
  - e.g., jmp [ebp]” or “call eax”
  - function pointer, dynamic linking, jump table, etc.
    - may desync the disassembler also because of overlapping instructions
  - execution flow not known; needs to be reconstructed as well
- important data are unavailable at the disassembler (they should not!)
  - names
  - data types
  - aggregation data, e.g., macros
  - comments
- difficult to correctly reconstruct functions and their prototypes
  - not clear where they start and end
  - not clear what the parameters (passed with the calling conventions) and what are just other data in the stack/registers

Binaries: state-of-the-art

- pointer aliasing
  - two pointers that refer to the same memory area
  - create uncertainty in the execution flow
    - in the case of RW operations
  - also compilation errors if optimizations are used
- self-modifying code
  - e.g. malware or super-optimized programs

Radare2

- static analysis
  - assemble and disassemble a large list of CPU instruction sets
- dynamic analysis
  - native debugger and integration with GDB, WINDBG, QNX and FRIDA
  - analyze and emulate code with ESIL
    - ESIL = Evaluable Strings Intermediate Language
- patching abilities
  - binaries, modify code or data
- advanced search
  - patterns, magic headers, function signatures
- full support for scripting
  - command line, C API, r2pipe to script in any language
- it is an extensible framework
  - new plugins, modifications to the architecture

Rabin2

- a tool of the radare2 framework
- get information about the binaries
  - Sections, Headers, Imports, Strings, Entrypoints, ...
  - may export the output in several formats
  - supports ELF, PE, Mach-O, Java CLASS
- rabin2 -I program
  - print binary info
  - operating system, language, endianness, architecture, mitigations (canary, pic, nx)
- rabin2 -Z program
  - prints the strings
    - a better formatted/organized output than strings

Other command line utilities

- Cutter a GUI for managing radare2
- r2pipe utility to script radare2 commands
- radiff2 a diffing utility, supports byte-level or delta diffing for binary files, and code-analysis diffing to find changes in basic code blocks obtained from the radare code analysis
- rafind2 to find byte patterns in files
- ragg2 compiles programs written in a simple high-level language into tiny
- binaries for x86, x86-64, and ARM.
- rasm2 command line assembler and disassembler for multiple architectures (including Intel x86 and x86-64, MIPS, ARM, PowerPC, Java, and myriad of others)

Radare2: useful commands

- i → info
  - ie show information about the “entrypoint”
  - iz lists the strings in data sections; izz lists the strings from everywhere
  - il show information about libraries
  - is prints the symbols
- a → analyse
  - aa analyze all; aaa analyse all + auto name; aaaa full analysis
  - af analyse the functions; afi information about analysed functions; afl analyse function list
  - aai analysis stats
  - agr reference graph; agf function graph; agc function call graph
  - ax x-ref: references to a given address

Radare2: useful commands

- p → print
  - pdf disassemble current function
  - pda disassemble all the possible code
  - pdc primitive decompiler
- f → flags
  - fs strings
- s → seek
  - s print the current address
  - s address moves to the given address
  - sf function
  - sl seek to line

Radare2: useful commands

- V → visual mode
  - VV graphical visual mode
  - p/P change visual mode
  - q back from visual mode
  - :command executes “command” in visual mode
  - h/j/k/l move the screen
- enable graphics
  - e scr.utf8=true and e scr.utf8.curvy=true
- scripting
  - @@ for each operator @@f @@b
  - ~ grep
    - afi @@f ~name

Radare2 load time options

- -A analyse the binaries at startup (aa)
- -AA analyse the binaries at startup (aaaa)
- -d attach the debugger
- -w allow binary writing

Ghidra

- suite of tools developed by NSA's Research Directorate to
  - analyze malicious code and
  - get an understanding of potential vulnerabilities in networks and systems
- software analysis tools
  - disassembly, assembly, decompilation, graphing, and scripting
  - several processor instruction sets and executable formats
  - user-interactive and automated modes
- extensible
  - develop Ghidra plug-in components and/or scripts using the exposed API
- https://ghidra-sre.org/

Tracing

- purpose
  - better understanding of the system/program behavior
  - as non-intrusive as possible
  - statistical data gathering
  - ...different from debugging
- some tools
  - ptrace, the most comprehensive one
  - SystemTap, observe the kernel and user-space/kernel switches
  - trace-cmd, non intrusive kernel observation
  - bpftrace, data aggregation at kernel level
  - strace and ltrace, user friendly for tracing system calls and calls to libraries