Software Reverse Engineering

Quanto mai sarà difficile andare da Binario a Sorgente?

Gianfranco Gallizia

Linux Day 2022

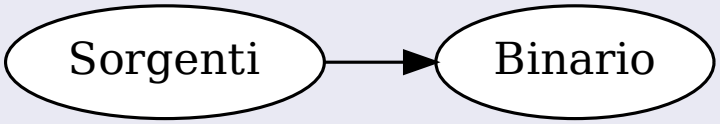
Introduzione

Disclaimer

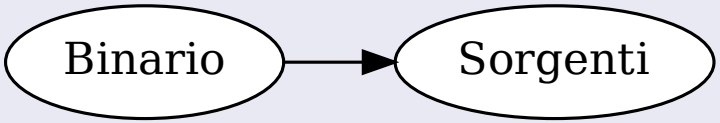
Non sono un avvocato, né un giurista. Tutto quello che troverete in questa presentazione sono miei appunti sugli strumenti e sulle tecniche utilizzate per l'analisi di file binari che ho raccolto e sviluppato nel corso degli anni. Questa presentazione ha un puro scopo didattico ed illustrativo.

Non eseguite le tecniche descritte per fini illeciti ed in caso di dubbi consultate un professionista.

Flusso normale



Flusso inverso



Analisi statica

Effettuata senza eseguire il software, guardando unicamente i bit su disco.

Analisi dinamica

Effettuata quando il software è in esecuzione guardando le attività ed il flusso di esecuzione.

Analisi statica - pro e contro

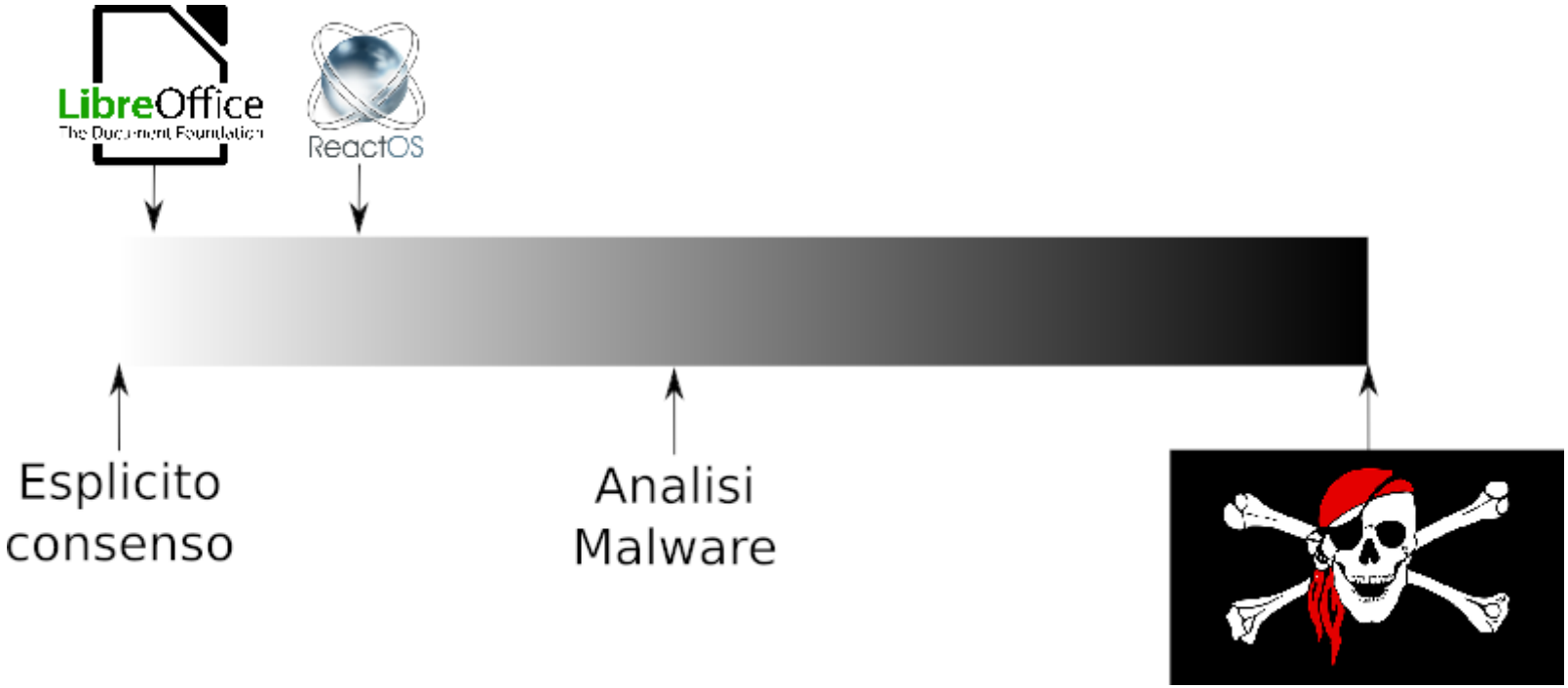
- Pro:
  - Non si esegue il binario in esame. Bypassando le protezioni a runtime ed evitando di eseguire potenziali malware.
  - L'analisi può essere fatta su una macchina di architettura diversa dalla macchina bersaglio (ad esempio: analisi di un binario ARM su un pc amd64).
- Contro:
  - Non si esegue il binario in esame. Se il binario è compresso/offuscato/criptato/modificato a runtime non lo si vedrà.

Quando è legale fare SRE nell'Unione Europea?

- Quando si ha l'esplicito consenso (meglio se scritto) del detentore dei diritti su software.
- Quando si effettua al fine di garantire l'interoperabilità del software con un altro software e/o con un altro sistema operativo/un'altra architettura. A patto che:
  - Si sia in possesso di regolare licenza d'uso per il software.
  - Non siano disponibili in altro modo le informazioni necessarie a garantire l'interoperabilità.
  - L'azione di SRE sia limitata alle sole parti necessarie a garantire l'interoperabilità.

Introduzione

Legalità del reverse engineering



Strumenti del SRE

Di cosa ho bisogno?

- Conoscenza
  - Manuali, presentazioni, post su forum. Raccogliete quante più informazioni possibile.
- Pazienza
- Alcuni software
  - Editor esadecimale
  - Disassembler/Decompiler
  - System trace
  - Debugger
  - Virtual Machine/Emulatori
  - (Almeno) un linguaggio di programmazione/scripting

Editor esadecimale

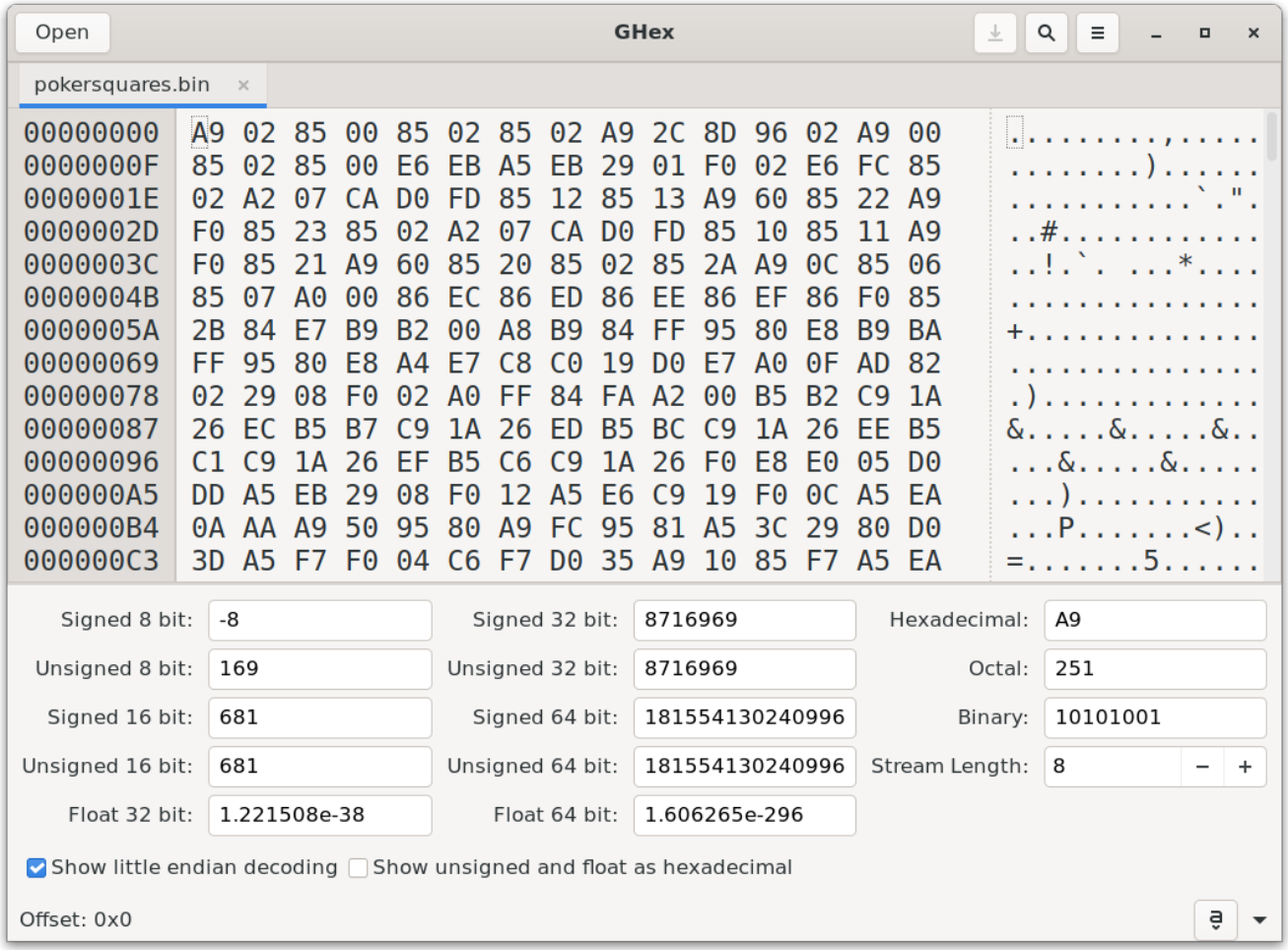
- Equivalente binario degli editor di testo.
- Usa la notazione esadecimale per rappresentare i byte del file (cifre da 0-9 e poi A=10, B=11, C=12, D=13, E=14, F=15). Due cifre esadecimali possono rappresentare i numeri da 00 a FF (255).
- Possono essere molto semplici (ad es. GHex) o con funzioni aggiuntive che semplificano l'analisi di strutture dati e la ricerca di pattern.
- Solitamente divisi in tre colonne: offset, dati in esadecimale, rappresentazione ASCII.

Editor esadecimale - Utilizzi

- Indagine preliminare:
  - È davvero un eseguibile?
  - Ci sono porzioni che contegono dati leggibili?
- Binary patching (sostituisco parte del binario per cambiarne il comportamento).

Analisi statica - Editor esadecimale

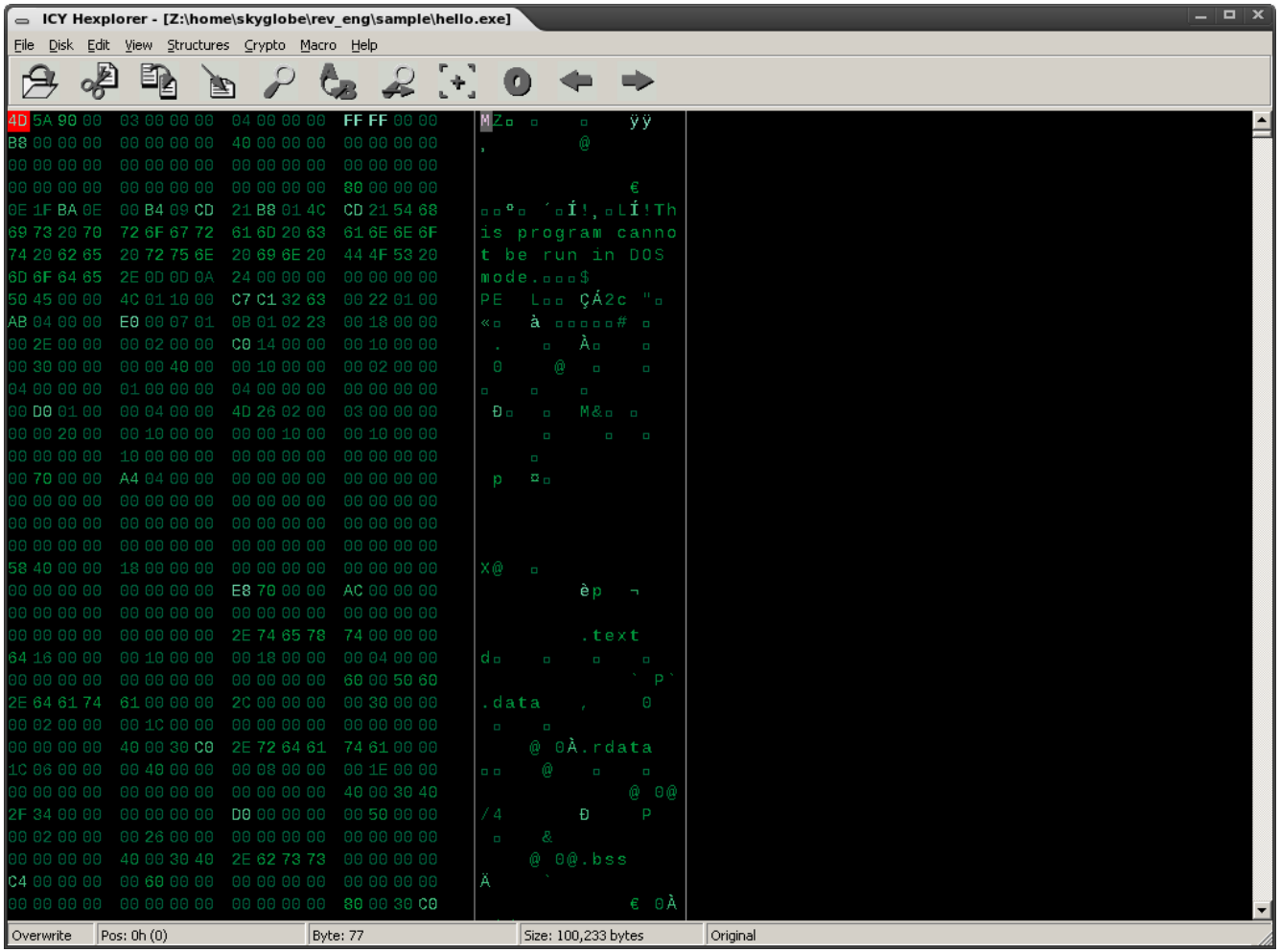
GHex



https://wiki.gnome.org/Apps/Ghex

Analisi statica - Editor esadecimale

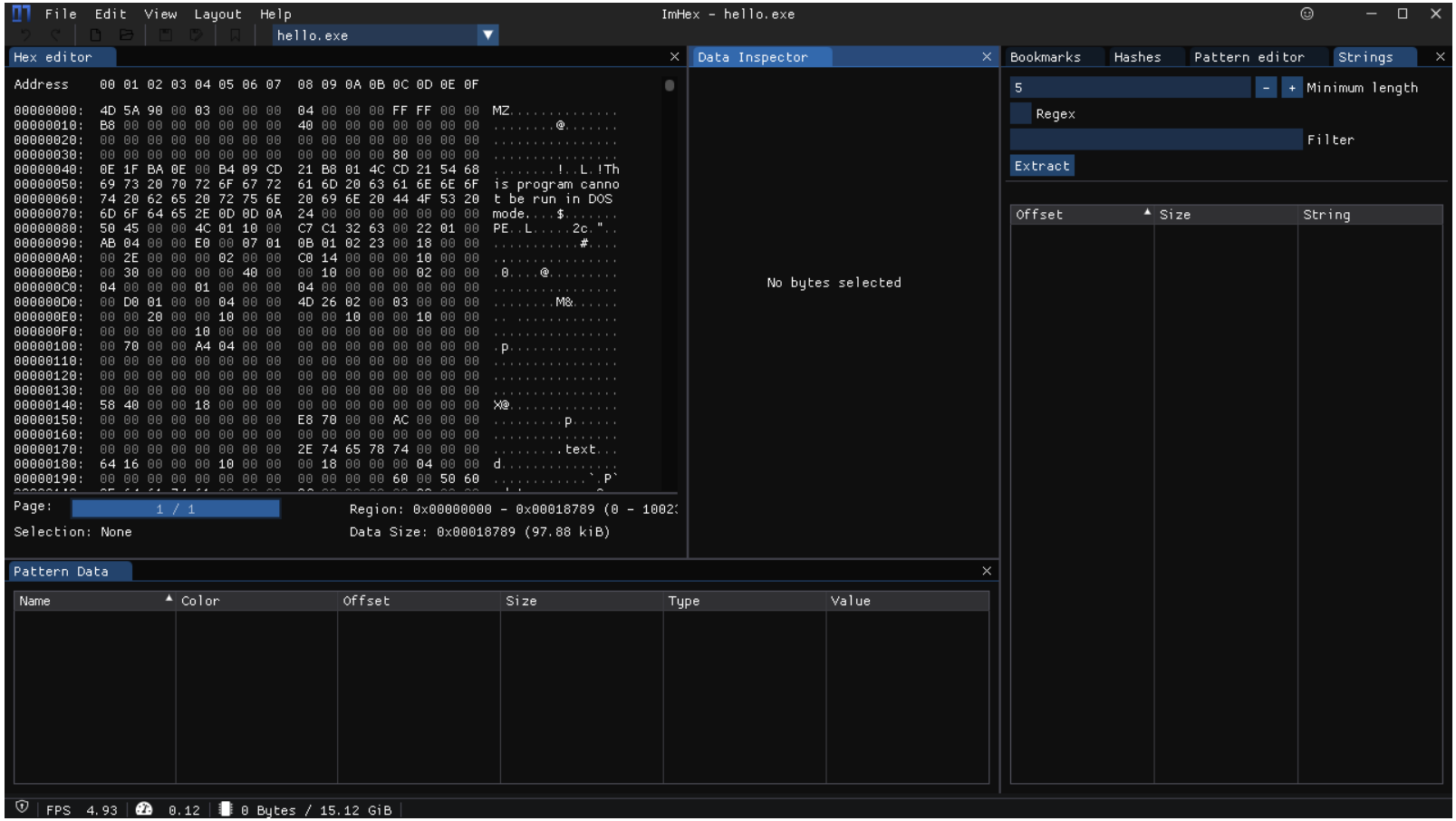
Hexplorer



https://sourceforge.net/projects/hexplorer/

Analisi statica - Editor esadecimale

ImHex



https://github.com/WerWolv/ImHex

Disassembler

- Effettua l'operazione inversa di un assembler (passare da codice macchina a codice assembly).
- Strettamente dipendente dall'architettura bersaglio.
- In alcuni casi (CPU/microcontrollori legacy ad 8 bit) il disassemblato è di fatto il codice sorgente.
- Più frequentemente è il risultato finale di numerose trasformazioni effettuate dal compilatore per ottenere il codice macchina.

Esempi

- objdump (Parte delle binutils)
- Radare2 (Suite di tools per SRE)
- Ghidra (Anche decompiler)

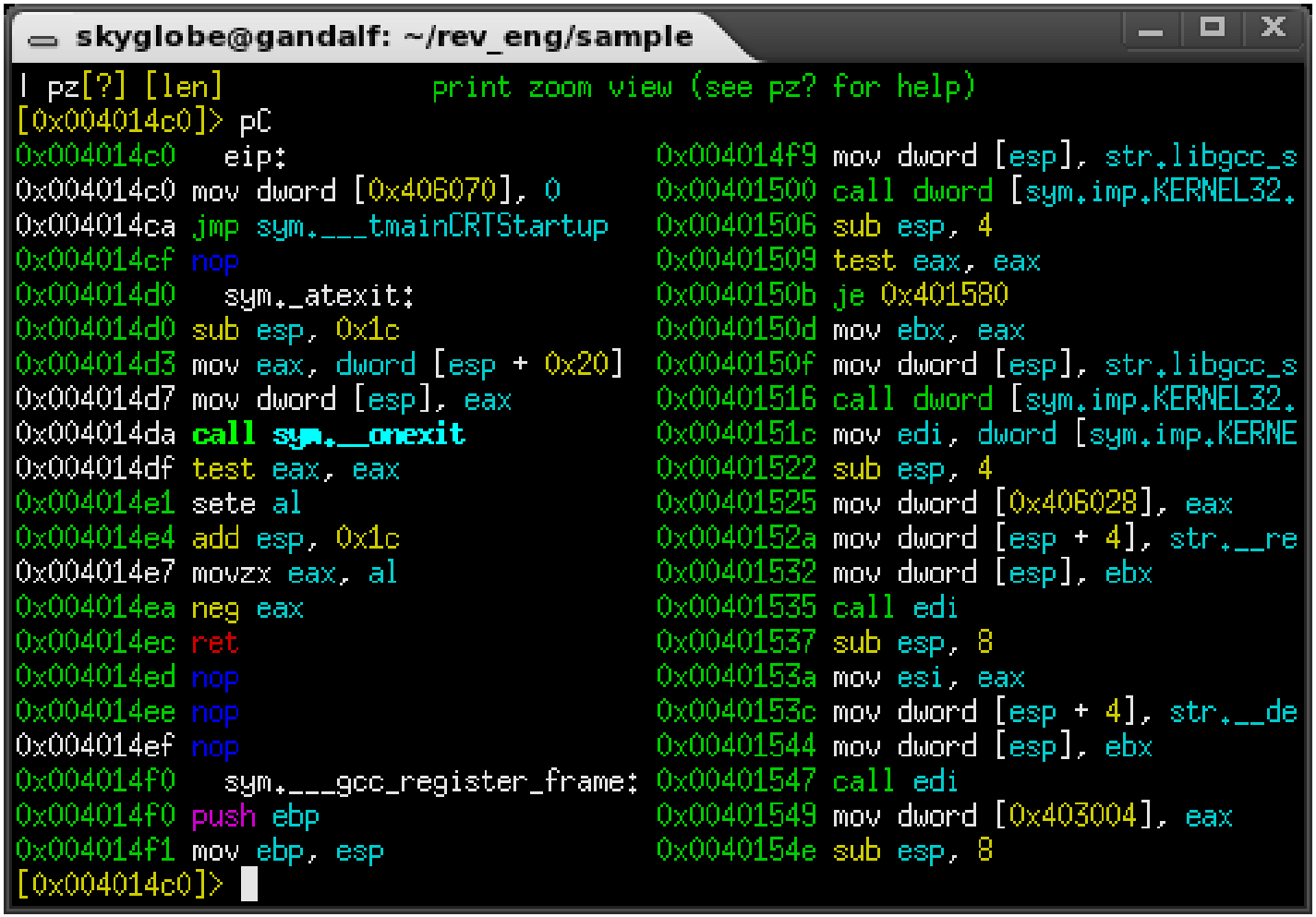
Analisi statica - Disassembler

Radare2

- Partito come tool per recuperare file da una partizione HFS+ si è evoluto fino a diventare un framework per il SRE.
- Ha un'interfaccia testuale scriptabile ed ha alcuni tool per compiti specifici: `rasm2` assembler/disassembler standalone, `rabin2` per l'analisi degli header binari, `radiff2` per l'analisi delle differenze tra due binari.
- Supporta architetture multiple: amd64, ARM, x86, MIPS, PowerPC, Atmel AVR, Motorola 68k, MSIL, Java bytecode, Dalvik VM bytecode, WebAssembly (WASM) e altre ancora.
- È disponibile una GUI (Iaito).

Analisi statica - Disassembler

Radare2



https://www.radare.org

Decompiler

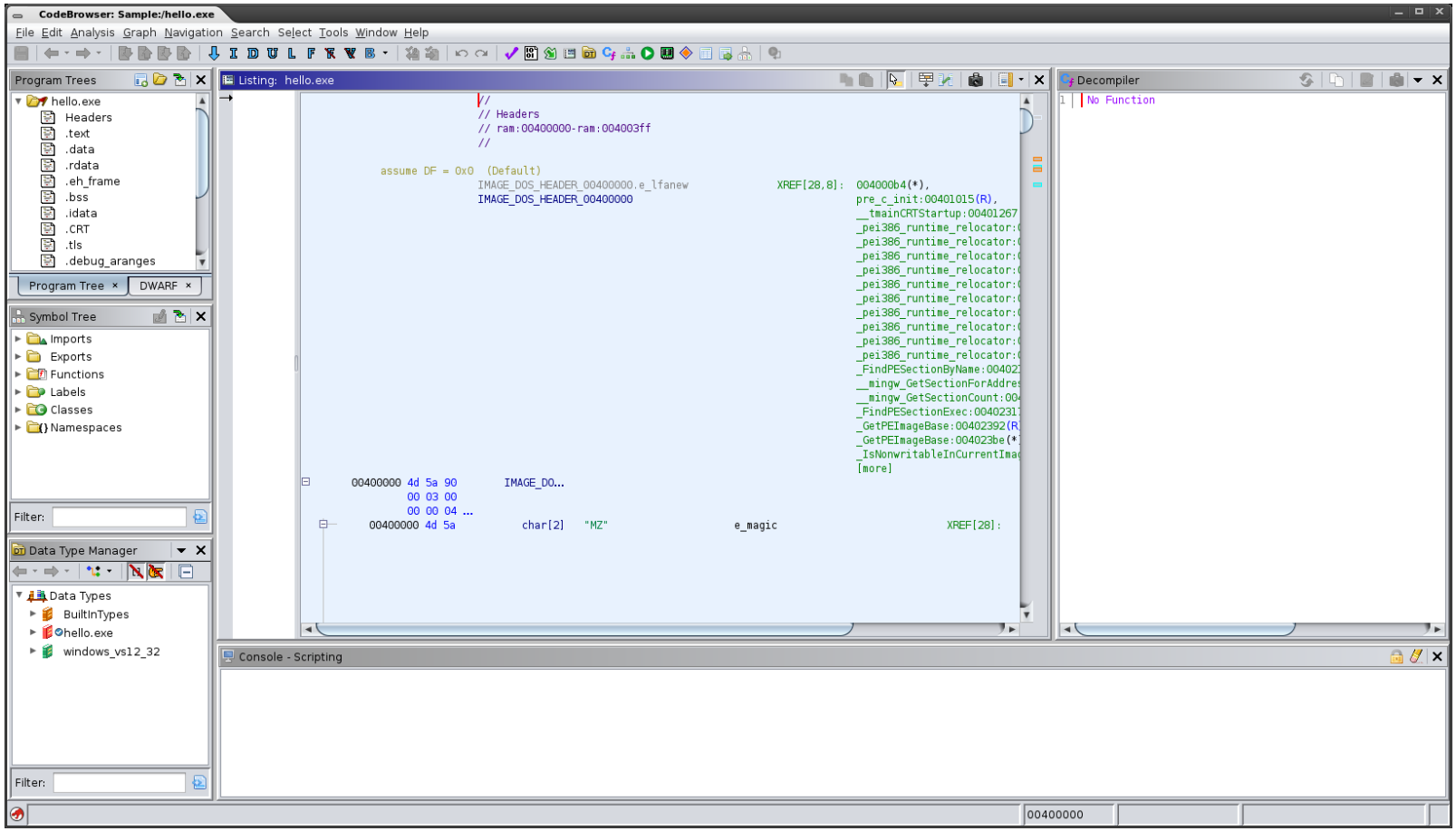
- Effettua l'operazione inversa del compilatore.
- Molto più complesso di un disassembler (vi è più perdita di informazione da sorgente ad assembly che da assembly a codice macchina).
- Richiede molto più lavoro manuale da parte dell'operatore (leggasi: serve avere esperienza di programmazione).
- Solitamente il linguaggio di programmazione "ricostruito" dal binario è il C (con risultati "interessanti" nel caso in cui non sia così).

Ghidra

- Progetto della NSA rilasciato come open source nel 2019.
- Scritto in Java e C++ (per le parti più computazionalmente intense).
- Può essere esteso con plug-in scritti in Java o in Python (grazie all'interprete Jython).
- Svolge sia funzioni di disassembler che di decompilatore (C/C++).
- Può interfacciarsi con un debugger (GDB in Linux, WinDbg in Windows).

Analisi statica - Decompiler

Ghidra



https://github.com/NationalSecurityAgency/ghidra

System trace

- La system trace (o, più precisamente, system call trace) è una tecnica di analisi dinamica che intercetta le chiamate alle API del sistema operativo effettuate da un programma.
- Non c'è bisogno di modificare il programma analizzato, basta lanciarlo tramite un wrapper che istruirà il sistema operativo a fare rapporto al wrapper.
- Il programma tracciato può cercare di individuare il tracciante e modificare il suo comportamento di conseguenza.

Esempi

- strace (Linux)
- ktrace (FreeBSD)
- Process Monitor (Windows - Sysinternals)

Analisi dinamica - Debugger

Introduzione

Debugger

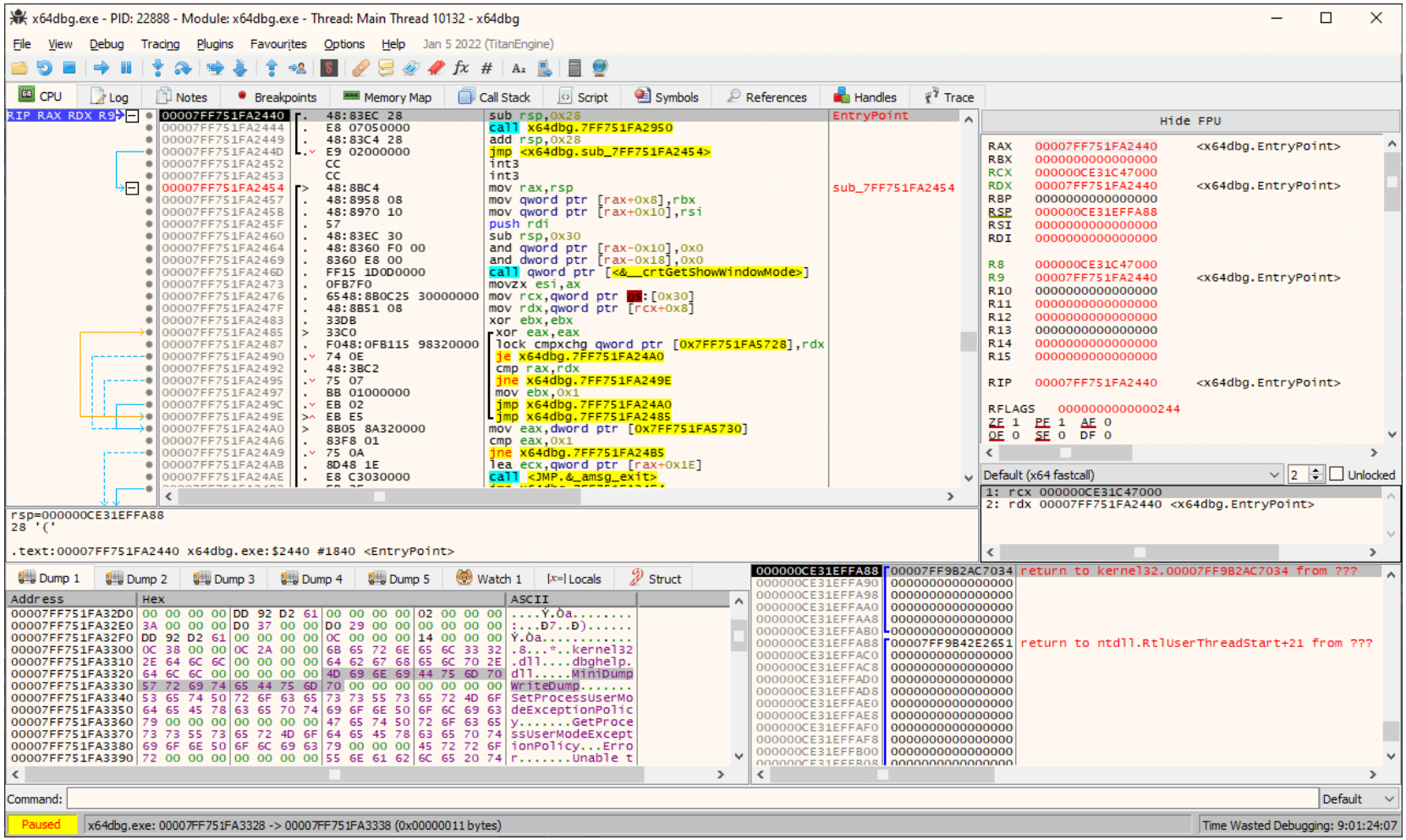
- Strumento usato dagli sviluppatori quando hanno bisogno di eseguire passo-passo il loro lavoro alla ricerca di problemi.
- Non è necessario avere i sorgenti di un programma per agganciare un debugger ad un processo: si riceverà un warning e l'analisi proseguirà a livello di disassemblato.

Esempi

- gdb
- x64dbg (Windows)

Analisi dinamica - Debugger

x64dbg



https://x64dbg.com

Virtual Machine ed Emulatori

- Vi sono casi in cui non è possibile usare la stessa macchina per analizzare un programma ed eseguirlo:
  - Il programma è scritto per un'architettura diversa da quella della macchina utilizzata per l'analisi.
  - Il programma richiede hardware non presente sulla macchina utilizzata per l'analisi.
  - Non ci si fida del programma che si sta analizzando.
  - Si vuole analizzare il programma in un contesto il più pulito possibile.

Virtual Machine ed Emulatori

- Le macchine virtuali (Virtual Machine, o VM) permettono di eseguire un programma in un contesto diverso da quello del sistema operativo della macchina ospitante:
  - Diverso kernel.
  - Diversa distribuzione.
  - Diverso set di pacchetti installati.
  - Diverso sistema operativo.

Analisi dinamica - Virtual Machine ed Emulatori

Introduzione - cont.

Virtual Machine ed Emulatori

- Quando i binari da analizzare non possono essere eseguiti sulla macchina corrente si può fare ricorso agli emulatori.
- Gli emulatori simulano hardware differente da quello presente sulla macchina che li sta eseguendo.

QEMU

- QEMU è un emulatore e virtualizzatore open source.
- Può operare in autonomia oppure appoggiarsi ad un Hypervisor (solitamente KVM o Xen).
- Può operare in due modalità:
  - System mode emulation, in cui emula un'intera macchina.
  - User mode emulation, in cui emula solo la CPU e consente di eseguire un processo compilato per un'architettura su di un'altra.
- Supporta il protocollo di comunicazione remota di `gdb` e consente quindi di fare debugging di binari compilati per architetture differenti da quella della macchina su cui gira.

https://www.qemu.org/

Linguaggi di programmazione

- Brian Kernighan, Dennis M. Ritchie "The C Programming Language" ISBN 0-13-110362-8 272 pagine
- Steve Oualline "Practical C++ Programming" ISBN 978-0-596-00419-4 550 pagine
- Randall Hyde "The Art of Assembly Language" ISBN 978-1-59327-207-4 732 pagine

Sistemi operativi

- Brian Ward "How Linux Works" ISBN 978-1-71850-040-2 464 pagine
- Michael Kerrisk "The Linux Programming Interface" ISBN 978-1-59327-220-3 1552 pagine
- Charles Petzold "Programming Windows" ISBN 978-1-57231-995-0 1479 pagine

Miscellanea

- Chris Eagle, Kara Nance "The Ghidra Book" ISBN 978-1-71850-102-7 608 pagine
- V. Anton Spraul "Think like a Programmer" ISBN 978-1-59327-424-5 256 pagine
- Brian Hook "Write Portable Code" ISBN 978-1-59327-056-8 272 pagine

Grazie per l'attenzione

Live Demo con domande?