CSE509 Computer System Security

2023-03-23

Reverse Engineering

Michalis Polychronakis

Stony Brook University

Reverse Engineering

The process of deconstructing a human-made object to extract its design, architecture, and other information

- Not only software! Machines, electronics, chemicals, …

Various motives

- Military or commercial espionage
- Product security analysis
- Competitive technical intelligence
- Interfacing with other systems
- Rescuing from obsolescence
- Repurposing

Software Reverse Engineering: Security

Malicious software analysis

Dissect and analyze potentially malicious samples ➔ develop countermeasures

Vulnerability discovery

Whitehat ➔ find bugs and develop patches

Blackhat ➔ write exploits and use them

Binary auditing

Verify functionality or security policy enforcement, discover backdoors, …

Cryptographic algorithms

Extract hard-coded keys, unknown logic/algorithms *(bad idea: security by obscurity)*

DRM cracking (media, software), game cheating, …

Software Reverse Engineering: Development

Interoperability with proprietary/legacy software/protocols

- Not enough/non-existent/inconsistent documentation, lost source code, …

Developing competing software

- Steal partial functionality
- Re-package whole applications (e.g., malicious Android apps)

Software quality evaluation (e.g., `cyber-itl.org`)

- Security features: many protections are introduced only at compilation time ➔ source code analysis is not enough
- Code hygiene: use of dangerous functions, consistency, …
- Code complexity: code size, number of dependencies, …
- Crash testing: fuzzing using bad inputs to assess robustness

Reversing at Different Levels

System

- Monitor the execution of a target program
- System calls, events, files, IPC, registry, …

Code

- Understand what a piece of code does
- Static (code disassembly) vs. runtime (debugging)
- Source code vs. intermediate representation code vs. machine code
- User space vs. kernel vs. firmware/BIOS/…

Network

- Monitor network traffic
- Understand protocols

Tools of the trade

System monitoring

Information mostly provided by the OS: Files, registry, network, system calls, …

Disassemblers

Convert machine code to assembly language code

Extract the control flow graph, variables, dependencies, …

Decompilers

Convert machine/assembly code to higher-level language code

Debuggers

Observe the execution of a process at instruction granularity

Disassembly, software/hardware breakpoints, register/memory info, …

Various other tools

Hex editors, memory dumpers, ELF/PE analyzers, hooking libraries, …

System Monitoring Tools

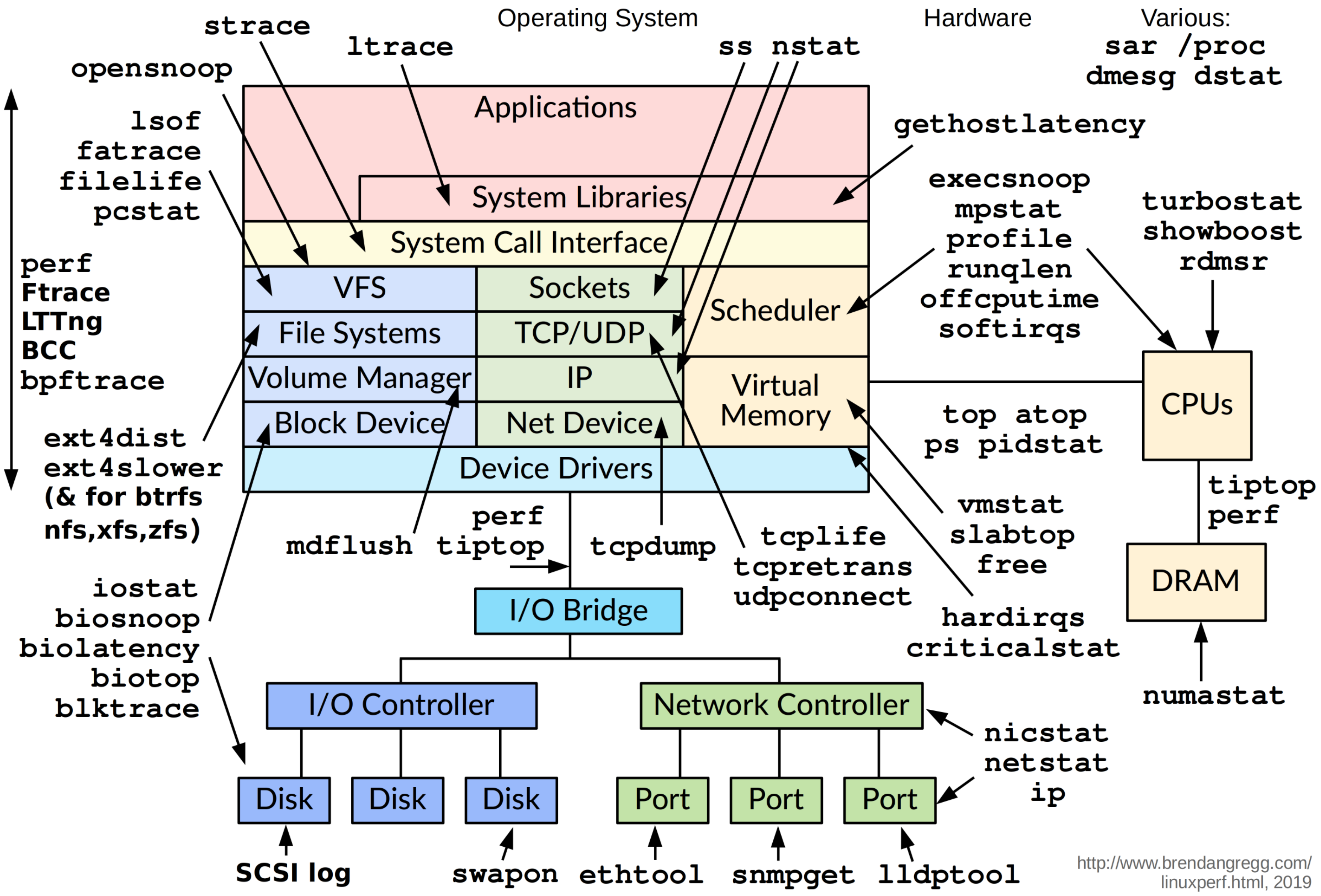
Linux

- strace, ltrace, netstat, ps, …
- perf_events, eBPF, LLTng, …
- osquery, sysdig, …

Windows

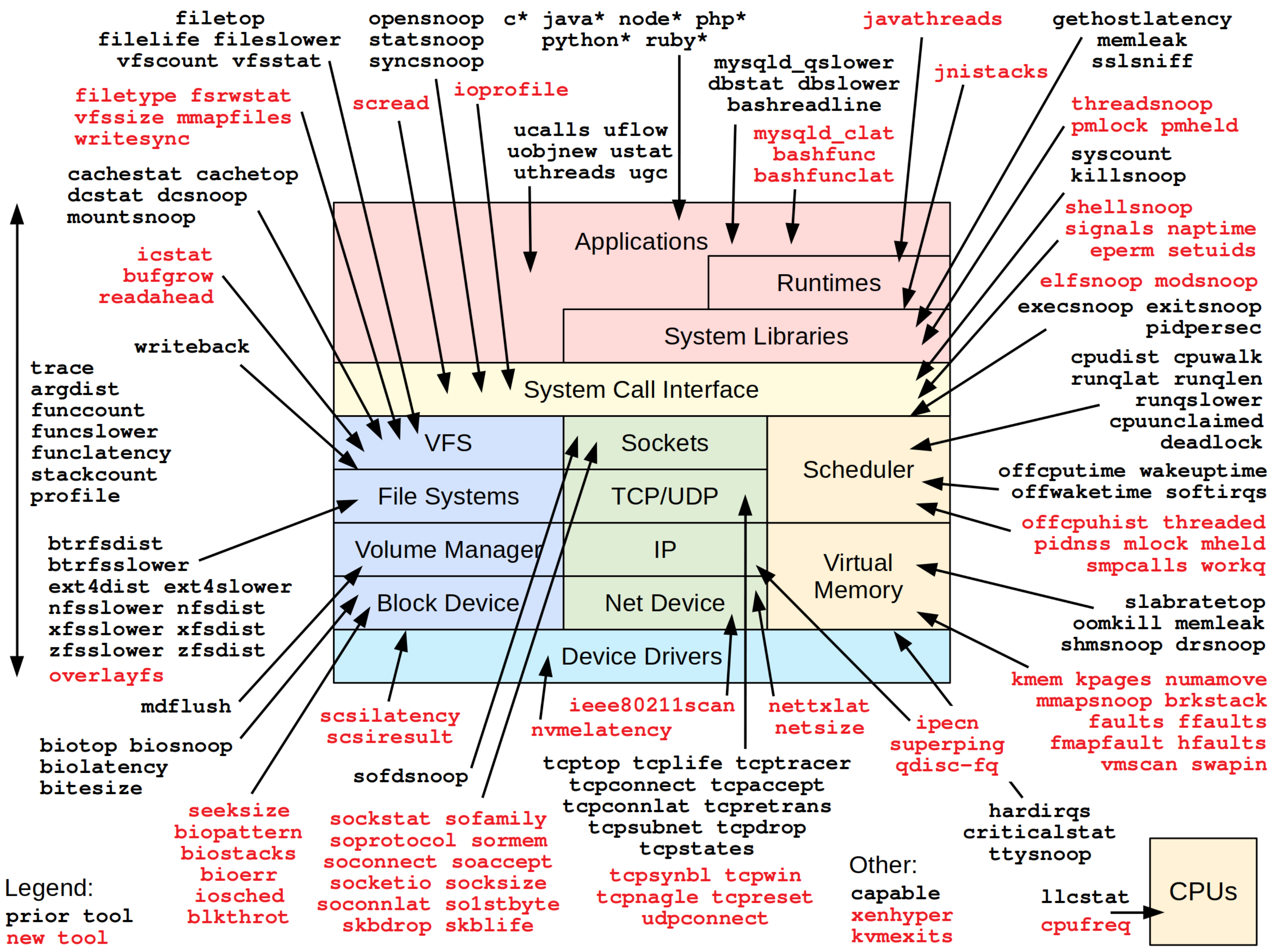
- Process Explorer
- procmon, filemon, regmon, sysmon,
- VMMap
- TCPView
- … and the rest of the Sysinternals suite: https://live.sysinternals.com/

Linux Performance Observability Tools



http://www.brendangregg.com/
linuxperf.html, 2019

New tools developed for the book **BPF Performance Tools: Linux System and Application Observability**
by Brendan Gregg (Addison Wesley, 2019), which also covers **prior BPF tools**



filetop
filelife fileslower
vfscount vfsstat

filetype fsrwstat
vfssize mmapfiles
writesync

cachestat cachetop
dcstat dcsnoop
mountsnoop

icstat
bufgrow
readahead

writeback

trace
argdist
funccount
funcslower
funclatency
stackcount
profile

btrfsdist
btrfsslower
ext4dist ext4slower
nfsslower nfsdist
xfsslower xfsdist
zfsslower zfsdist
overlayfs

mdflush

biotop biosnoop
biolatency
bitesize

seeksize
biopattern
biostacks
bioerr
iosched
blkthrot

Legend:
prior tool
new tool

opensnoop
statsnoop
syncsnoop

scread

ioprofile

c* java* node* php*
python* ruby*

ucalls uflow
uobjnew ustat
uthreads ugc

mysqld_qslower
dbstat dbslower
bashreadline

mysqld_clat
bashfunc
bashfunclat

javathreads

jnistacks

gethostlatency
memleak
sslsniff

threadsnoop
pmlock pmheld

syscount
killsnoop

shellsnoop
signals naptime
eperm setuids

elfsnoop modsnoop

execsnoop exitsnoop
pidpersec

cpudist cpuwalk
runqlat runqlen
runqslower
cpuunclaimed
deadlock

offcputime wakeuptime
offwaketime softirqs

offcpuhist threaded
pidnss mlock mheld
smpcalls workq

slabratetop
oomkill memleak
shmsnoop drsnoop

kmem kpages numamove
mmapsnoop brkstack
faults ffaults
fmapfault hfaults
vmscan swapin

scsilatency
scsiresult

sofdsnoop

sockstat sofamily
soprotocol sormem
soconnect soaccept
socketio socksize
soconnlat so1stbyte
skbdrop skblife

ieee80211scan
nvmelatency

nettxlat
netsize

ipecn
superping
qdisc-fq

tcptop tcplife tcptracer
tcpconnect tcpaccept
tcpconnlat tcpretrans
tcpsubnet tcpdrop
tcpstates

tcpsynbl tcpwin
tcpnagle tcpreset
udpconnect

hardirqs
criticalstat
ttysnoop

Other:
capable
xenhyper
kvmexits

llcstat
cpufreq

Disassemblers

IDA Pro *commercial, with free demo version*

Ghidra *NSA's own open-source reverse engineering tool suite (!)*

Radare2 *powerful reversing framework*

Capstone *multi-platform, multi-architecture framework*

distorm3 *disassembler library*

Hopper *commercial, with demo version*

Binary Ninja *commercial, with free demo version*

BinNavi *control flow and program analysis*

objdump, ndisasm, gdb, …

Decompilers

IDA Pro, Boomerang, JEB, …

Many others for specific languages: Java/.NET/…

Binary Analysis and Lifting

angr, BAP, Miasm, rev.ng, …

Beyond code disassembly and decompilation

- Lifting to intermediate representation (IR)
- Symbolic/concolic execution
- Control-flow graph extraction
- Support for multiple platforms and architectures

Extensible frameworks for building binary analysis applications

- Automated ROP chain construction
- Binary patching and hardening
- Automated exploit generation

Debuggers

gdb, LLDB (LLVM), Visual Studio, …

OllyDbg *simple and powerful, with intuitive GUI (Windows)*

Windbg *both user and kernel space*

IDA Pro

x64dbg *open-source x64/x32 debugger for Windows*

edb *inspired by Ollydbg, cross platform, still under development*

SoftICE *popular kernel debugger (last release was in 2000)*

rr *reverse debugging under gdb!*

Binary Instrumentation and Tracing

Pin, DynamoRio, Valgrind, Frida, …

Intel Processor Trace *HW tracing, supported by gdb and perf*

Emulators/VMs

Qemu, Unicorn, VMware, Virtual Box, Xen, KVM, …

Debugger Basics

Software breakpoints

- Replace target instruction with `int 3` (breakpoint interrupt)
- Once triggered, execution freezes and `int 3` is replaced with the original instruction

Hardware breakpoints

- Managed directly by the processor through debug registers
- Triggered on code or data access
- Main benefit: no modification in the program
- Main drawback: just a few debug registers are available (only six for x86)

Single-stepping

- When the trap flag (TF) in the `EFLAGS` register is set, the processor generates an interrupt after the execution of every instruction

Modern Debugger Features

Flexible breakpoints

- Column breakpoints: break at specific point within a source code line
- Conditional breakpoints: break only after hit count is reached, conditional expressions, …
- Tracepoints: don’t break but just log information (e.g., current value of a variable)
- Data breakpoints: break when specific memory location is written

Data visualization

- Beyond printing variables: data formatting, custom object views, bitmap images, …

Expression evaluation

- Execute new code within the context (using the state) of the debugged process

Concurrency and multithreading

- Dependencies across call stacks of different threads, freeze/unfreeze threads, …

Many other: hot patching, time travel/reverse debugging, …

https://werat.dev/blog/what-a-good-debugger-can-do/