Towards the Higher Level Debugging with QEMU

Pavel Dovgalyuk, ISP RAS
About us

- Ivannikov Institute for System Programming of the RAS
- Emulation-related projects
- Full system record/replay in mainline QEMU
- VM introspection and instrumentation
- Stealth WinDbg stub for QEMU
- Reverse debugging patches ready for 4.3 (or 5.0?)

- https://github.com/ispras/swat
Plan

- Approaches to system-wide debugging
- Problems of system-wide debugging
- New ideas and proposals
Debugging with QEMU/KVM

• QEMU/KVM
  - Debugging OS/drivers/BIOS
  - Malware analysis
• QEMU only
  - Execution recording (time travel debugging)
  - Cross-platform debugging
Debugger functions

• Processes
  – Pages
  – Threads/Fibers
  – Process switches

• Executables
  – Memory areas
  – Function names
  – Variable names
  – Call stack

• Breakpoints
  – Memory access
  – Register access

• Events
  – Exceptions
  – Interrupts
  – System calls
  – I/O
Full system debugging with WinDbg

- OS debug mode has to be enabled
- Has complete kernel information
- Can debug separate processes
- Unofficial stub for QEMU
- Windows only
Developer’s view to the debugging

- Run gdb server in the guest
- Run gdb client on the host
- Attach to guest process
- Load symbols
- Debug the program

- Run gdb client
- Load kernel symbols
- Connect to guest/emulator gdb server
- Debug the kernel
Reverser’s view to the debugging

- Run gdb server in the guest
- Run gdb client on the host
- Attach to guest process
- Load symbols
- Debug the program

- Run gdb client
- Load kernel symbols
- Connect to guest/emulator gdb server
- Debug the kernel
Full system debugging with GDB

- Need to figure out the address for loading symbols from the binaries
- Not usable for Windows
- Can’t distinguish the processes even when having the symbols
Jedi debugging

- Use the Force to figure out CR3
- break *0xdeadf00d if $cr3=0x1ee7
Debugging problems

• VM Introspection to extract OS-level information
  – Processes and threads
  – Call stack
  – Address spaces and page tables
  – Executed images and symbol/debug information

• Client which capable of full-system debugging
  – Process and thread support
  – Support for switching the address spaces
Introspection: guest agents

• Have full control to the guest data structures and API
• Require SDK inside the image
  – or debug mode for WinDbg
  – or running gdbserver
• Side effects
  – behavior change
  – can be detected by malware
  – can’t be recorded/replayed
Introspection: memory analysis

- Rekall/Volatility
- Parse memory dumps
- Include many OS profiles
- Hardly applicable for custom kernels and esoteric OSes
- Too slow for runtime monitoring
Introspection: event hooking

- Volatility-like profiles and event monitoring (PANDA)
  - Needs configuring for every kernel
  - Requires SDK for the guest
- Profile-less and agent-less event monitoring (SWAT)
  - Single config for all Linux kernels 2.x-4.x
  - Lacks some details of the kernel internals
pyvmidbg

- OS-agnostic debug interface
- Uses Rekall for introspection
- Intended to support
  - Linux and Windows
  - all debuggers

- [https://github.com/Wenzel/pyvmidbg](https://github.com/Wenzel/pyvmidbg)
LibVMI

- Extracts CPU and memory state from running VM
- Supports runtime events
  - Memory access, privileged registers access, debug events, ...
- Suitable for GDB and WinDbg stubs
- Doesn’t support QEMU yet

- [https://github.com/libvmi/libvmi](https://github.com/libvmi/libvmi)
Instrumenting the code

- Debugger can’t parse call stack when frame pointer is omitted
- Break on specific opcode
  - syscall – ok for libvmi (exception)
  - call/ret – not ok for libvmi
- Break on register access
  - CR3 – ok for libvmi (privileged)
  - ESP – not ok for libvmi
- Impossible for HW hypervisors
- Possible with QEMU, but not implemented yet
More debugging problems

- Too dumb breakpoints
- Can’t inspect hardware state except the CPU registers
Breakpoints: emulator-side conditions

- Set breakpoint
- Run
- Stop at breakpoint
- Check condition
- Run
- Stop at breakpoint
- Check condition
- Run
- Stop at breakpoint
- Check condition
- Run
- Stop at breakpoint
- Check condition
- Run
- Stop at breakpoint
- Check condition
- Run
- Stop execution
More breakpoints

- I/O breakpoints
- Memory area (e.g. whole array) watchpoints
- Breakpoints at specific process
- Breakpoints at interrupts and exceptions

- Need to extend QEMU and the debugger
Device introspection and debugging

- Hardware-software codesign
- Driver debugging
- Emulator debugging

- Not very handy approaches
  - Debug logs in QEMU
  - Running two debuggers
Conclusion

- Only WinDbg supports system-wide view
- LibVMI is not enough for extracting all the details
- Need synchronized QEMU-GDB efforts to extend the protocol

- Solutions
  - use only Windows as a guest
  - create new debugger (maybe based on the existing one)
LibVMII

NewDbg

system-wide debugging

introspection and instrumentation

QEMU+

and maybe others