Tutorial 3: eBPF

CSC 469 / CSC 2208
Fall 2019
Angela Demke Brown
eBPF

- “extended Berkeley Packet Filters”
- Language-level VM within Linux kernel
  - Register-based VM
  - Custom 64-bit RISC instruction set
  - Bytecode verifier
- Restrictions are placed on eBPF programs for safety
  - Limited number of instructions
  - Controlled memory referencing
  - Originally, no loops allowed
- Bounded loops are available starting with Linux 5.3!
History: BPF

  • Register-based language-level virtual machine to run user programs for packet capture & filtering inside the BSD Unix kernel.
    • 2 registers
    • 22 instructions
    • No backward branches (no loops)
  • Safety / restrictions not mentioned in paper
History: eBPF

- BPF instruction set was too limited
- Linux introduced new “internal” BPF circa 2013
  - User programs written in “classic” BPF were translated to internal BPF
  - New virtual machine had ten (10) 64-bit registers (enough to pass function arguments in regs), new BPF_CALL instruction to call kernel functions, ~100 instructions, and other features
- “internal” BPF was made available to users as “extended” BPF soon after (patch mid-2014)
  - Verifier checks user programs at load time
    - Termination (no loops), no uninitialized reads, no out-of-bounds memory access, etc.
  - Added support for data “maps” (key-value structures) shared between kernel and user-space.
Running eBPF Programs

• Must be “attached” to code points in kernel
  • Event triggers execution of eBPF code

• Used for:
  • Classic network filtering and monitoring
  • Restricting system calls (seccomp)
  • Debugging and performance analysis
    • Attach to tracepoints, kprobes, perf events
How does it work?

• Userspace has one overloaded system call, bpf()
  
  int bpf(int cmd,
           union bpf_attr *attr,
           unsigned int size);

  • Meaning of attr depends on the command.

• For loading an eBPF program, cmd = BPF_PROG_LOAD
  
  • For load, attr includes a program type, and the list of instructions in the program
  
  • Type determines what program can access in kernel

• Other commands attach, create and access maps
  
  • Need to specify map type, max # of elements, key size and value size (in bytes)
Try it out

- Samples in linux/samples/bpf
- Need LLVM and Clang to compile them
  ```
sudo apt-get install clang
sudo apt-get install llvm
  ```
- LLVM needs to have bpf backend. Check:
  ```
llc -version | grep bpf
  ```
- In top-level linux source directory (note trailing “/” at the end)
  ```
make samples/bpf/
  ```