What is eBPF

- The extended Berkeley Packet Filter (eBPF) is a virtual machine that allows the kernel to run custom programs in response to specific events such as network packets, system calls, and tracepoints [1].
- By using eBPF, code can be injected from application space into the kernel without the need to recompile or install additional kernel modules during runtime [1].
- eBPF can be used by executing a program that is compiled into instructions that are executed in kernel space [2].
- eBPF has a virtual CPU, which consists of a 512 byte memory for storing variables and 11 registers. The 11th register is the frame pointer [2].
- Code execution at the kernel level is a very powerful tool that can be abused if it falls into the hands of someone with malicious intentions.

References


The eBPF Verifier

The eBPF verifier does a static analysis to check the validity of each eBPF program before it is loaded into the kernel. This to ensure that it does not violate security requirements and thus compromise the kernel. It simulates execution of all parts of the program in the virtual machine.

The verifier performs the following checks:

1) Ensures that the program terminates by building a control flow graph to detect and reject programs with unbounded loops or backward jumps.
2) Symbolic execution for each instruction, recording the state changes to the registers and the stack. This is to ensure memory safety to prevent out of bounds memory access.
3) Value tracking by tracking the state of each value stored in the register and each bit in the binary number to prevent invalid memory usage.
4) Verifier tracks spill/fill of registers into the stack, which may otherwise result in too many states being inspected.
5) Path pruning is used to solve the problem of large numbers of states, resulting in a dramatic reduction in verification time and allowing very complicated programs with many branches that would normally pass verification to pass.
6) After successful verification, the BPF instructions and registers are mapped to the actual architectural instructions and registers. The BPF assembly is converted to the actual machine code using a JIT compiler, which gives BPF programs their native execution speed with very low or no overhead.

Ongoing Work

Currently collaborating with an external party on:

- How the eBPF runtime in the Linux kernel has evolved over time.
- How various components of the eBPF runtime works.
- Dependable safety guarantees of eBPF for users.
- Basic operation of the eBPF verifier to determine program safety.

Research Questions

The overarching research questions for this project are:

- When is eBPF safe or unsafe to use?
- How can we increase the trust in the eBPF verifier?

Contacts

- Gbadamosi Bolaji, bolaji.gbadamosi@kau.se
- Tobias Pulls, tobiass.pulls@kau.se
- Per Hurtig, per.hurtig@kau.se
- Anna Brunstrom, anna.brunstrom@kau.se
- Toke Hailand-Jørgensen, take@redhat.com
- Simo Sorce, simo@redhat.com

Acknowledgments

The project is a collaboration between Karlstad University and Red Hat Research. For more information and additional resources, see the project page.