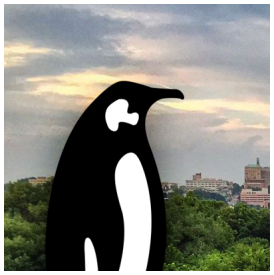

Contribution ID: **293** Type: **not specified**

Overflowing the kernel stack with BPF

Monday, 13 November 2023 12:30 (30 minutes)

eBPF is accelerating waves of innovation allowing applications to enhance the kernel's capabilities at runtime, while guaranteeing stability and security. Such guaranteed safety is made possible by the verifier engine which statically verifies BPF code. However, the verifier implicitly makes assumptions about the runtime execution environment, which must hold for safety to be upheld. One such component of the execution environment is the availability of stack space for use by the BPF program. In this talk, we highlight two fundamental problems in the setup of the BPF runtime environment that allowed us to overflow the kernel stack.

First, the BPF program, when attached to the kernel, often inherits or reuses the kernel stack, which is limited in size. Depending on the attachment point, the kernel stack may already be approaching the limit, and a BPF program can overflow the stack, despite being verified to use less than 8KB of stack space. The verifier makes an incorrect assumption about the runtime execution environment: that the kernel stack will always have 8KB stack space available.

Second, while in most cases the BPF execution environment restricts nesting of BPF programs to limit the resultant stack depth, it is incomplete. We find that, by hooking BPF programs on helper functions, we can nest multiple BPF programs in the kernel in a way that inherits the same stack state, ultimately exhausting the stack. The verifier once again makes an incorrect assumption about the runtime execution environment: that BPF program nesting is disallowed or carefully controlled in all cases.

In this talk we intend to touch upon:

- BPF program attachment and its interaction with the stack.
- an overview of the well-known difficulties encountered due to the limited kernel stack, especially with paths in the kernel, like XFS, that can bloat the stack in some cases.
- a demonstration of stack overflow due to BPF program attachment on an otherwise-innocuous attachment point.
- all methods of BPF program nesting, and their effects on stack reuse.
- a demonstration of stack overflow due to uncontrolled BPF program nesting.
- a discussion of how to mitigate these and future problems that are caused by implicit verifier assumptions about the runtime execution environment.

Primary authors: WILLIAMS, Dan (Virginia Tech); SOMARAJU, Sai Roop (Virginia Tech); CHINTAMANENI, Siddharth (Virginia Tech)

Presenters: WILLIAMS, Dan (Virginia Tech); SOMARAJU, Sai Roop (Virginia Tech); CHINTAMANENI, Siddharth (Virginia Tech)

Session Classification: eBPF & Networking

Track Classification: eBPF & Networking Track