Unsafe kernel extension composition via BPF program nesting

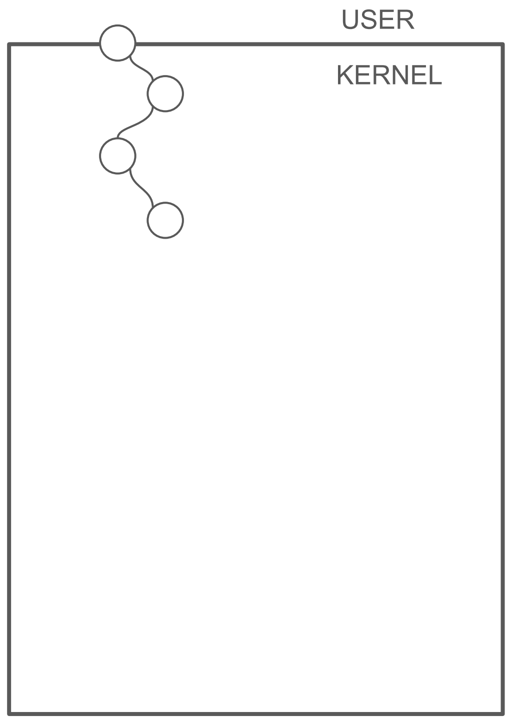
Siddharth Chintamaneni, Sai Roop Somaraju and Dan Williams

VIRGINIA TECH

Safe extension with BPF

- BPF program loaded into the kernel
- Safety checked with in-kernel static verifier
- Attached to hook point
- Run on event
- Many use cases:
  - Tracing, networking, security, scheduling, …

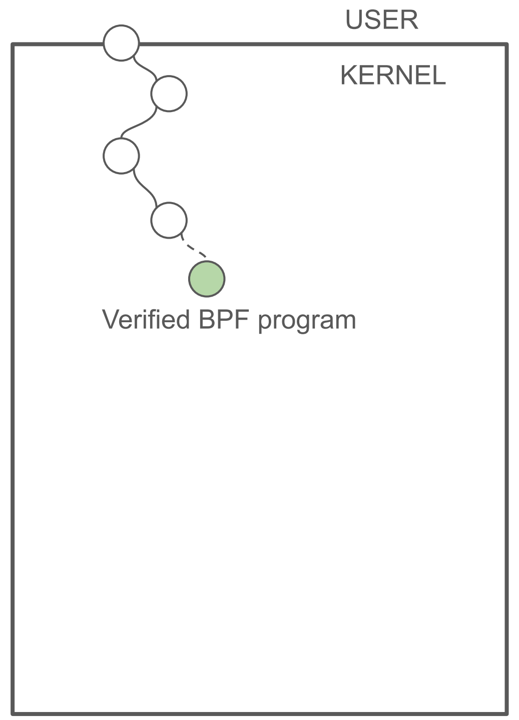
- Safety is key!!



Safe extension with BPF

- BPF program loaded into the kernel
- Safety checked with in-kernel static verifier
- Attached to hook point
- Run on event
- Many use cases:
  - Tracing, networking, security, scheduling, …

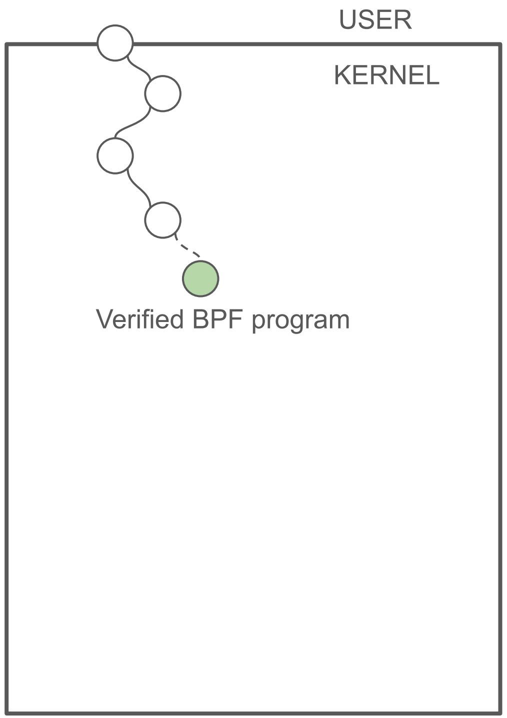
- Safety is key!!



Verifying individual extensions is not enough!!

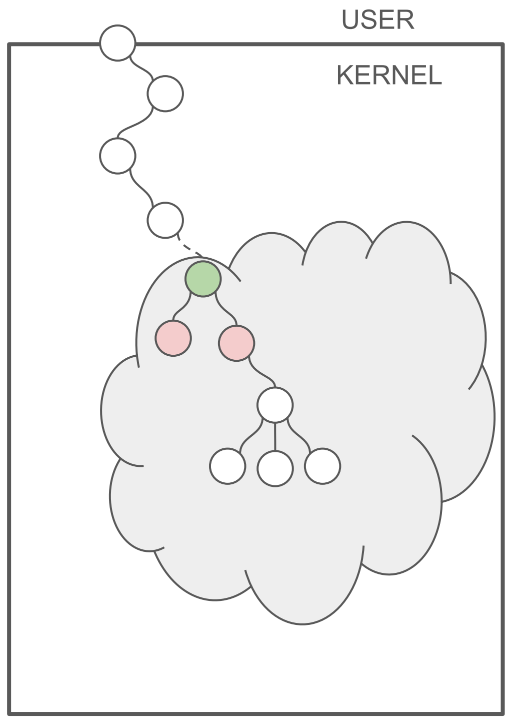
Incompleteness of Verification

- BPF programs interact with the kernel through helper functions or kfuncs



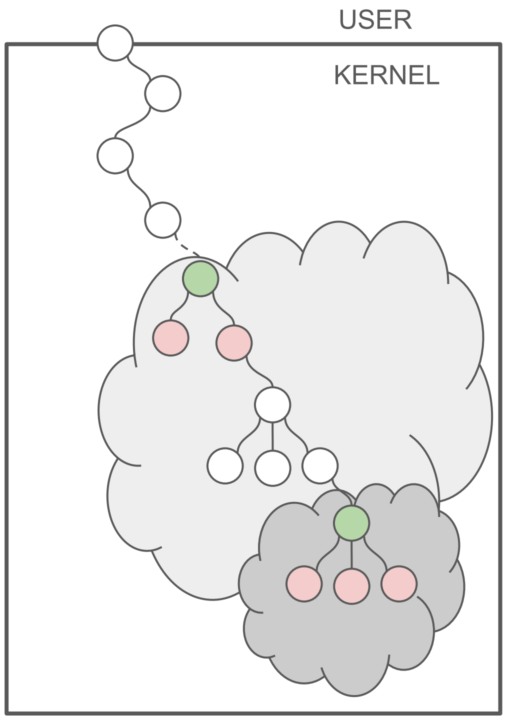
Incompleteness of Verification

- BPF programs interact with the kernel through helper functions or kfuncs



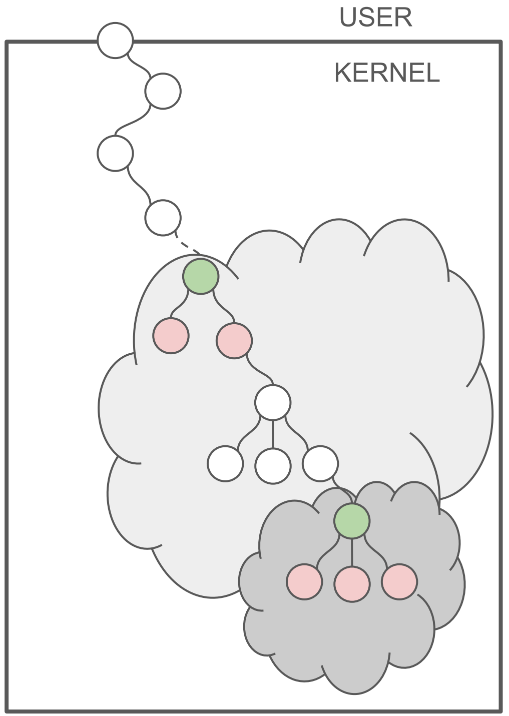
Incompleteness of Verification

- BPF programs interact with the kernel through helper functions or kfuncs
- Other BPF programs may be attached to those creating **BPF nesting**



What could go wrong?

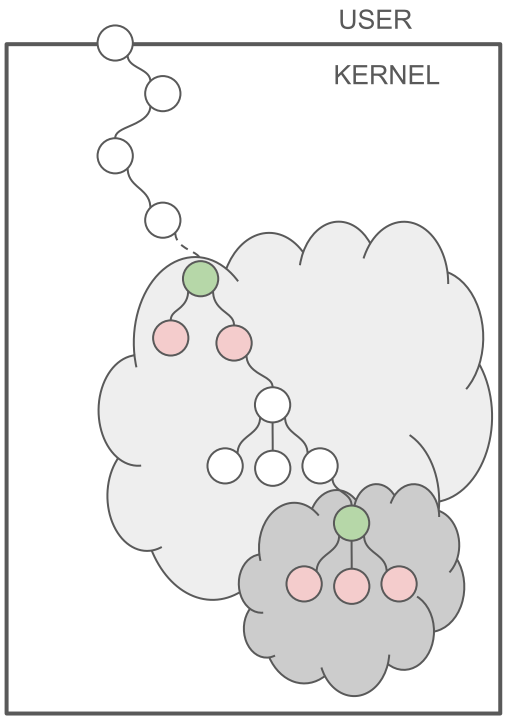
- BPF stack checks
  - Ensure program does not use more than 512 bytes of stack
  - How deep can new control flow stack become? [1]
- Deadlock
- Other issues?



[1] **Overflowing the kernel stack with BPF.** *In Linux Plumbers Conference, Richmond, VA, November 2023.* Siddharth Chintamaneni, Sai Roop Somaraju, and Dan Williams

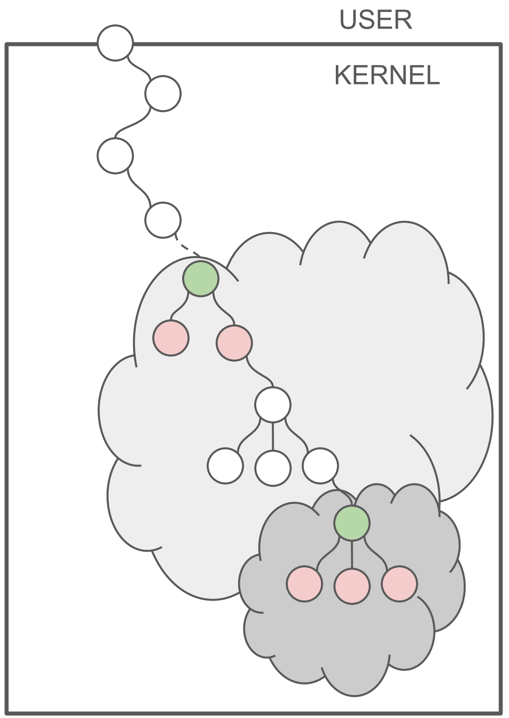
Root cause:

verifier does not know
enough about **composition**
of extensions
with kernel and each other



Callgraph-based solution

- Can we teach verifier about composition?
- Key idea: statically compute helper/kfunc rooted callgraph
  - Compute possible stack usage per node
  - Dynamically track changing callgraphs as extensions are attached and nested



Key Challenges and Approaches

- How to generate callgraphs of Linux
  - Lots of indirect calls
- Idea: limit focus to helper functions to start
  - More tractable than entire kernel
  - Utilize state-of-the-art type-based inference tools
  - Eliminate indirect calls in helpers when possible
  - Focus on common helpers/nesting use cases

Summary

- Safety involves more than just the BPF program
- Unsafe composition with kernel and nested BPF programs
- Callgraph approach can catch such issues

Summary

- Safety involves more than just the BPF program
- Unsafe composition with kernel and nested BPF programs
- Callgraph approach can catch such issues

THANK YOU!!

sidchintamaneni@vt.edu
djwillia@vt.edu