Flexible and Low-Overhead System-Call Aggregation using BPF

FGBS2022-Spring Talk
March 17, 2022

Luis Gerhorst, Benedict Herzog, Stefan Reif, Wolfgang Schröder-Preikschat, Timo Hönig

Ruhr-Universität Bochum
Friedrich-Alexander-Universität Erlangen-Nürnberg

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

RUHR UNIVERSITÄT BOCHUM

RUB

Introduction

Operating Systems

- Isolation: safety, security
- Communication: IO, IPC

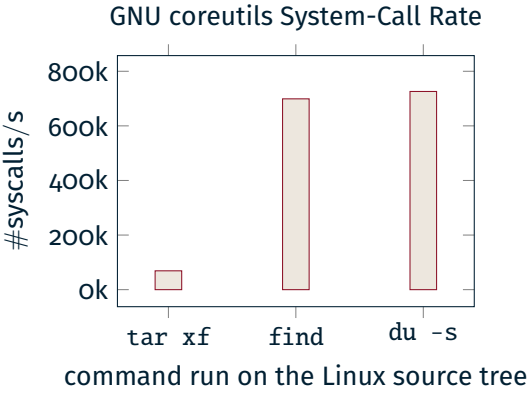
System Calls

- (+) Controlled, privileged operations
- (+) Stable interface
- (-) User/kernel transition overhead

Motivation

High system-call rates of

- HTTP servers
- `memcached`
- Interactive workloads
- GNU coreutils
- …

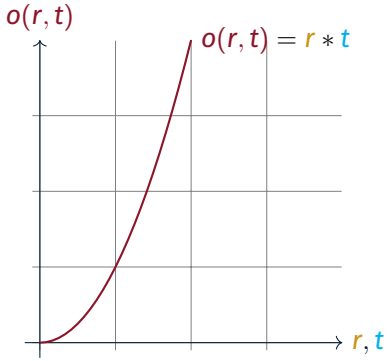


Problem

Increasing Syscall Overheads $o(r, t)$

- System-call rate r
- Transition overhead t

$\rightarrow$ System-Call Aggregation

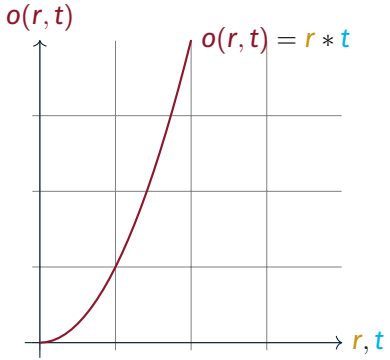


Approach

Increasing Syscall Overheads $o(r, t)$

$\rightarrow$ System-Call Aggregation

- Fast: one transition, many system calls
- Flexible: programmable



Outline

System Calls

- Problem
- Approach

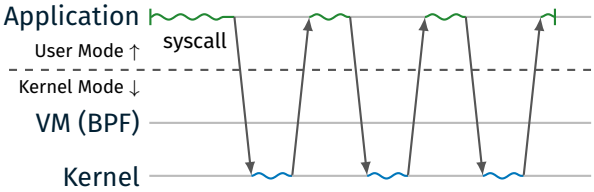
System-Call Aggregation

BPF

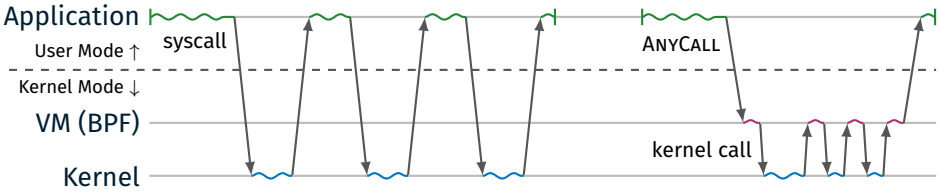
AnyCall

- Architecture
- Programming Model
- Evaluation

System-Call Aggregation

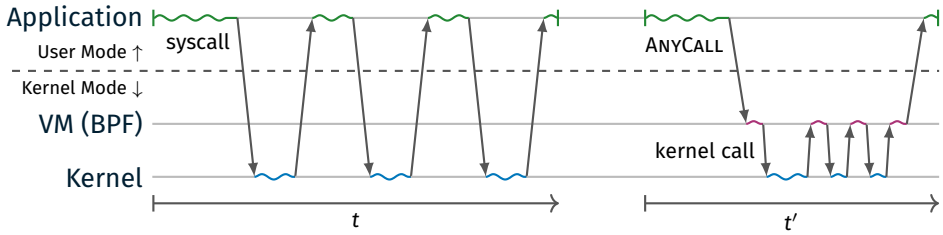


System-Call Aggregation



AnyCall: Kernel Calls without User/Kernel Transitions

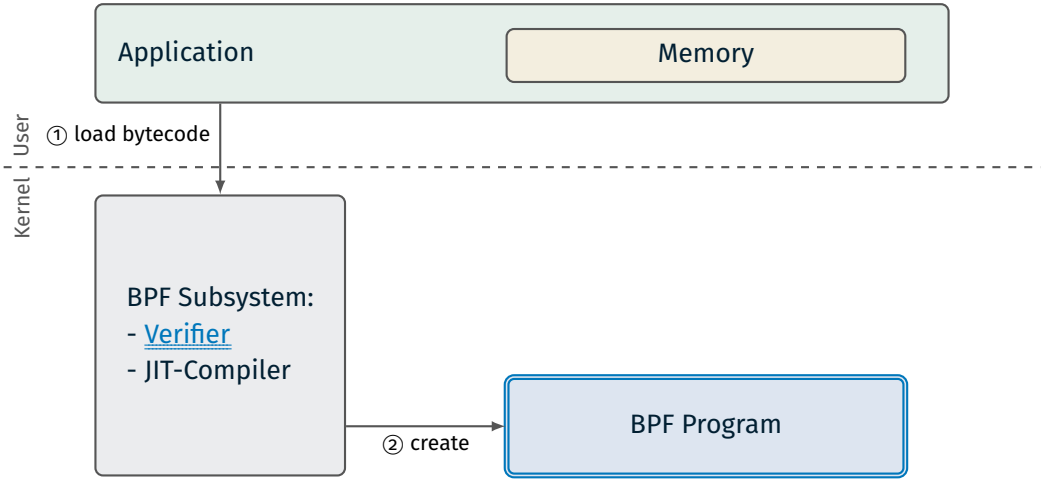
System-Call Aggregation



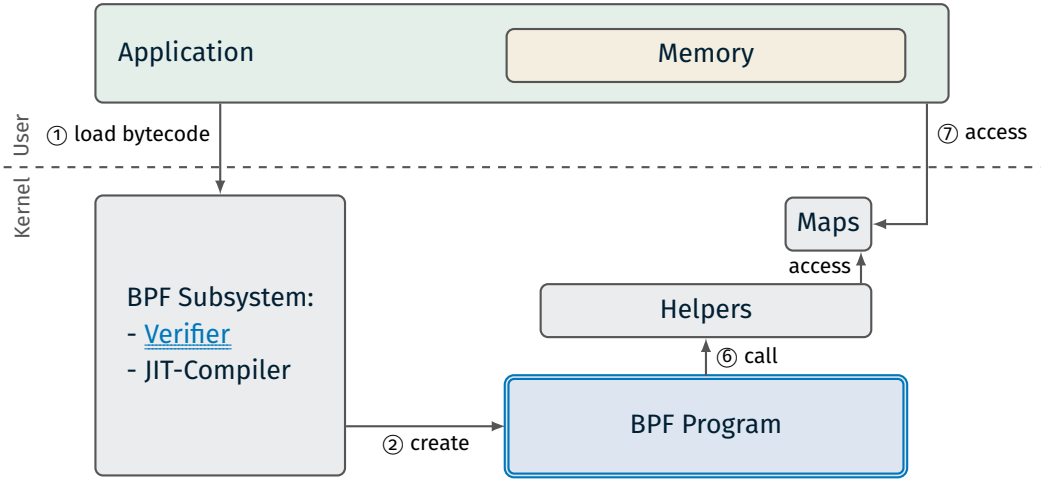
AnyCall: Kernel Calls without User/Kernel Transitions

- Reduced direct overhead
- Cache stays hot → faster user and kernel code

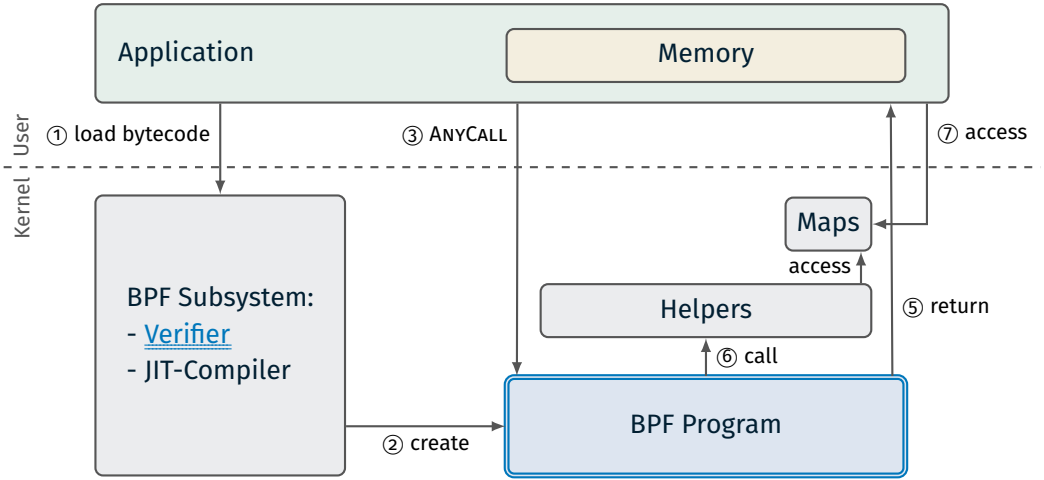
Extended Berkeley Packet Filter (BPF)



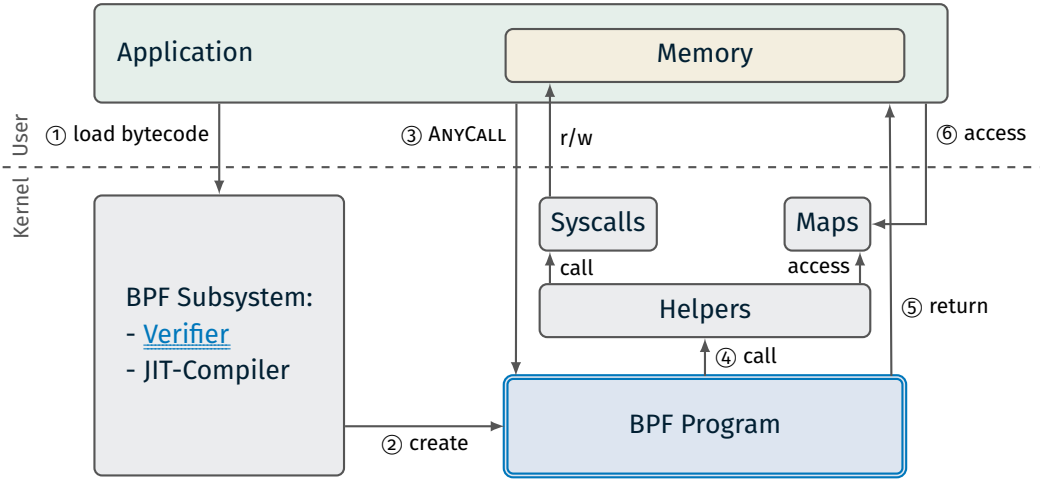
Extended Berkeley Packet Filter (BPF)



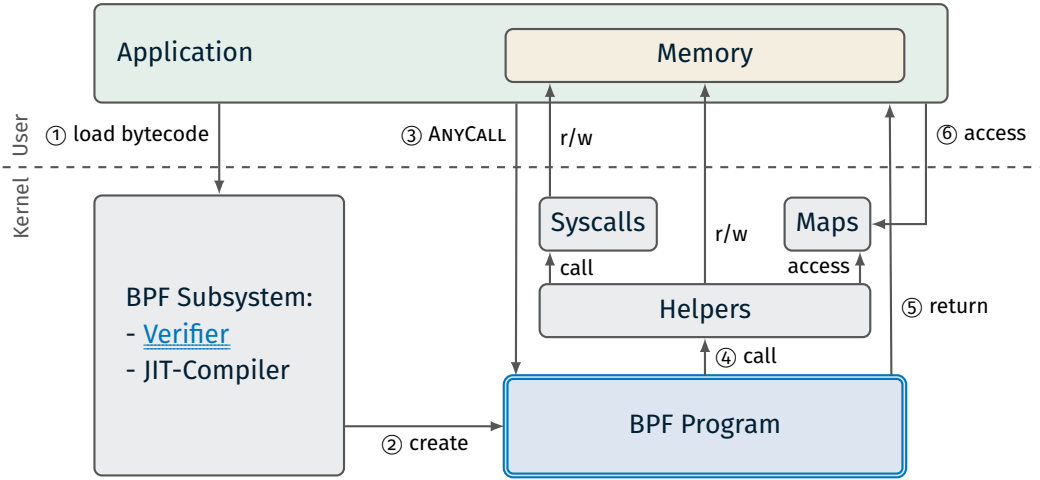
AnyCall


Application
Memory
User
Kernel
① load bytecode
③ AnyCall
⑦ access
Maps
access
BPF Subsystem:
- Verifier
- JIT-Compiler
Helpers
⑤ return
⑥ call
② create
BPF Program

AnyCall


Application
Memory
User
Kernel
① load bytecode
③ AnyCall
r/w
⑥ access
Syscalls
Maps
call
access
Helpers
⑤ return
④ call
BPF Subsystem:
- Verifier
- JIT-Compiler
② create
BPF Program

AnyCall


Application
Memory
User
Kernel
① load bytecode
③ AnyCall
r/w
⑥ access
Syscalls
r/w
Maps
call
access
BPF Subsystem:
- Verifier
- JIT-Compiler
Helpers
⑤ return
④ call
BPF Program
② create

Outline

System Calls
- Problem
- Approach

System-Call Aggregation

BPF

AnyCall
- Architecture
- Programming Model
- Evaluation

Disk Usage Estimation Example

Traditional System Calls

- C program:

```
int[N] files;      // input
size_t total = 0;

struct stat s;
for (i = 0; i < n; i++) {
    fstat(files[i], &s);

    total += s.st_size;
}

gui_display(total);
```

Disk Usage Estimation Example

AnyCall

- C program:

```
gui_display(total);
```

- BPF program:

```
int[N] files;
size_t total = 0;

struct stat s;
for (i = 0; i < n && i < N; i++) {
    fstat(files[i], uaddr);
    copy(&s, sizeof(s), uaddr);
    total += s.st_size;
}
```

Disk Usage Estimation Example

AnyCall

- C program:

```
// write bpfprog->files,uaddr
int fd = load(bpfprog);
anycall(fd);



gui_display(bpfprog->total);
```

- BPF program:

```
int[N] files;
size_t total = 0;

struct stat s;
for (i = 0; i < n && i < N; i++) {
    fstat(files[i], uaddr);
    copy(&s, sizeof(s), uaddr);
    total += s.st_size;
}
```

Outline

System Calls

- Problem
- Approach

System-Call Aggregation

BPF

AnyCall

- Architecture
- Programming Model
- Evaluation

Evaluation

Benchmarks

- Vector and `getpid()` benchmarks
- Real-world applications
  - File searching
  - Disk usage estimation

Evaluation

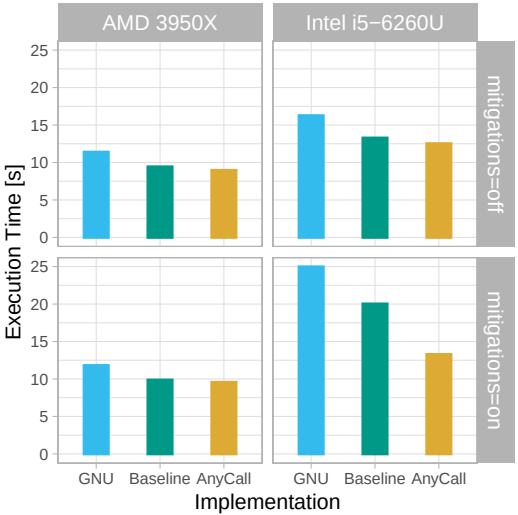
Benchmarks

- Vector and `getpid()` benchmarks
- Real-world applications
  - File searching
  - Disk usage estimation

Platforms and Mitigated Vulnerabilities

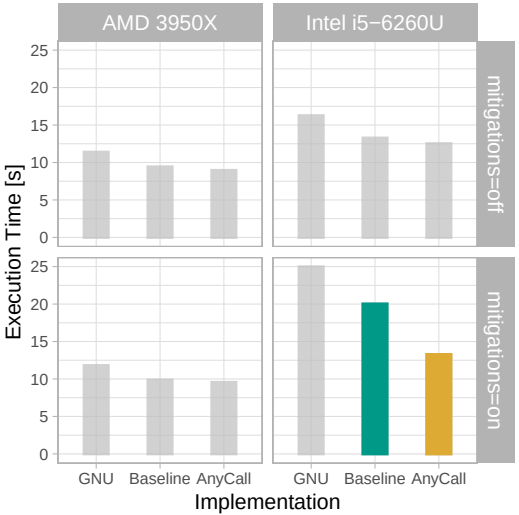
	AMD 3950X (2019)	Intel i5-6260U (2015)
`mitigations=off`	-	L1TF
`mitigations=on`	Spectre, SSB	Meltdown, L1TF, MDS, Spectre, SSB

Disk Usage Estimation Tool



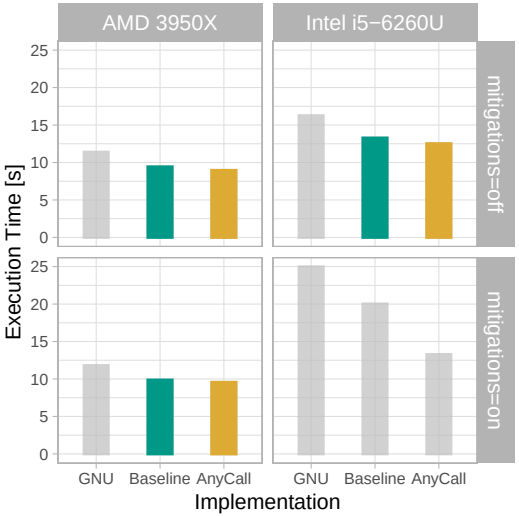
- Estimate disk usage of the Poky Linux distribution build tree
  - 1.7M files, 48GB
  - Warm page cache
- 34 % speedup with KPTI
- 3.1 % to 5.7 % speedup without KPTI

Disk Usage Estimation Tool



- Estimate disk usage of the Poky Linux distribution build tree
  - 1.7M files, 48GB
  - Warm page cache
- 34 % speedup with KPTI
- 3.1 % to 5.7 % speedup without KPTI

Disk Usage Estimation Tool



- Estimate disk usage of the Poky Linux distribution build tree
  - 1.7M files, 48GB
  - Warm page cache
- 34 % speedup with KPTI
- 3.1 % to 5.7 % speedup without KPTI

Summary

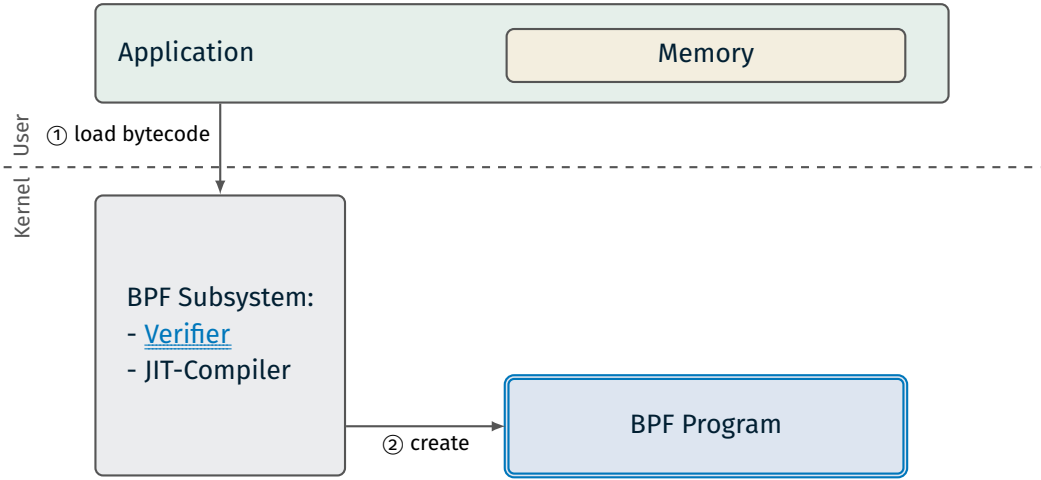
Problem: Increasing user/kernel-transition-overheads for applications

Approach: Programmable system-call aggregation

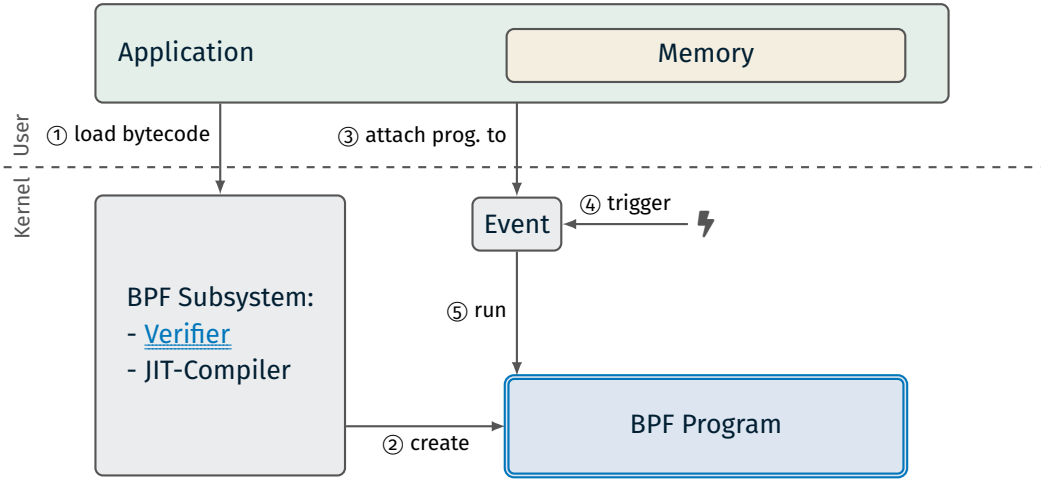
AnyCall:
- ✓ *Fast:* up to 34 % speedup with KPTI
- ✓ *Flexible:* programmable, per-application capability
- ✓ *Safe:* checked by verifier
- ✓ Extensible Linux/BPF-based implementation

[Ger+21] Luis Gerhorst, Benedict Herzog, Stefan Reif, Wolfgang Schröder-Preikschat, and Timo Hönig. "AnyCall: Fast and Flexible System-Call Aggregation". In: *Proceedings of the 11th Workshop on Programming Languages and Operating Systems*. PLOS'21. 2021, pp. 1–8

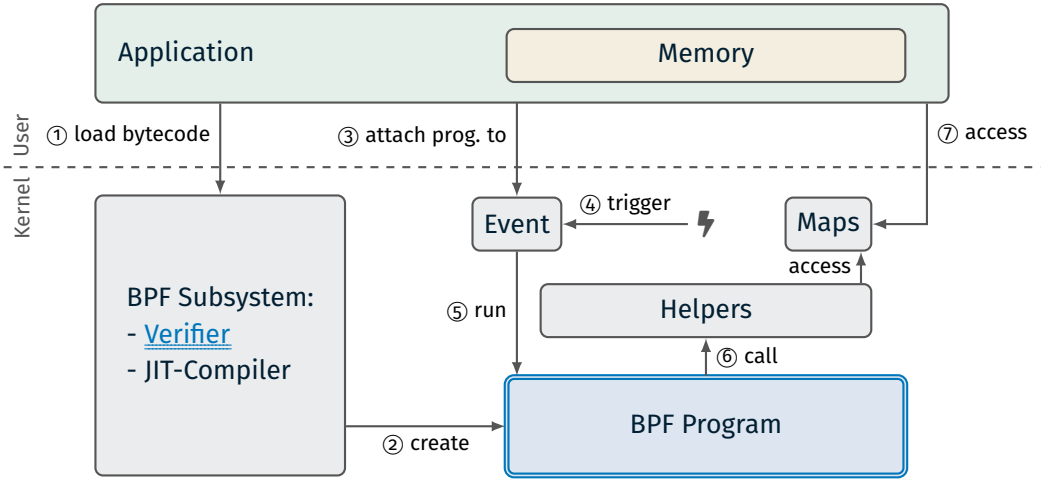
Extended Berkeley Packet Filter (BPF)



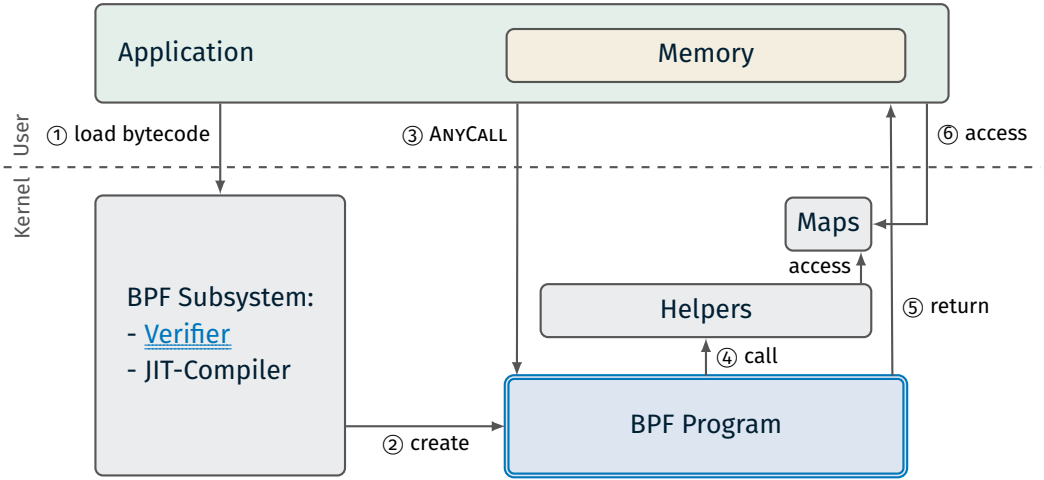
Extended Berkeley Packet Filter (BPF)



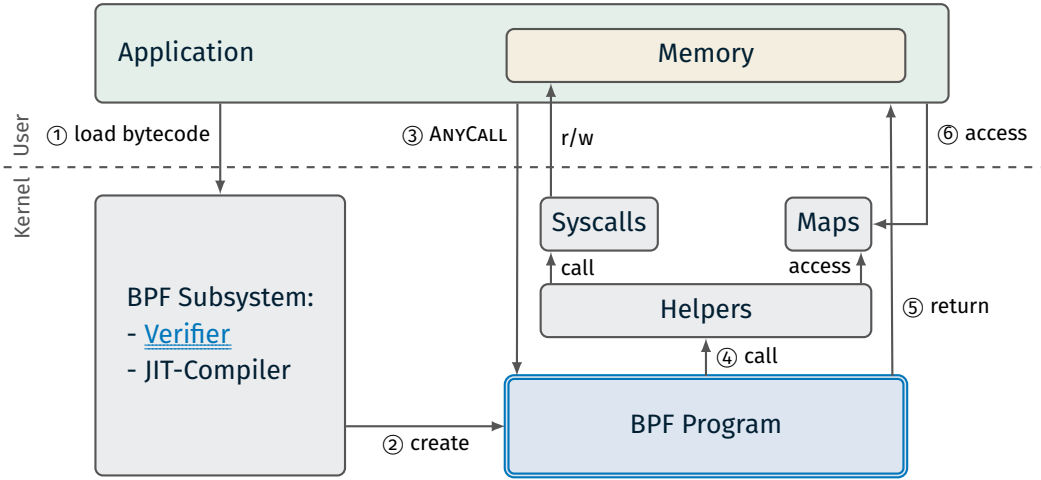
Extended Berkeley Packet Filter (BPF)



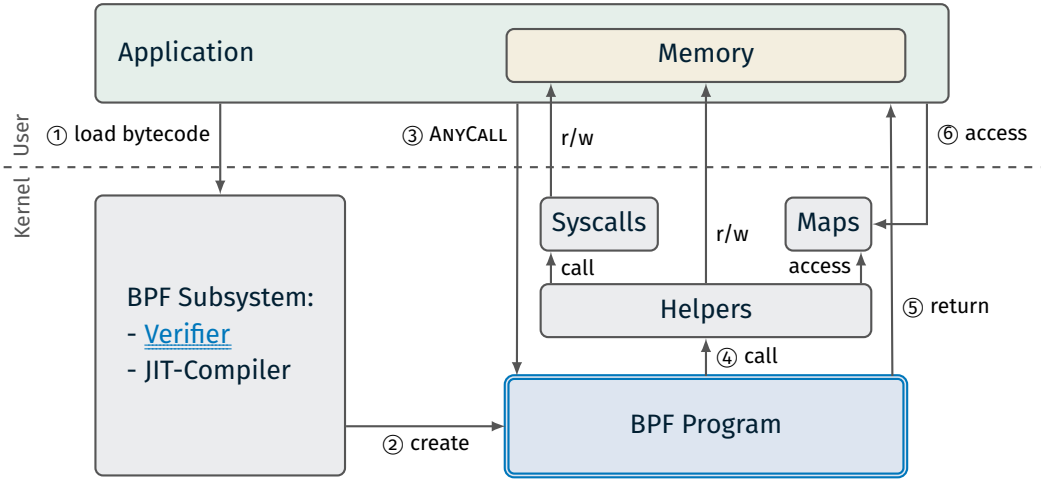
AnyCall


Application
Memory
User
Kernel
① load bytecode
③ AnyCall
⑥ access
Maps
access
BPF Subsystem:
- Verifier
- JIT-Compiler
Helpers
⑤ return
④ call
BPF Program
② create

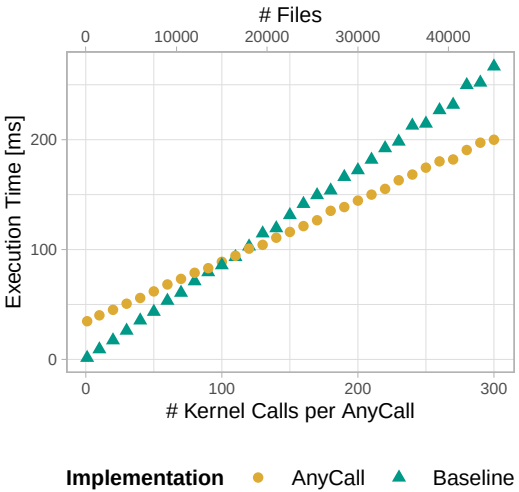
AnyCall


Application
Memory
User
Kernel
① load bytecode
③ AnyCall
r/w
⑥ access
Syscalls
Maps
call
access
Helpers
⑤ return
④ call
BPF Subsystem:
- Verifier
- JIT-Compiler
② create
BPF Program

AnyCall

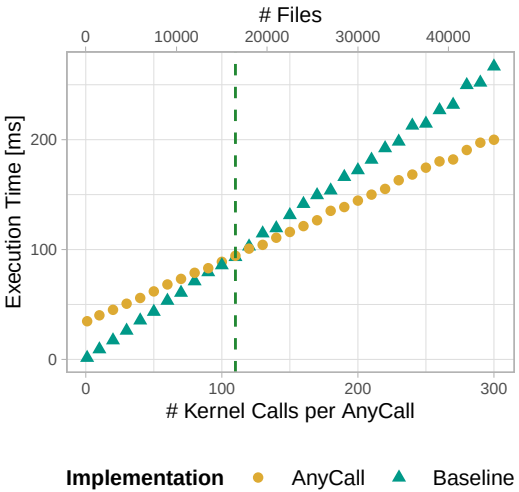

Application
Memory
User
Kernel
① load bytecode
③ AnyCall
r/w
⑥ access
Syscalls
r/w
Maps
call
access
BPF Subsystem:
- Verifier
- JIT-Compiler
Helpers
⑤ return
④ call
BPF Program
② create

Vector AnyCalls (Intel i5-6260U, Mitigations Active)



- open/close a requested number of temporary files
- two implementations
  - load AnyCall, invoke 150×
  - traditional syscalls

Vector AnyCalls (Intel i5-6260U, Mitigations Active)



- open/close a requested number of temporary files
- two implementations
  - load AnyCall, invoke 150×
  - traditional syscalls
- amortization of loading overhead
  - 16 500 trad. syscalls
  - GNU coreutils:
    70 000 to 726 000 syscalls/s
    when processing the Linux sources

References (1)

[Jou+01] Philippe Joubert et al. “High-Performance Memory-Based Web Servers: Kernel and User-Space Performance”. In: *USENIX Annual Technical Conference, General Track*. 2001, pp. 175–187.

[KS15] Ake Koomsin and Yasushi Shinjo. “Running application specific kernel code by a just-in-time compiler”. In: *Proceedings of the 8th Workshop on Programming Languages and Operating Systems*. Monterey California: ACM, Oct. 2015. ISBN: 9781450339421. DOI: 10.1145/2818302.2818305. URL: http://dx.doi.org/10.1145/2818302.2818305.

References (2)

[Zad+05] E Zadok et al. “Efficient and safe execution of user-level code in the kernel”. In: *19th IEEE International Parallel and Distributed Processing Symposium*. Denver, CO, USA: IEEE, 2005. ISBN: 9780769523125. DOI: 10.1109/ipdps.2005.189. URL: http://dx.doi.org/10.1109/ipdps.2005.189.

[Ger+21] Luis Gerhorst et al. “AnyCall: Fast and Flexible System-Call Aggregation”. In: *Proceedings of the 11th Workshop on Programming Languages and Operating Systems*. PLOS’21. 2021, pp. 1–8.