Artifacts Available acm V1.1

Artifacts Evaluated acm Functional V1.1

Results Reproduced acm V1.1

Revealing the Unstable Foundations of eBPF-Based Kernel Extensions

Shawn Zhong, Jing Liu, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Microsoft

eBPF: Powerful Framework to Extend Kernel

eBPF: Framework to safely extend Linux kernel functionality

- 🚀 Run custom programs in kernel, triggered by hooks (e.g., on function entry point)
- 🔍 Read and traverse internal kernel data structures (via in-kernel helper function)

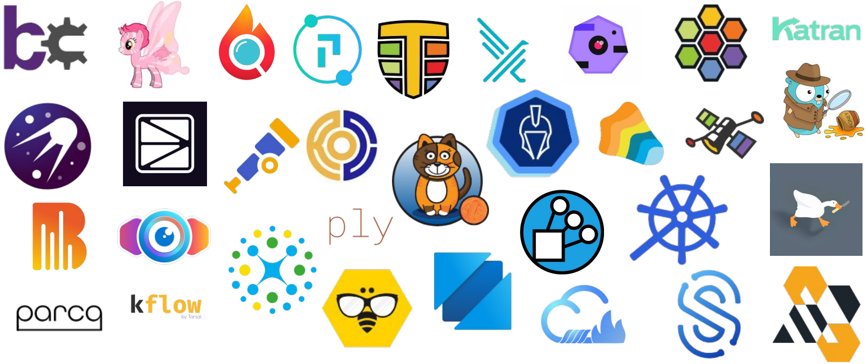
eBPF: Popularity and Use Cases

eBPF: Framework to safely extend Linux kernel functionality

- 🚀 Run custom programs in kernel, triggered by hooks (e.g., on function entry point)
- 🔍 Read and traverse internal kernel data structures (via in-kernel helper function)

Use cases

- 👀 Observability: trace kernel functions
- 🔒 Security: enforce security policies
- 🌐 Network: filter packets

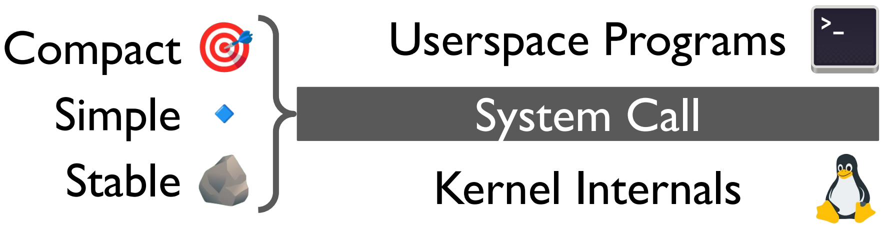


Problem: eBPF Programs Depend on Kernel Internals

Userspace programs depend on syscall

eBPF programs depend on kernel internals

- E.g., functions & structs
- Large, complex, unstable



Compact 🎯

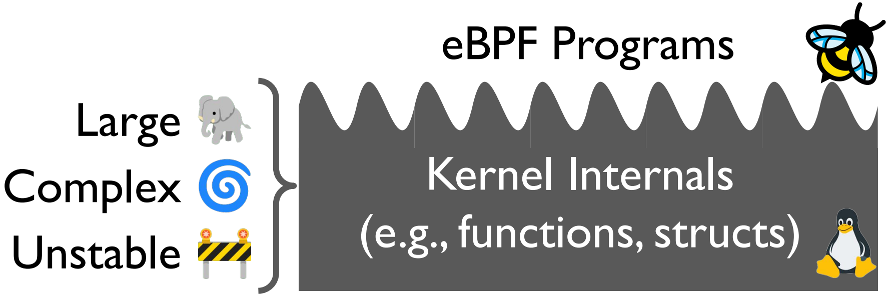
Simple 🔹

Stable 🪨

Userspace Programs

System Call

Kernel Internals 🐧



eBPF Programs 🐝

Large 🐘

Complex 🌀

Unstable 🚧

Kernel Internals

(e.g., functions, structs) 🐧

Problem: Unstable Kernel Internals ⇒ Unportable eBPF Program

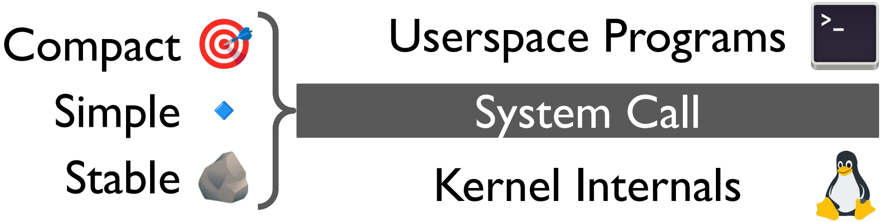
Userspace programs depend on syscall

eBPF programs depend on kernel internals

- E.g., functions & structs
- Large, complex, unstable

Impact on eBPF programs

- Fundamentally unportable
- Frequently break on different kernels
- Unclear if it work on another kernel?

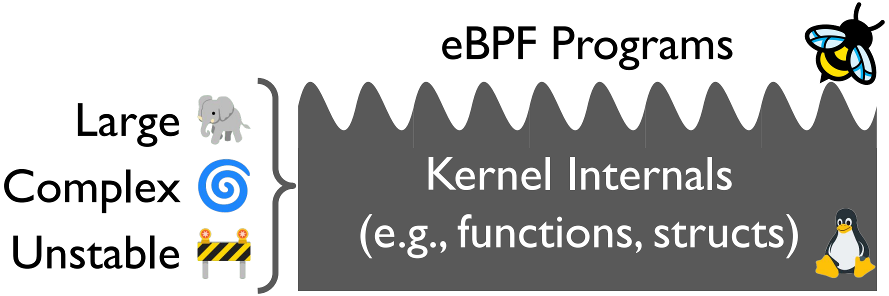


Compact 🎯
Simple 🔹
Stable 🪨

Userspace Programs

System Call

Kernel Internals



eBPF Programs

Large 🐘
Complex 🌀
Unstable 🚧

Kernel Internals
(e.g., functions, structs)

Our Contribution: DepSurf

DepSurf: a tool to analyze dependency mismatches between

- Program Dependency Set: a set of dependencies used by an eBPF program

Program Dependency Set

```
void foo(int i)
```

Our Contribution: DepSurf

DepSurf: a tool to analyze dependency mismatches between

- Program Dependency Set: a set of dependencies used by an eBPF program
- Kernel Dependency Surface: all dependencies exposed by a kernel

Program Dependency Set

```
void foo(int i)
```

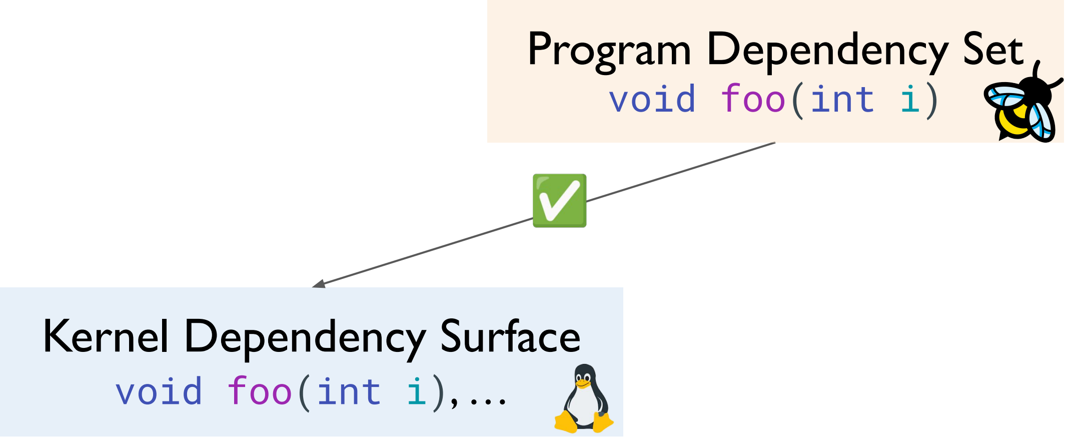
Kernel Dependency Surface

```
void foo(int i),...
```

Our Contribution: DepSurf

DepSurf: a tool to analyze dependency mismatches between

- Program Dependency Set: a set of dependencies used by an eBPF program
- Kernel Dependency Surface: all dependencies exposed by a kernel



Program Dependency Set

`void foo(int i)`

✅

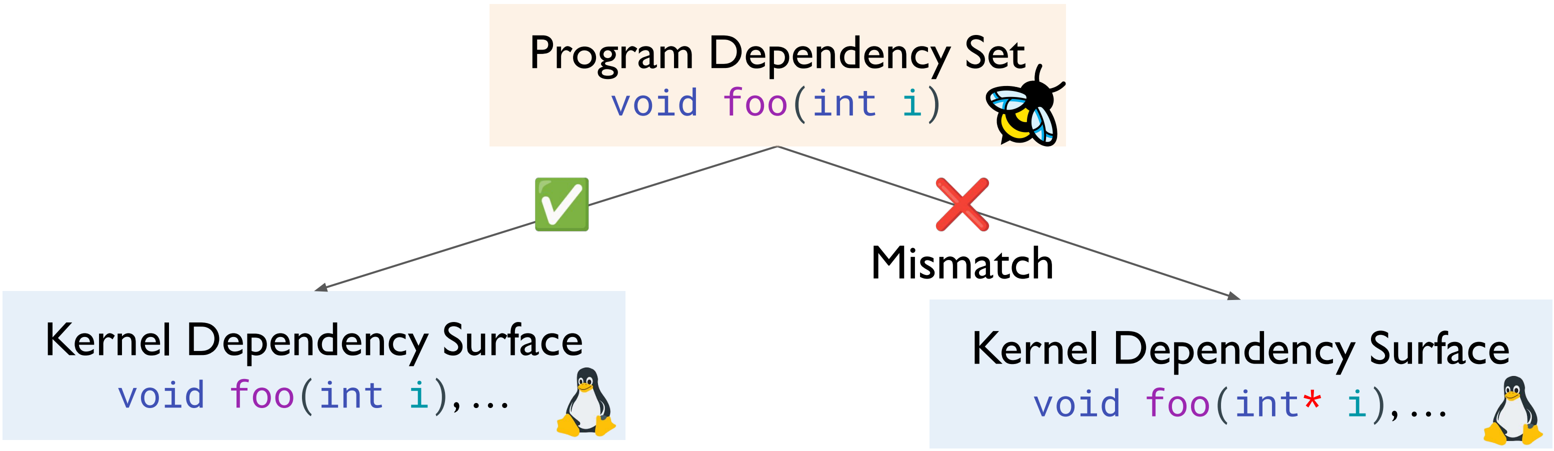
Kernel Dependency Surface

`void foo(int i),...`

Our Contribution: DepSurf

DepSurf: a tool to analyze dependency mismatches between

- Program Dependency Set: a set of dependencies used by an eBPF program
- Kernel Dependency Surface: all dependencies exposed by a kernel



Program Dependency Set

```
void foo(int i)
```

✅

❌ Mismatch

Kernel Dependency Surface

```
void foo(int i),...
```

Kernel Dependency Surface

```
void foo(int* i),...
```

Kernel Dependency Surface Analysis 🐧

Analyzed 25 kernel images ⇒ Kernel dependency surface is highly unstable

- 📝 Kernel Source Code
- ⚙️ Kernel Configuration
- 📦 Kernel Compilation

Kernel Dependency Surface Analysis 🐧

Analyzed 25 kernel images ⇒ Kernel dependency surface is highly unstable

- 📝 Kernel Source Code
- ⚙️ Kernel Configuration
- 📦 Kernel Compilation

Consequences for eBPF programs

- 🚨 Fail to compile, load, or attach ⇒ Explicit error
- 🗑️ Stray read ⇒ Incorrect garbage results
- 🤔 Missing invocation ⇒ Apparently correct but incomplete results

Program Dependency Set Analysis

Analyzed 53 eBPF programs ⇒ All depend on some unstable kernel internals

- Majority depend on internal structs and fields
- Half depend on internal functions
- Half depend on kernel tracepoints (i.e., static markers)

Program Dependency Set Analysis

Analyzed 53 eBPF programs ⇒ All depend on some unstable kernel internals

- Majority depend on internal structs and fields
- Half depend on internal functions
- Half depend on kernel tracepoints (i.e., static markers)

Dependency mismatches are widespread (83%)

- Function optimization
- Missing fields in structs
- Changed tracepoints

Outline

Introduction

Motivation

Dependency Mismatch

DepSurf

- Kernel Dependency Surface Analysis
- Program Dependency Set Analysis

Conclusion

eBPF Portability: Misconceptions and Expectations

🧑‍💻 eBPF Dev

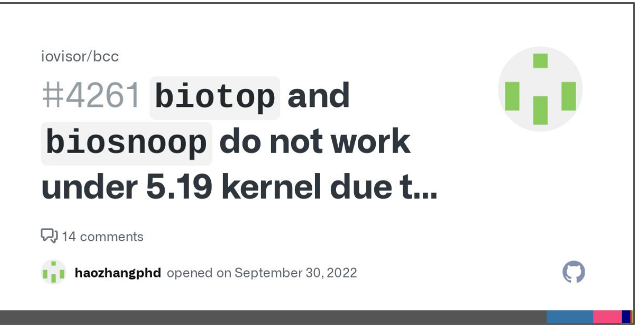
"[The eBPF infrastructure] guarantees that existing eBPF programs keep running with newer kernel versions"

"eBPF programs are portable across different architectures"

"[Their product] runs on most common Linux distributions and kernels."

🧑‍💻 eBPF Dev

eBPF Portability: Reality

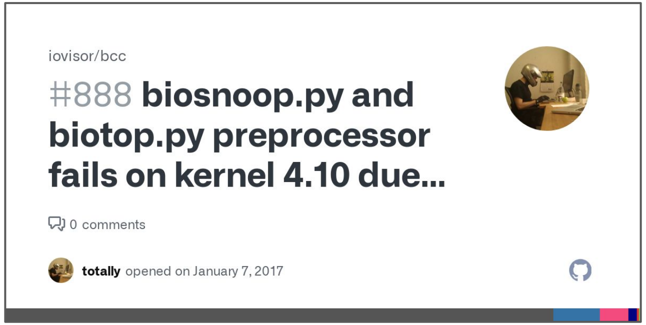


iovisor/bcc

#4261 `biotop` and `biosnoop` do not work under 5.19 kernel due t...

14 comments

haozhangphd opened on September 30, 2022

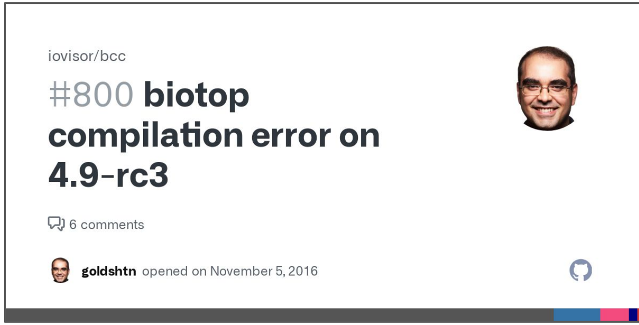


iovisor/bcc

#888 biosnoop.py and biotop.py preprocessor fails on kernel 4.10 due...

0 comments

totally opened on January 7, 2017

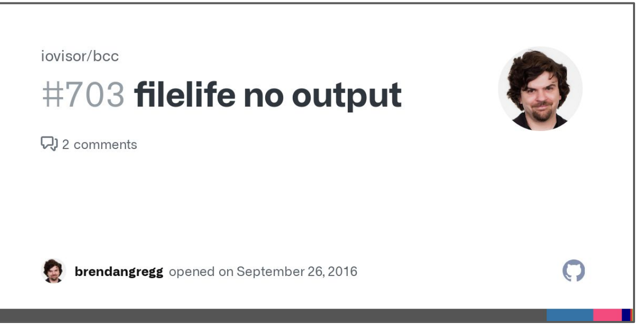


iovisor/bcc

#800 biotop compilation error on 4.9-rc3

6 comments

goldshtn opened on November 5, 2016

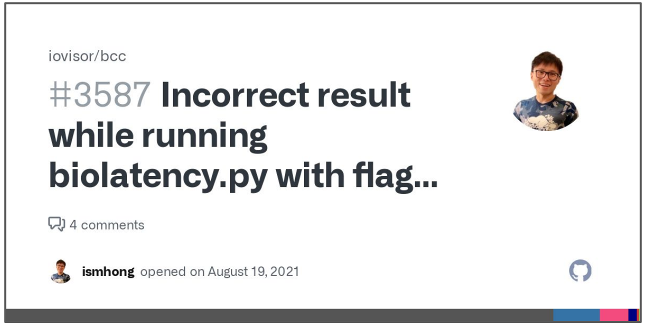


iovisor/bcc

#703 filelife no output

2 comments

brendangregg opened on September 26, 2016

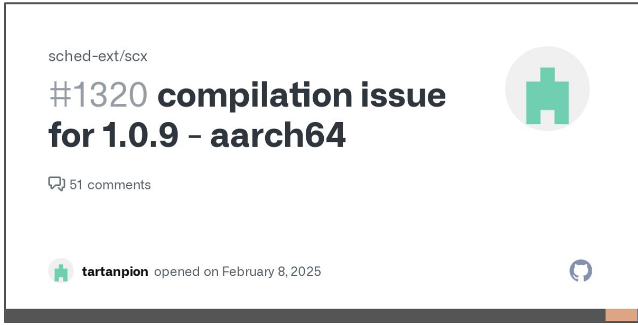


iovisor/bcc

#3587 Incorrect result while running biolatency.py with flag...

4 comments

ismhong opened on August 19, 2021

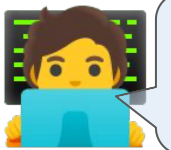


sched-ext/scx

#1320 compilation issue for 1.0.9 - aarch64

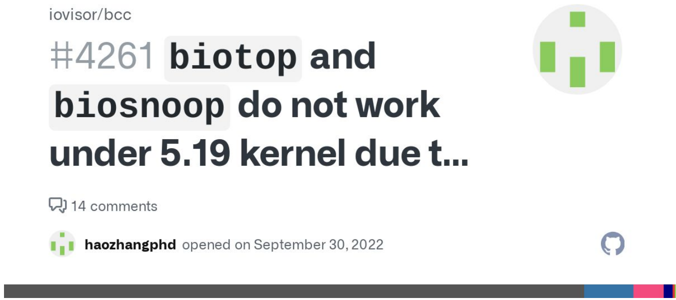
51 comments

tartanpion opened on February 8, 2025



"It is difficult to write eBPF programs that work correctly on all kernels."

Case Study: biotop



iovisor/bcc

#4261 `biotop` and `biosnoop` do not work under 5.19 kernel due t...

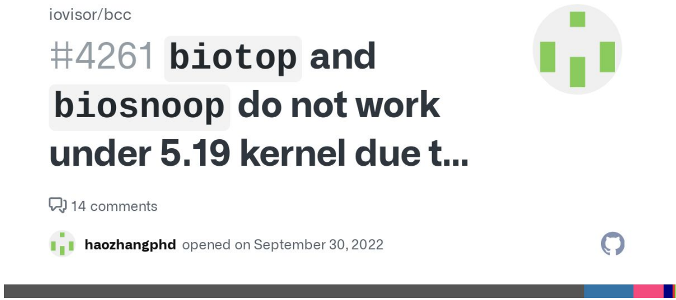
14 comments

haozhangphd opened on September 30, 2022

`biotop`: trace block I/O operations

- Whenever the kernel function `blk_account_io_start` is called
- Access 1st arg `struct request *req` and read field `req->__data_len`

Case Study: biotop



iovisor/bcc

#4261 biotop and biosnoop do not work under 5.19 kernel due t...

14 comments

haozhangphd opened on September 30, 2022

biotop: trace block I/O operations

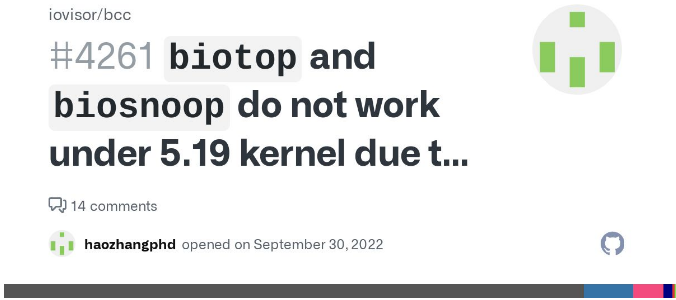
- Whenever the kernel function blk_account_io_start is called
- Access 1st arg struct request *req and read field req->__data_len

Issue #4261

- On 5.19 kernel, the function blk_account_io_start is missing
- As a result, biotop fail to start: attachment error

User
Bug
Report

Case Study: biotop



iovisor/bcc

#4261 `biotop` and `biosnoop` do not work under 5.19 kernel due t...

14 comments

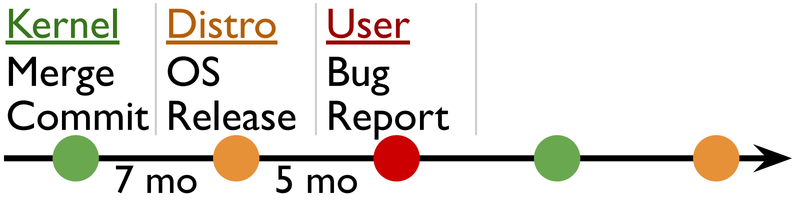
haozhangphd opened on September 30, 2022

`biotop`: trace block I/O operations

- Whenever the kernel function `blk_account_io_start` is called
- Access 1st arg `struct request *req` and read field `req->__data_len`

Cause: commit "block: inline hot paths of blk_account_io_*()"

- `void blk_account_io_start()` →
- `static inline void blk_account_io_start()`



Kernel
Merge Commit

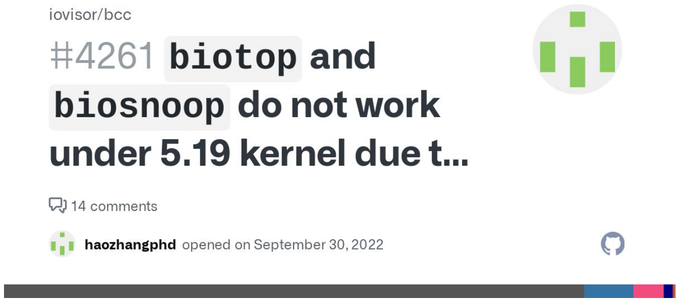
Distro
OS Release

User
Bug Report

7 mo

5 mo

Case Study: biotop



iovisor/bcc

#4261 `biotop` and `biosnoop` do not work under 5.19 kernel due t...

14 comments

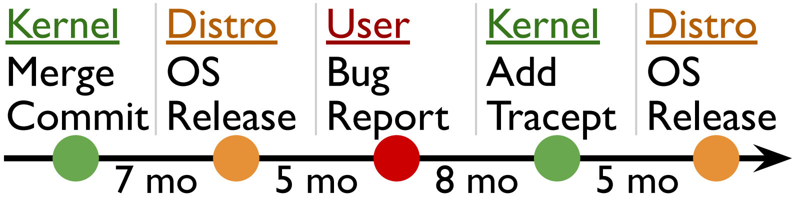
haozhangphd opened on September 30, 2022

`biotop`: trace block I/O operations

- Whenever the kernel function `blk_account_io_start` is called
- Access 1st arg `struct request *req` and read field `req->__data_len`

Solution

- 8 months: Tracepoint `block_io_start` added to kernel
- 5 months: New OS version released with newer kernel



Kernel
Merge Commit

Distro
OS Release

User
Bug Report

Kernel
Add Tracept

Distro
OS Release

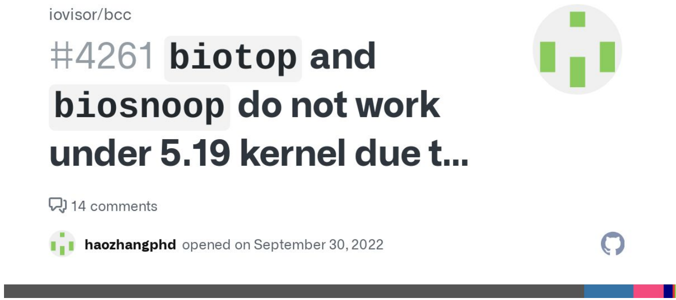
7 mo

5 mo

8 mo

5 mo

Case Study: biotop



iovisor/bcc

#4261 `biotop` and `biosnoop` do not work under 5.19 kernel due t...

14 comments

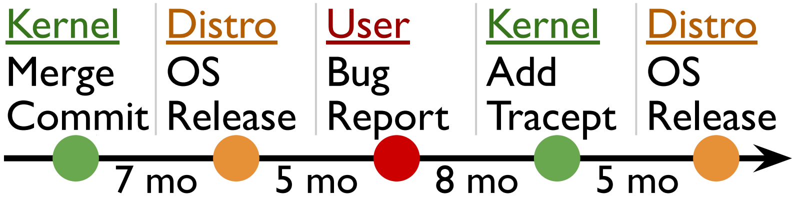
haozhangphd opened on September 30, 2022

biotop: trace block I/O operations

- Whenever the kernel function `blk_account_io_start` is called
- Access 1st arg `struct request *req` and read field `req->__data_len`

Changing kernel is difficult

- 2 years end-to-end
- `biotop` still broken on older kernels



Kernel
Merge Commit

7 mo

Distro
OS Release

5 mo

User
Bug Report

8 mo

Kernel
Add Tracept

5 mo

Distro
OS Release

Outline

Introduction

Motivation

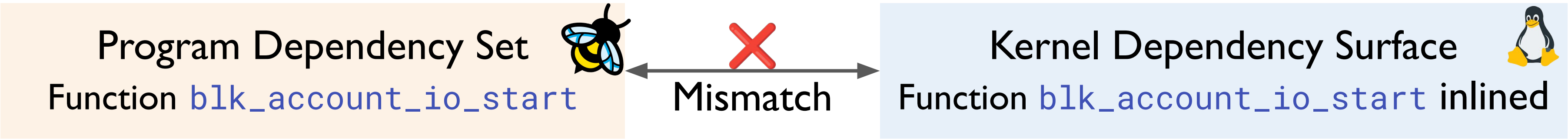
Dependency Mismatch

DepSurf

- Kernel Dependency Surface Analysis
- Program Dependency Set Analysis

Conclusion

Dependency Mismatch ⇐ Unstable Kernel Dependency Surface



Program Dependency Set

Function `blk_account_io_start`

❌

Mismatch

Kernel Dependency Surface

Function `blk_account_io_start` inlined

Dependency mismatch caused by unstable kernel dependency surface

- >35k functions and >5k structs are unstable but widely used
- 500-1000 tracepoints are desired to be stable, but are not

Brendan Gregg:
“Tracepoints provide a stable API.”

Brendan Gregg:
“I have seen tracepoints change.”

Outline

Introduction

Motivation

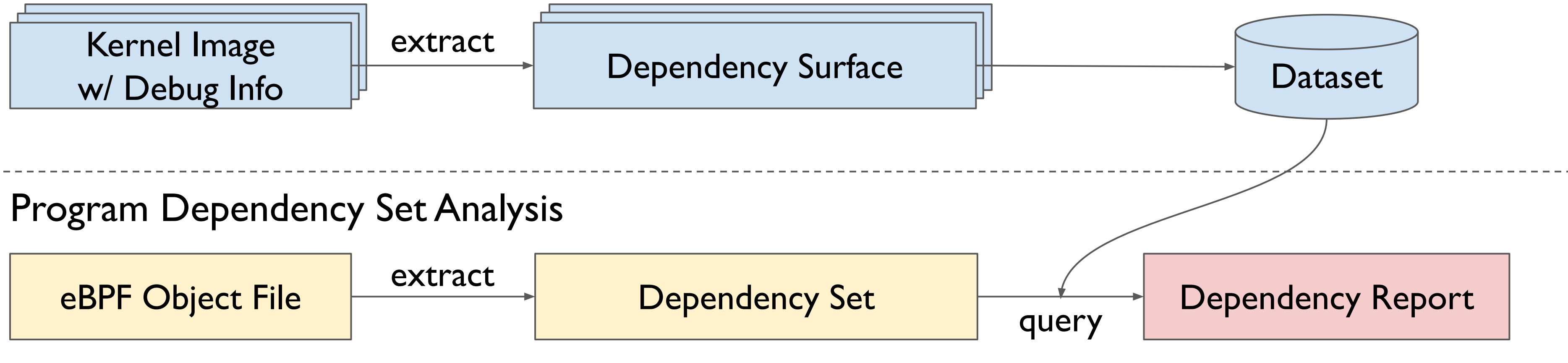
Dependency Mismatch

DepSurf

- Kernel Dependency Surface Analysis
- Program Dependency Set Analysis

Conclusion

DepSurf

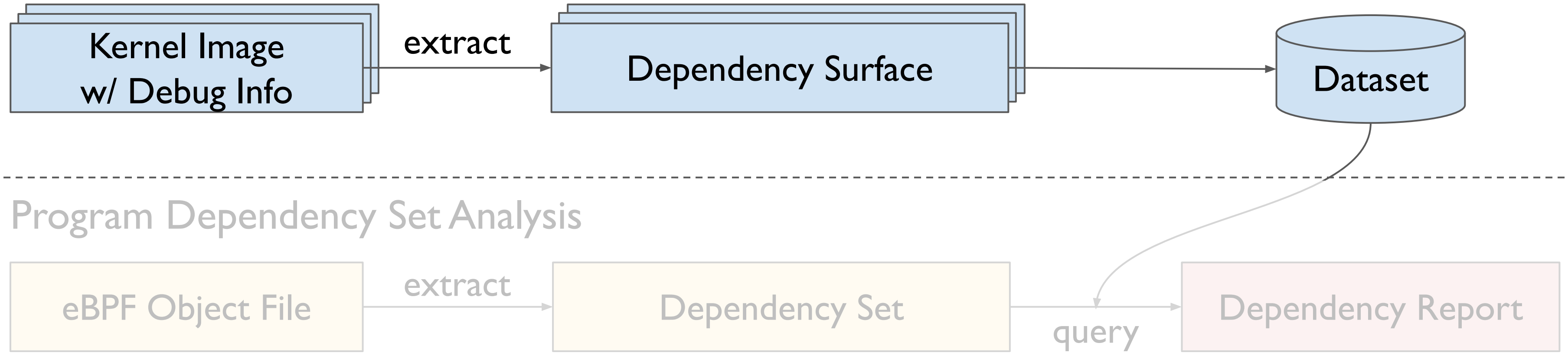


Kernel Dependency Surface Analysis
Kernel Image w/ Debug Info
extract
Dependency Surface
Dataset
Program Dependency Set Analysis
eBPF Object File
extract
Dependency Set
query
Dependency Report

See paper for implementation

DepSurf

Kernel Dependency Surface Analysis



Kernel Image w/ Debug Info

extract

Dependency Surface

Dataset

Program Dependency Set Analysis

eBPF Object File

extract

Dependency Set

query

Dependency Report

Causes of Unstable Kernel Dependency Surface

Kernel Source Code

- New features (e.g., folio)
- Depreciations (e.g., single-queue bio)
- Perf. optimization
- Bug fixes

Kernel Configuration

- Arch-specific definitions (e.g., register, syscall)
- Features (e.g., NUMA)
- Parameters (e.g., timer)
- Set by OS distro

Kernel Compilation

- Function Optimizations (e.g., inline)
- Driven by compiler
- Opaque to developers

Causes of Unstable Kernel Dependency Surface

Kernel Source Code

- New features (e.g., folio)
- Depreciations (e.g., single-queue bio)
- Perf. optimization
- Bug fixes

Kernel Configuration

- Arch-specific definitions (e.g., register, syscall)
- Features (e.g., NUMA)
- Parameters (e.g., timer)
- Set by OS distro

Kernel Compilation

- Function Optimizations (e.g., inline)
- Driven by compiler
- Opaque to developers

Causes of Unstable Kernel Dependency Surface

Kernel Source Code

- New features (e.g., folio)
- Depreciations (e.g., single-queue bio)
- Perf. optimization
- Bug fixes

Kernel Configuration

- Arch-specific definitions (e.g., register, syscall)
- Features (e.g., NUMA)
- Parameters (e.g., timer)
- Set by OS distro

Kernel Compilation

- Function Optimizations (e.g., inline)
- Driven by compiler
- Opaque to developers

Kernel Dependency Surface Dataset

25 kernel images build by Ubuntu

- 17 versions over 8 years
  - Linux kernel 4.4 to 6.8
  - Ubuntu 16.04 to 24.04
- 5 architectures and 5 build flavors
  - x86, arm64, arm32, PowerPC, RISC-V
  - Generic, Low-latency, AWS, Azure, GCP
- 14 compiler versions

Highly extensible: effortless to add new kernel images

Summary of Dependency Mismatches

Kernel Source Code

Function: Absence, Change

Struct: Absence, Change

Tracepoint: Absence, Change

Kernel Configuration

Function: Absence, Change

Struct: Absence, Change

Tracepoint: Absence

Syscall: Availability, Traceability

Register: Layout Difference

Kernel Compilation

Function:

- Full / Selective Inline
- Transformation
- Duplication
- Name Collision

📝 Kernel Source Code: Function

Function absence

Example: page ➡️ folio

- `account_page_dirtied` ➡️ `folio_account_dirtied`
- `migrate_misplaced_page` ➡️ `migrate_misplaced_folio`
- `mark_page_accessed` ➡️ `folio_mark_accessed`
- `...`

Every 2 years, 24% functions added, 10% functions removed

Takeaway: Kernel functions are constantly added and removed, causing explicit attachment error for dependent eBPF programs

📝 Kernel Source Code: Function

Function signature change

- Parameter added or removed
  - `int vfs_rename(struct inode *, ... /* 5 more parameters */)`
  - `int vfs_rename(struct renamedata *)`
- Parameter type or return type changed
- Parameter reordered

Every 2 years, 6% functions changed signature

> Takeaway: Function changes are common, causing eBPF programs to silently read garbage data

📝 Kernel Source Code: Tracepoint

New features (e.g., folio)

- `writeback_dirty_page` ➡️ `writeback_dirty_folio`

Code maintenance

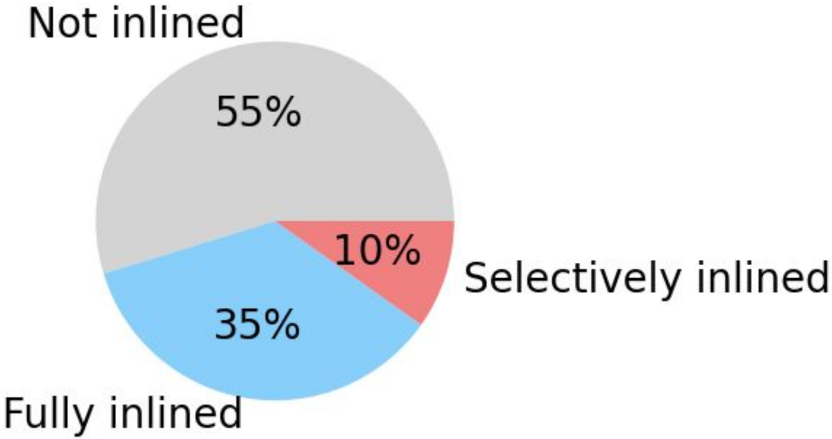
- Commit "mm/slab_common: unify NUMA and UMA version of tracepoints"
- Removed `kmem_alloc` and renamed `kmem_alloc_node` to the removed one
- "This will break some tools, but maintaining both does not makes sense."

Takeaway: Tracepoints are not as stable as presumed

Brendan Gregg:
"I have seen tracepoints change."

📦 Kernel Compilation: Inline

Full inline: function copied to all call sites and disappeared from the symbol table



Takeaway: 1/3 of kernel functions are fully inlined, causing attachment error

📦 Kernel Compilation: Inline

Selective inline: function inlined at some call sites, but not others

Example: eBPF program tracing `vfs_fsync`

```
fs/sync.c:
          int vfs_fsync() { /* logic */ }  // func definition
         long sys_fsync() { vfs_fsync(); } // inlined       ❌ NOT traced
fs/aio.c:
   extern int vfs_fsync();                 // func declaration
    void aio_fsync_work() { vfs_fsync(); } // not inlined ✅ Traced
```

Takeaway: 10% of kernel functions are selectively inlined, causing incomplete results

Summary of Dependency Mismatches

Kernel Source Code

~~Function: Absence, Change~~

Struct: Absence, Change

~~Tracepoint: Absence, Change~~

Kernel Configuration

Function: Absence, Change

Struct: Absence, Change

Tracepoint: Absence

Syscall: Availability, Traceability

Register: Layout Difference

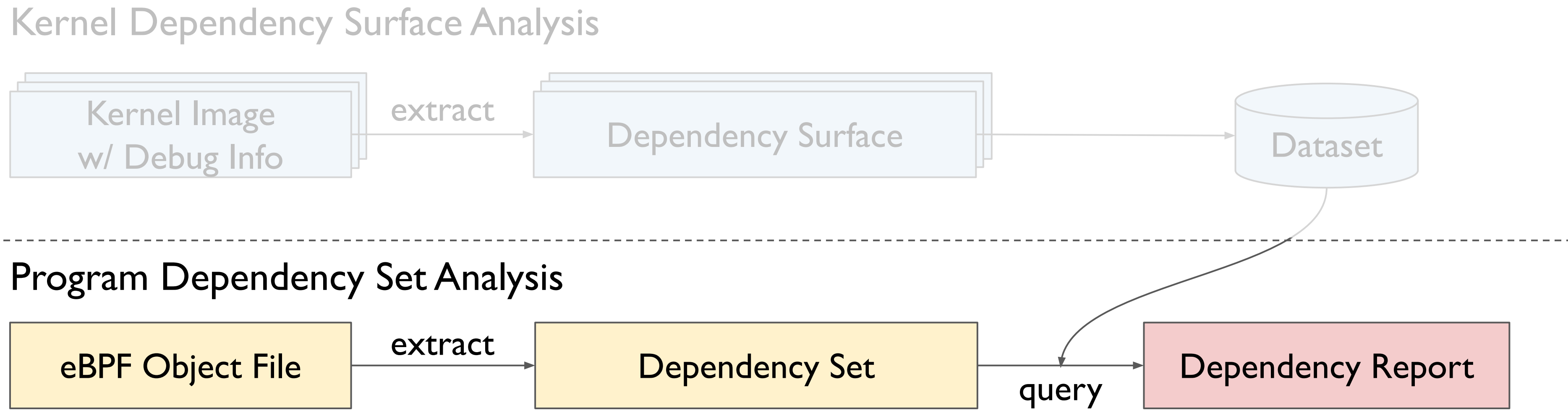
Kernel Compilation

Function:

- ~~Full / Selective Inline~~
- Transformation
- Duplication
- Name Collision

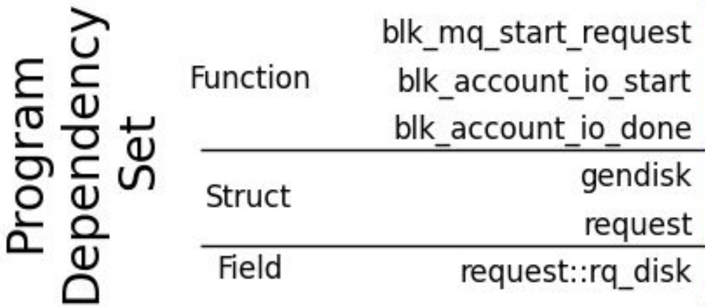
See the rest in paper

DepSurf


Kernel Dependency Surface Analysis
Kernel Image w/ Debug Info
extract
Dependency Surface
Dataset
Program Dependency Set Analysis
eBPF Object File
extract
Dependency Set
query
Dependency Report

Program Dependency Set Analysis

1. Extract Dependency Set from an eBPF program

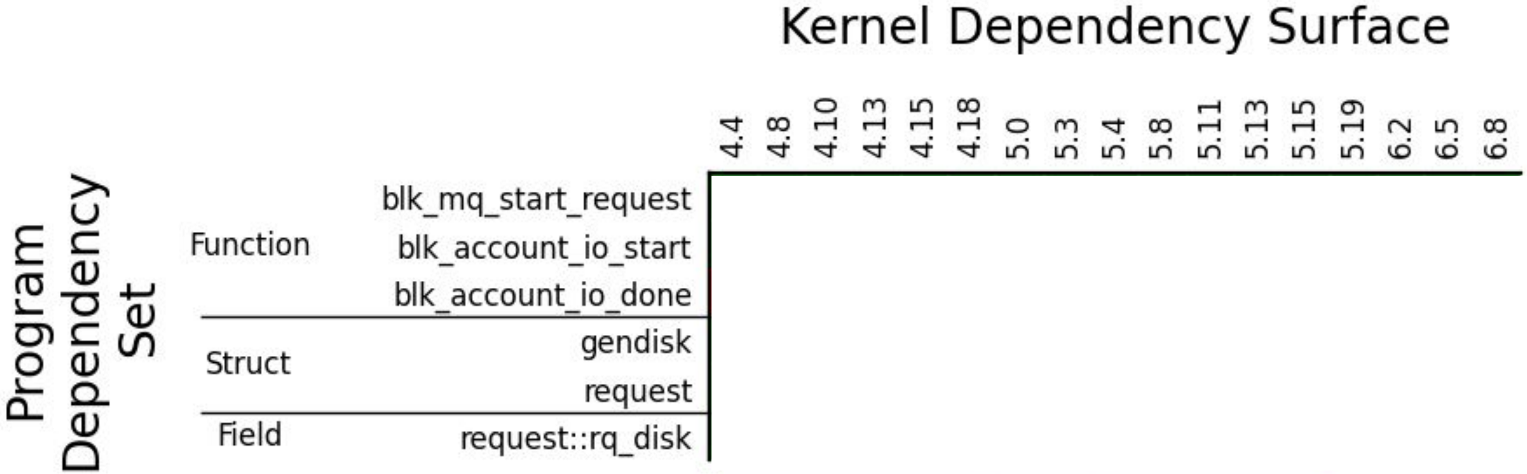


Program Dependency Set	Function	blk_mq_start_request
		blk_account_io_start
		blk_account_io_done
	Struct	gendisk
		request
	Field	request::rq_disk

*Simplified example

Program Dependency Set Analysis

1. Extract Dependency Set from an eBPF program
2. Query Kernel Dependency Surface dataset



Kernel Dependency Surface

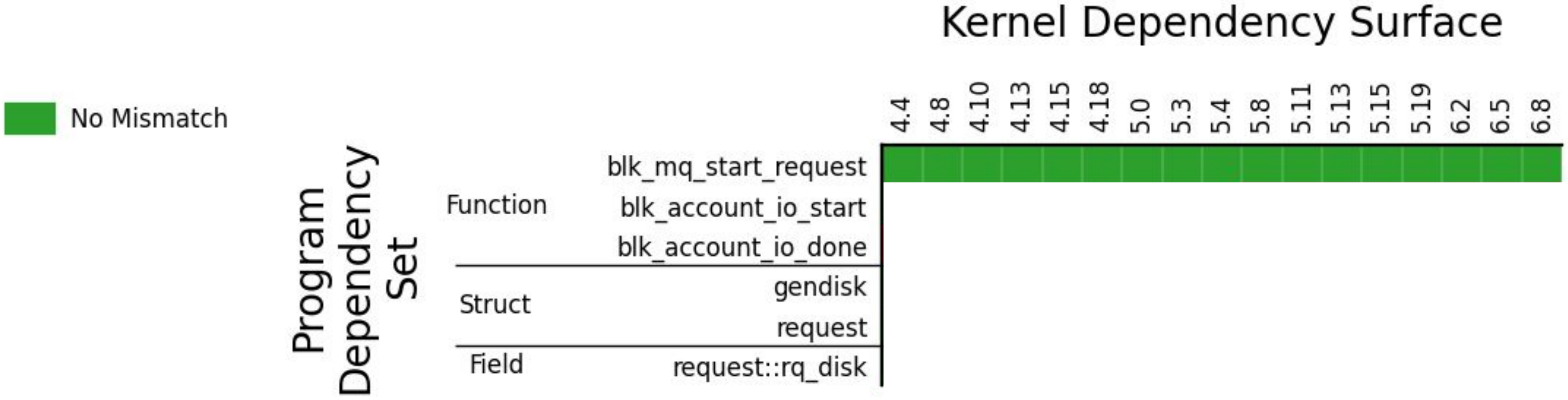
Program Dependency Set

		4.4	4.8	4.10	4.13	4.15	4.18	5.0	5.3	5.4	5.8	5.11	5.13	5.15	5.19	6.2	6.5	6.8
Function	blk_mq_start_request																	
	blk_account_io_start																	
	blk_account_io_done																	
Struct	gendisk																	
	request																	
Field	request::rq_disk																	

*Simplified example

Program Dependency Set Analysis

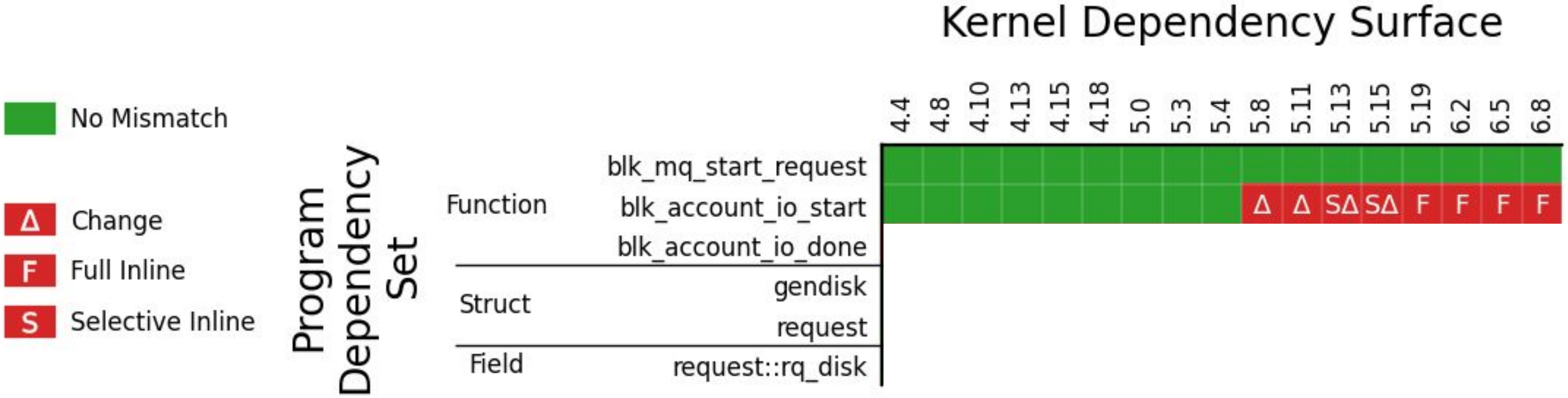
1. Extract Dependency Set from an eBPF program
2. Query Kernel Dependency Surface dataset



*Simplified example

Program Dependency Set Analysis

1. Extract Dependency Set from an eBPF program
2. Query Kernel Dependency Surface dataset

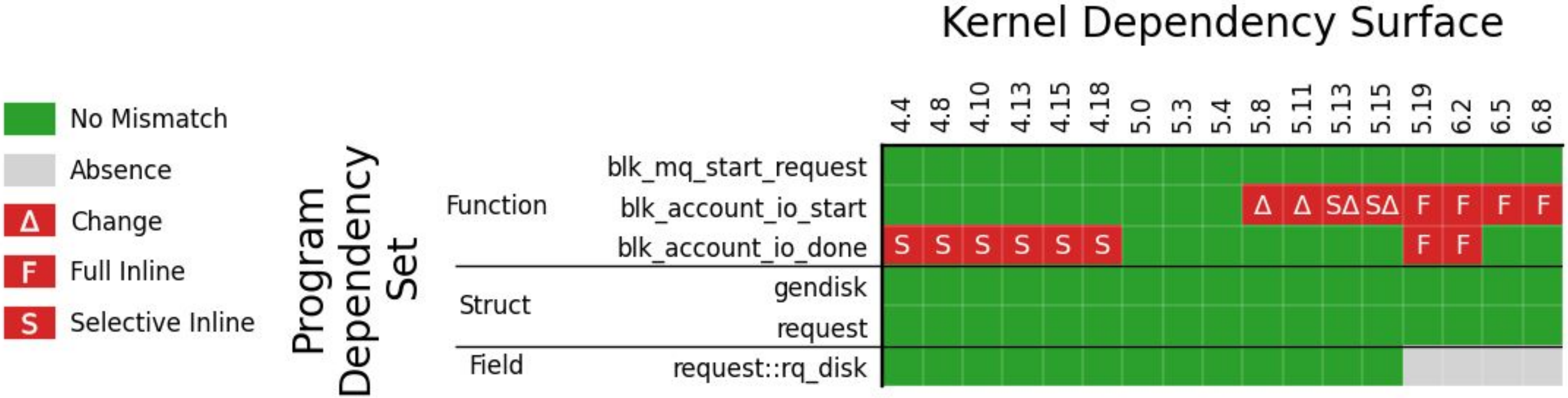


		4.4	4.8	4.10	4.13	4.15	4.18	5.0	5.3	5.4	5.8	5.11	5.13	5.15	5.19	6.2	6.5	6.8
Function	blk_mq_start_request	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
	blk_account_io_start	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	Δ	Δ	SΔ	SΔ	F	F	F	F
	blk_account_io_done																	
Struct	gendisk																	
	request																	
Field	request::rq_disk																	

*Simplified example

Program Dependency Set Analysis

1. Extract Dependency Set from an eBPF program
2. Query Kernel Dependency Surface dataset



Kernel Dependency Surface

- 🟩 No Mismatch
- ⬜ Absence
- Δ Change
- F Full Inline
- S Selective Inline

Program Dependency Set		4.4	4.8	4.10	4.13	4.15	4.18	5.0	5.3	5.4	5.8	5.11	5.13	5.15	5.19	6.2	6.5	6.8
Function	blk_mq_start_request	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
	blk_account_io_start	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	Δ	Δ	SΔ	SΔ	F	F	F	F
	blk_account_io_done	S	S	S	S	S	S	🟩	🟩	🟩	🟩	🟩	🟩	🟩	F	F	🟩	🟩
Struct	gendisk	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
	request	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Field	request::rq_disk	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	⬜	⬜	⬜	⬜

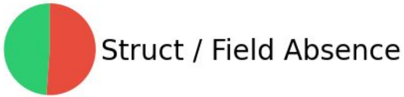
Takeaway: DepSurf allows developers to easily identify dependency mismatches

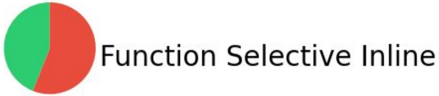
*Simplified example

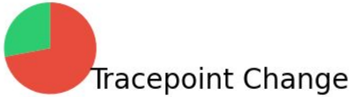
Program Dependency Set Analysis

Analyzed 53 eBPF programs from BCC and Tracee

- 43 programs depend on structs / fields
  - 22 absence ⇒ explicit error
- 25 programs depend on functions
  - 14 selective inline ⇒ incomplete results
- 25 programs depend on tracepoints
  - 18 change ⇒ explicit error / incorrect results







Takeaway: Dependency mismatches are widespread in eBPF programs

Outline

Introduction

Motivation

Dependency Mismatch

DepSurf

- Kernel Dependency Surface Analysis
- Program Dependency Set Analysis

Conclusion

Conclusion

Kernel Dependency Surface is fundamentally unstable

- eBPF programs built on it are not portable, requiring careful development and constant maintenance
- Developers lack tools for dependency mismatches

We developed DepSurf to systematically study dependency issue

- Development: Guide decisions for kernel internal usage
- Maintenance:Validate compatibility among kernels

Raise awareness of eBPF dependency issue & facilitate a robust eBPF ecosystem

depsurf.github.io

Backup Slides

Potential Solutions

Inherent Challenges: Unstable Kernel Dependency Surface

- Community-built compatibility layer
- Stability guarantee from the kernel
- Dependency tooling: DepSurf

Technical Challenges: Silent error with information gaps

- Linux kernel: Function inlining, transformations, …
- eBPF program: Type expectations, dependency fallbacks, …

See discussion in paper

Stability of Kernel Internals

	Location	Examples	Stability

uAPI Header	`include/uapi/`	`struct stat`
		`__NR_stat`
			Stable

Kernel Header	`include/linux/`
	`include/net/`
		`struct ext4_sb_info`
		`vfs_stat`
			Unstable

In-Tree Header	`fs/internal.h` `fs/ext4/ext4.h`	`do_statx` `ext4_getattr`	Very Unstable
C Source Code	`fs/sync.c` `fs/ext4/inode.c`	`ext4_do_update_inode` `ext4_chksum`	Extremely Unstable