Improving Indirect-Call Analysis in LLVM with Type and Data-Flow Co-Analysis

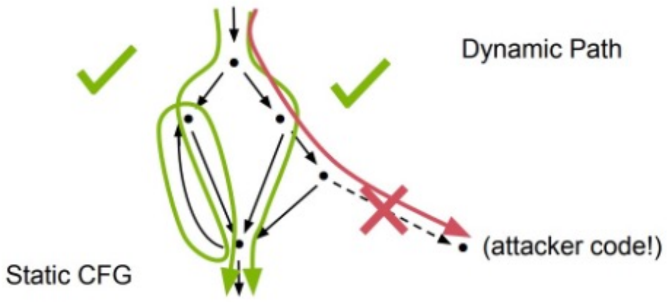
Dinghao Liu Shouling Ji Kangjie Lu Qinming He

USENIX Security 2024

Background

Background

- Control-flow graph is fundamental in program analysis



Control-Flow Integrity

Bug Detection

Program Optimization

Building precise CFG requires precise indirect call analysis

Background

➢ Indirect call analysis methods

- **Dynamic analysis**

  - Precise, but unsound

- **Pointer analysis**

  - Precise, but unscalable

- **Type analysis**

  - Sound & scalable, but imprecise

- **One-layer type analysis**

  - Matching types of functions and indirect call pointers

- **Two-layer / Multi-layer type analysis**

  - Many indirect calls are stored and used in composite types like structures

  - The outer layer struct type information could be used in type analysis

Background

➢ Type analysis methods

```c
typedef void (*f_ptr)(int a, int b);
struct S {f_ptr field1; f_ptr field2};

void address_taken_func1(int a, int b){...}
void address_taken_func2(int a, int b){...}
void address_taken_func3(int a, int b){...}
void address_taken_func4(int a, int b){...}

struct S s1 = {.field1 = address_taken_func1,
               .field2 = address_taken_func2};
struct S s2 = {.field1 = address_taken_func3,
               .field2 = address_taken_func4};

void main() {
    ...
    s1.field1(100, 200); // address_taken_func1 is called here
    ...
}
```

One-layer type analysis

- Matching type: f_ptr

- Target set:

[address_taken_func1, address_taken_func2

address_taken_func3, address_taken_func4]

Background

➢ Type analysis methods

```c
typedef void (*f_ptr)(int a, int b);
struct S {f_ptr field1; f_ptr field2};

void address_taken_func1(int a, int b){...}
void address_taken_func2(int a, int b){...}
void address_taken_func3(int a, int b){...}
void address_taken_func4(int a, int b){...}

struct S s1 = {.field1 = address_taken_func1,
               .field2 = address_taken_func2};
struct S s2 = {.field1 = address_taken_func3,
               .field2 = address_taken_func4};

void main() {
    ...
    s1.field1(100, 200); // address_taken_func1 is called here
    ...
}
```

One-layer type analysis

- Matching type: f_ptr
- Target set:

[address_taken_func1, address_taken_func2
address_taken_func3, address_taken_func4]

Multi-layer type analysis

- Matching type: S.field1 + f_ptr
- Target set:

[address_taken_func1,
address_taken_func3]

Eliminating 50% targets

Background

➢ **Insight: combining type and data-flow information**

```c
typedef void (*f_ptr)(int a, int b);
struct S {f_ptr field1; f_ptr field2};

void address_taken_func1(int a, int b){...}
void address_taken_func2(int a, int b){...}
void address_taken_func3(int a, int b){...}
void address_taken_func4(int a, int b){...}

struct S s1 = {.field1 = address_taken_func1,
               .field2 = address_taken_func2};
struct S s2 = {.field1 = address_taken_func3,
               .field2 = address_taken_func4};

void main() {
    ...
    s1.field1(100, 200); // address_taken_func1 is called here
    ...
}
```

Type analysis

- Target set:

[address_taken_func1, address_taken_func3]

Data-flow analysis

- Target set:

[address_taken_func1, address_taken_func2]

Ground truth

- Target set:

[address_taken_func1]

Challenges & Solutions

- **Challenge 1: Problems with the type analysis in LLVM**
  - Broken types in LLVM IRs & Type information omission in optimized IRs

  Solution: **Type recovery**

- **Challenge 2: Data-flow analysis is inefficient**
  - Iterative inter-procedural analysis is required

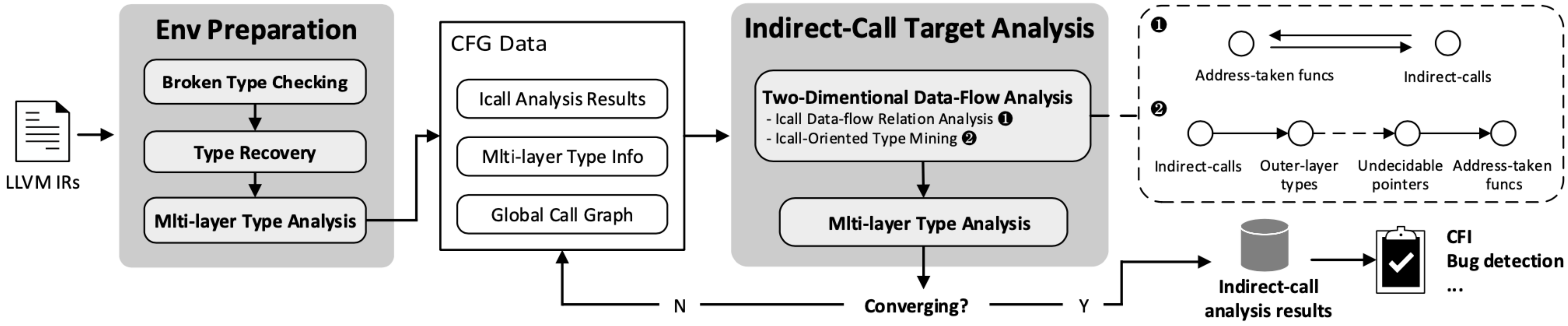
  Solution: **Two-dimensional data-flow analysis**

System Design

Overview

TFA (Type and Flow co-Analysis)

- Statically identifying indirect call targets through co-analysis.
- LLVM-based inter-procedural static analyzer.



Problems with the type analysis in LLVM

➢ Omitting function pointer fields

```
struct A {
    int i;
    int (*f)(int, struct A*);
    int (*g)(char, struct A*);
};
```

- Expected LLVM IR

```
%struct.A = type {i32, i32 (i32, %struct.A*), i32
(i8, %struct.A*)}
```

- Expected LLVM IR

```
%struct.A = type {i32, {}*, i32 (i8, %struct.A*)}
```

➢ Type unfolding

```
struct dvb_usb_adapter_properties adapter[2];
```

- Expected LLVM IR

```
[2 x %struct.dvb_usb_adapter_properties]
```

- Expected LLVM IR

```
<{{i8, i8, i32 (%struct.dvb_usb_adapter*, i32)*,
i32 (%struct.dvb_usb_adapter*, i32, i16, i32)*,
{i8, i8, i8,
{%struct.anon.163, [8 x i8]}}},
%struct.dvb_usb_adapter_properties}>
```

Problems with the type analysis in LLVM

➢ Missing struct names

```
static struct platform_driver omap_rtc_driver = {
    .probe = omap_rtc_probe,
    .remove = omap_rtc_remove,
    ...
}
```

17.8% global struct variables are defined without type names!

- Expected LLVM IR

```
@omap_rtc_driver = internal global %struct.platform_driver{definition}{initializer}
```

- Actual LLVM IR

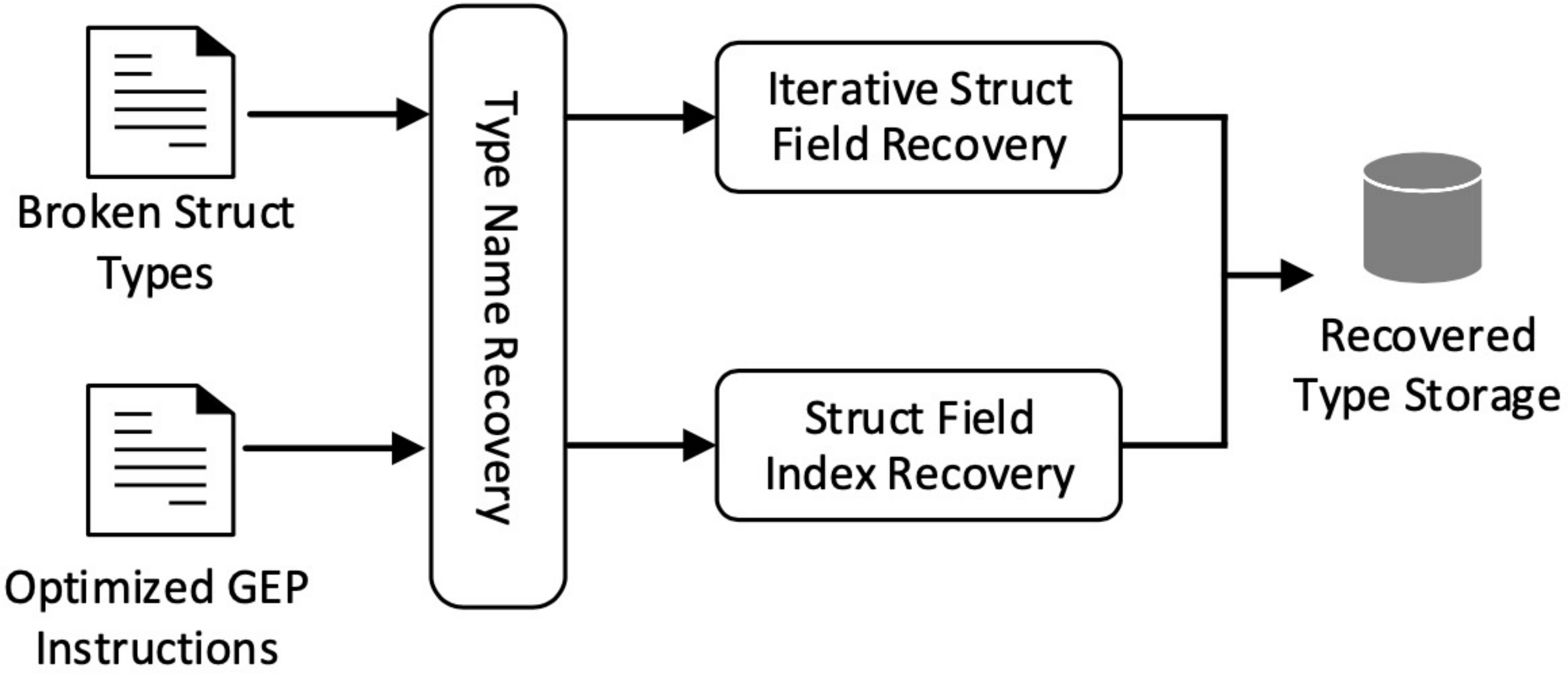
```
@omap_rtc_driver = internal global {definition}{initializer}
```

➢ Type information omission in optimized code

- O0 optimized GEP inst: `%ptr = getelementptr inbounds %struct.S* %0, i64 0, i32 5`
- O2 optimized GEP inst: `%ptr = getelementptr inbounds i8, i8* %0, i64 28`

Type Recovery

- Target: Recover the missing struct type information
- Solution: Use the source code information to recover broken types



Struct type comparison ➡ **Type name comparison**

Type Recovery

➢ Type name recovery

```
/* drivers/rtc/rtc-omap.c */
static struct platform_driver omap_rtc_driver = {
      .probe    = omap_rtc_probe,
      .remove  = omap_rtc_remove,
    ...
};
```

Source code

Actual LLVM IR (struct name is missing):

```
@omap_rtc_driver = internal global { ...} {...} align 8, !dbg !4378
```

```
!4378 = !DIGlobalVariableExpression(var: !4379, expr: !DIExpression())

!4379 = distinct !DIGlobalVariable(name: "omap_rtc_driver", scope: !2, file: !4352,
line: 1018, type: !4380, isLocal: true, isDefinition: true)

!4380 = distinct !DICompositeType(tag: DW_TAG_structure_type, name:
"platform_driver", file: !4381, line: 204, size: 1600, elements: !4382)
```

Source code info of the global variable expression

Source code info of the global variable's type

Type Recovery

➢ **Struct field recovery**

- Recover the missed type name
- Search the LLVM type definition list to get the intact struct
- Match all fields of the broken and intact struct type

➢ **Field index recovery for optimized GEP inst**

- Recover the missed type name
- Compute the struct field index according to the byte offset

S = a->b->c

O0: GEP1: a->b GEP2: b->c

O2: GEP: a->c

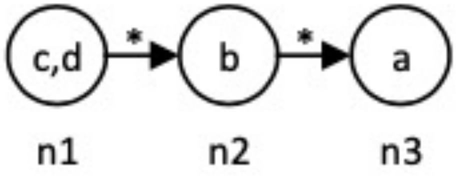
→ Recursively compute the address offset layer by layer

Alias Analysis

➢ Alias representation

```
int a = 10;
int *b = &a;
int **c = &b;
int **d = &b;
```

Source code

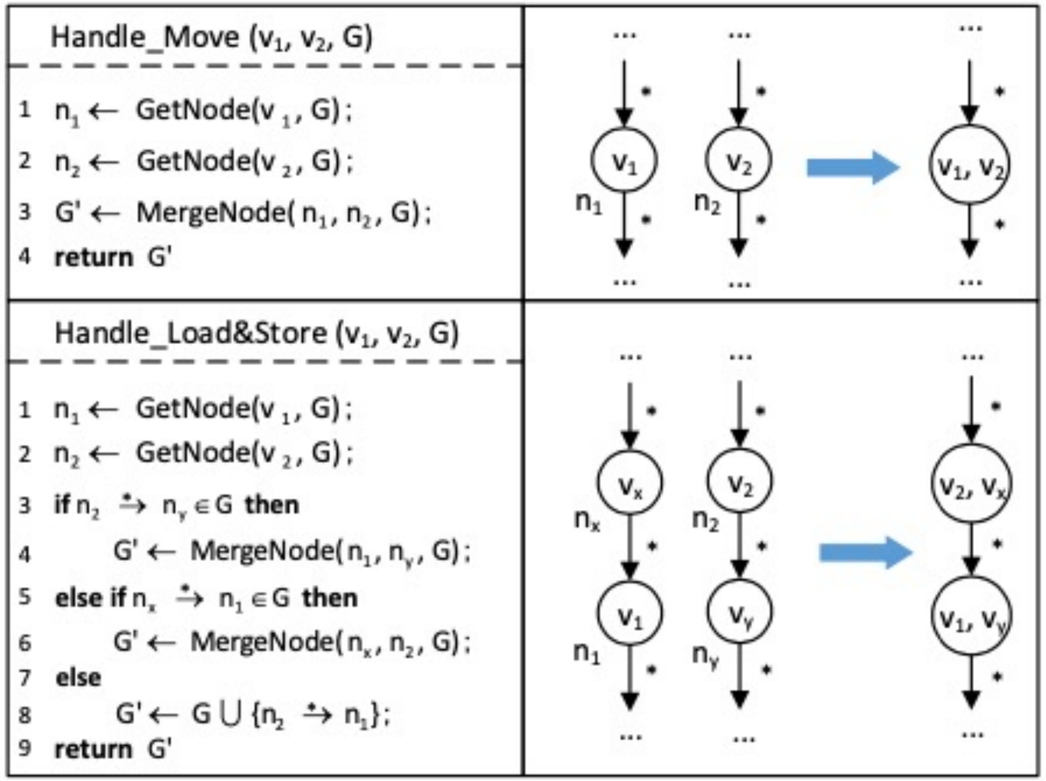


Alias graph

AliasSet1: {c,d}
AliasSet2: {b, *c, *d}
AliasSet3: {a, *b, **c, **d}

Alias sets

➢ Graph updating



Algorithms of alias handing

Graph updating

- Kastrinis G, Balatsouras G, Ferles K, et al. *An efficient data structure for must-alias analysis.* CC 2018
- Li T, Bai J J, Sui Y, et al. *Path-sensitive and alias-aware typestate analysis for detecting OS bugs*. ASPLOS 2022

Alias Analysis

- **Case study**

Source code

```
static void (*ps_con)(void);

static void ps_tq_int(...) {
  void (*con)(void);
  con = ps_con;
  ...
  con();    //Seed value
}

static void ps_set_intr(
        void (*acon)(void), ...){
  ps_con = acon;
  ...
}

static void do_pf_write(void){
 ps_set_intr(do_pf_write_start, ...);
}

static void do_pf_read(void){
  ps_set_intr(do_pf_read_start, ...);
}
```

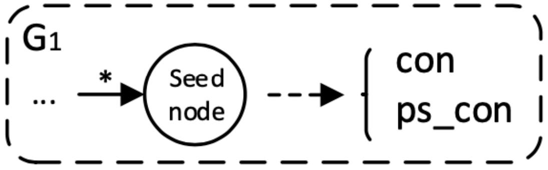
Alias Analysis

- **Case study**

Source code

```c
static void (*ps_con)(void);

static void ps_tq_int(...) {
  void (*con)(void);
  con = ps_con;
  ...
  con();    //Seed value
}

static void ps_set_intr(
        void (*acon)(void), ...){
  ps_con = acon;
  ...
}

static void do_pf_write(void){
 ps_set_intr(do_pf_write_start, ...);
}

static void do_pf_read(void){
  ps_set_intr(do_pf_read_start, ...);
}
```

Alias graph **Alias set**



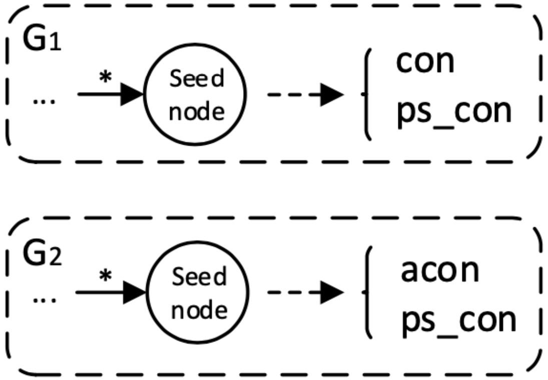
Alias Analysis

- **Case study**

Source code

```
static void (*ps_con)(void);

static void ps_tq_int(...) {
  void (*con)(void);
  con = ps_con;
  ...
  con();    //Seed value
}

static void ps_set_intr(
       void (*acon)(void), ...){
  ps_con = acon;
  ...
}

static void do_pf_write(void){
 ps_set_intr(do_pf_write_start, ...);
}

static void do_pf_read(void){
  ps_set_intr(do_pf_read_start, ...);
}
```

Alias graph **Alias set**



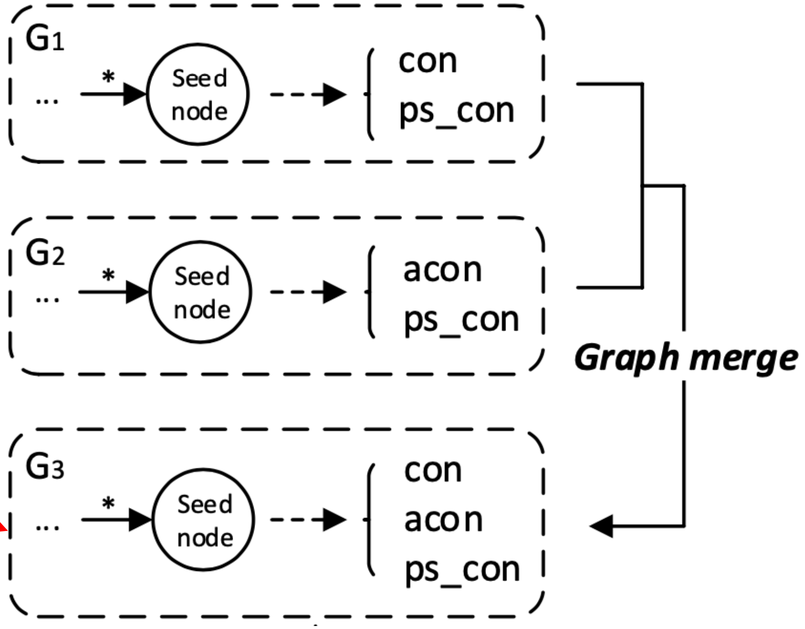
Alias Analysis

- **Case study**

Source code

```c
static void (*ps_con)(void);

static void ps_tq_int(...) {
  void (*con)(void);
  con = ps_con;
  ...
  con();    //Seed value
}

static void ps_set_intr(
       void (*acon)(void), ...){
  ps_con = acon;
  ...
}

static void do_pf_write(void){
 ps_set_intr(do_pf_write_start, ...);
}

static void do_pf_read(void){
  ps_set_intr(do_pf_read_start, ...);
}
```

Alias graph **Alias set**

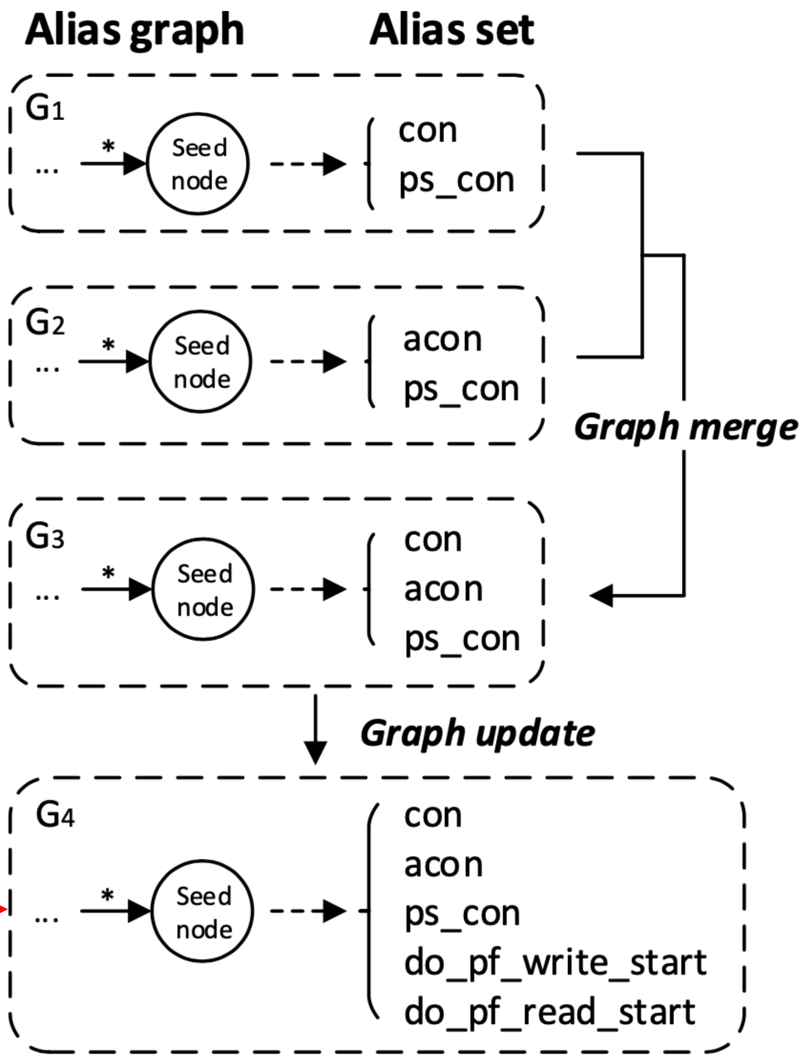


Alias Analysis

➢ Case study

Source code

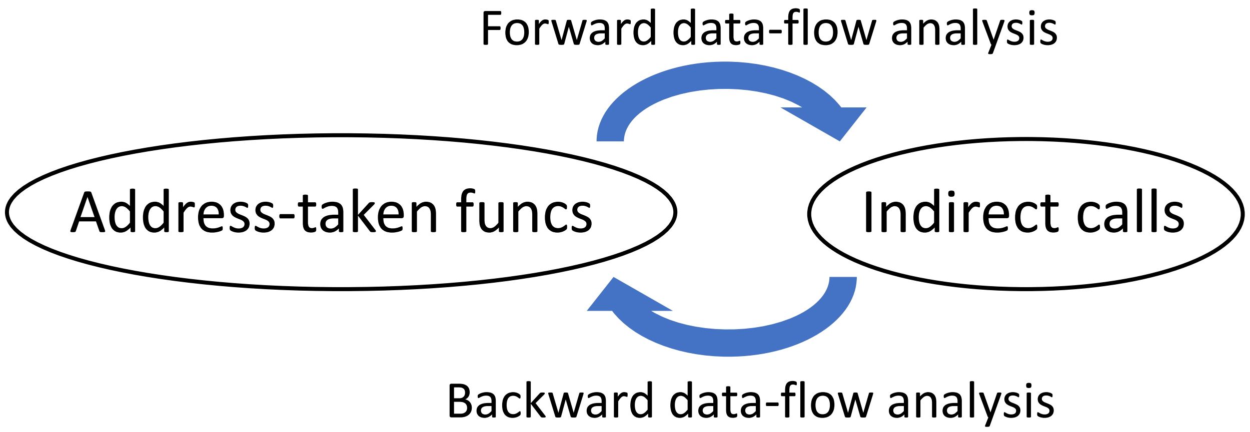
```
static void (*ps_con)(void);

static void ps_tq_int(...) {
  void (*con)(void);
  con = ps_con;
  ...
  con();    //Seed value
}

static void ps_set_intr(
        void (*acon)(void), ...){
  ps_con = acon;
  ...
}

static void do_pf_write(void){
 ps_set_intr(do_pf_write_start, ...);
}

static void do_pf_read(void){
  ps_set_intr(do_pf_read_start, ...);
}
```



Two-Dimensional Data-Flow Analysis

➢ **Target 1: Use data-flow analysis to resolve simple icall targets**

- **Bidirectional data-flow analysis**



Fall back strategies

- The size of alias set reaches the threshold
- The input value is aliased with assembly code or arithmetic operated values

➢ **Target 2: Use data-flow analysis to facilitate MLTA**

- **Icall-oriented type mining**
  - **Type mining for icalls**
  - **Type mining for undecidable targets**

Icall-Oriented Type Mining

➢ Type mining for icalls

```
/* sound/core/pcm_lib.c */
static int interleaved_copy(..., pcm_transfer_f transfer) {
    ...
    return transfer(...); //No outer-layer type in MLTA
}

snd_pcm_sframes_t __snd_pcm_lib_xfer(...) {
    ...
    pcm_copy_f writer;
    pcm_transfer_f transfer;
    ...
    writer = interleaved_copy; //An icall of interleaved_copy
    ...
    transfer = substream->ops->copy_kernel;
    ...
    err = writer(..., transfer);
}
```

It seems that '*transfer*' does not have outer-layer types.

MLTA

Icall-Oriented Type Mining

➢ Type mining for icalls

```
/* sound/core/pcm_lib.c */
static int interleaved_copy(..., pcm_transfer_f transfer) {
    ...
    return transfer(...); //No outer-layer type in MLTA
}

snd_pcm_sframes_t __snd_pcm_lib_xfer(...) {
    ...
    pcm_copy_f writer;
    pcm_transfer_f transfer;
    ...
    writer = interleaved_copy; //An icall of interleaved_copy
    ...
    transfer = substream->ops->copy_kernel;
    ...
    err = writer(..., transfer);
}
```

It seems that ‘*transfer*’ does not have outer-layer types.

MLTA

‘*transfer*’ has an outer-layer type <u>substream->ops->copy_kernel</u>.

TFA

Icall-Oriented Type Mining

➢ **Type mining for undecidable targets**

```c
/* drivers/gpu/drm/drm_aperture.c */
static int devm_aperture_acquire(struct drm_device *dev, ...
            void (*detach)(struct drm_device *)) {
    ...
    struct drm_aperture *ap;
    ...
    ap->detach = detach; //An undetermined pointer is stored
}

static void drm_aperture_detach_drivers(resource_size_t base,
            resource_size_t size) {
    ...
    struct drm_aperture *ap = container_of(...);
    struct drm_device *dev = ap->dev;
    ...
    ap->detach(dev); //Indirect call site
    ..
}

int devm_aperture_acquire_from_firmware(...) {
    ...
    return devm_aperture_acquire(...,
        drm_aperture_detach_firmware);
}
```

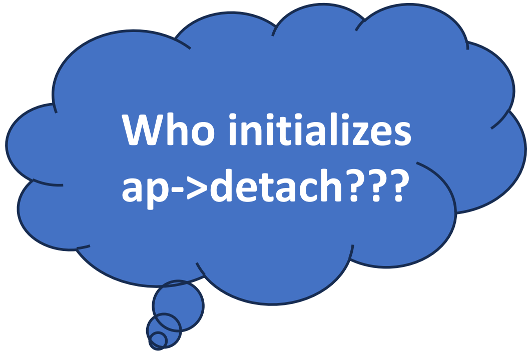
Who initializes ap->detach???

MLTA

Icall-Oriented Type Mining

- **Type mining for undecidable targets**

```
/* drivers/gpu/drm/drm_aperture.c */
static int devm_aperture_acquire(struct drm_device *dev, ...
            void (*detach)(struct drm_device *)) {
    ...
    struct drm_aperture *ap;
    ...
    ap->detach = detach; //An undetermined pointer is stored
}

static void drm_aperture_detach_drivers(resource_size_t base,
            resource_size_t size) {
    ...
    struct drm_aperture *ap = container_of(...);
    struct drm_device *dev = ap->dev;
    ...
    ap->detach(dev); //Indirect call site
    ..
}

int devm_aperture_acquire_from_firmware(...) {
    ...
    return devm_aperture_acquire(...,
        drm_aperture_detach_firmware);
}
```



MLTA

TFA

Evaluation

Evaluation Settings

Environment

- Use a Linux server with 126 GB RAM and an Intel Xeon Silver 4316 CPU
- Use Clang-15 to implement TFA
- Use TypeDive[1] for type analysis

Target

- Linux kernel v5.18 (allyesconfig & localmodconfig)
- FreeBSD kernel v12.4
- OpenSSL library v3.0.6
- OpenCV library v4.9.0
- MongoDB database v8.0.0

[1] https://github.com/umnsec/mlta

Evaluation – Indirect Call Analysis

Performance on eliminating icall targets

System	Language	Bitcode Files	Total Icalls	Avg. (Sig)	Avg. (MLTA)	Avg. (MLTA+VH)	Avg. (TFA)	Analysis Rounds	Analysis Time
OpenSSL	C	1,309	2,200	32.3	27.5	27.5	20.9 (24%↓)	2	34s
Linux-loc	C	2,978	9,527	52.5	18.6	18.6	8.3 (55%↓)	2	4m 8s
FreeBSD	C	3,826	20,901	38.1	20.2	20.2	11.6 (43%↓)	3	19m 4s
MongoDB	C++	4,406	23,885	34.8	30.0	11.7	6.6 (44%↓)	2	1h 57m
OpenCV	C++	1,583	33,602	44.5	44.5	32.6	14.2 (56%↓)	2	42m 2s
Linux-all	C	21,438	73,163	161.7	44.9	44.9	18.6 (59%↓)	3	1h 59m

TFA could eliminate 24% - 59% icall targets compared with MLTA

Evaluation – Indirect Call Analysis

Performance of different analysis rounds

Systems	Init	Round1	Round2	Round3
Linux-all	3,288,024	1,465,868	1,360,894	1,358,831
OpenSSL	60,417	46,224	46,038	-
MongoDB	279,272	158,971	158,177	-

Performance of different analysis phases

Systems	BDA	TM-I	TM-UT
Linux-all	1,157,344	362,363	409,486
OpenSSL	7,204	1,501	5,674
MongoDB	94,968	8,633	17,494

Evaluation – False Negative Analysis

Dataset

- Collect runtime icall traces through LLVM instrumentation
- Run the *Linux Test Project* and *openssl speed*
- 6,452 unique valid traces in the Linux kernel, 683 in the OpenSSL library

Results

- TFA misses 2 callees in the Linux kernel
- TFA misses 58 callees in the OpenSSL library

→ **Introduced since (one-layer) type analysis!**

The data-flow analysis of TFA does not introduce more false negatives than existing type-based analysis methods.

Evaluation – Type Recovery

Broken struct type recovery

```
/* drivers/tty/tty_ioctl.c */
unsigned int tty_write_room(struct tty_struct *tty){
    if (tty->ops->write_room)
        return tty->ops->write_room(tty);
    return 2048
}

/* drivers/tty/vt/vt.c */
static unsigned int con_write_room(struct tty_struct *tty){...}

static const struct tty_operations con_ops = {
    ...
    .write_room = con_write_room,
    ...
};

// struct tty_operations in tty_ioctl.bc
%struct tty_ioctl_operations = type {... {}*, {}*, ...}

// struct tty_operations in vt.bc
%struct tty_ioctl_operations = type {... i32 (%struct.tty_struct*)*,
i32 (%struct.tty_struct*)*, ...}
```

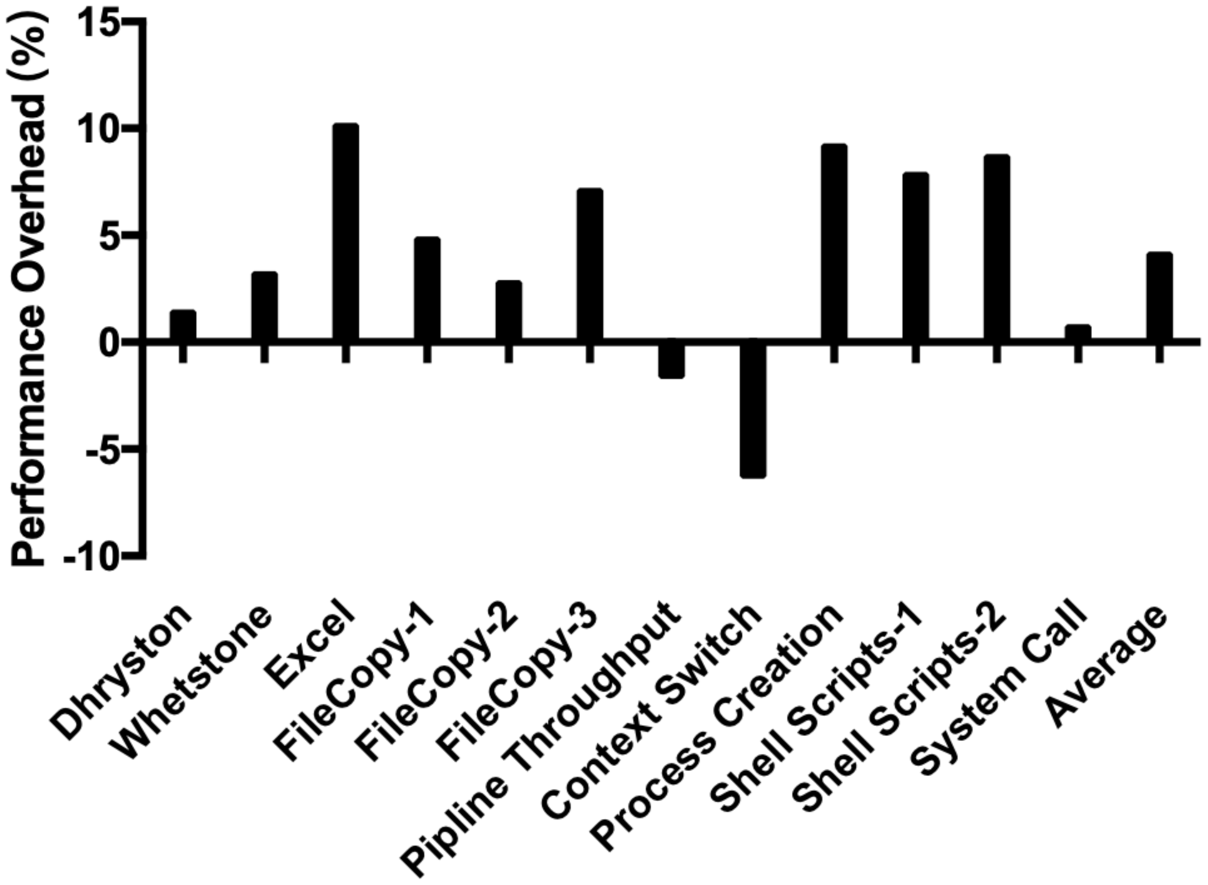
- Turn off the type recovery of TFA
- Use the kernel icall traces to evaluate the soundness

879 false negatives!

Evaluation – Application

Application 1: Fine-grained forward-edge CFI

- **Design**
  - Based on KCFI
  - Use the LLVM prefix data to determine valid icall targets
- **Evaluation**
  - Ubuntu machine with 4 CPUs (Intel Core i7-8770 CPU, 3.20Ghz)
  - UnixBench 5.1.2



4.1% performance overhead

Evaluation – Application

Application 2: Kernel bug detection

- **Design**
  - Analyze whether the indirect release callbacks are correctly used in the Linux kernel
  - Redundant callback: double-free
  - Missing callback: memleak

Bug function	Impact	Status
zfcp_port_enqueue	Double-free	A
ptp_ocp_device_init	Double-free	A
ocxl_file_register_afu	Double-free	F
rpmsg_virtio_add_ctrl_dev	Double-free	F
rpmsg_probe	Double-free	F
stm_register_device	Double-free	S
css_alloc_subchannel	Memleak	A
i3c_master_register_new_i3c_devs	Memleak	A

- **Result**
  - 8 new kernel bugs
  - TFA effectively refined the analysis scope

Conclusion

- **Existing approaches for indirect call analysis does not fully utilize type and data-flow information**

- **We presented TFA to analyze indirect call targets**
  - Type recovery
  - Two-dimensional data-flow analysis

- **We evaluated TFA on five popular large programs**
  - TFA could eliminate 24% to 59% icall targets compared with MLTA
  - The fine-grained icall analysis could support security applications

Q&A

dinghao.liu@zju.edu.cn