DeepType: Refining Indirect Call Targets with Strong Multi-layer Type Analysis

Tianrou Xia, *Hong Hu, Dinghao Wu*

August 16, 2024

PennState

Background

- Indirect calls are common in C/C++ programs
  - Mozilla Firefox
  - Google Chrome
  - LibreOffice
  - Apache HTTP Server
- Determining the target of an indirect call is non-trivial

Background

- Existing approaches

  - Data-based analysis [1]
    - Track data-flow

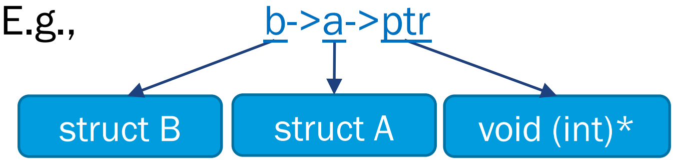
Precise | Time-consuming

  - Type-based analysis
    - Check function signatures

Efficient | High false positive rate

improved

- Multi-Layer Type Analysis (MLTA) [2]
  - Leverage composite type information
  - E.g.,



b->a->ptr

struct B

struct A

void (int)*

Multi-layer type: void (int)* | struct.A | struct.B

  - Check if the multi-layer types of functions and indirect calls match

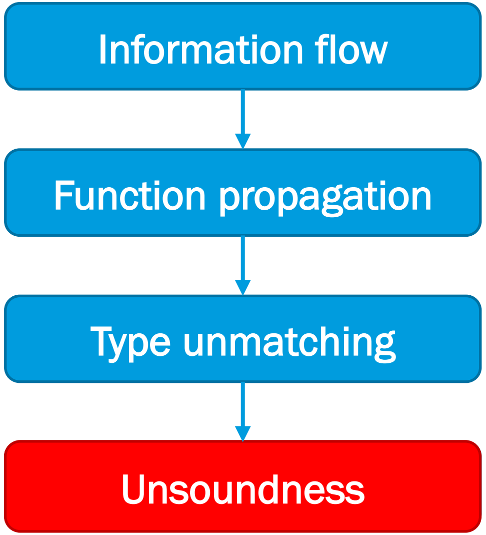
PennState

[1] Yulei Sui and Jingling Xue. 2016. SVF: interprocedural static value-flow analysis in LLVM. In Proceedings of the 25th International Conference on Compiler Construction (CC 2016). Association for Computing Machinery, New York, NY, USA, 265–266.

[2] Kangjie Lu and Hong Hu. 2019. Where Does It Go? Refining Indirect-Call Targets with Multi-Layer Type Analysis. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19). Association for Computing Machinery, New York, NY, USA, 1867–1881.

Challenges

- Multi-layer Type Matching

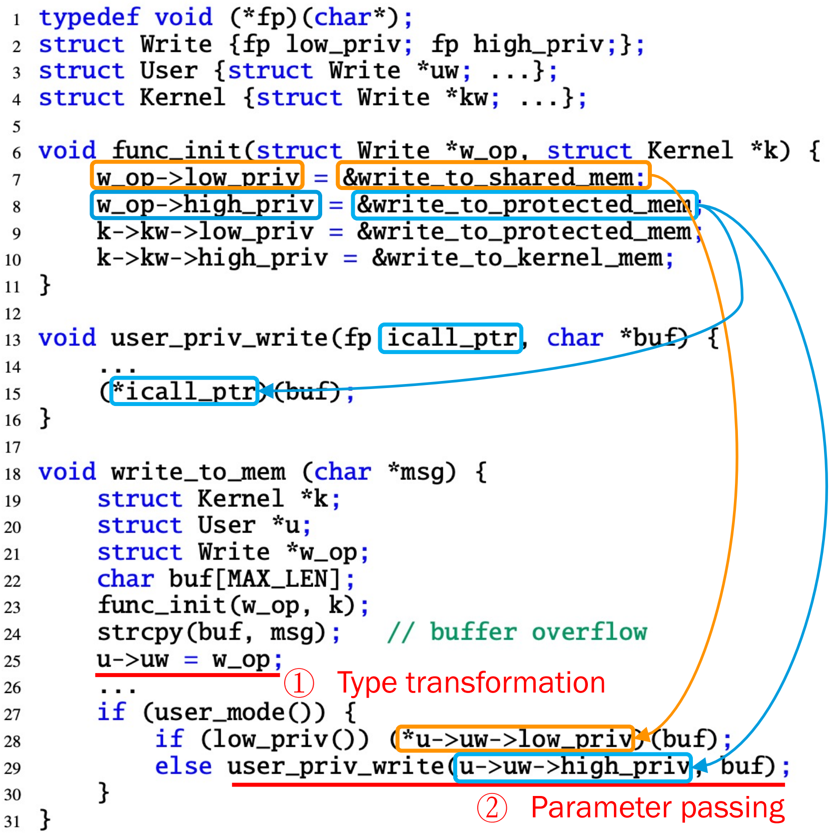


Information flow

Function propagation

Type unmatching

Unsoundness

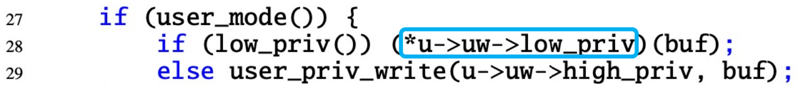


```
typedef void (*fp)(char*);
struct Write {fp low_priv; fp high_priv;};
struct User {struct Write *uw; ...};
struct Kernel {struct Write *kw; ...};

void func_init(struct Write *w_op, struct Kernel *k) {
    w_op->low_priv = &write_to_shared_mem;
    w_op->high_priv = &write_to_protected_mem;
    k->kw->low_priv = &write_to_protected_mem;
    k->kw->high_priv = &write_to_kernel_mem;
}

void user_priv_write(fp icall_ptr, char *buf) {
    ...
    (*icall_ptr)(buf);
}

void write_to_mem (char *msg) {
    struct Kernel *k;
    struct User *u;
    struct Write *w_op;
    char buf[MAX_LEN];
    func_init(w_op, k);
    strcpy(buf, msg);    // buffer overflow
    u->uw = w_op;
    ...
    if (user_mode()) {
        if (low_priv()) (*u->uw->low_priv)(buf);
        else user_priv_write(u->uw->high_priv, buf);
    }
}
```

① Type transformation

② Parameter passing

Motivation



```
27      if (user_mode()) {
28          if (low_priv()) (*u->uw->low_priv)(buf);
29          else user_priv_write(u->uw->high_priv, buf);
```

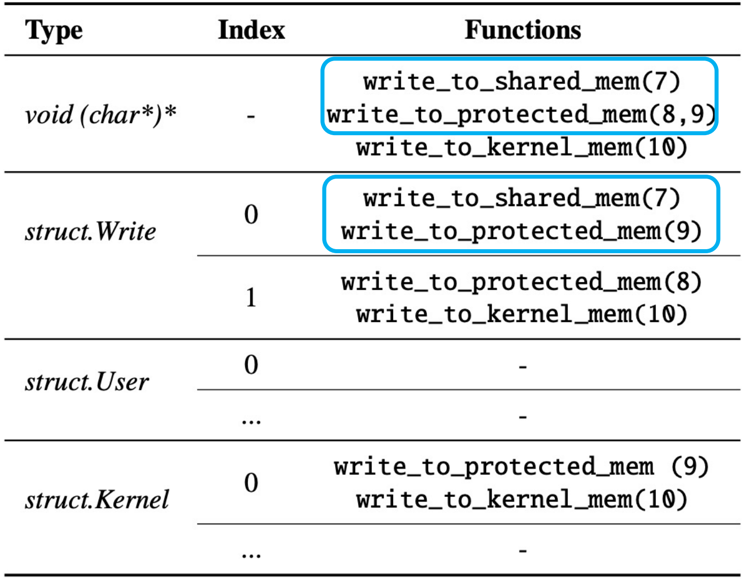
- MLTA **splits** multi-layer types
    - Record mappings between split types and associated functions

      Weaken the restrictions provided by multi-layer types

    - Resolve each layer and calculate intersection of their target sets

      Produce false positive target(s)

Line 28 real target: write_to_shared_mem



Type	Index	Functions

void (char)**	-	write_to_shared_mem(7)
		write_to_protected_mem(8,9)
		write_to_kernel_mem(10)

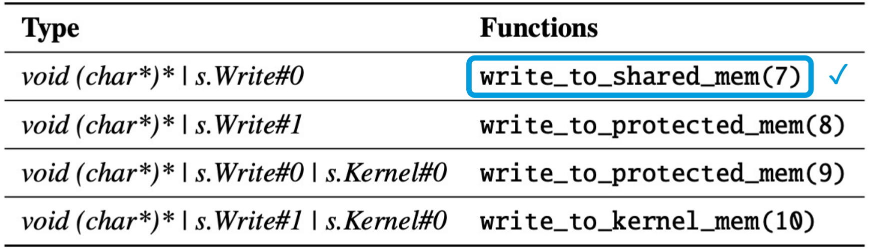
struct.Write	0	write_to_shared_mem(7)
		write_to_protected_mem(9)

	1	write_to_protected_mem(8)
		write_to_kernel_mem(10)

struct.User	0	-
	...	-
struct.Kernel	0	write_to_protected_mem (9) write_to_kernel_mem(10)
	...	-

Strong Multi-layer Type Analysis (SMLTA)

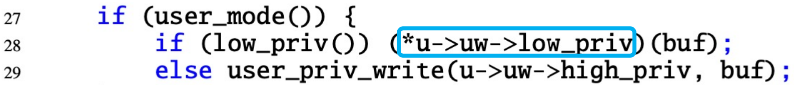
- Keep **strong restrictions** provided by multi-layer types
  - Record mappings between entire multi-layer types and associated functions in Type-Func Map



Type	Functions
void (char)* | s.Write#0*	write_to_shared_mem(7) ✓
void (char)* | s.Write#1*	write_to_protected_mem(8)
void (char)* | s.Write#0 | s.Kernel#0*	write_to_protected_mem(9)
void (char)* | s.Write#1 | s.Kernel#0*	write_to_kernel_mem(10)

Line 28 real target: write_to_shared_mem

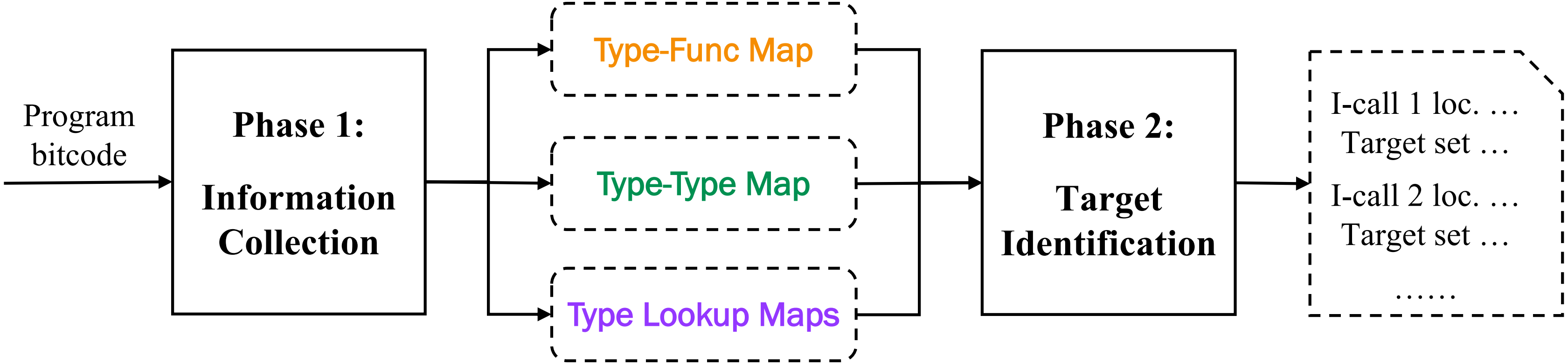
  - Resolve the entire multi-layer type of each indirect call



```
27      if (user_mode()) {
28          if (low_priv()) (*u->uw->low_priv)(buf);
29          else user_priv_write(u->uw->high_priv, buf);
```

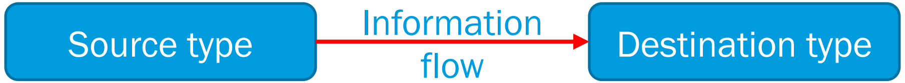
DeepType

- Workflow

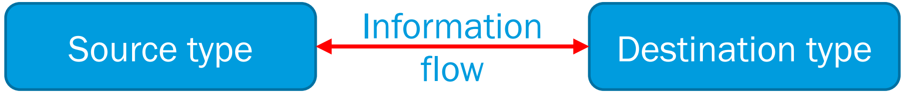


Phase 1 – Information collection

- Type relationship resolving → Address type transformation
  - Type assignment



  - Type casting

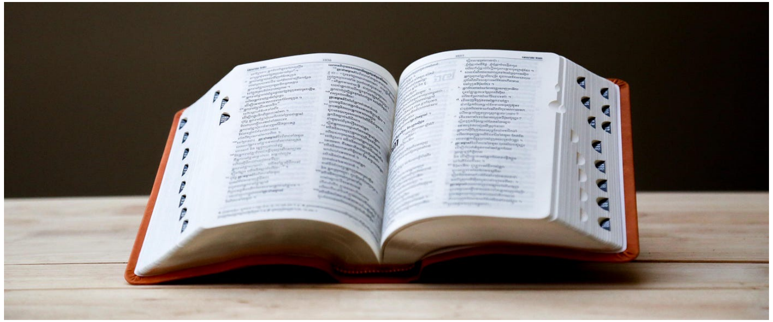


  - **Friend type**
    - A is a friend type relative to B if there exists information flow from A to B
  - Record type relationships in Type-Type Map

SHARe

Phase 1 – Information collection

- Multi-layer type organization
  - **Multi-layer mappings**
  - Record multi-layer types in Type Lookup Maps

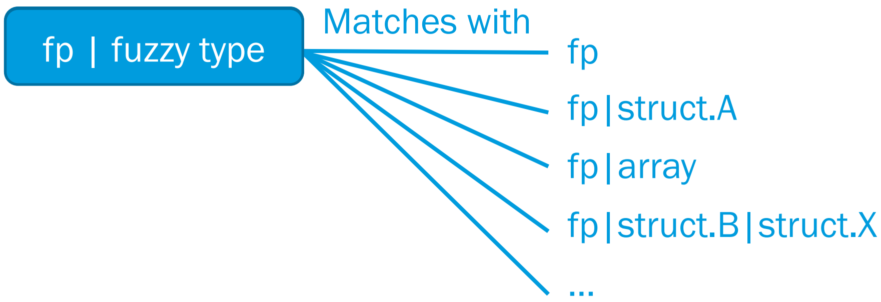


  - Provide efficient access and retrieval

Phase 2 – Target Identification

- **Fuzzy type** → Address parameter passing
    - Mark the type of uncertain layer(s)

```
void user_priv_write(fp icall_ptr, char *buf) {
    ...
    (*icall_ptr)(buf);
}
```



- Match with any type

Phase 2 - Target Identification

1. Ascertain the multi-layer type of the indirect call
2. Query Type-Type Map for friend types relative to each **fragment**
   - A fragment is one or multiple continuous layers in a multi-layer type
   - Use adapted **breadth-first search** to exhaustively search for friend types
3. Generate friend types for the **entire multi-layer type**
4. Look for **matched types** in Type Lookup Maps
5. Query Type-Func Map to achieve **valid targets**

Optimization

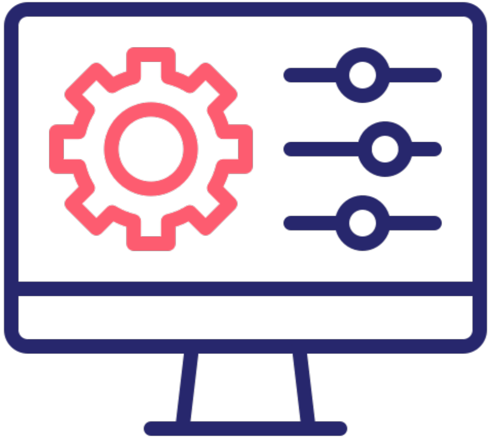
- Special handlings
  - Address corner cases
  - Reduce FPs and FNs

Precision improvement

- Caches
  - Store the result of resolved multi-layer types

Efficiency improvement

Evaluation



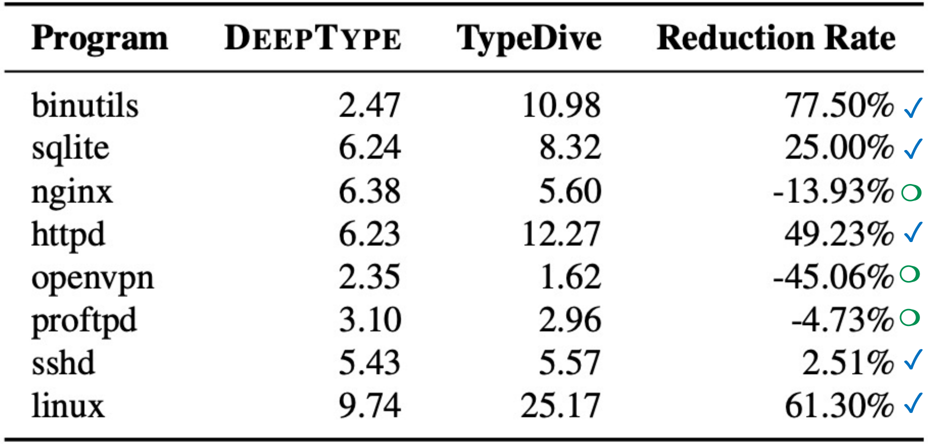
- **Experiment environment**
  - Ubuntu 20.04 (64bit)
  - 8-core Intel Core i9-9880H CPU @ 2.30GHz
  - 6GB DDR4 RAM
- **Benchmarks**
  - **Linux kernel:** Linux-5.1
  - **5 severs:** Nginx, httpd, openVPN, proftpd, sshd
  - **14 user applications:** binutils-2.35, SQLite-3.45.1
- **Compare with TypeDive** (MLTA prototype)

Evaluation

- **DeepType is more effective than TypeDive**
  - Metric: Average Number of Targets (ANT) $\frac{Num(T)}{Num(IC)}$

Num(T): Total number of targets

Num(IC): Total number of indirect calls with targets

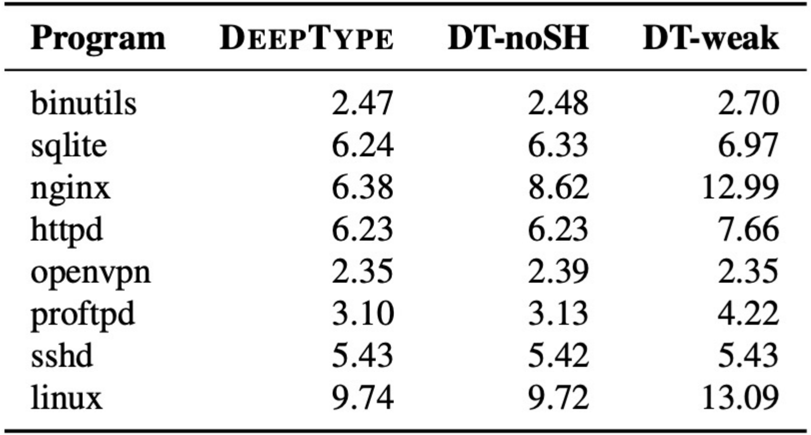


Program	DEEPTYPE	TypeDive	Reduction Rate
binutils	2.47	10.98	77.50% ✓
sqlite	6.24	8.32	25.00% ✓
nginx	6.38	5.60	-13.93% ○
httpd	6.23	12.27	49.23% ✓
openvpn	2.35	1.62	-45.06% ○
proftpd	3.10	2.96	-4.73% ○
sshd	5.43	5.57	2.51% ✓
linux	9.74	25.17	61.30% ✓

- DeepType **reduces** the ANT by **43.11%** on average
  - SMLTA and special handling **reduce FPs**
- DeepType does not consistently reduce ANT
  - The special handling **reduces FNs**
  - TypeDive produces FNs

Evaluation

- Contribution of SMLTA

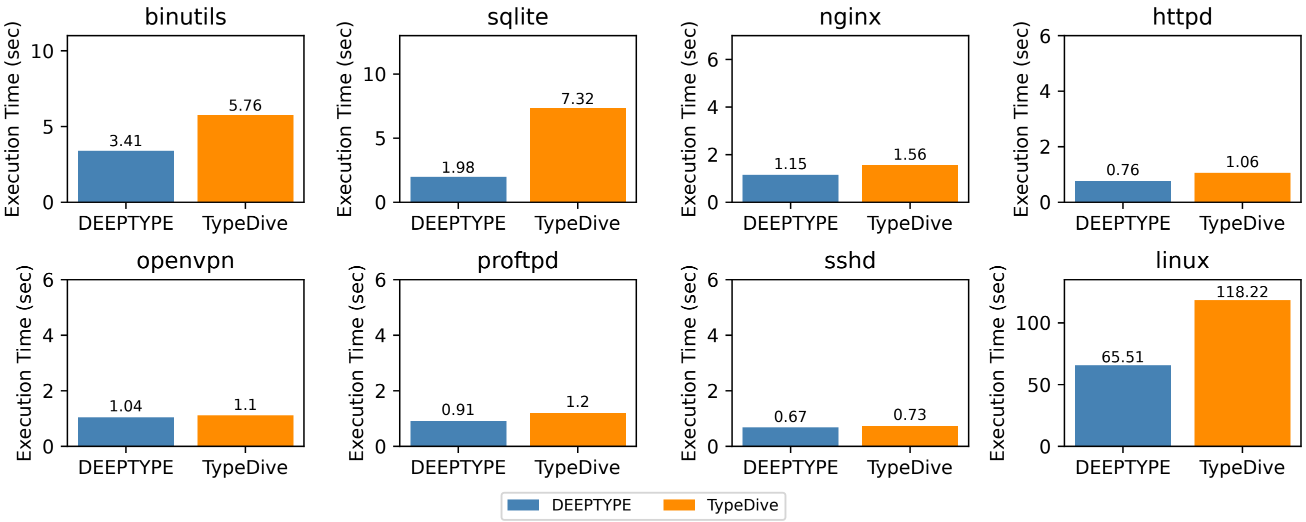


Program	DeepType	DT-noSH	DT-weak
binutils	2.47	2.48	2.70
sqlite	6.24	6.33	6.97
nginx	6.38	8.62	12.99
httpd	6.23	6.23	7.66
openvpn	2.35	2.39	2.35
proftpd	3.10	3.13	4.22
sshd	5.43	5.42	5.43
linux	9.74	9.72	13.09

- DT-noSH: No special handling
  - Reveal the significant impact of **SMLTA** on effectiveness
- DT-weak: Store split types
  - Show the impact of **storing entire multi-layer types** in reducing FPs

Evaluation

- **DeepType is more efficient than TypeDive**



- DeepType **outperforms** TypeDive, showing an **average reduction** of **37.02%**.

Conclusion

- We introduced **strong multi-layer type analysis** (SMLTA), a novel approach in refining indirect call targets.
- We implemented a prototype, DeepType, which is equipped with special handling to address diverse code patterns.
- DeepType is **scalable** to large applications with superior **effectiveness** as well as **performance** over TypeDive.

Artifact

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

- Available at https://github.com/s3team/DeepType

Thank you