Goshawk: Hunting Memory Corruptions via Structure-Aware and Object-Centric Memory Operation Synopsis

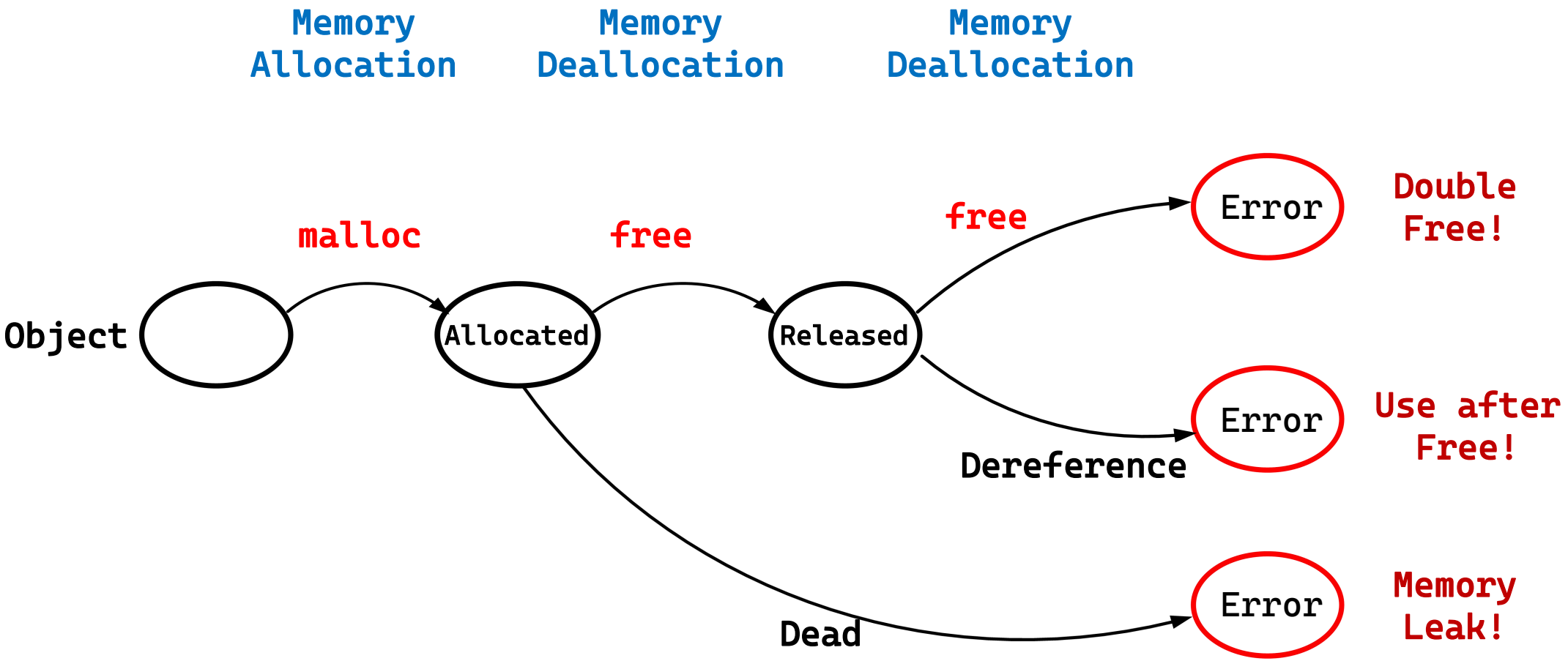
Yunlong Lyu, Yi Fang, Yiwei Zhang, Qibin Sun, Siqi Ma,
Elisa Bertino, Kangjie Lu, and Juanru Li

University of Science and Technology of China, Feiyu Security, Purdue University,
The University of New South Wales, University of Minnesota, Shanghai Qi Zhi Institute,
G.O.S.S.I.P, Shanghai Jiao Tong University

Outline

1. Motivation
2. Design
3. Evaluation
4. Conclusion

Memory corruption bugs detection routine



Custom Memory Management Functions

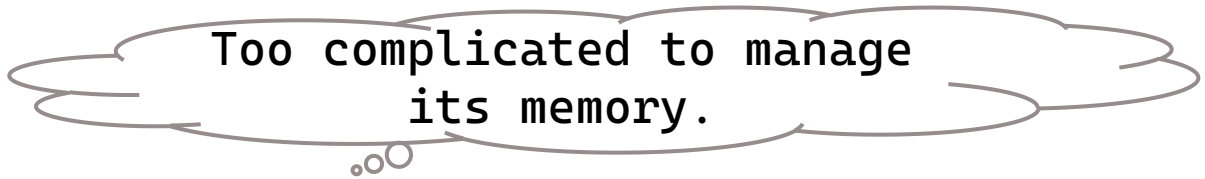
Structure Definition

```
struct sk_buff {
    union {
        struct {
            struct sk_buff              *next;
            struct sk_buff              *prev;

                union {
                    struct net_device   *dev;
                    unsigned long       dev_scratch;
                };
            };
            struct rb_node      rbnode;
            struct list_head    list;
    };
      …
};//205 lines, used more than 20,000 times
```

Structure Memory Management

```
1 void l2cap_create_connless_pdu(...)
2 {
3     struct sk_buff* skb;
4
5   ❶ skb = l2cap_sock_alloc_skb_cb(...);
6     ...
7   ❷ kfree_skb(skb);
8     ...
9 }
```



l2cap_sock_alloc_skb_cb()->bt_skb_send_alloc()->sock_alloc_send_skb()->
sock_alloc_send_pskb()->alloc_skb_with_frags()->alloc_skb()->__alloc_skb()->
napi_skb_cache_get()->kmem_cache_alloc_bulk()->__kmem_cache_alloc_bulk()-
>kmem_cache_alloc()

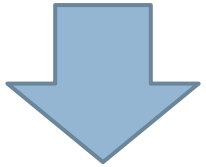
Complex Custom Memory Management Behaviors


```c
/*File: Linux/mm/dmapool.c*/
void *dma_pool_alloc(struct dma_pool *pool, gfp_t
                        mem_flags, dma_addr_t *handle)
{
     unsigned long flags;
     struct dma_page *page;
     size_t offset;
     void *retval;
     ...
     /*ALLOCATE OBJECT info*/
     page = pool_alloc_page(pool, mem_flags &
                                    (~__GFP_ZERO));
     if (!page)
           return NULL;
     retval = offset + page->vaddr;
     return retval;
}
```

```c
/*File: Linux/mm/dmapool.c*/
static struct dma_page *pool_alloc_page(struct
                        dma_pool *pool, gfp_t mem_flags)
{
     struct dma_page *page;

     page = kmalloc(sizeof(*page), mem_flags);
     if (!page)
           return NULL;

     page->vaddr = dma_alloc_coherent(pool->dev,
           pool->allocation,&page->dma, mem_flags);
      ...
     return page;
}
```

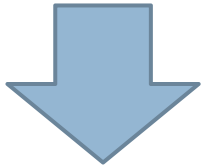


Feature 1: Multi-object and Structure nested allocation

Complex Custom Memory Management Behaviors


```
9       /*ALLOCATE OBJECT info*/
10    ❶ info = framebuffer_alloc(sizeof(struct hvfb_par),
11                                &hdev->device);
12      if (!info)
13              return -ENOMEM;
14      ...
15      ret = hvfb_getmem(hdev, info);
16
17      if (ret) {
18              pr_err("No memory for framebuffer\n");
19              goto error2;
20      }
21      ...
22  error2:
23      ...
24      /*DEALLOCATE OBJECT info*/
25    ❷ framebuffer_release(info);
26      ... //info->apertures double free!
27      return ret;
28  }
```

```
37      return info;
38  }
39  static int hvfb_getmem(struct hv_device *hdev, ...)
40  {
41      ...
42    ❸ info->apertures = alloc_apertures(1);
43      ...                 //info->apertures allocation
44      pdev = pci_get_device(...);
45      if (!pdev) {
46        ❹ kfree(info->apertures);//info->apertures free
47          return -ENODEV;
48      }
49  }
50  void framebuffer_release(struct fb_info *info)
51  {
52      if (!info) return;
53    ❺ kfree(info->apertures);
54      kfree(info);
55  }
```

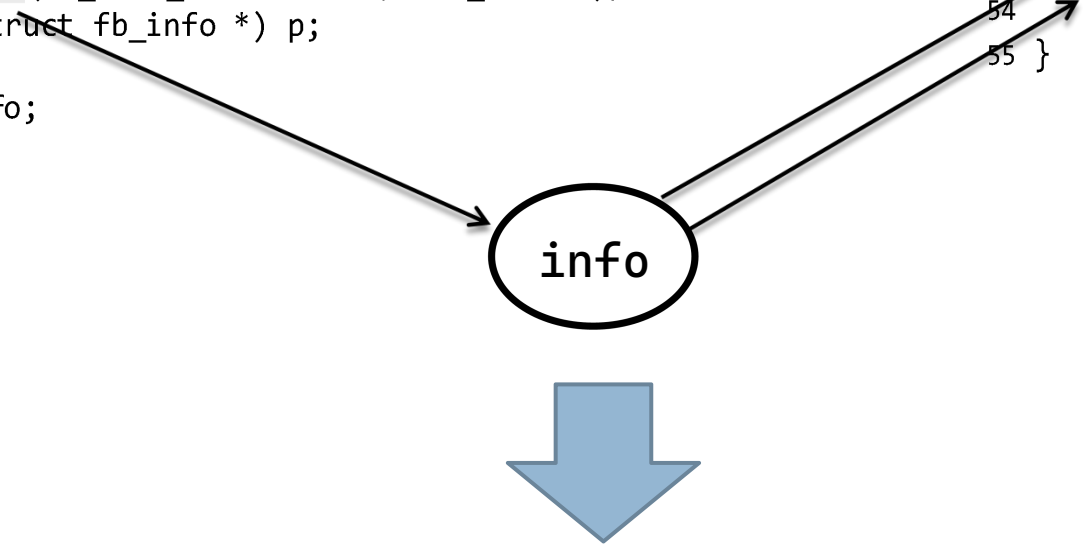


Feature 2: Require Stucture-aware deallocation

Complex Custom Memory Management Behaviors


```
29 struct fb_info *framebuffer_alloc(size_t size, ...)
30 {
31     struct fb_info *info;
32     char *p;
33     ...
34     p = kzalloc(fb_info_size + size, GFP_KERNEL);
35     info = (struct fb_info *) p;
36     ...
37     return info;
38 }
```

```
50 void framebuffer_release(struct fb_info *info)
51 {
52     if (!info) return;
53     kfree(info->apertures);
54     kfree(info);
55 }
```



Feature 3: Follow an Unpaired-use model

Objects & Challenges

Custom memory management functions

Traditional bug detection

How to fill the Gap?

```
l2cap_sock_alloc_skb_cb
pool_alloc_page
framebuffer_alloc
framebuffer_release
```

- Simple memory object
- Naive data flow tracking
- Paired-use model

- Multi-object allocation
- Structure nested allocation
- Structure-aware deallocation
- Unpaired-use model

MOS

Structure-aware and Object-centric Memory Operation Synopsis

MOS: Structure-aware and Object-centric Memory Operation Synopsis

Memory managemnt function interface

```
1  static struct dma_page *pool_alloc_page(struct
2                  dma_pool *pool, gfp_t mem_flags)
3  {
4      struct dma_page *page;
5
6      page = kmalloc(...);
9
10     page->vaddr = dma_alloc_coherent(...);
11     ...
12
13     return page;
14 }
```

Internal allocation/deallocation

Memory Operation Synopsis

MOS Representation:

```
{
```

Function name		Property
pool_alloc_page	:	Allocator

Memory object list		Object type
RetVal	:	struct dma_page*
RetVal->vaddr	:	void*

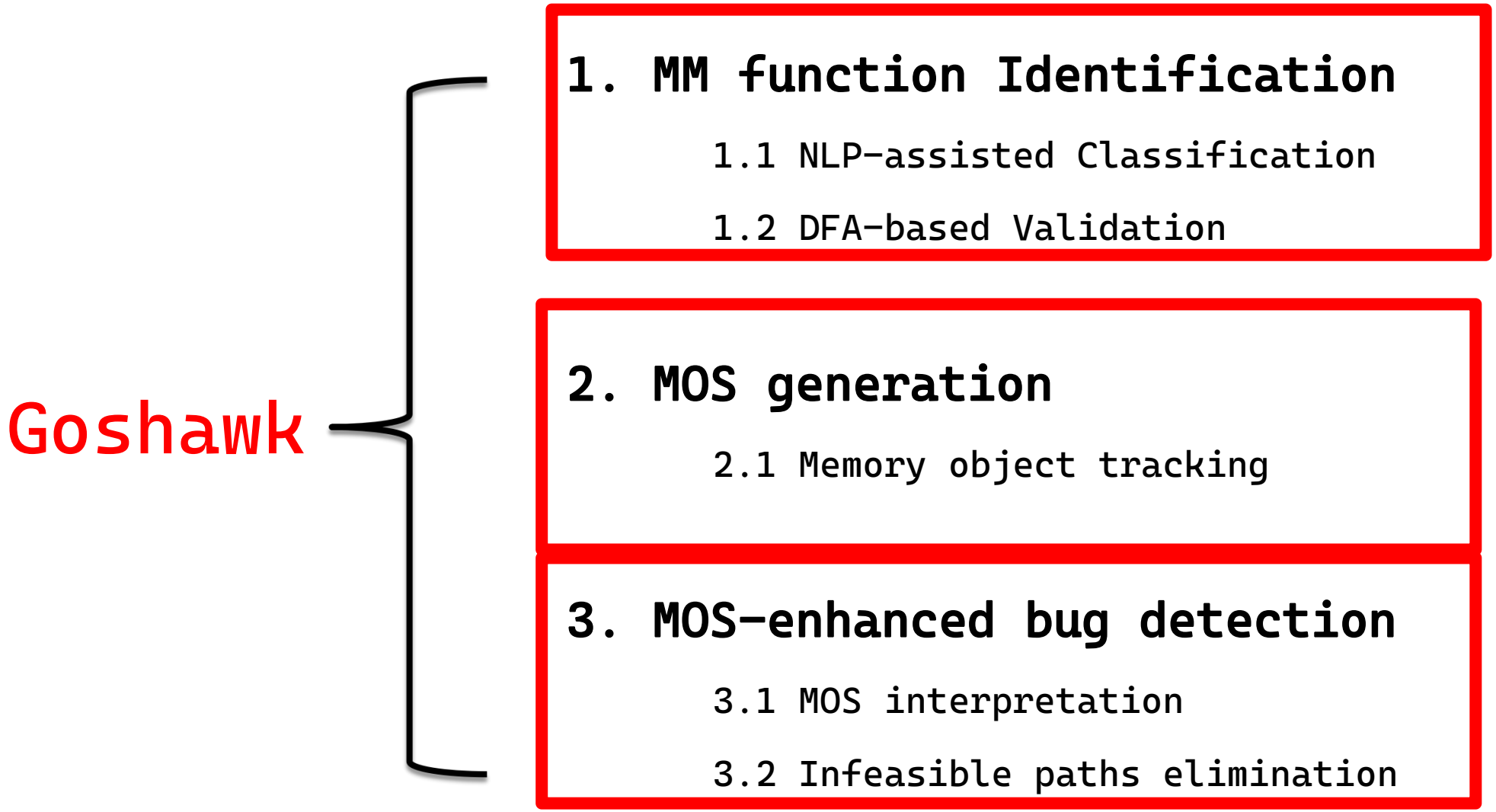
```
}
```

Object-centric **Structure-aware**

Outline

1. Motivation
2. **Design**
3. Evaluation
4. Conclusion

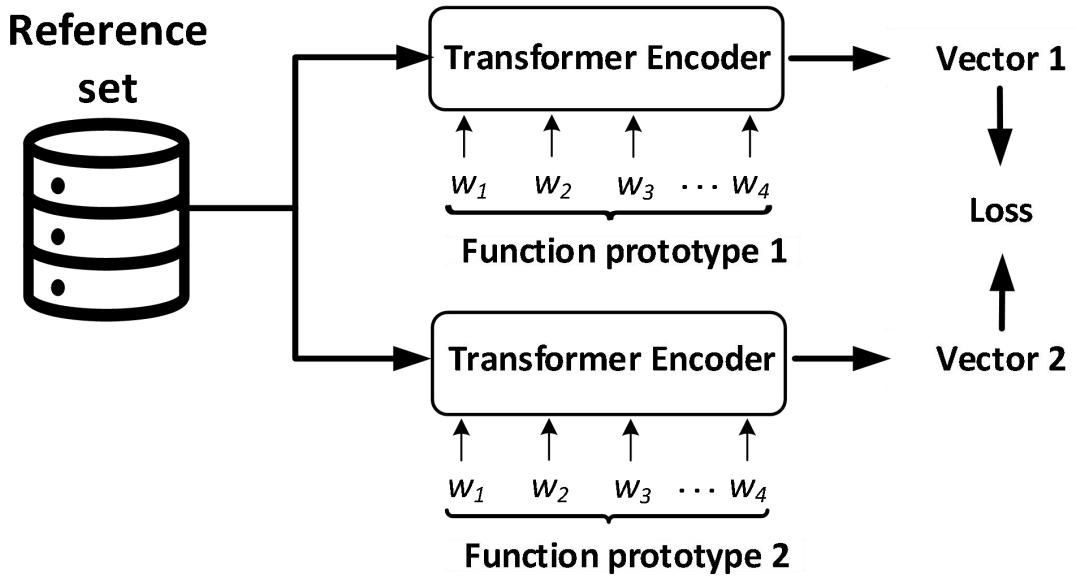
Goshawk: MOS-enhanced memory bugs detection system



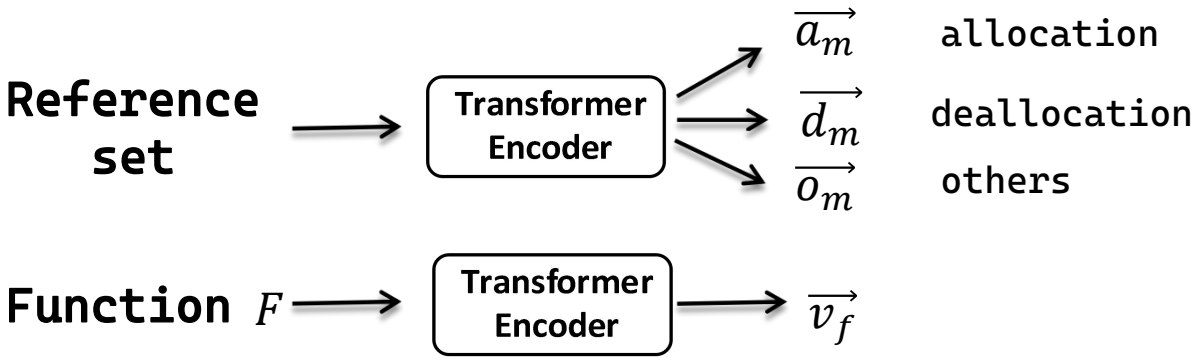
MM function Identification

NLP-assisted Classification

1. Segment and normalize function prototypes.
2. Convert function prototype into numeric vectors.
3. Compute the similarity with the reference set.



Siamese network training



$$Sim(F, X) = cosine(\overrightarrow{v_f}, \overrightarrow{x_m})$$

$$class(F) = \begin{cases} A, Sim(F, A) > max(Sim(F, D), Sim(F, O)) \\ D, Sim(F, D) > max(Sim(F, A), Sim(F, O)) \\ O, others \end{cases}$$

Function classification

MM function Identification

DFA-based Validation

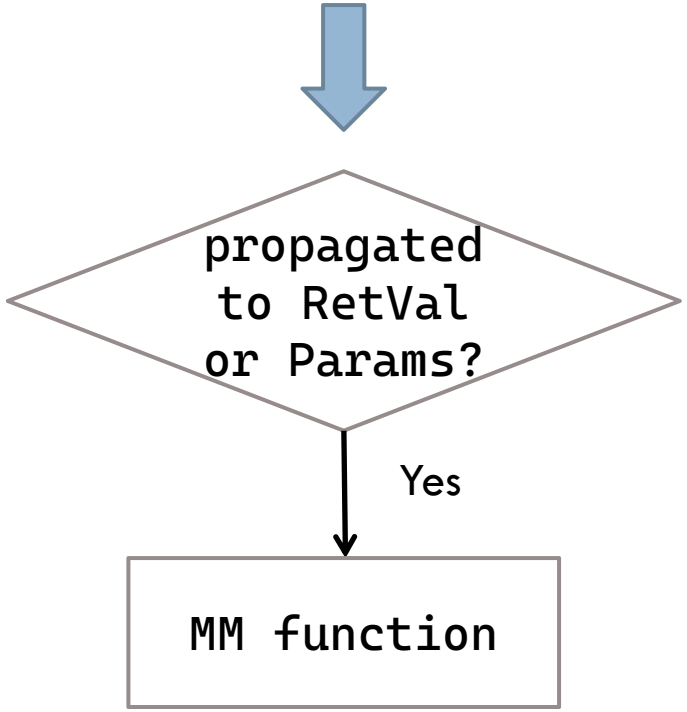
Interprocedural & backward analysis

```
1  static struct dma_page *pool_alloc_page(struct
2                  dma_pool *pool, gfp_t mem_flags)
3  {
4      struct dma_page *page;
5      page = kmalloc(...);
6      page->vaddr = dma_alloc_coherent(...);
7
8      return page;
9  }
10
11 static inline void *dma_alloc_coherent(...)
12 {
13     return dma_alloc_attrs(...);
14 }
15
16 void *dma_alloc_attrs(...)
17 {
18     cpu_addr = dma_direct_alloc(...);
19     return cpu_addr;
20 }
Official MM function set: kmalloc,...
```

❹ ❸ ❷ ❶

Official MM function set:

- collected from official documents
- 32 functions: *kmalloc*, *kfree*, *malloc*, *free*, *vmalloc*, *vfree*, etc.



MM function Identification

DFA-based Validation

Interprocedural & backward analysis

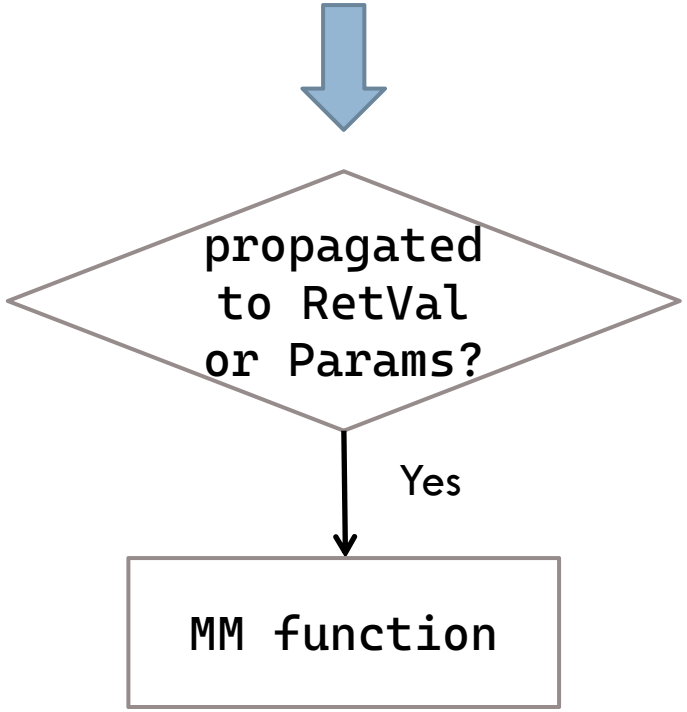
```
1  static void tpci200_uninstall(struct tpci200_board
2                                *tpci200)
3  {
4      tpci200_unregister(tpci200);
5      kfree(tpci200->slots);
6  }
7
8  static void tpci200_unregister(struct tpci200_board
9                                 *tpci200)
10 {
11     pci_iounmap(tpci200->info->pdev,
12                 tpci200->info->interface_regs);
13     pci_iounmap(tpci200->info->pdev,
14                 tpci200->info->cfg_regs);
15 }
16
17 void pci_iounmap(struct pci_dev *dev, void *addr)
18 {
19     ...
20     iounmap(addr);
21 }
```

❸ ❷ ❶

Official MM function set: kfree, iounmap, ...

Official MM function set:

- collected from official documents
- 32 functions: *kmalloc*, *kfree*, *malloc*, *free*, *vmalloc*, *vfree*, etc.



MOS Generation

Interprocedural & backward analysis

```
1  static struct dma_page *pool_alloc_page(struct
2                  dma_pool *pool, gfp_t mem_flags)
3  {
4      struct dma_page *page;
5      page = kmalloc(...);
6      page->vaddr = dma_alloc_coherent(...);
7
8      return page;
9  }
10
11 static inline void *dma_alloc_coherent(...)
12 {
13     return dma_alloc_attrs(...);
14 }
15
16 void *dma_alloc_attrs(...)
17 {
18     cpu_addr = dma_direct_alloc(...);
19     return cpu_addr;
20 }
Official MM function set: kmalloc,...
```

❹ ❸ ❷ ❶

Collected Data Flows

↓

Merge and abstract

↓

MOS Representation:

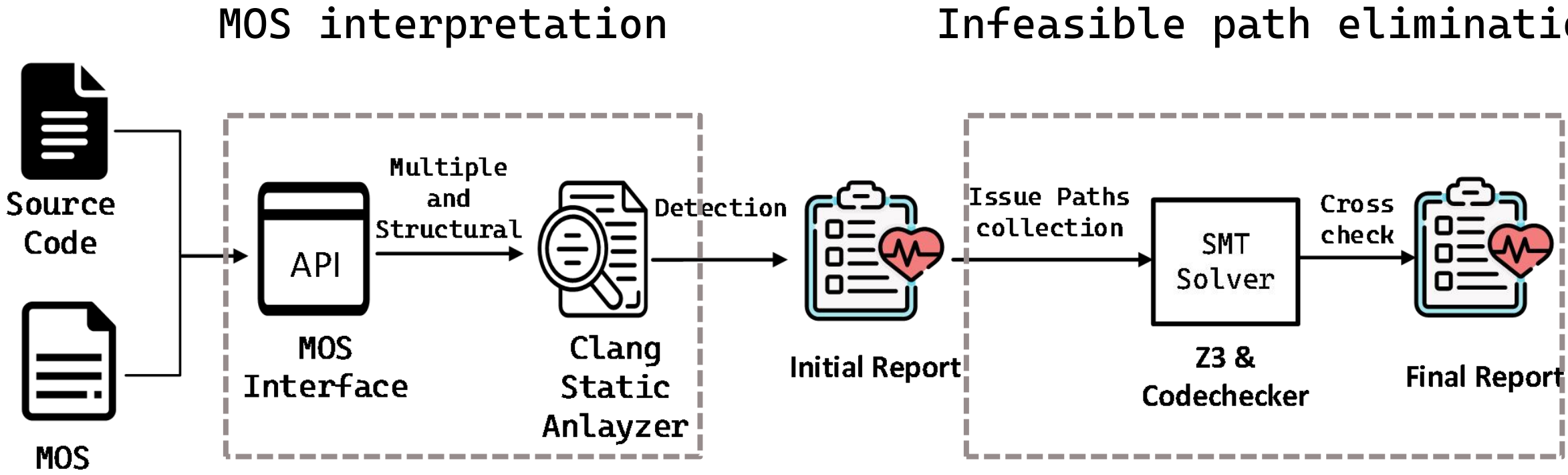
{

Function name		Property
pool_alloc_page	:	Allocator

Memory object list		Object type
RetVal	:	struct dma_page*
RetVal->vaddr	:	void*

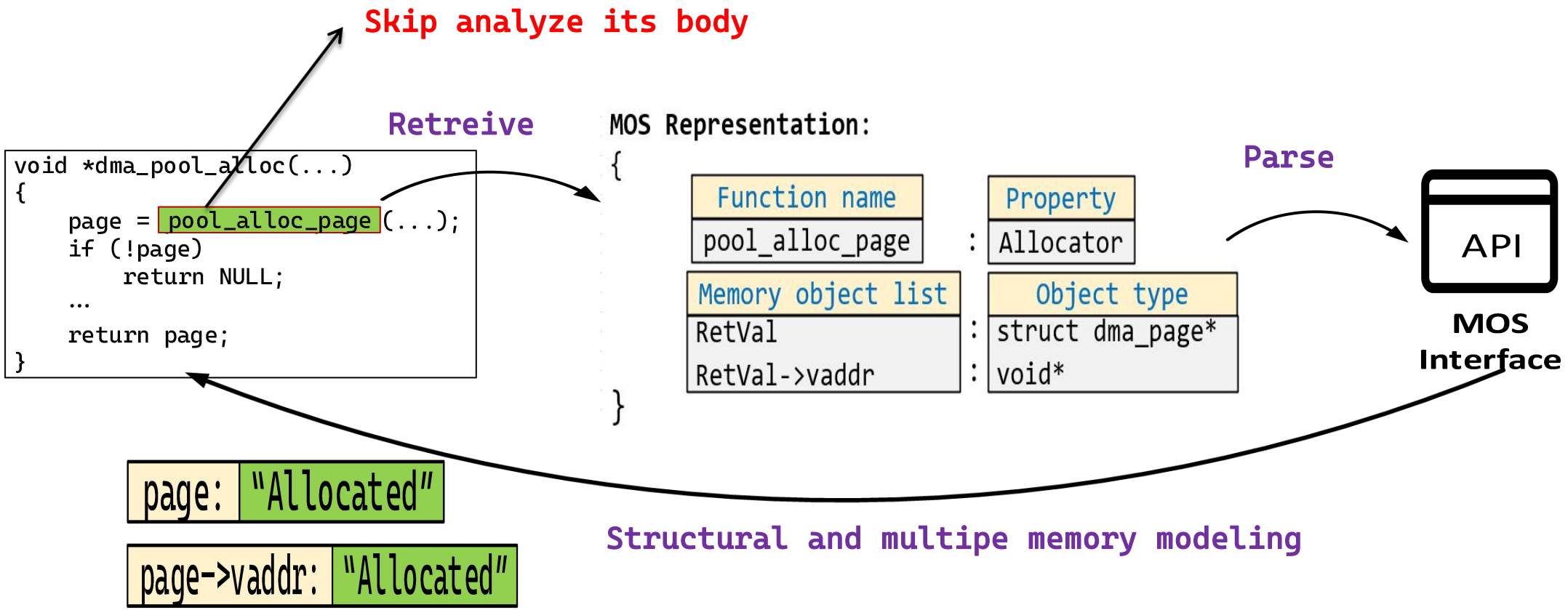
}

MOS-enhanced memory bugs detection



MOS-enhanced memory bugs detection

MOS interpretation



Infeasible paths elimination

```
1  static int hvfb_probe(struct hv_device *hdev,
2                const struct hv_vmbus_device_id *dev_id)
3  {
4       struct fb_info *info;
5
6       ❶ALLOCATE OBJECT info,info->apertures
7       info = framebuffer_alloc  (...);
9
10      ❷Deallocate OBJECT info->apertures
10       kfree(info->apertures);
11      info->status = Success ;
12
13 error2:
14
15      ❸Deallocate OBJECT info,info->apertures
16     framebuffer_release (info);
17
18      ❹info->apertures Double Free!
19 }
```

collect issue paths → CodeChecker with Z3 → cross check

```
1 void framebuffer_release(struct fb_info *info)
2 {
3       if ( info->status )
4             kfree(info->apertures);
5       kfree(info);
6 }
```

Infeasible Path !

Initial report

Outline

1. Motivation
2. Design
3. **Evaluation**
4. Conclusion

Overall Results

TABLE I
RESULTS OF CUSTOMIZED MEMORY MANAGEMENT FUNCTION IDENTIFICATION AND MEMORY BUG DETECTION IN POPULAR OPEN-SOURCE PROJECTS

Program	Version	Lines of Code	Number of Functions	Identified Allocators		Identified Deallocators		Detected Bugs		
				Primitive	Extended	Primitive	Extended	UAF	DF	Confirmed
Linux	5.12-rc2	20.1M	549K	2,916	1,805	1,812	4,266	32	17	40
FreeBSD	13.0.0	5.9M	99K	482	273	393	853	9	9	17
OpenSSL	3.0.0	456K	19K	93	15	134	193	1	8	9
Redis	6.2.1	149K	3K	51	9	28	41	1	0	1
Azure	2021_Ref01	661K	4.8K	126	7	58	177	0	4	1
QcloudE	3.1.8	86K	759	21	1	26	21	2	5	7
QcloudH	3.2.3	79K	638	20	1	25	21	2	2	4
Total	-	-	-	**3,709**	**2,111**	**2,476**	**5,572**	**47**	**45**	**79**

QcloudE refers to Qcloud IoT Explorer Device SDK and QcloudH refers to Qcloud IoT Hub Device SDK.
UAF: use-after-free bug; **DF**: double-free bug

- Identified more than 10,000 custom MM functions
- MM function identification is precise.
- Detected 92 new use-after-free and double-free bugs
- Achieves a low false positive.

	Allocator		Deallocator	
	Precision	Recall	Precision	Recall
w/o DFA	89.6%	91.0%	90.2%	97.0%
w/ DFA	100%	84.5%	100%	89.5%

Comparison with MallocChecker

TABLE II
COMPARISON WITH MALLOCCHECKER IN TERMS OF DETECTION EFFECTIVENESS AND EFFICIENCY (IN MINUTES)

	Goshawk		MalChk-S		MalChk-E	
	Bugs	Time	Bugs	Time	Bugs	Time
Linux	49	328.91	2	367.72	-	-
FreeBSD	18	20.84	5	19.73	6	664.76
OpenSSL	9	3.51	1	3.66	1	79.13
Redis	1	1.03	0	0.78	0	24.01
Azure	4	0.22	2	0.26	2	6.41
QcloudE	7	0.10	4	0.11	7	1.77
QcloudH	4	0.07	4	0.08	4	8.66
Presicion	**63.4%**		**20.6%**		**35.7%**	

MalChk-S : MallocChecker with AMD 3990x, 192G RAM.
MalChk-E : MallocChecker with Xeon 4126, 1T RAM.

Goshawk vs. MalChk-S:

- under the **same** experiment setting
- AMD 3990x, 192G RAM
- AD=5, AB=225,000, CTU=100

MalChk-E:

- more **extensive** experiment setting
- Xeon 4126, 1TG RAM
- AD=10, AB=2,250,000, CTU=20,000

AD: Analysis depth, AB: Analysis breadth, CTU: Cross translations units

Comparison with related works

TABLE IV
COMPARISON OF MM FUNCTION IDENTIFICATION IN THE LINUX KERNEL

Version	K-MELD		SinkFinder		Goshawk			
	A	D	A	D	A	D	MN	MD
v5.2.13	806	461	-	-	4,571	4,847	19	129
v4.19	-	-	256	182	4,396	4,704	8	46

A : # of allocators; **D** : # of deallocators;
MN : # of MM functions missed by NLP-assisted classification;
MD : # of MM functions missed by DFA based validation.

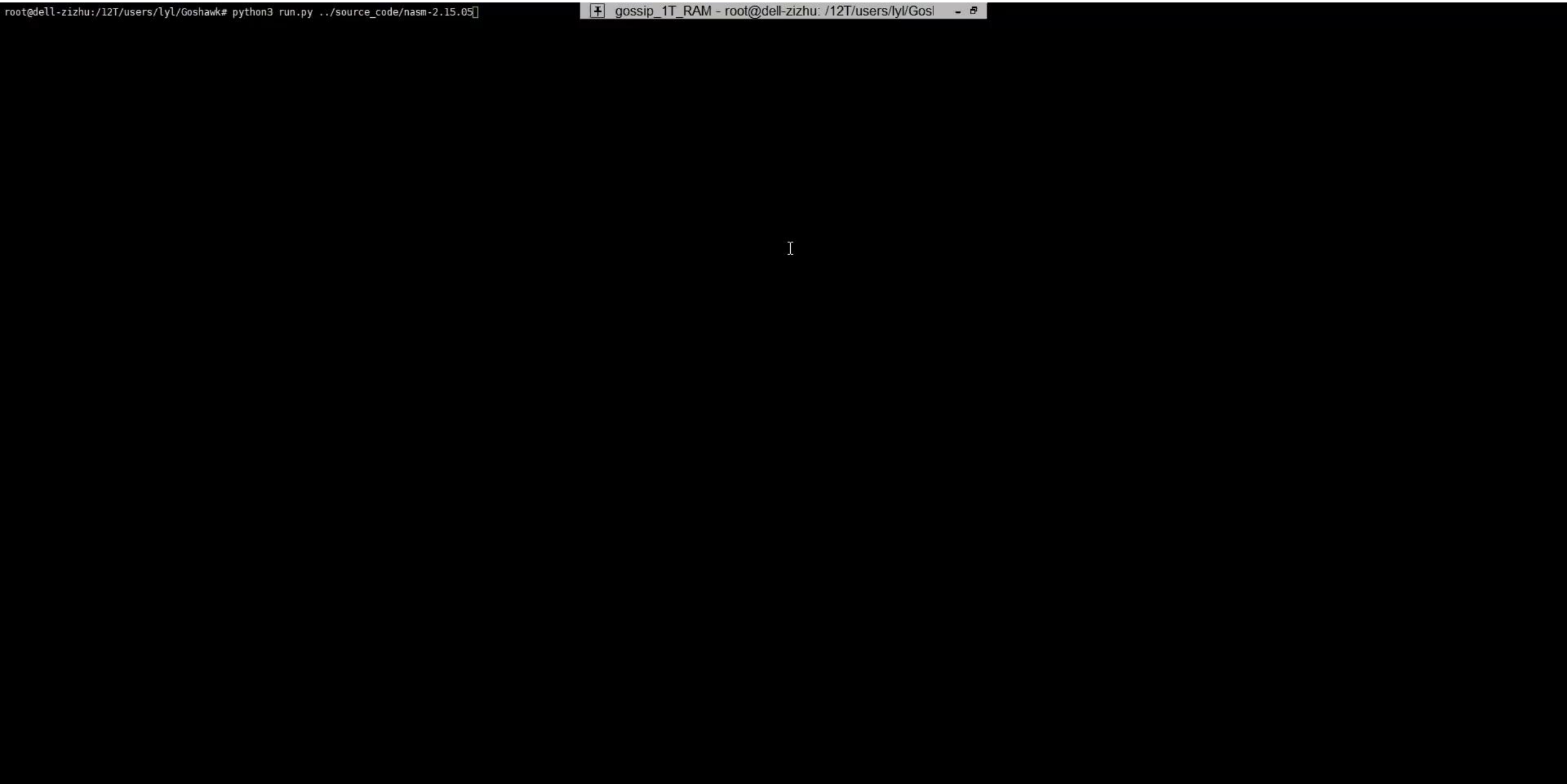
MM function identification:

- Outperforms by orders of magnitude
- Applicable to different scales codes

Bug detection:

- High coverage
- Less false warnings

Demonstration



Outline

1. Motivation
2. Design
3. Evaluation
4. **Conclusion**

Conclusion

- MOS: a novel concept to enhance memory bug detection
- Goshawk: a easy-to-use bug detection tool
- Found 92 new bugs in Linux/BSD kernels, OpenSSL, Redis, etc.
- Found new bugs in CAIRO, Flite, NASM after our submission!

Download and try Goshawk by visiting https://goshawk.code-analysis.org/

Q&A
Thanks!