AsiaCCS 2023 Tutorial

Custom Memory Functions Demystified: A tutorial of memory corruptions detection using Goshawk

Xiang Chen, Siqi Ma, Zihan Ni, Shangzhi Xu

Shanghai Jiao Tong University

The University of New South Wales, Australia

CSIRO, AUS

Agenda

- Preparation - 5’
- Introducing Goshawk - 20’
- Simple demo using Goshawk - 10’
- Interaction - 30’
- Result Review - 15’
- Q&A

It takes about **80** minutes in total.

Preparation

Connect to Goshawk server

Select a username from user001 to user100 and use this command:

```
ssh -p 12750 user001@sky.gossip.team # password: asiaccs2023
```

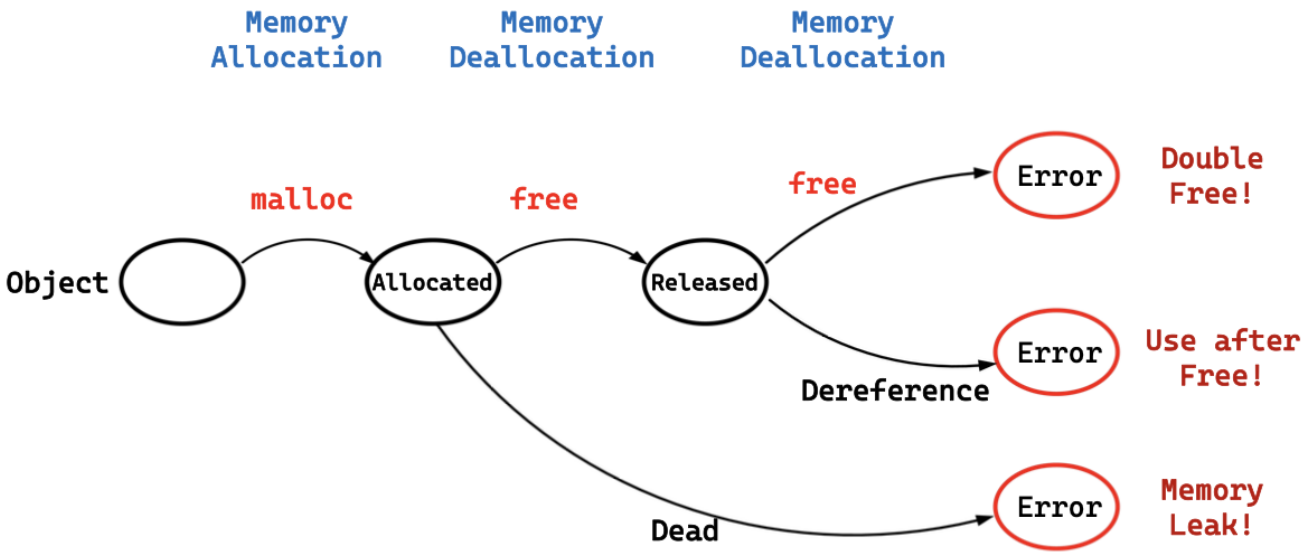
Everyone will have the following directories:

- Goshawk project
- Two C/C++ projects ready to be analyzed
- ...(Other C/C++ projects you like to analyze)

Introducing Goshawk

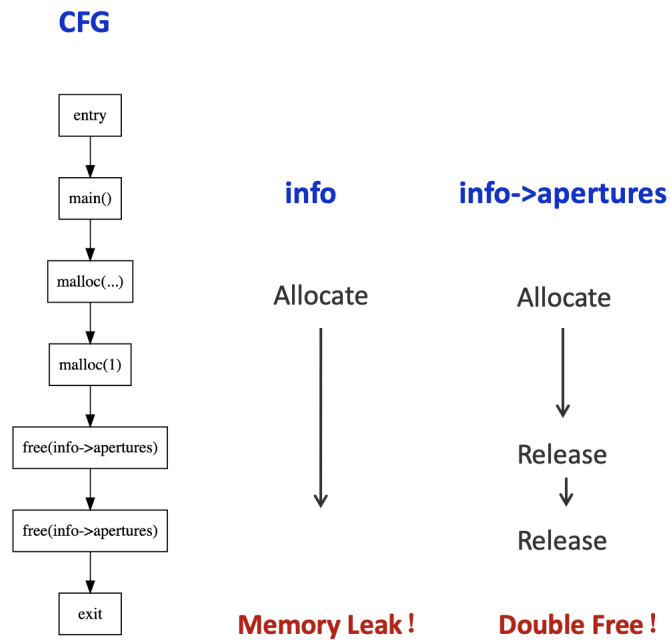
Traditional memory corruption detection

- Create a **CFG** (control flow graph) & Scan the CFG - ClangStaticAnalyzer MallocChecker



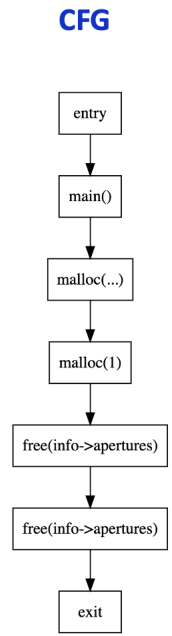
A simple case

```
1   int main(){
2       struct fb_info *info;
3
4       info = malloc(...);
5
6       info->apertures = malloc(1);
7
8       free(info->apertures);
9
10      free(info->apertures);
11
12      return 0;
13  }
```



A simple case ✅

```
1   int main(){
2       struct fb_info *info;
3
4       info = malloc(...);
5
6       info->apertures = malloc(1);
7
8       free(info->apertures);
9
10      free(info->apertures);
11
12      return 0;
13  }
```



info

Allocate

Memory Leak !

info->apertures

Allocate

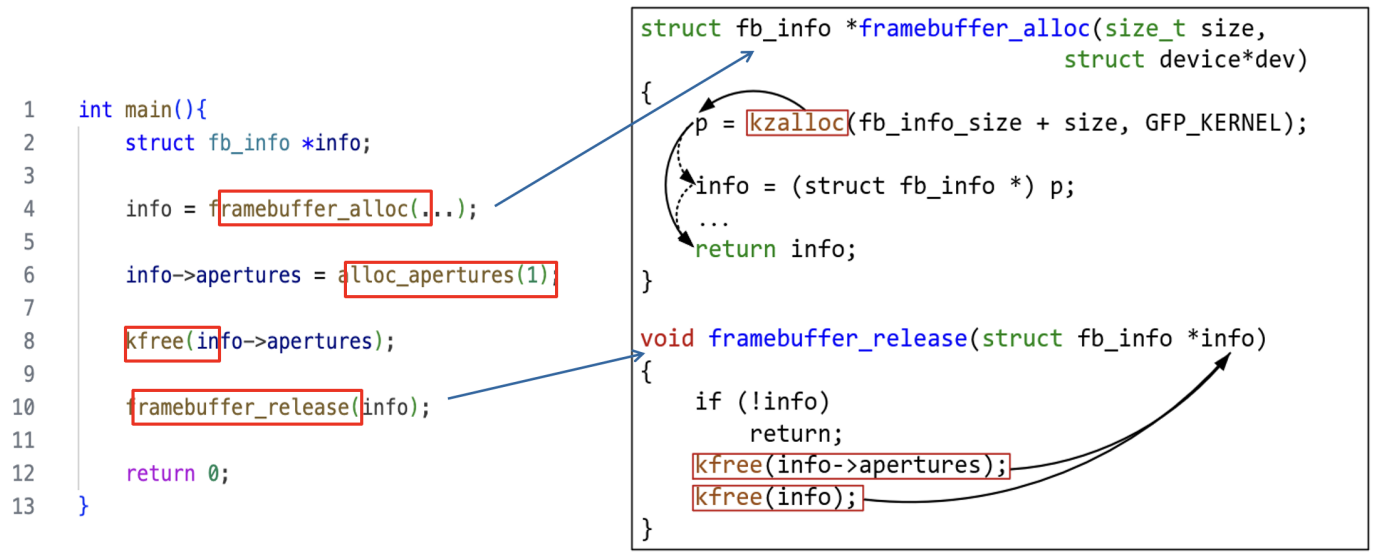
Release

Release

Double Free !

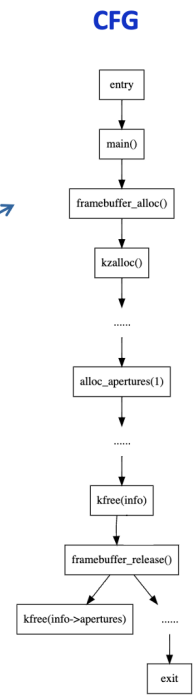
PS: more cases can be found at: ClangStaticAnalyzer testsuite

A complex (real-world) case



A complex (real-world) case

```
1   int main(){
2       struct fb_info *info;
3
4       info = framebuffer_alloc(...);
5
6       info->apertures = alloc_apertures(1);
7
8       kfree(info->apertures);
9
10      framebuffer_release(info);
11
12      return 0;
13  }
```



info	info->apertures
Allocate	Allocate
↓	↓
	Release
↓	↓
Release	Release

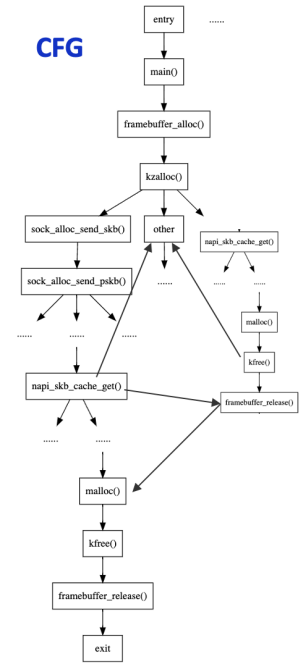
Double Free !

A complex (real-world) case ❌

```c
1   int main(){
2       struct fb_info *info;
3
4       info = framebuffer_alloc(...);
5
6       info->apertures = alloc_apertures(1);
7
8       kfree(info->apertures);
9
10      framebuffer_release(info);
11
12      return 0;
13  }
```

```c
struct fb_info *framebuffer_alloc(size_t size,
                                  struct device*dev)
{
    p = kzalloc(fb_info_size + size, GFP_KERNEL);
    info = (struct fb_info *) p;
    ...
    return info;
}
```

```
framebuffer_alloc()
-> kzalloc()
-> 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()
-> malloc()
```



Diving into custom MM functions is not scalable, and in most cases not feasible 🤦

Another complex (real-world) case ❌

```
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  }
```

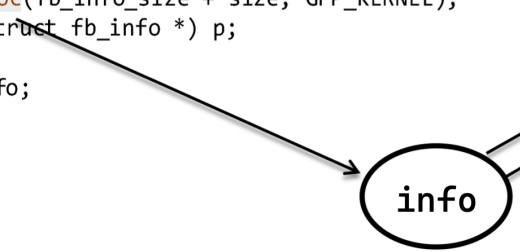
```
37      return info;
38  }
39  static int hvfb_getmem(struct hv_device *hdev, ...)
40  {
41      ...
42    ❸ info->apertures = alloc_apertures(1);
43      ...               //info->apertures allocation
44      pdev = pci_get_device(...);
45      if (!pdev) {
46        ❹ kfree(info->apertures);//info->apertures 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

Another complex (real-world) case ❌

```
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

Challenge - A Huge GAP

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

Our solution - MOS model

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

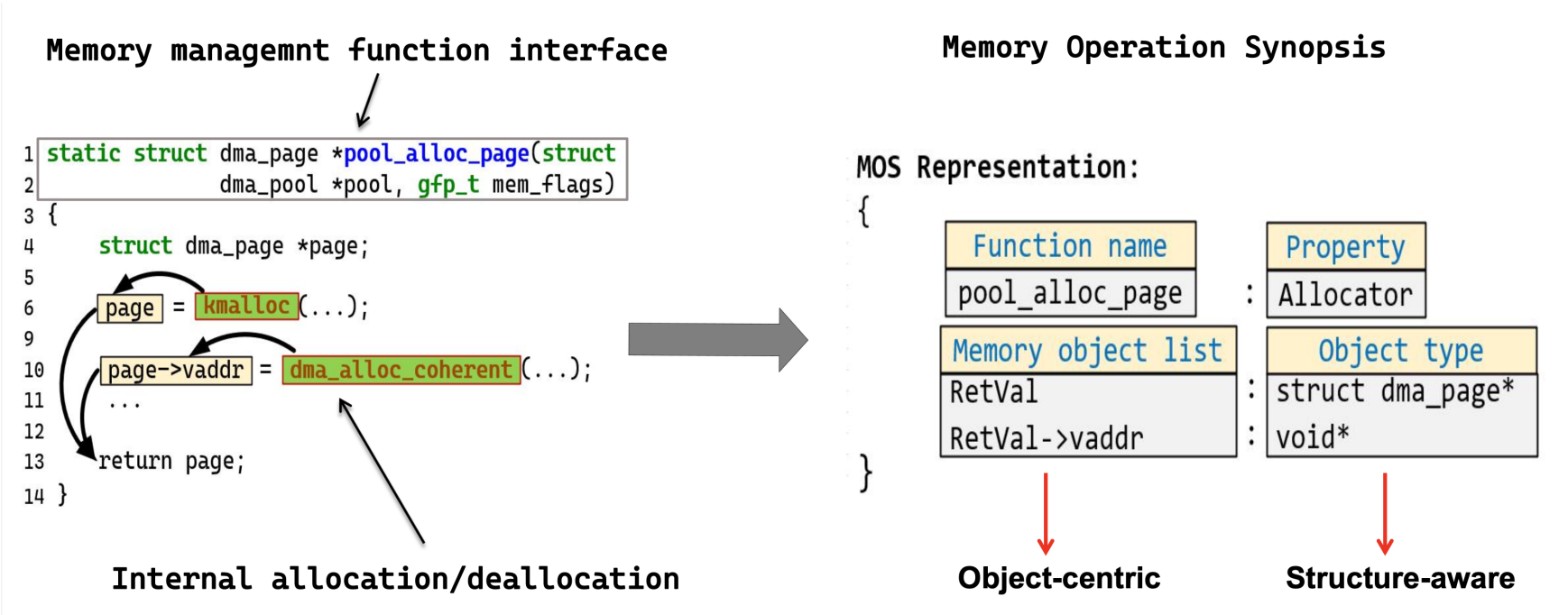
Yunlong Lyu, Yi Fang, Yiwei Zhang, Qibin Sun, Siqi Ma,
Elisa Bertino, Kangjie Lu, and Juanru Li

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

IEEE S&P 2022

The MOS model

Below is a MOS model summarized from `Linux/mm/dmapool.c`



MOS model showcase

Applying MOS model to the previous mentioned complex case:

```
1  int main(){
2      struct fb_info *info;
3
4      info = framebuffer_alloc(...);
5
6      info->apertures = alloc_apertures(1);
7
8      kfree(info->apertures);
9
10     framebuffer_release(info);
11
12     return 0;
13 }
```

MOS Interface

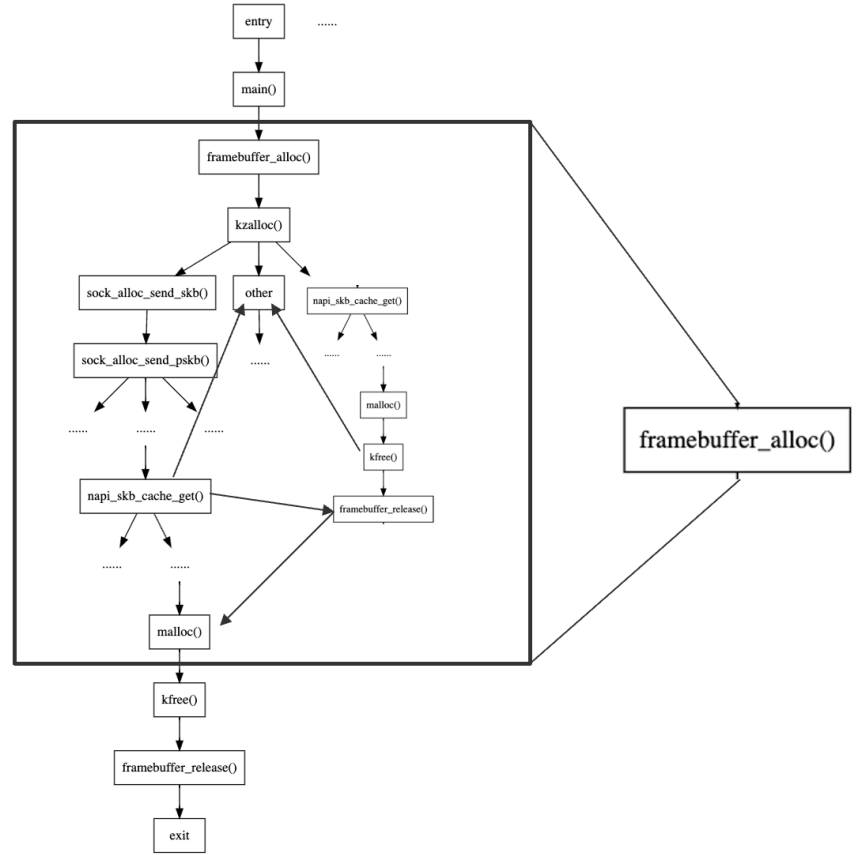
MOS translate

MOS translate

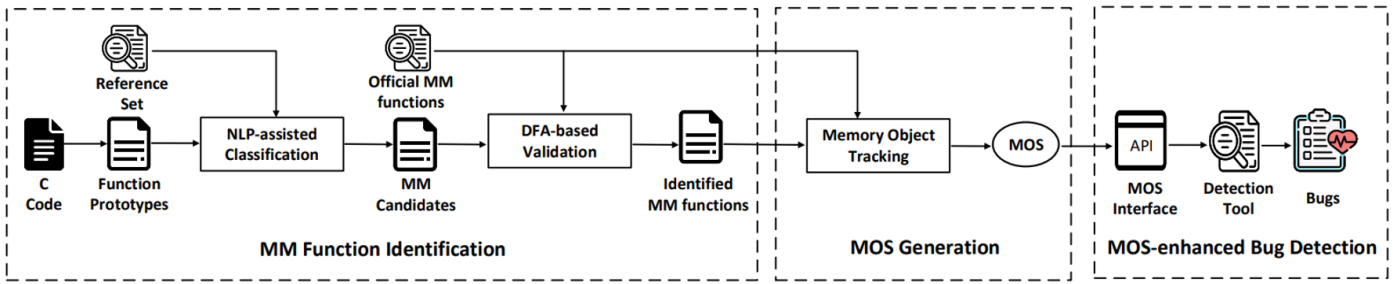
MOS translate

MOS translate

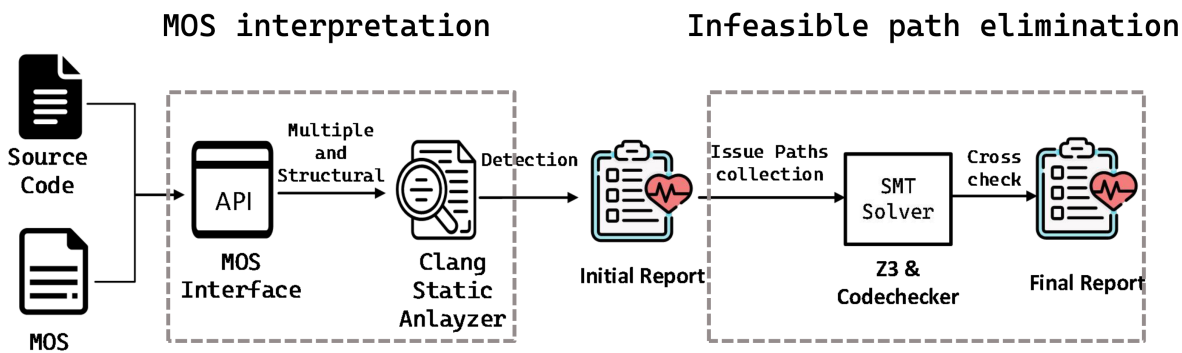
info	info->apertures
Allocate	Allocate
	Release
Release	Release
	Double Free !



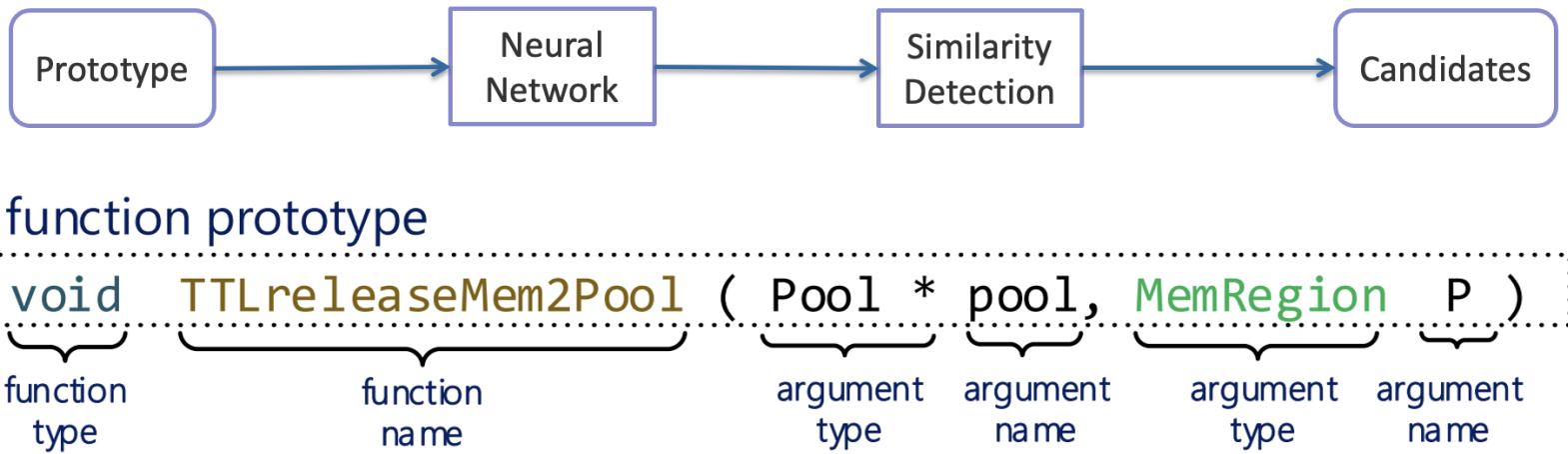
Goshawk workflow



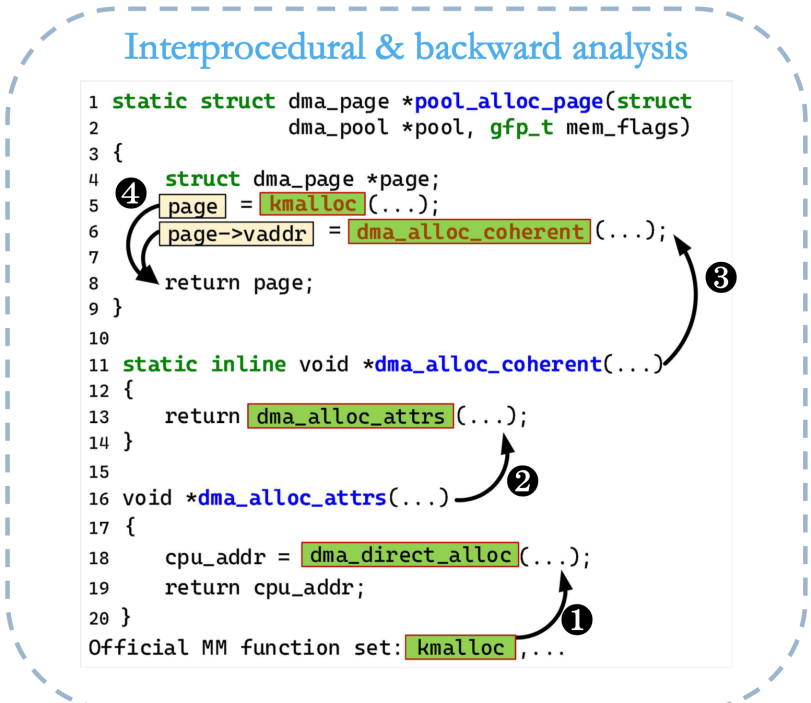
The last step is the core idea - **simplifying CFG and data-flow analysis**



Technical details - NLP-asstisted clasification

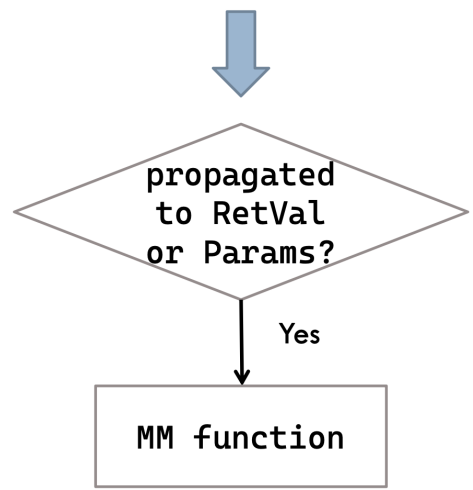


Technical details - DFA-based validation

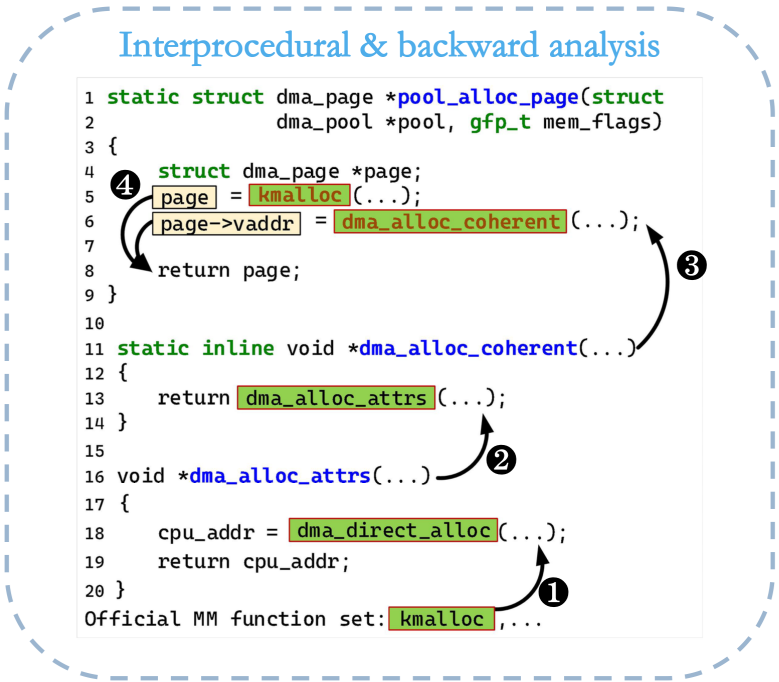


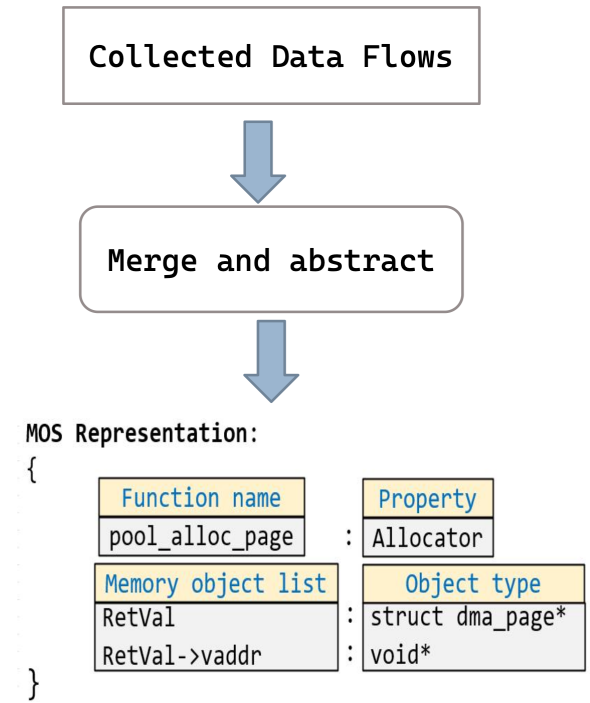
Official MM function set:

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



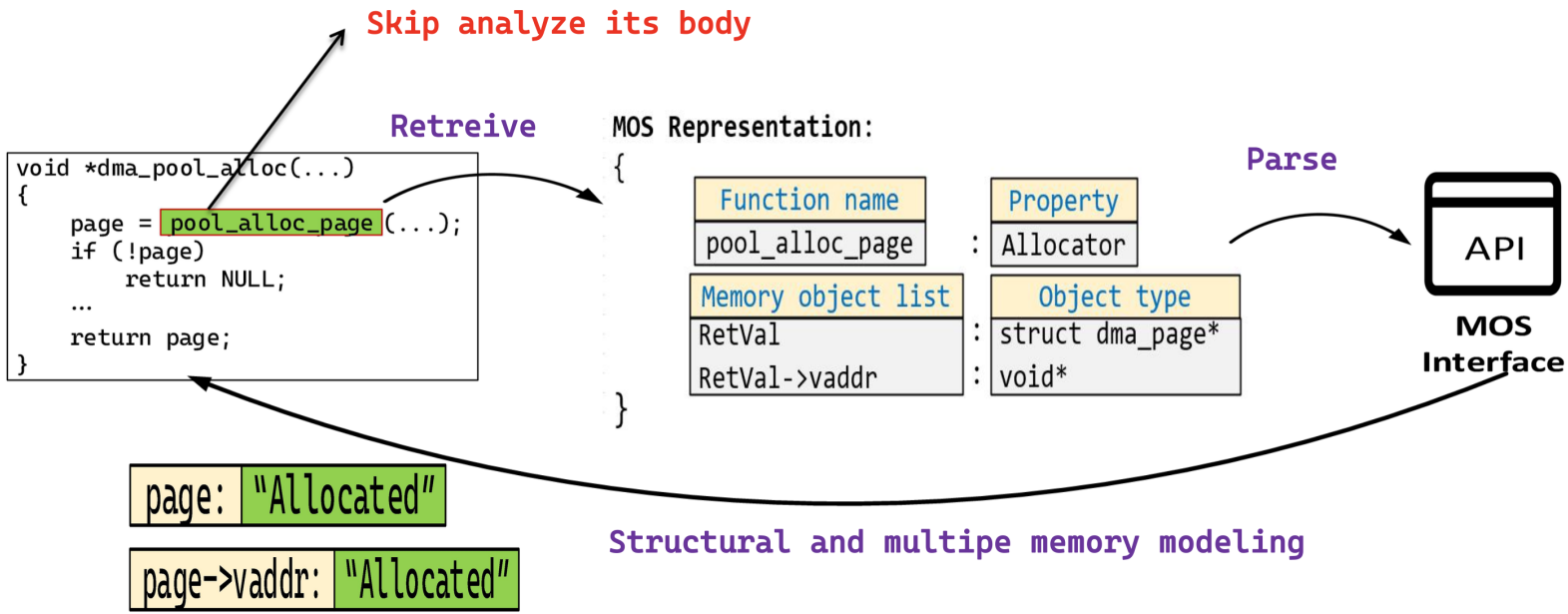
Technical details - MOS model generation



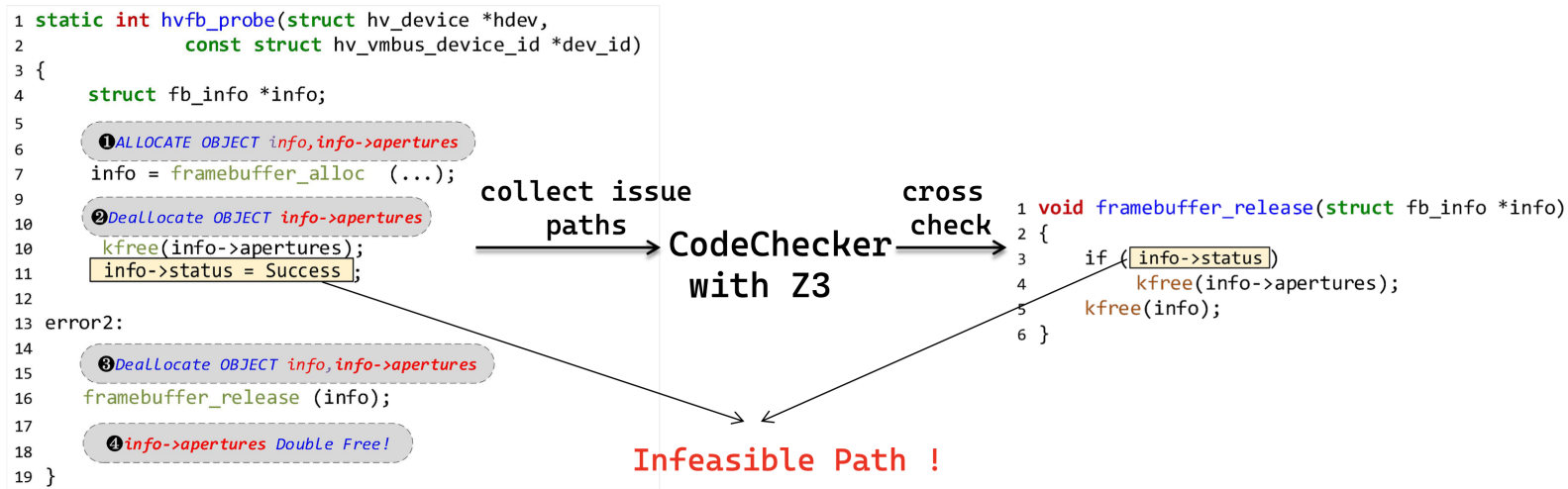


Technical details - MOS model interpretation

MOS interpretation



Technical details - Infeasible paths elimination



Initial report

Goshawk Trophies

By the time of paper publication, Goshask has identified **10K+** custom MM functions, detected **92** new use-after-free and double-freebugs while kept a low FP rate.

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

Motivation of tutorial

- Goshawk is still evloving
  - ported to Clang-15
  - more user-friendly run script
  - re-train script for other kinds of custom functions
- Goshawk has been applied to a wider range of softwares
  - **19** bugs (13 UAFs and 2 Double Frees) in 90 OpenWrt pacakges are discovered and confirmed
  - the average length of these bugs' data-flow trace is 16.86

Simple demo

Apply Goshawk to redsocks

Interaction

Try Goshawk self!

Software environment

- Ubuntu 22.04.2 LTS
- Python 3.10.6
- Bear 3.1.2
- Clang 15.0.0
- CodeChecker 6.22.1

Useful Commands

- Get compilation database: `cd <project_path>; bear -- make CC=clang HOSTCC=clang`
- Identify MM functions: `cd <Goshawk_path>; python3 run.py -s=extract <project_path>`
- Dataflow analysis: `cd <Goshawk_path>; python3 run.py -s=analyze <project_path>`

Result review

http://code-analysis.net

Q&A

All the resources has been public available via:

https://github.com/cascades-sjtu/Goshawk-tutorial

Feel free to create an issue about this tutorial

Try other analysis tools made by us: https://code-analysis.org