Prevention of C/C++ Pointer Vulnerability

Dexter An
Division of Science & Mathematics
University of Minnesota, Morris
April 20, 2019

List of Contents

- Background of C/C++ Memory Allocation and Pointer
- Pointer Vulnerability and Attacks
- Type-after-Type Type Safe Memory Reuse
- Machine-Learning-Guided Static UAF Detection
- Delta Pointer
- Conclusion

Background Information

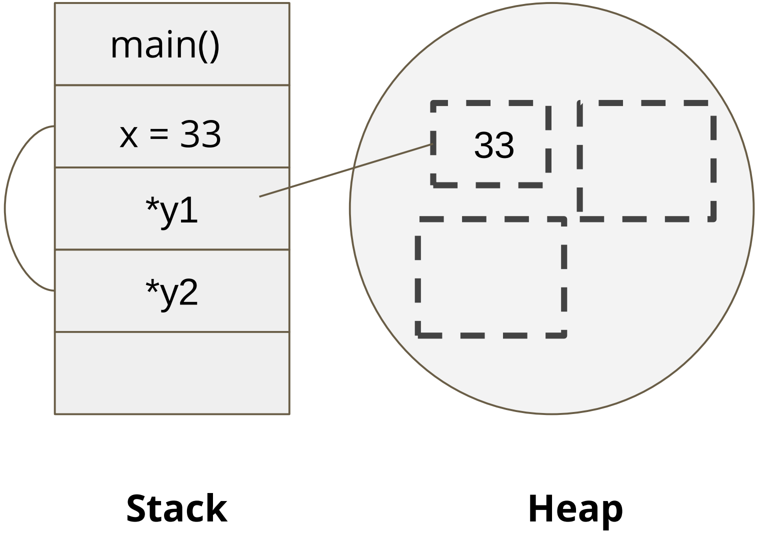
Actual values of y1 will be like 0x7fffa057dd4

Memory Allocation of C/C++ Languages

- Dynamic memory allocation
- Stack & Heap
- Basic procedures: malloc() and free()
- Invention of C/C++: 1978 & 1980s

Pointer

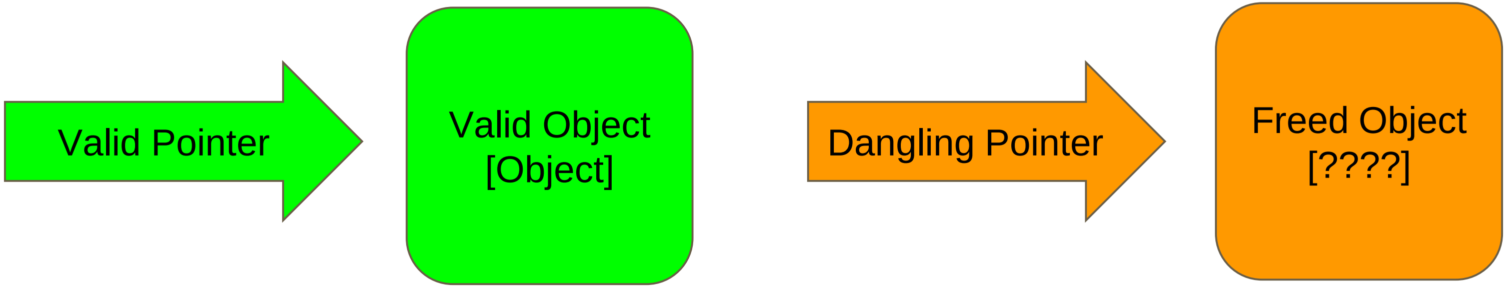
- Address locator of C/C++
- y1 & y2 are pointers



Pointer Attack Scenarios

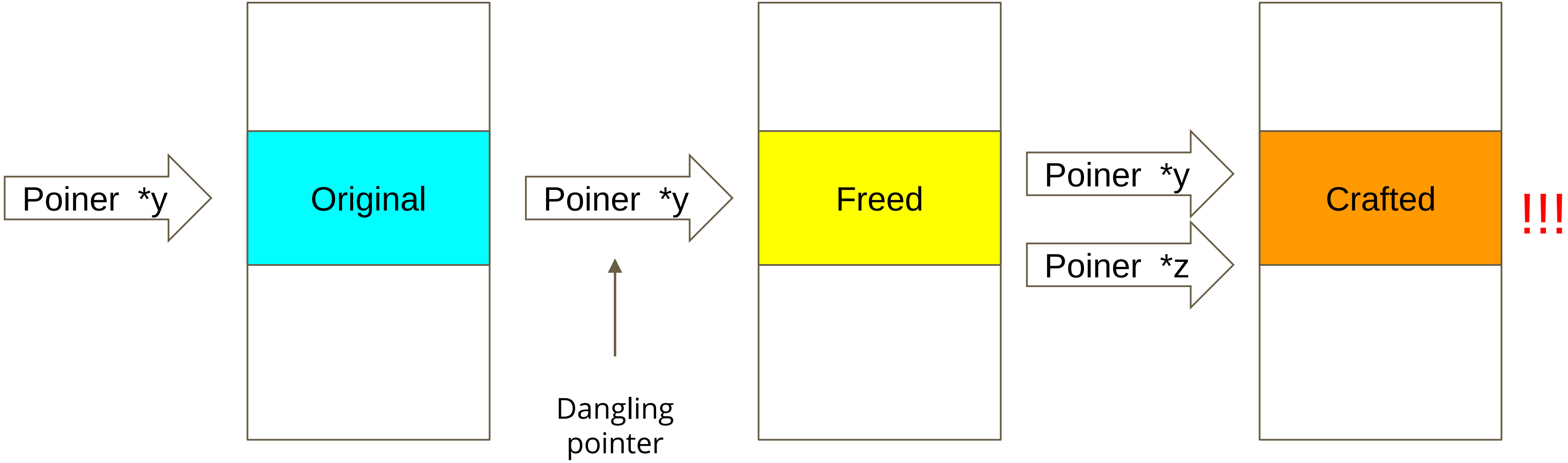
Use-After-Free Attack (UAF):

- Dangling pointer
- Reallocation to attacker-controlled data



Pointer Attack Scenarios

Use-After-Free Attack (UAF):

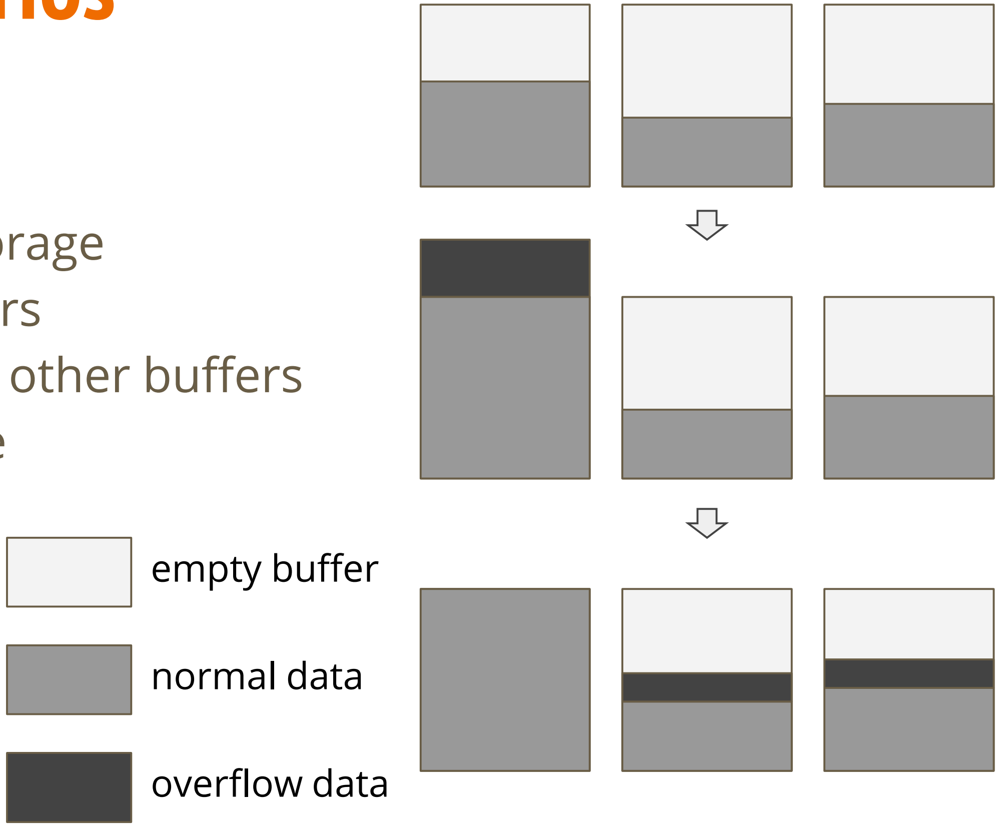


Pointer Attack Scenarios

Buffer Overflow Attack:

- Buffer -- temporary data storage
- Data leak out to other buffers
- Corrupt & overwrite data of other buffers
- Inplant harmful data & code

Buffer overflow



Type-after-Type (TAT) Memory Reuse

Type Specification for Memory

- Prevents attackers take of advantage of dangling pointers
- Lower resource cost comparing the existing methods
- Heap site of specification
- Stack site of specification

Type-after-Type (TAT) Memory Reuse

Type Specification for Memory

On stack:

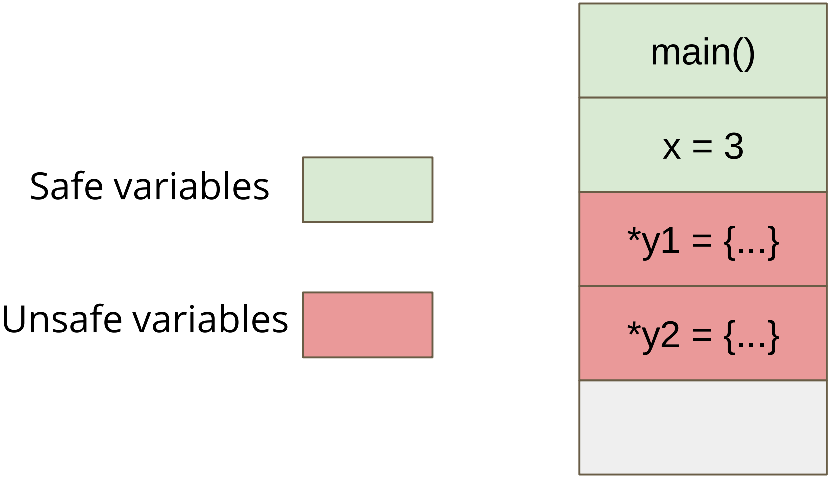
- Safe variables:

    function names,

    global / local variables

- Unsafe variables:

    pointers



Safe variables 🟩

Unsafe variables 🟥

main()

x = 3

*y1 = {...}

*y2 = {...}

Type-after-Type (TAT) Memory Reuse

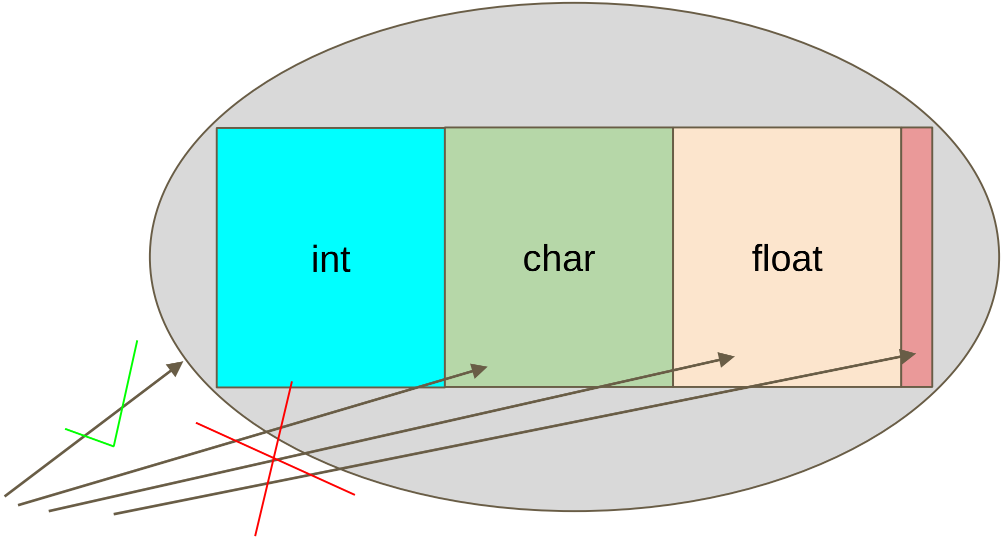
Type Specification for Memory

On heap:

*x = 20

free(x)

*x = ??



int

char

float

Available memory on heap

Machine-Learning-Guided Detection

Static Use-After-Free Vulnerability Detector

Focus on large scale program and reducing false detection

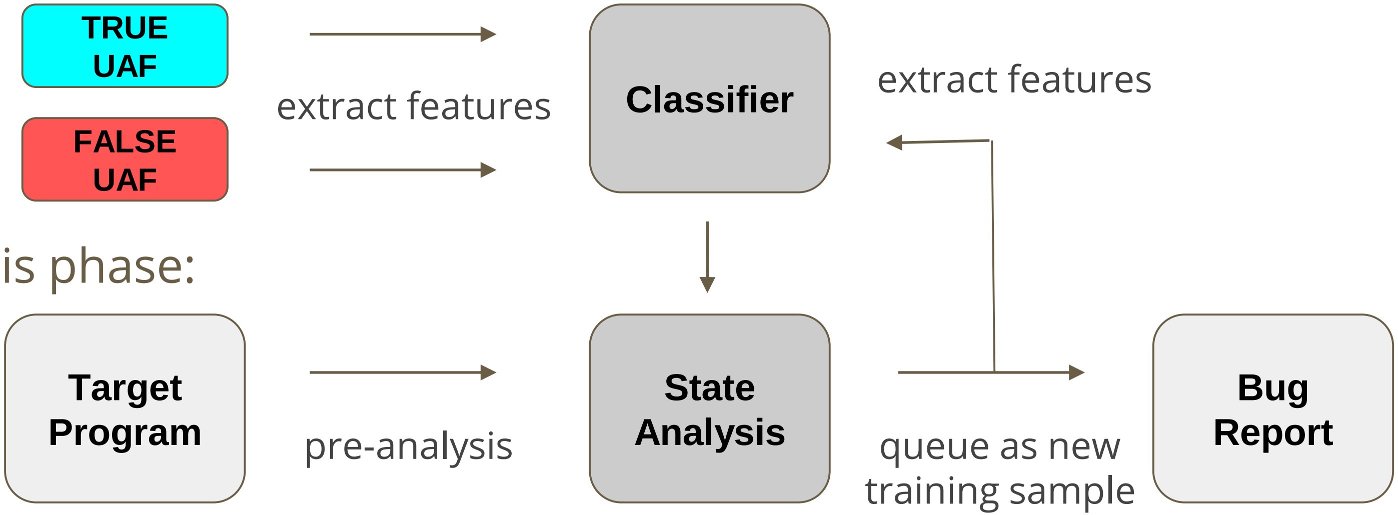
Features of the samples:

- Type information (e.g., global, array and struct)
- Control overflow (e.g., loop and recursion)
- Common practice (e.g., pointer casting and reference counting)
- Points-to information (e.g., the number of objects that may be used at a use site and the number of UAF pairs sharing the same free site)

Machine-Learning-Guided Detection

Static Use-After-Free Vulnerability Detector

- Training phase:



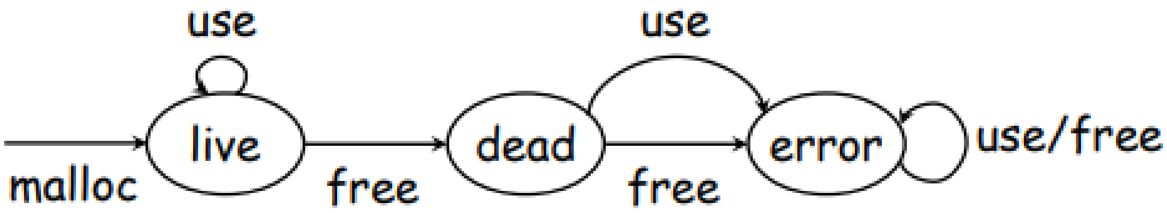
- Analysis phase:

Machine-Learning-Guided Detection

Static Use-After-Free Vulnerability Detector

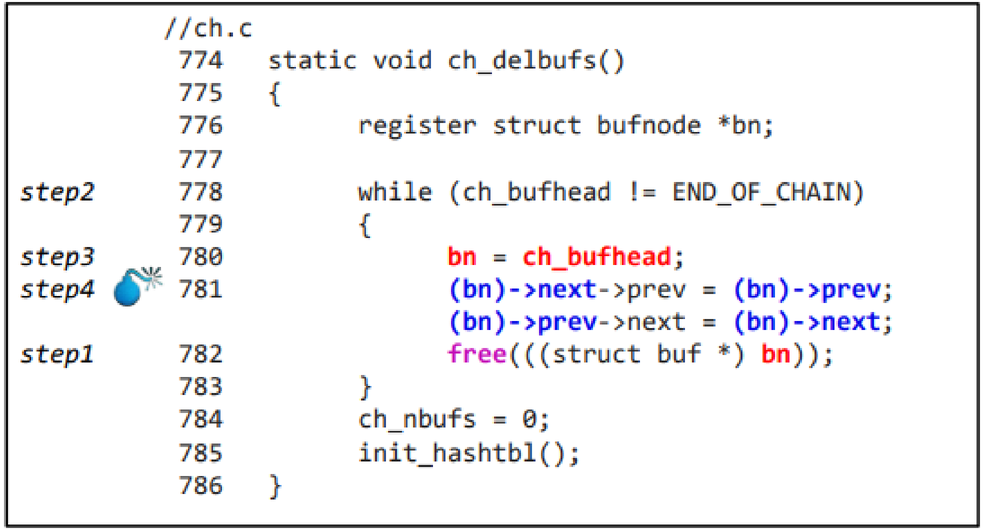
- Pre-analysis
- State analysis

State graph



UAF error in state graph

Machine-Learning-Guided Detection



```
          //ch.c
          static void ch_delbufs()
          {
                  register struct bufnode *bn;

step2             while (ch_bufhead != END_OF_CHAIN)
                  {
step3                     bn = ch_bufhead;
step4  💣                 (bn)->next->prev = (bn)->prev;
                          (bn)->prev->next = (bn)->next;
step1                     free(((struct buf *) bn));
                  }
                  ch_nbufs = 0;
                  init_hashtbl();
          }
```

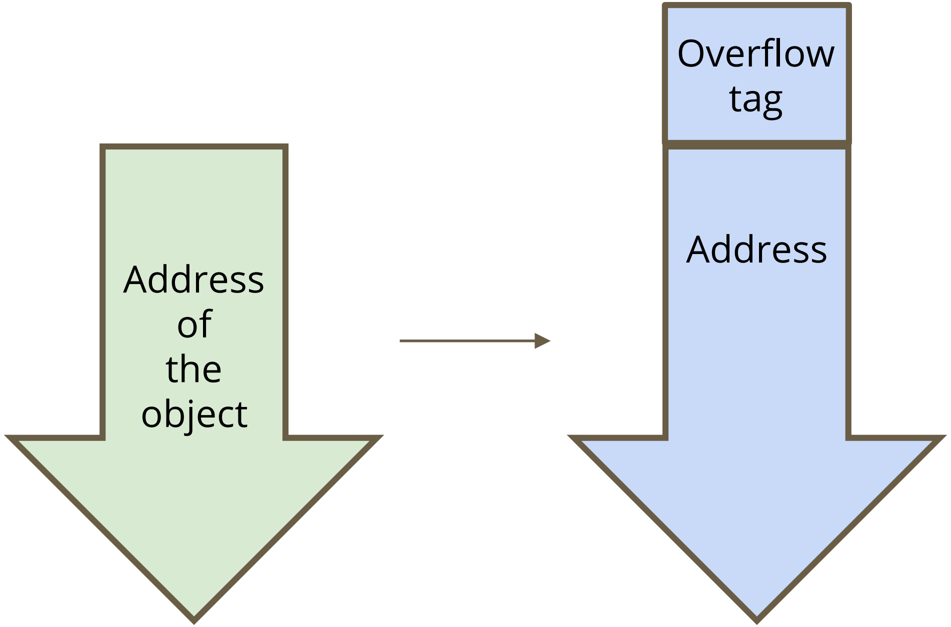
bn is dereferenced 4 times without updating its information

Example of detecting UAF error in early version of ***less***

Delta Pointers

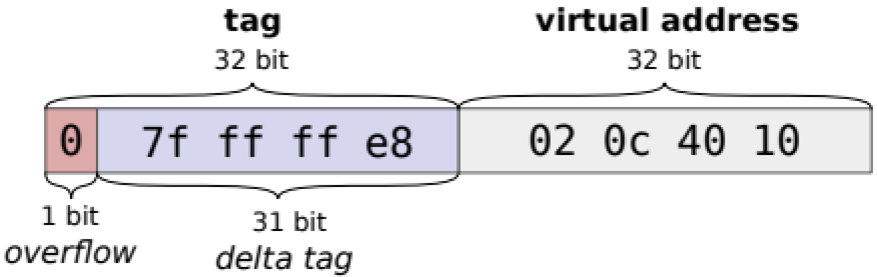
Low Resource Cost Pointer Tagging

- Each Delta pointer has its "overflow" tag
- Cause run-time error to stop the program avoiding the further damage



Delta Pointers

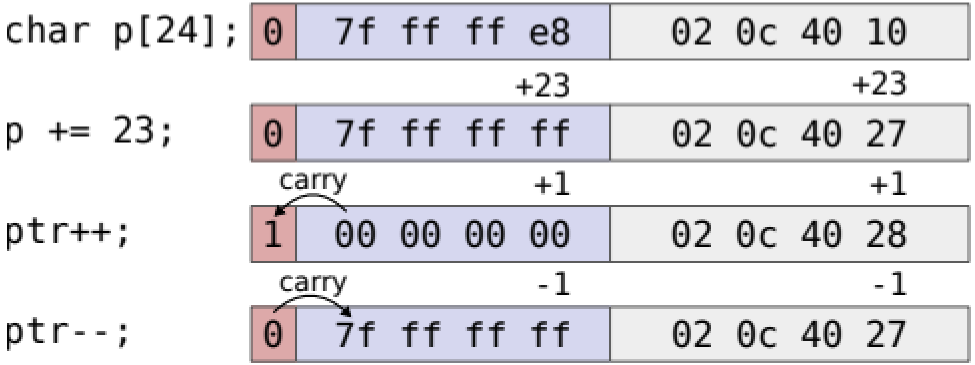
Low Resource Cost Pointer Tagging



Delta pointer structure

Delta Pointers

Low Resource Cost Pointer Tagging

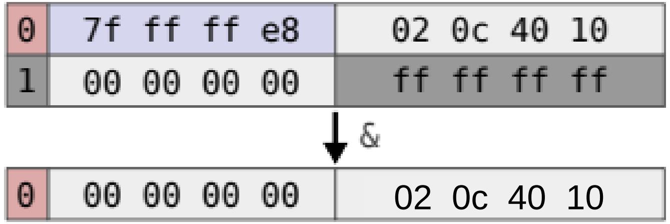


char p[24];	0	7f ff ff e8	02 0c 40 10
		+23	+23
p += 23;	0	7f ff ff ff	02 0c 40 27
	carry	+1	+1
ptr++;	1	00 00 00 00	02 0c 40 28
	carry	-1	-1
ptr--;	0	7f ff ff ff	02 0c 40 27

Overflow detection

Delta Pointers

Low Resource Cost Pointer Tagging



Pointer:	0	7f ff ff e8	02 0c 40 10
Mask:	1	00 00 00 00	ff ff ff ff
Returning address:	0	00 00 00 00	02 0c 40 10

Retrieving address

Conclusion

- C/C++ are widely used
- Dynamic allocation has both good and bad parts
- Balance between resource cost vs. effectiveness on defense
- Overview:

  Type-after-Type type safe memory reuse (low cost)

  Machine-Learning-Guided UAF detector (high accuracy & precision)

  Delta pointers (fast)

References & Acknowledgments

Hua Yan, Yulei Sui, Shiping Chen, and Jingling Xue. 2017. Machine-Learning_Guided Typestate Analysis for Static Use-After-Free Detection.

Taddeus Kroes, Koen Koning, Erik van der Kouwe, Herbert Bos. 2018. Delta Pointers: Buffer Overflow Checks Without the Checks.

Van der Kouwe, Erik and Kroes, Taddeus and Ouwehand, Chris and Bos, Herbert and Giuffrida, Cristiano. 2018. Type-After-Type: Practical and Complete Type-Safe Memory Reuse.

Thank the writers, my advisor Elena Machkasova, and peers who gave feedbacks to the presentation.

Questions?