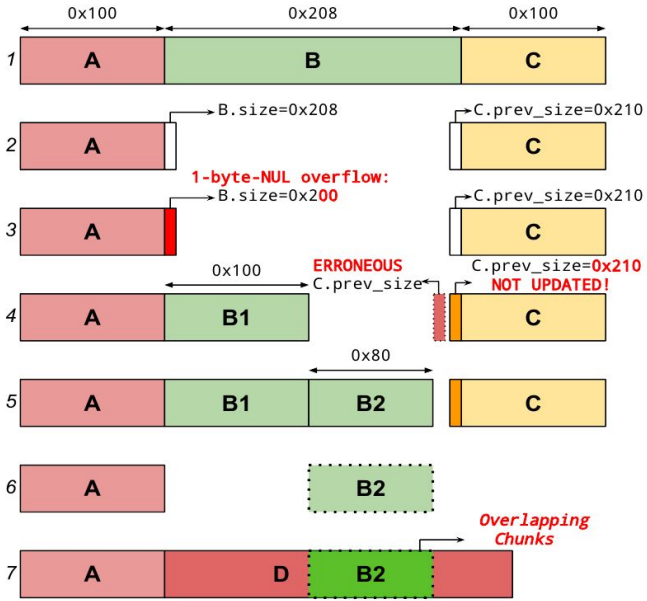
HeapHopper

Bringing Bounded Model Checking to Heap Implementation Security

Moritz Eckert* , Antonio Bianchi*†, Ruoyu Wang*°,
Yan Shoshitaishvili°, Christopher Kruegel*, and Giovanni Vigna*

*University of California, Santa Barbara
°Arizona State University
†The University of Iowa

Poison NULL Byte Attack



- Complex attacking-technique discovered by Chris Evans
- Only needs an overflow of a single NULL byte
- Leverages that to a full overlapping chunk
  - Attacker gains full control over chunk and metadata
- A patch was introduced by Chris Evans himself:

“Did we finally nail off-by-one NULL byte overwrites in the glibc heap? Only time will tell!”

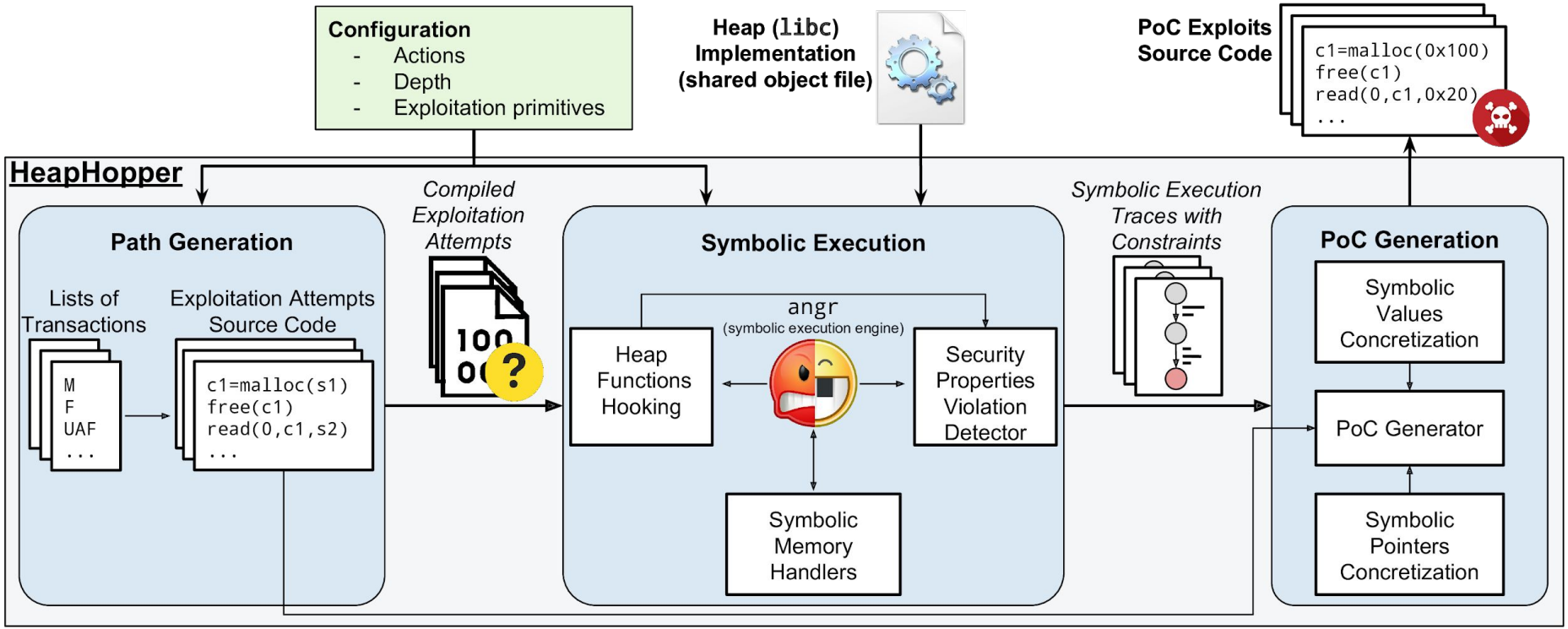
Poison NULL Byte Attack

- The answer is No.
- After the usual long proposal phase the patch was considered being “good” and finally merged
- Within days someone found a bypass

Motivation

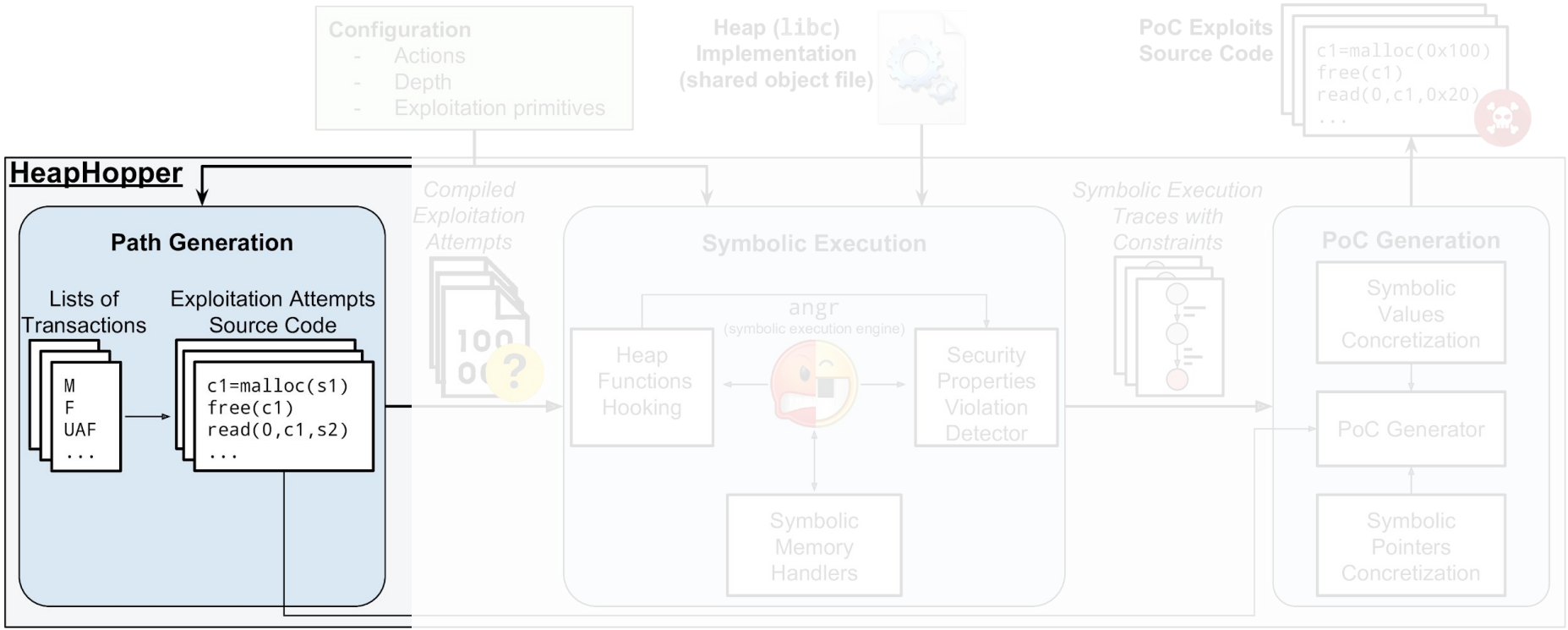
- Manually managing dynamic memory is *hard* → Bugs are *common*
- Metadata corruption is a *valuable target* for attackers
- Checks are introduced in a *nonsystematic* way

HeapHopper

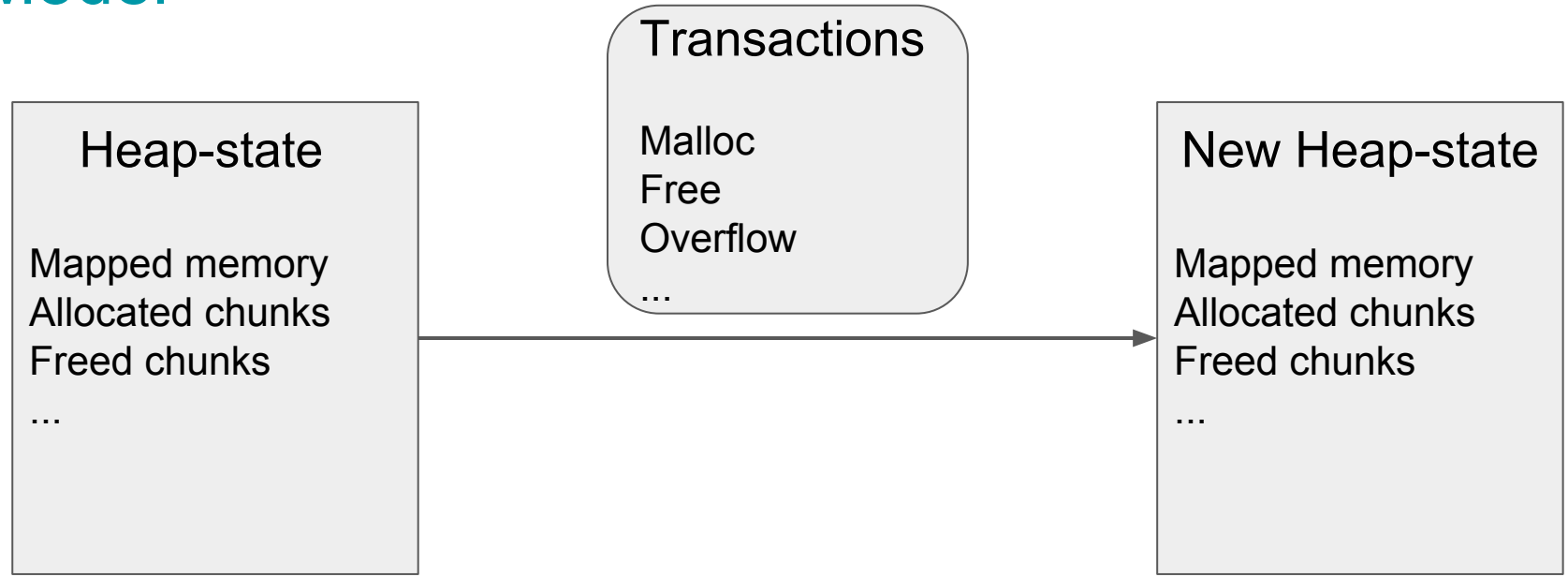

Configuration
- Actions
- Depth
- Exploitation primitives
Heap (libc) Implementation (shared object file)
PoC Exploits Source Code
c1=malloc(0x100)
free(c1)
read(0,c1,0x20)
...
HeapHopper
Path Generation
Lists of Transactions
M
F
UAF
...
Exploitation Attempts Source Code
c1=malloc(s1)
free(c1)
read(0,c1,s2)
...
Compiled Exploitation Attempts
Symbolic Execution
angr
(symbolic execution engine)
Heap Functions Hooking
Security Properties Violation Detector
Symbolic Memory Handlers
Symbolic Execution Traces with Constraints
PoC Generation
Symbolic Values Concretization
PoC Generator
Symbolic Pointers Concretization

Heap Interaction Models

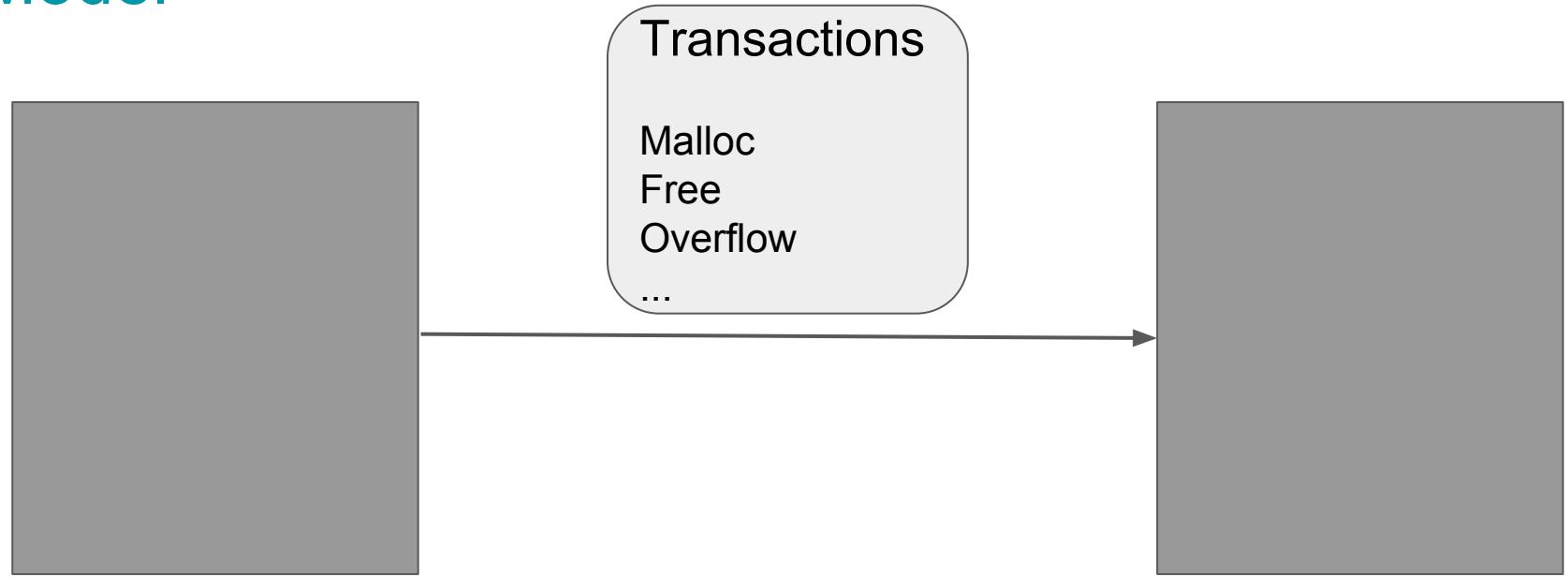
Heap Interaction Models



Model


Heap-state
Mapped memory
Allocated chunks
Freed chunks
...
Transactions
Malloc
Free
Overflow
...
New Heap-state
Mapped memory
Allocated chunks
Freed chunks
...

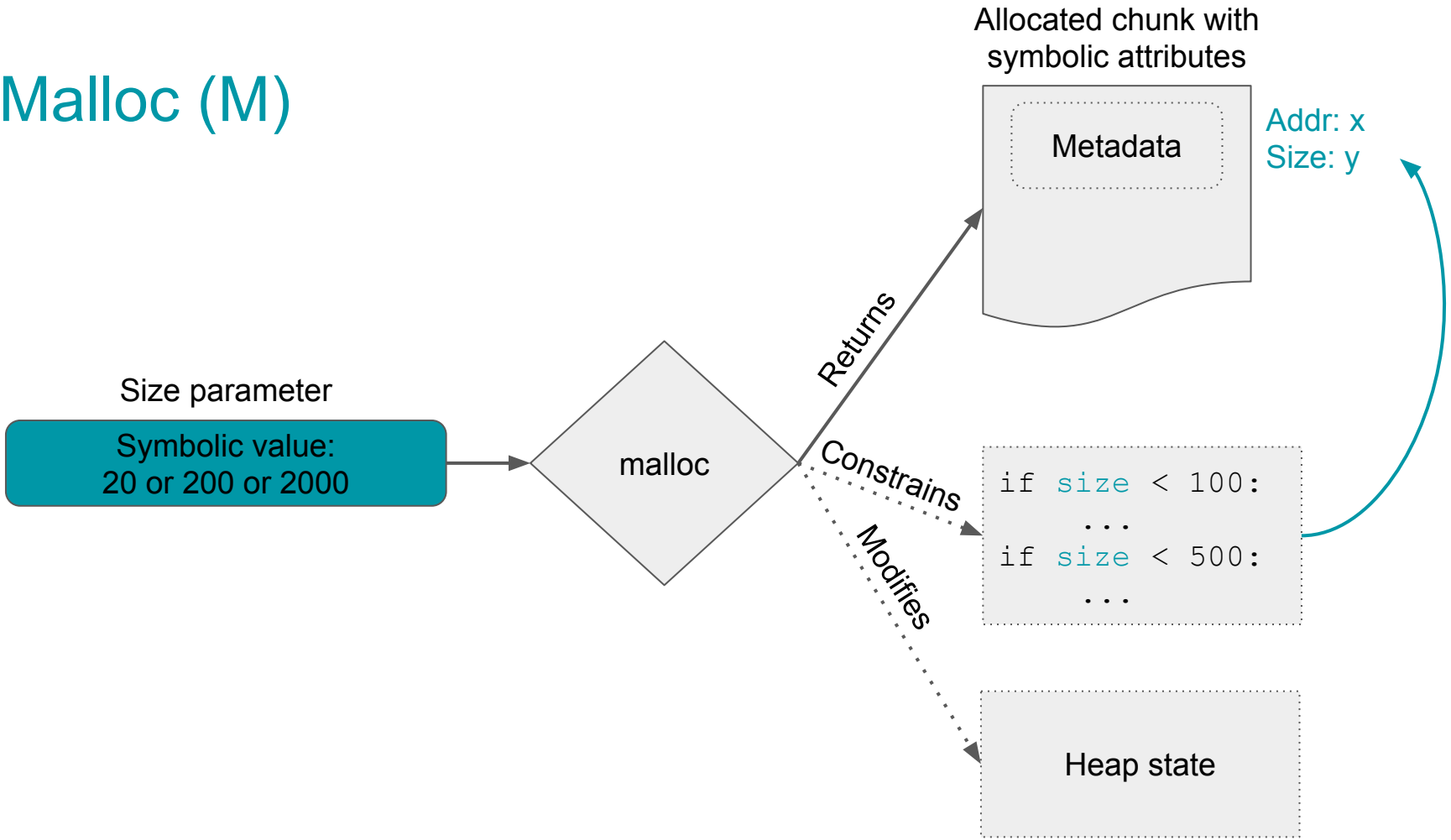
Model



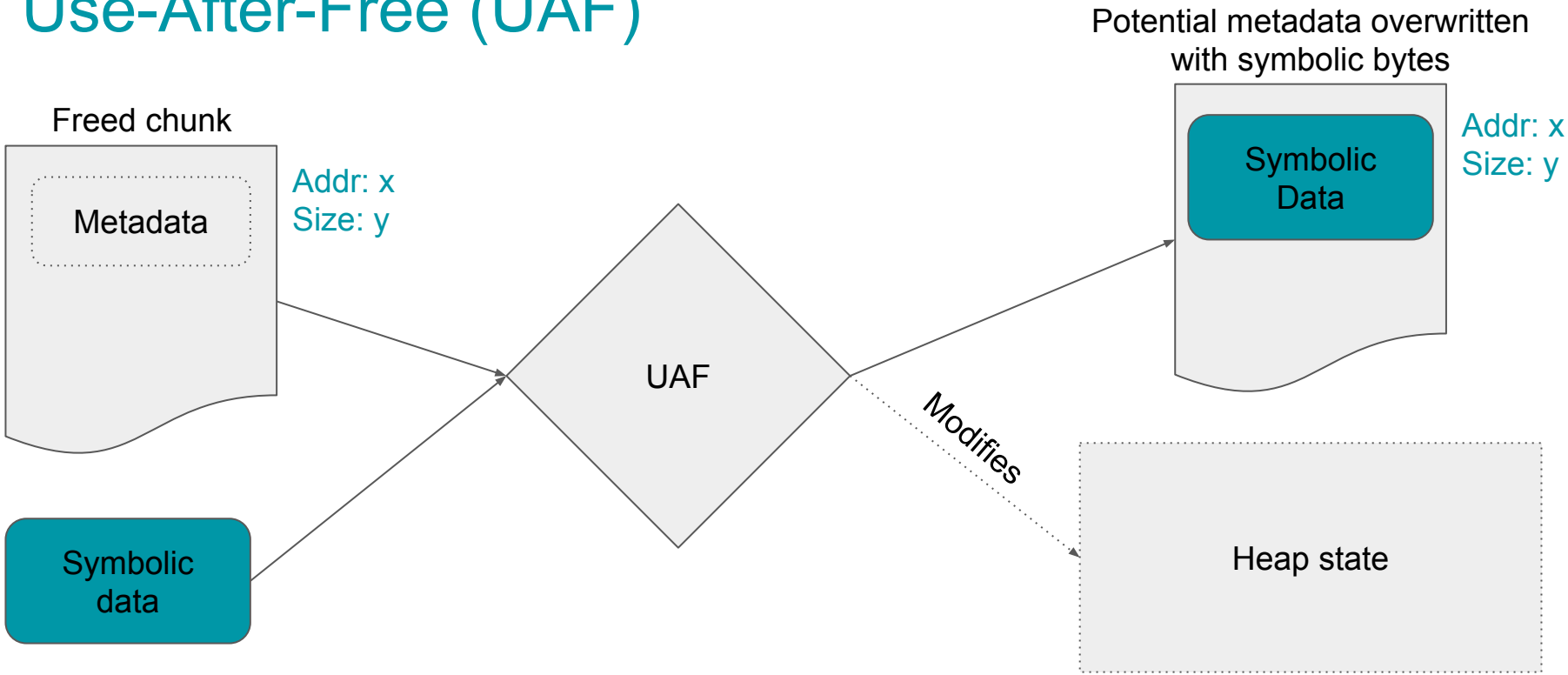
Transactions

- Currently supported transactions
  - Usages
    - **Malloc**
    - Free
  - Miss-Usages
    - Overflow
    - **Use-After-Free (UAF)**
    - Double Free
    - Fake Free

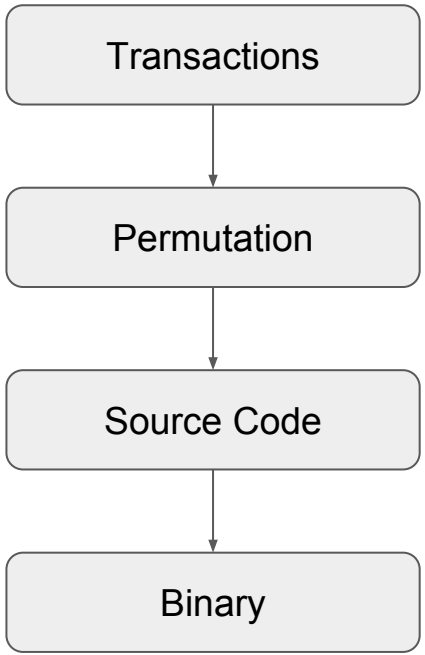
Malloc (M)


Allocated chunk with symbolic attributes
Metadata
Addr: x
Size: y
Size parameter
Symbolic value:
20 or 200 or 2000
malloc
Returns
Constrains
Modifies
if size < 100:
...
if size < 500:
...
Heap state

Use-After-Free (UAF)



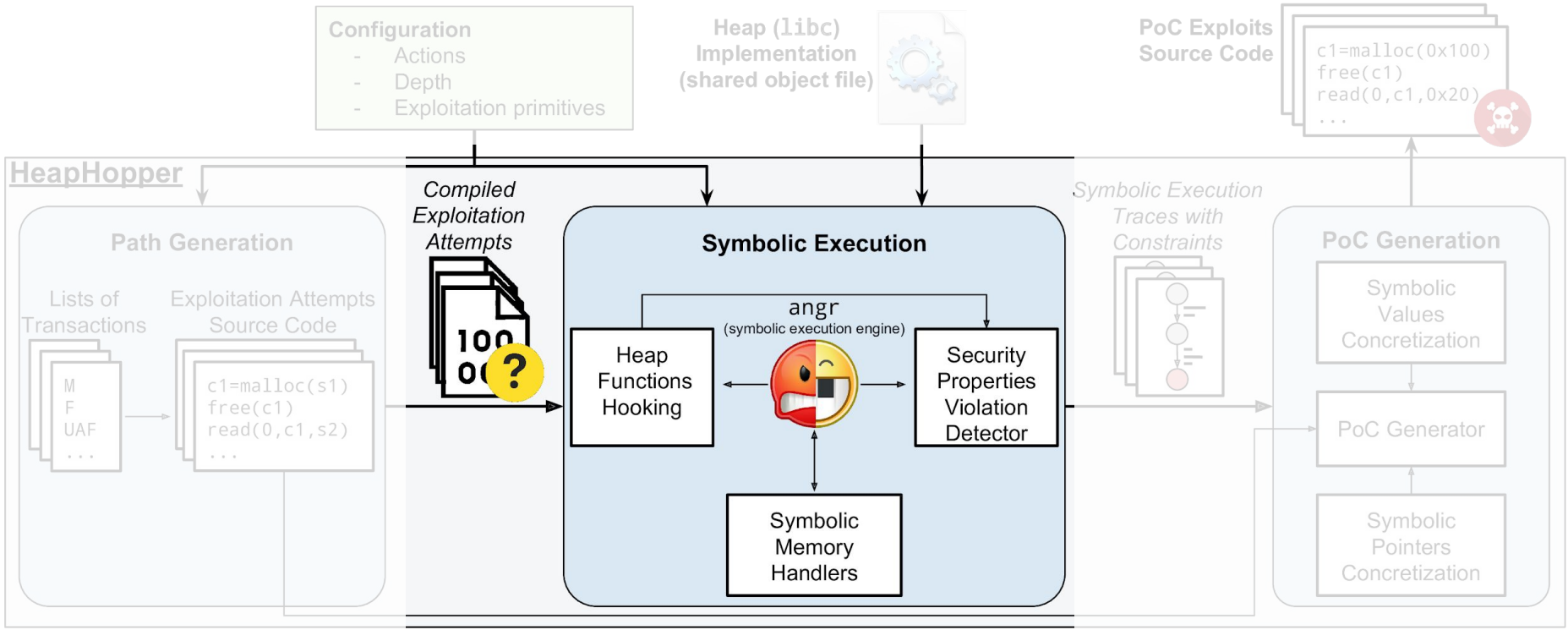
Interaction Models



- All permutations of Transactions *bounded* by a maximum depth
- Filtered with a set of rules
  - Consider semantics
  - Existence of at least one malicious transactions
- Transform to source code
  - Placeholders for the symbolic memory
- Compiled to binaries

Model Checking

Model Checking



Symbolic Execution

angr

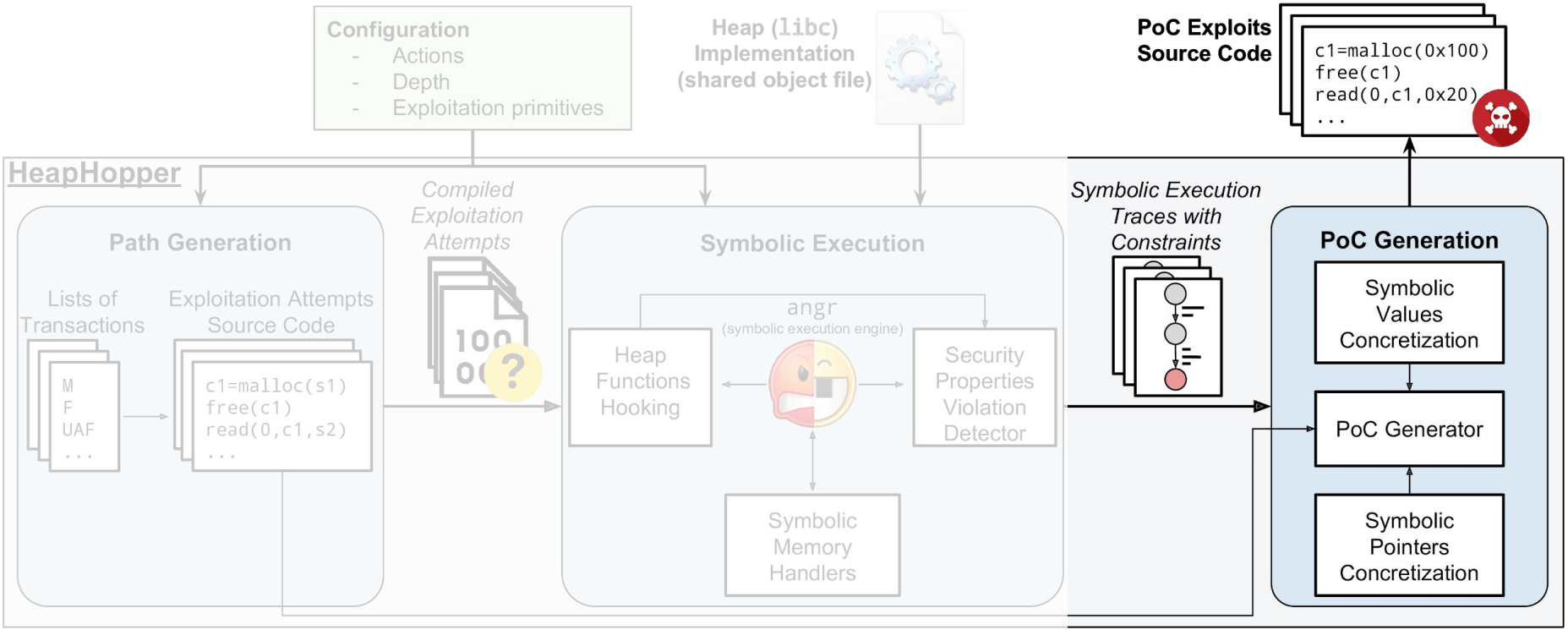
- Executing the library code
- Emulating system calls such as *mmap, brk*
- Using Depth First Search

Identifying Security Violations

- Checking for one of the following states
  - Overlapping Allocation (OA)
  - Non-Heap Allocation (NHA)
  - Arbitrary Write (AW) / Arbitrary Write Constraint (AWC)
    - Memory write issued in allocator code with a symbolic address as the destination
    - Representing a attacker controlled write

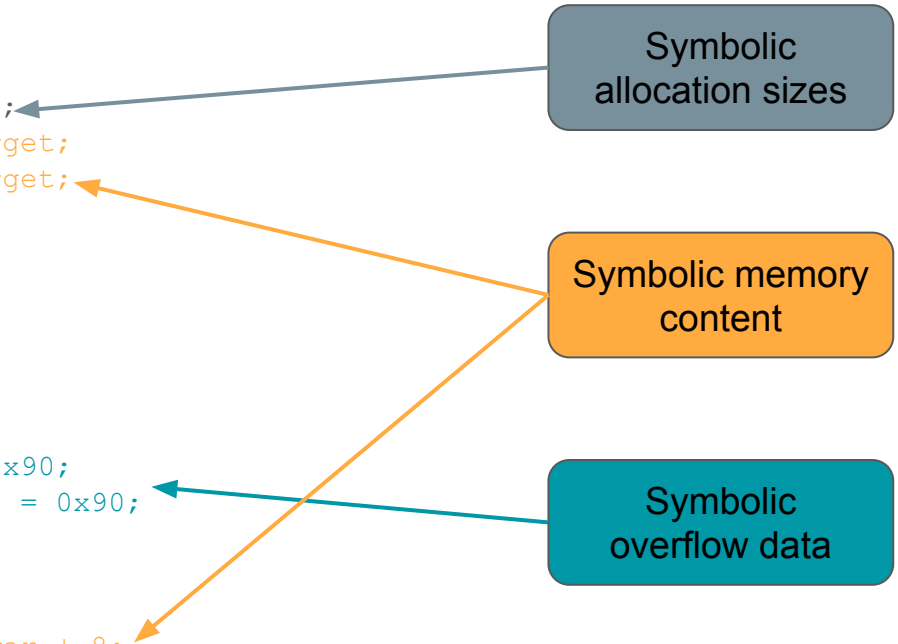
PoC Generation

PoC Generation

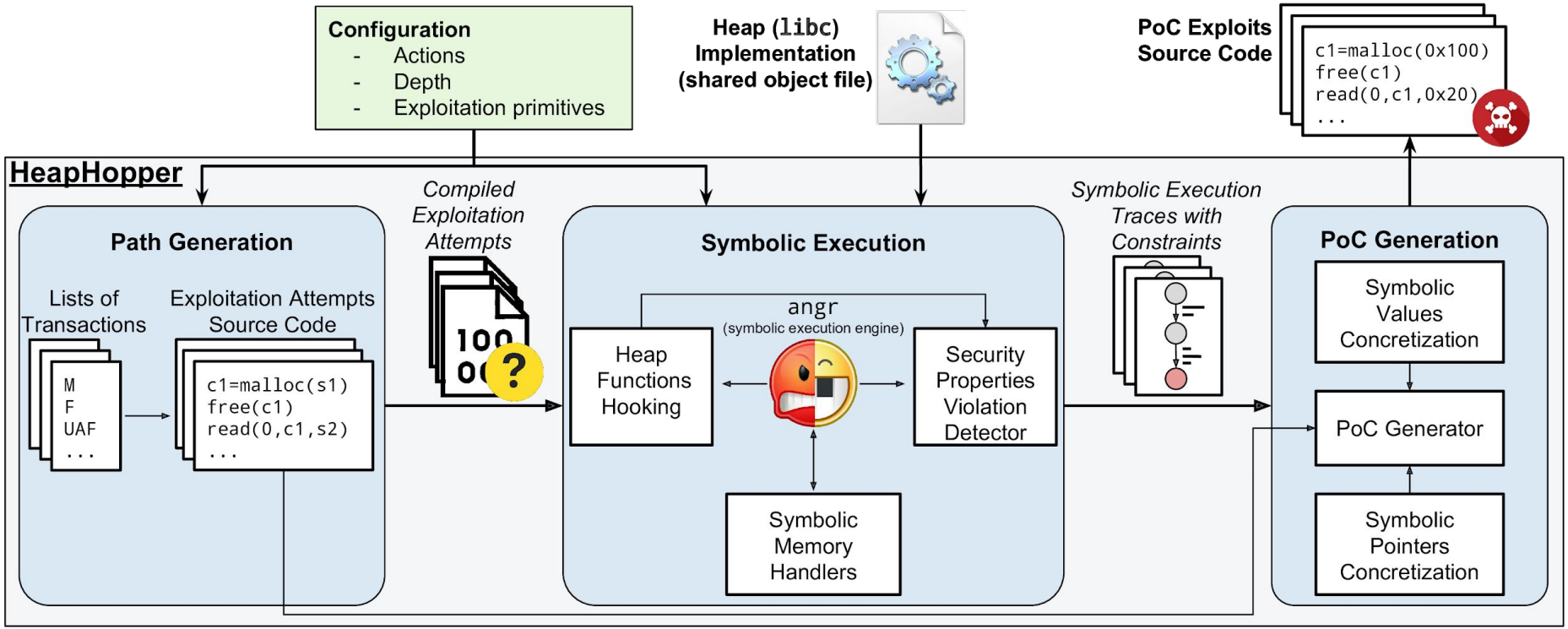


PoC Generation

```
int main(void) {
    // Allocation
    ctrl_data_0.global_var = malloc( 0x80);
    ctrl_data_0.global_var[0] = &write_target;
    ctrl_data_0.global_var[1] = &write_target;
    ctrl_data_0.global_var[2] = 0x0;
    ctrl_data_0.global_var[3] = 0x0;

      ...

    // VULN: Overflow
    offset = mem2chunk_offset;
    (ctrl_data_1.global_var-offset)[0] = 0x90;
    (ctrl_data_1.global_var-offset+0x8)[0] = 0x90;

    write_target[0] = 0x0;
    write_target[1] = 0x0;
    write_target[2] = ctrl_data_0.global_var + 8;
    write_target[3] = ctrl_data_0.global_var + 0;
    free(ctrl_data_1.global_var);
}
```



HeapHopper


Configuration
- Actions
- Depth
- Exploitation primitives
Heap (libc) Implementation (shared object file)
PoC Exploits Source Code
c1=malloc(0x100)
free(c1)
read(0,c1,0x20)
...
HeapHopper
Path Generation
Lists of Transactions
M
F
UAF
...
Exploitation Attempts Source Code
c1=malloc(s1)
free(c1)
read(0,c1,s2)
...
Compiled Exploitation Attempts
Symbolic Execution
angr
(symbolic execution engine)
Heap Functions Hooking
Security Properties Violation Detector
Symbolic Memory Handlers
Symbolic Execution Traces with Constraints
PoC Generation
Symbolic Values Concretization
PoC Generator
Symbolic Pointers Concretization

Limitations

- Bounded by depth when creating permutations
- Bounded by memory
- Bounded by time

Evaluation

Allocator Comparison

Allocator	OA	NHA	AWC	AW
dlmalloc 2.7.2	(M,F,O): *M-M-M-F-O-M* (M,F,UAF): *M-M-M-F-UAF-M-M*	(M,FF): *FF-M* (M,F,O): *M-M-O-F-M* (M,F,UAF): *M-M-F-UAF-M-M*		(M,F,FF): *M-FF-F* (M,F,O): *M-M-O-F* (M,F,UAF): *M-M-F-UAF-M*
dlmalloc 2.8.6	(M,F,O): *M-M-M-F-O-M* (M,F,UAF): *M-M-M-F-UAF-M-M*		(M,F,O): *M-M-M-F-O-O-F*	
musl 1.1.9	(M,F,O): *M-M-M-F-O-M* (M,F,UAF): *M-M-M-F-UAF-M-M*	(M,FF): *FF-M* (M,F,UAF): *M-M-F-UAF-M-M*	(M,F,FF): *M-FF-F*	(M,F,UAF): *M-M-F-UAF-M* (M,F,FF): *M-M-F-FF-M-M*
ptmalloc 2.23	(M,F,O): *M-M-M-F-O-M* (M,F,UAF): *M-M-M-F-UAF-M-M*	(M,FF): *FF-M* (M,F,O): *M-M-M-O-F-M* (M,F,UAF): *M-M-F-UAF-M-M*	(M-F-FF): *M-FF-F* (M,F,O): *M-M-O-F*	(M,F,UAF): *M-M-F-UAF-M*
ptmalloc 2.26	(M,F,O): *M–M-O-F-M* (M,F,UAF): *M-M-M-F-UAF-M-M*	(M,FF): *FF-M* (M,F,UAF): *M-M-F-UAF-M-M*		(M,F,UAF): *M-M-F-UAF-M* (M-F-FF): *M-FF-F*

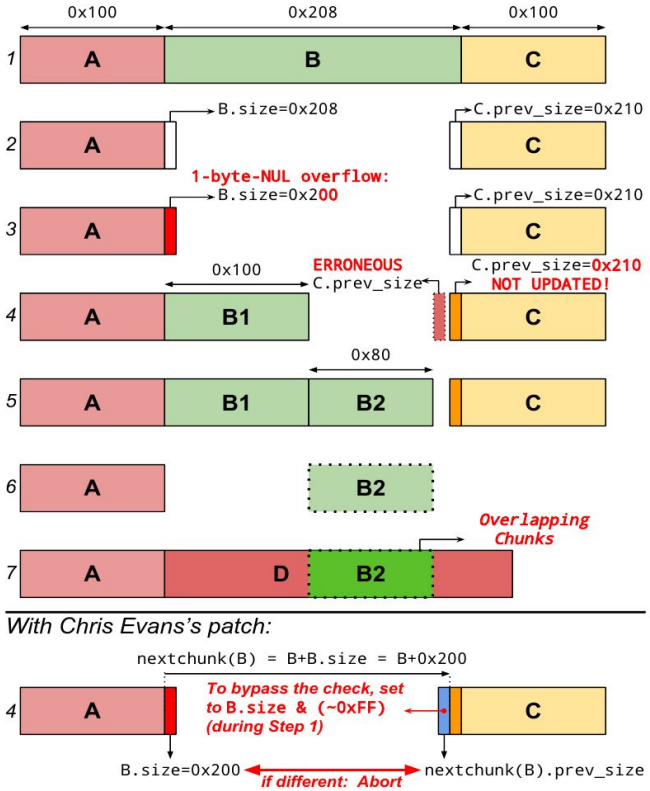
Overflow (O), Free (F), Use-After-Free (UAF), Double Free (DF), Fake Free (FF)

Allocator Comparison

Allocator	OA	NHA	AWC	AW
dlmalloc 2.7.2	(M,F,O): *M-M-M-F-O-M* (M,F,UAF): *M-M-M-F-UAF-M-M*	(M,FF): *FF-M* (M,F,O): *M-M-O-F-M* (M,F,UAF): *M-M-F-UAF-M-M*		(M,F,FF): *M-FF-F* (M,F,O): *M-M-O-F* (M,F,UAF): *M-M-F-UAF-M*
dlmalloc 2.8.6	(M,F,O): *M-M-M-F-O-M* (M,F,UAF): *M-M-M-F-UAF-M-M*		(M,F,O): *M-M-M-F-O-O-F*	
musl 1.1.9	(M,F,O): *M-M-M-F-O-M* (M,F,UAF): *M-M-M-F-UAF-M-M*	(M,FF): *FF-M* (M,F,UAF): *M-M-F-UAF-M-M*	(M,F,FF): *M-FF-F*	(M,F,UAF): *M-M-F-UAF-M* (M,F,FF): *M-M-F-FF-M-M*
ptmalloc 2.23	(M,F,O): *M-M-M-F-O-M* (M,F,UAF): *M-M-M-F-UAF-M-M*	(M,FF): *FF-M* (M,F,O): *M-M-M-O-F-M* (M,F,UAF): *M-M-F-UAF-M-M*	**(M-F-FF): *M-FF-F*** (M,F,O): *M-M-O-F*	(M,F,UAF): *M-M-F-UAF-M*
ptmalloc 2.26	(M,F,O): *M–M-O-F-M* (M,F,UAF): *M-M-M-F-UAF-M-M*	(M,FF): *FF-M* (M,F,UAF): *M-M-F-UAF-M-M*		(M,F,UAF): *M-M-F-UAF-M* **(M-F-FF): *M-FF-F***

Overflow (O), Free (F), Use-After-Free (UAF), Double Free (DF), Fake Free (FF)

Poison NULL Byte Attack



- Challenging because of high depth
- Verified that HeapHopper finds attack
- Verified that HeapHopper finds patch bypass
- Developed a new patch and verified that HeapHopper does not find a bypass
- We are trying to upstream this patch

Questions?

https://github.com/angr/heaphopper