Inroduction of Kint

Department of Computer Science & Technology
Tsinghua University

Main Content

- **What is Kint ?**
- The features of Kint
- The design of Kint
- The evaluation of Kint
- NaN integer semantics

What is Kint ?

- Scalable
- Static analysis
- Detect integer errors in C program

Main Content

- What is Kint ?
- **The features of Kint**
- The design of Kint
- The evaluation of Kint
- NaN integer semantics

The features of Kint

- coverage：
  - statically generates a constraint capturing the path condition leading to an integer error
- false positives：
  - int fun(int a, int b){
  - if(a<0 || b<0 || **a+b<a**)
  - return -1;
  - return a+b;
  - }

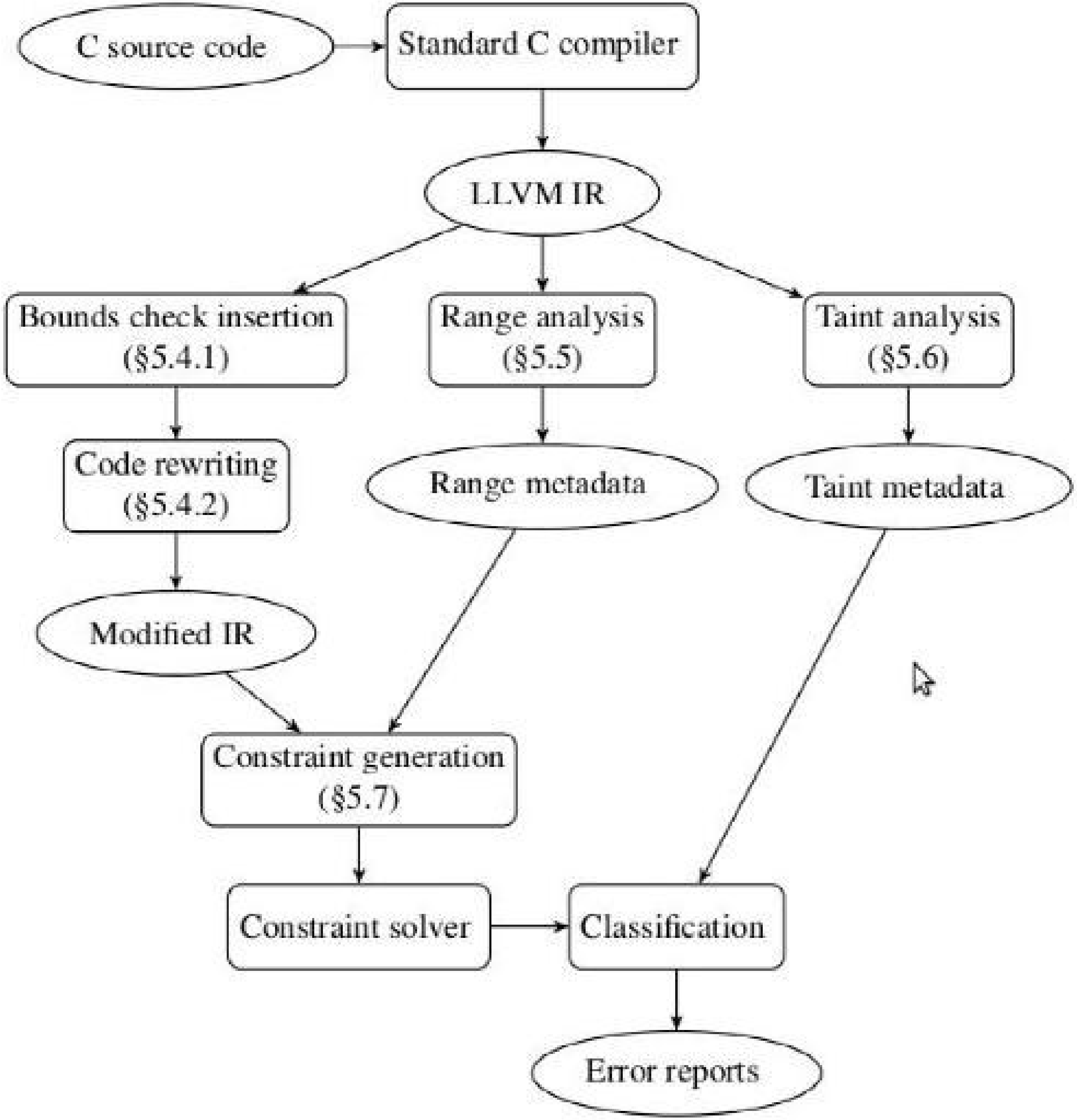
The features of Kint

- Five contributions to help developers find and deal with integer errors:
- 1. the pragmatic definition of integer errors
- 2. whole-program analysis
- 3. range annotation
- 4. a family of overflow-checked integer for C
- 5. a less error-prone API for memory allocation in the Linux kernel

Main Content

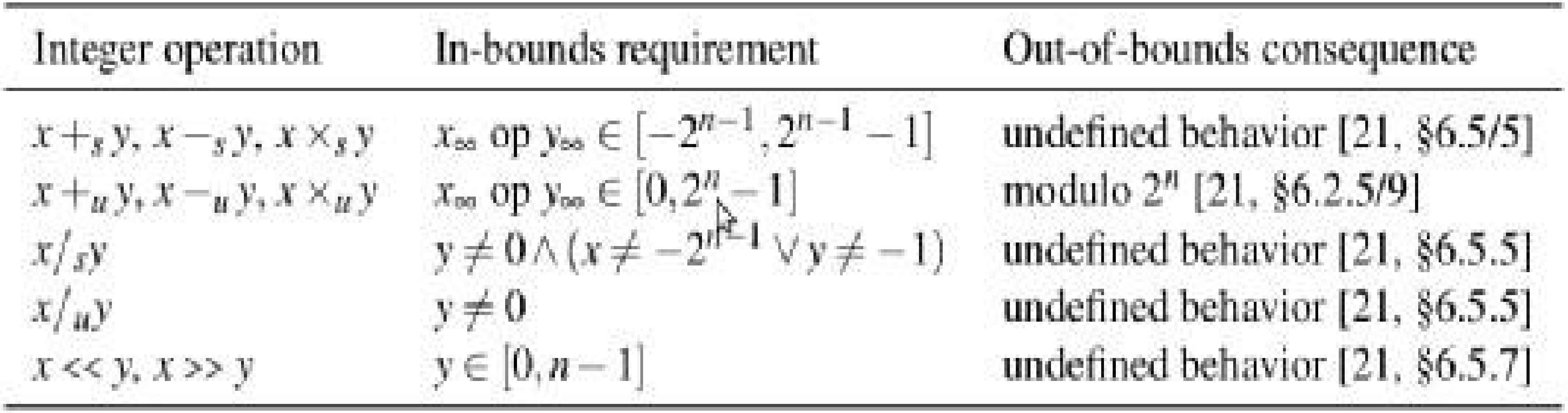
- What is Kint ?
- The features of Kint
- **The design of Kint**
- The evaluation of Kint
- NaN integer semantics

The design of Kint



The design of Kint

- In-bounds requirement of integer operations



Integer operation	In-bounds requirement	Out-of-bounds consequence
$x +_s y$, $x -_s y$, $x \times_s y$	x_∞ op $y_\infty \in [-2^{n-1}, 2^{n-1} - 1]$	undefined behavior [21, §6.5/5]
$x +_u y$, $x -_u y$, $x \times_u y$	x_∞ op $y_\infty \in [0, 2^n - 1]$	modulo 2^n [21, §6.2.5/9]
$x /_s y$	$y \neq 0 \wedge (x \neq -2^{n-1} \vee y \neq -1)$	undefined behavior [21, §6.5.5]
$x /_u y$	$y \neq 0$	undefined behavior [21, §6.5.5]
$x << y$, $x >> y$	$y \in [0, n-1]$	undefined behavior [21, §6.5.7]

The design of Kint

- Function-level analysis :

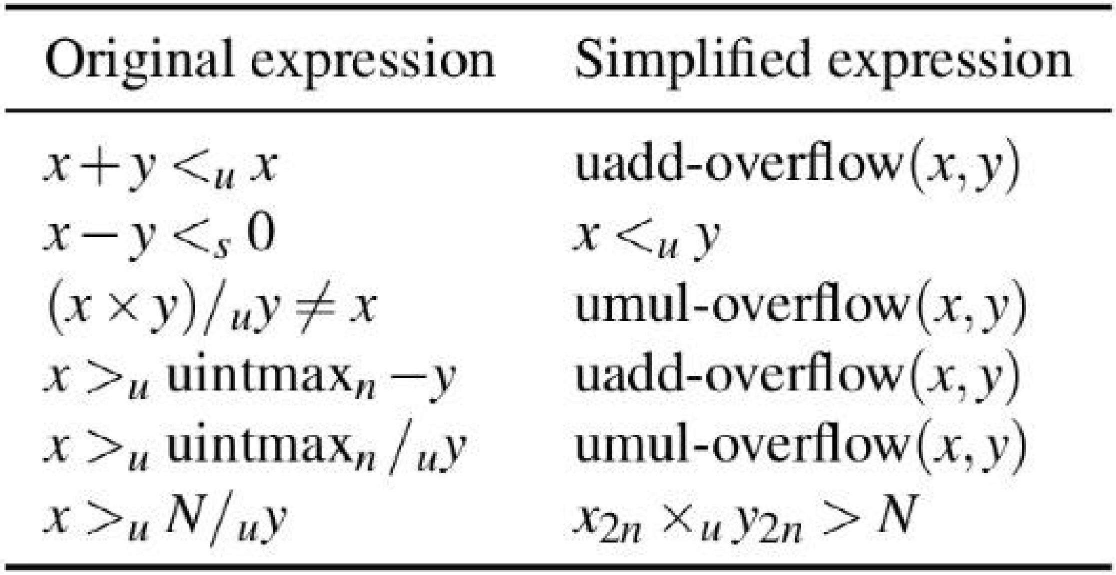
1. Bounds check insertion

- On the level of IR, Kint marks the potential integer errors through LLVM intrinsic functions

The design of Kint - Function-level analysis

2. Code rewriting：

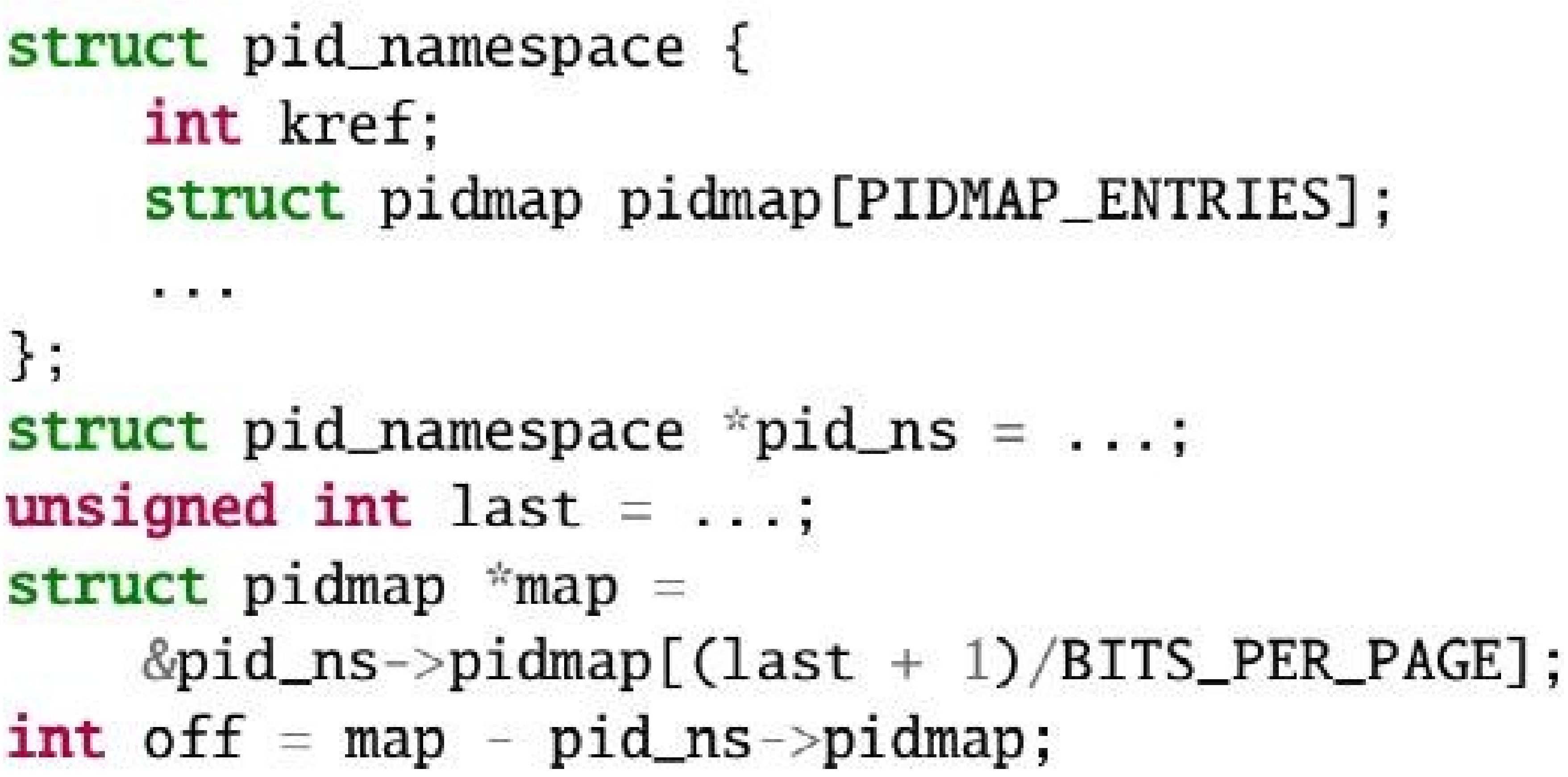
（1）Simplifying common idioms



Original expression	Simplified expression
$x+y <_u x$	$\text{uadd-overflow}(x, y)$
$x-y <_s 0$	$x <_u y$
$(x \times y)/_u y \neq x$	$\text{umul-overflow}(x, y)$
$x >_u \text{uintmax}_n - y$	$\text{uadd-overflow}(x, y)$
$x >_u \text{uintmax}_n /_u y$	$\text{umul-overflow}(x, y)$
$x >_u N/_u y$	$x_{2n} \times_u y_{2n} > N$

The design of Kint-Funtion level analysis

（2）Simplifying pointer arithmetic



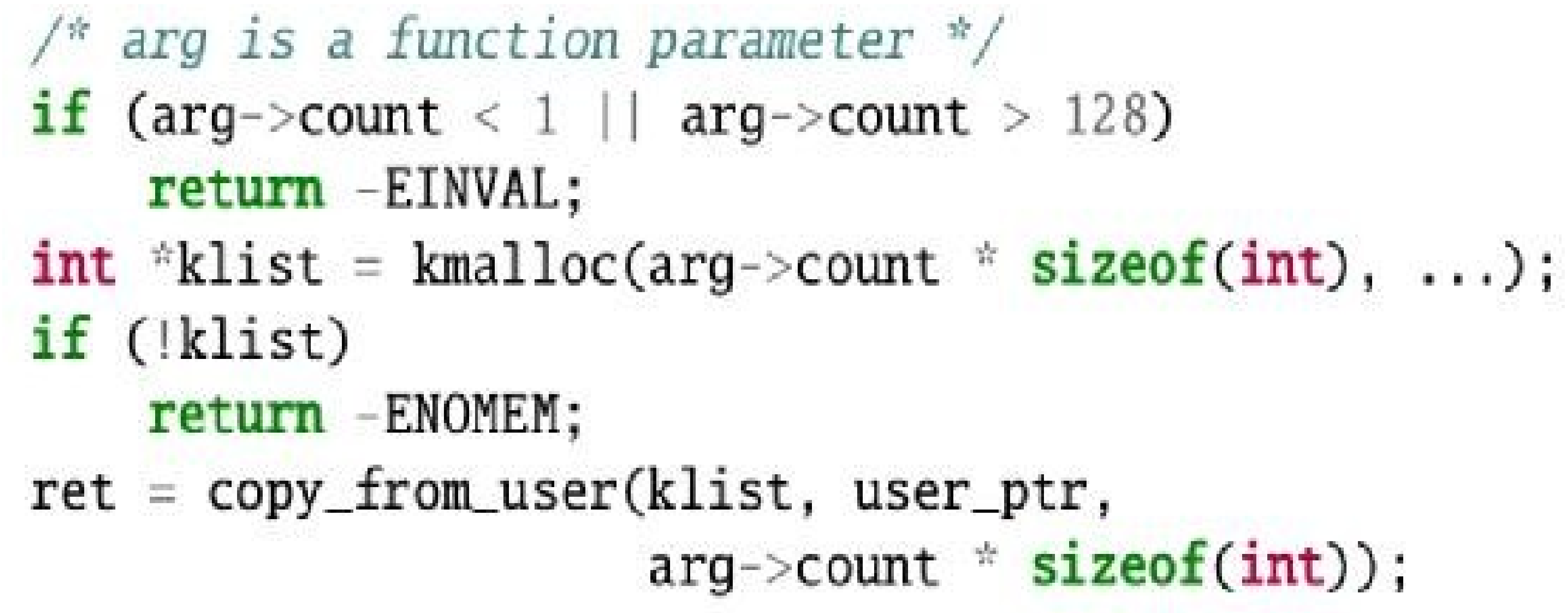
```
struct pid_namespace {
    int kref;
    struct pidmap pidmap[PIDMAP_ENTRIES];
    ...
};
struct pid_namespace *pid_ns = ...;
unsigned int last = ...;
struct pidmap *map =
    &pid_ns->pidmap[(last + 1)/BITS_PER_PAGE];
int off = map - pid_ns->pidmap;
```

The design of Kint-Function level analysis

- i=(last+1)/BITS_PER_PAGE
- map=pid_ns+4+i×8
- pid_ns->pidmap=pid_ns+4
- off=(pid_ns+4+i×8)-(pid_ns+4)=i×8

The design of Kint-Function level analysis

- （3）Merging memory loads



```
/* arg is a function parameter */
if (arg->count < 1 || arg->count > 128)
    return -EINVAL;
int *klist = kmalloc(arg->count * sizeof(int), ...);
if (!klist)
    return -ENOMEM;
ret = copy_from_user(klist, user_ptr,
                     arg->count * sizeof(int));
```

The design of Kint-Function level analysis

- （4）Eliminating checks using compiler optimizations
- if optimizer infer that the potential integer error can't be satisfied, it will remove the corresponding LLVM intrinsic function

The design of Kint

- Range analysis
- one of the limitations of Function level analysis is that it can't capture the invariants that hold across functions
- Kint stores a range in the global range table for every cross-function entity

The design of Kint-Range analysis

- Strict-aliasing rules：one memory location can't be used by two different type
- for example：
- int a=9; double b=(double)a;

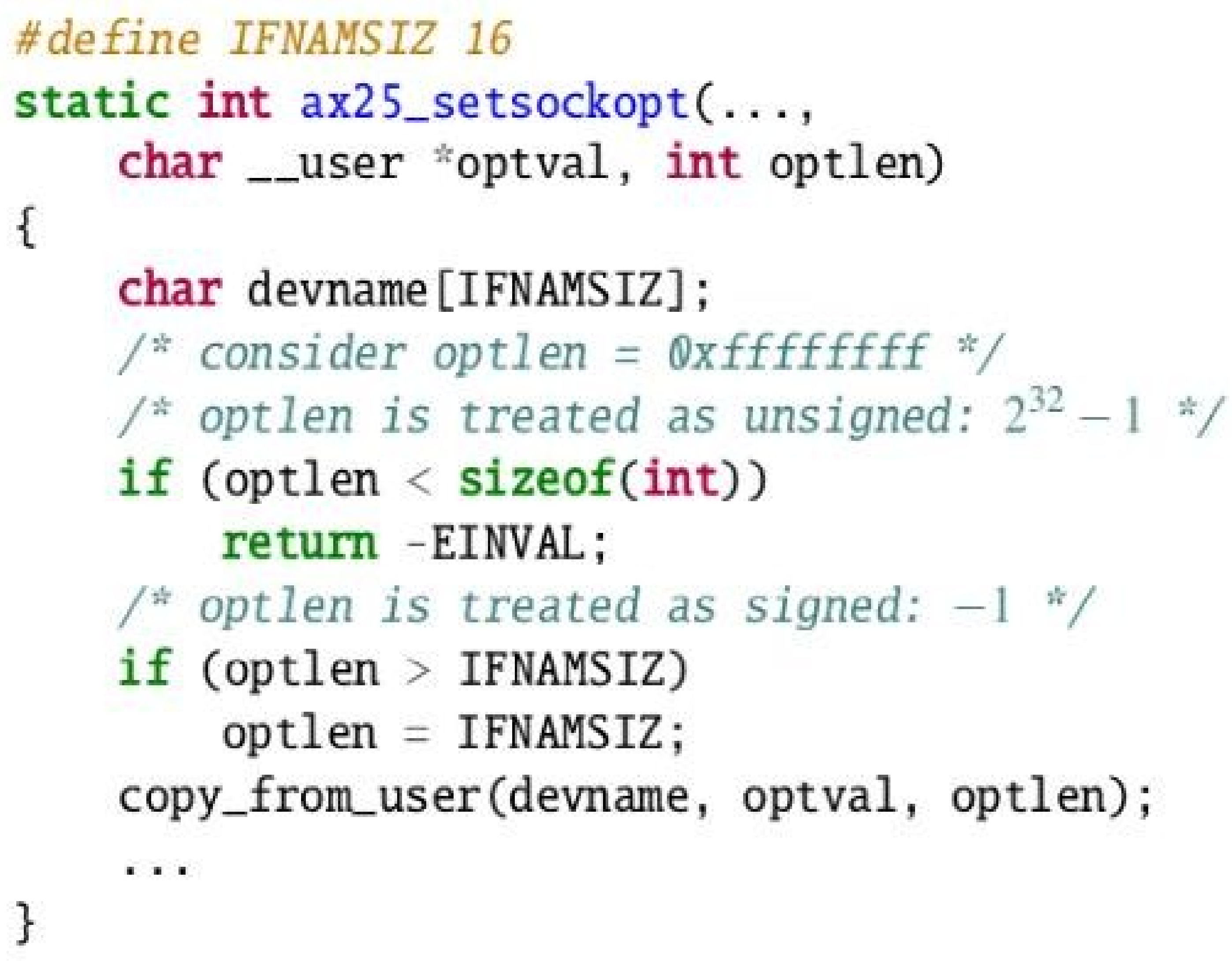
The design of Kint

- Taint analysis
- classify error reports through an untrusted input(source) or sensitive context(sink)

The design of Kint

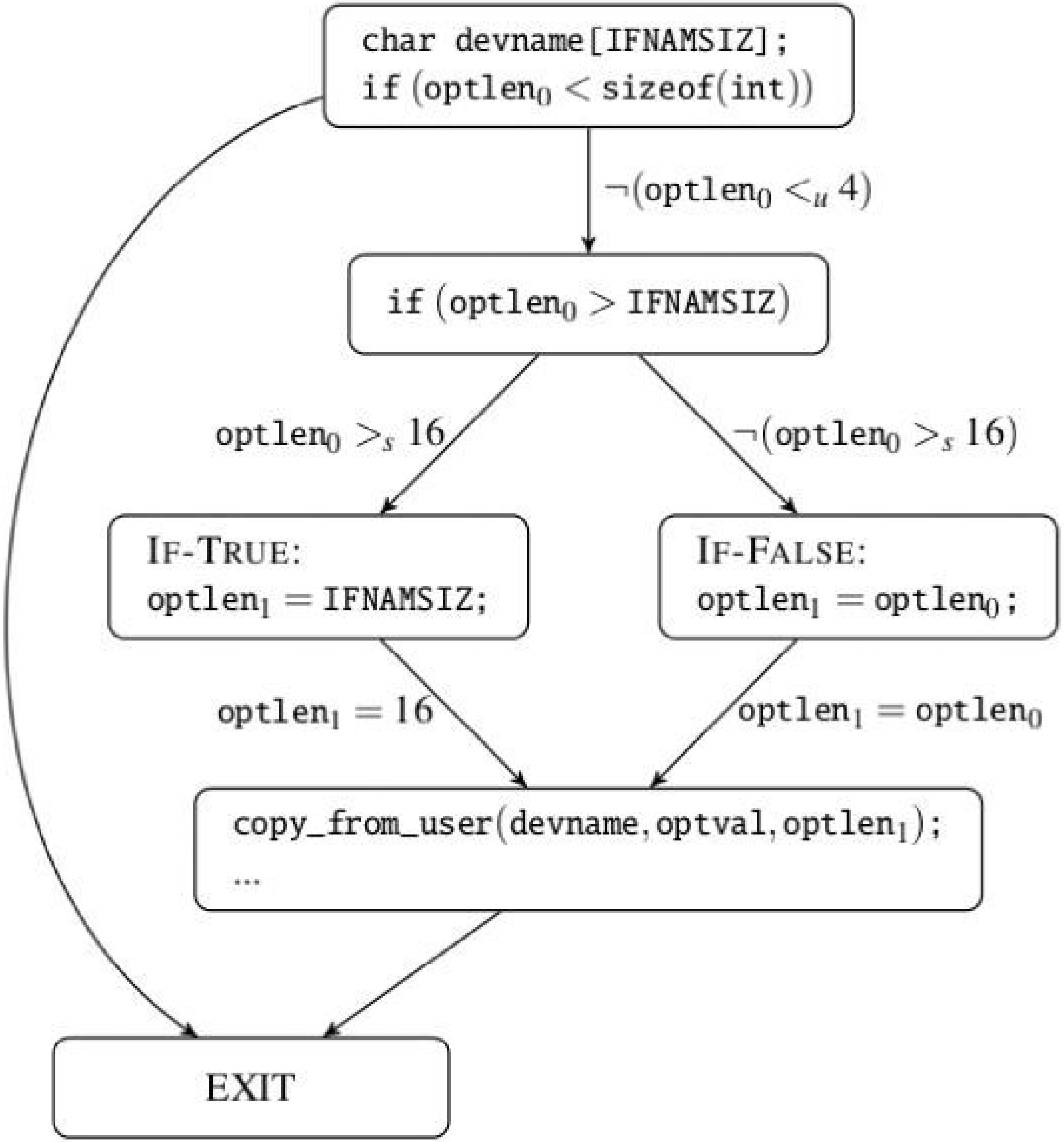
- Constraint generation
- the source of constraint：
  - assignments to variables by preceding operations
  - conditional branches along the execution path

The design of Kint- Contraint generation



```
#define IFNAMSIZ 16
static int ax25_setsockopt(...,
    char __user *optval, int optlen)
{
    char devname[IFNAMSIZ];
    /* consider optlen = 0xffffffff */
    /* optlen is treated as unsigned: 2^32 - 1 */
    if (optlen < sizeof(int))
        return -EINVAL;
    /* optlen is treated as signed: -1 */
    if (optlen > IFNAMSIZ)
        optlen = IFNAMSIZ;
    copy_from_user(devname, optval, optlen);
    ...
}
```

The design of Kint - Constraint generation



char devname[IFNAMSIZ];
if $(\texttt{optlen}_0 < \texttt{sizeof(int)})$

$\neg(\texttt{optlen}_0 <_u 4)$

if $(\texttt{optlen}_0 > \texttt{IFNAMSIZ})$

$\texttt{optlen}_0 >_s 16$

$\neg(\texttt{optlen}_0 >_s 16)$

IF-TRUE:
$\texttt{optlen}_1 = \texttt{IFNAMSIZ};$

IF-FALSE:
$\texttt{optlen}_1 = \texttt{optlen}_0;$

$\texttt{optlen}_1 = 16$

$\texttt{optlen}_1 = \texttt{optlen}_0$

copy_from_user(devname, optval, $\texttt{optlen}_1$);
...

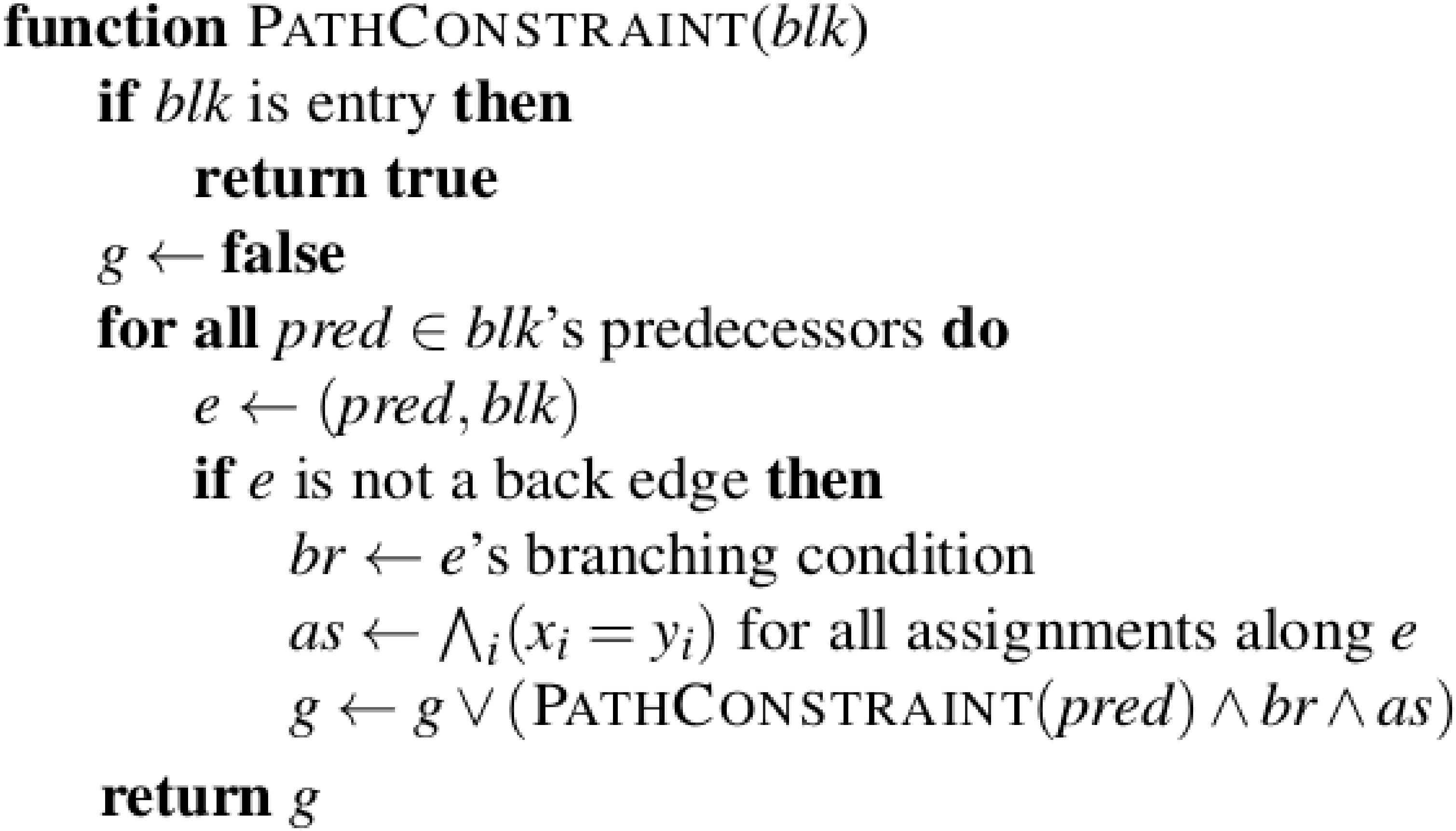
EXIT

The design of Kint-Constraint generation

- **((optlen1 = 16) ∧ PathConstraint(I F -TRUE))∨((optlen1 = optlen0) ∧ PathConstraint(I F -FALSE))**
- **((optlen1 = 16) ∧ (optlen0 >s 16) ∧ ¬(optlen0 <u 4))∨((optlen1 = optlen0) ∧ ¬(optlen0 >s 16)∧ ¬(optlen0 <u 4))**

The design of Kint-Constraint generation

- For programs that contain loops, the path constraint generation algorithm unrolls each loop once and ignores branching edges that jump back in the control flow



```
function PATHCONSTRAINT(blk)
    if blk is entry then
        return true
    g ← false
    for all pred ∈ blk's predecessors do
        e ← (pred, blk)
        if e is not a back edge then
            br ← e's branching condition
            as ← $\bigwedge_i (x_i = y_i)$ for all assignments along e
            g ← g ∨ (PATHCONSTRAINT(pred) ∧ br ∧ as)
    return g
```

The design of Kint-Constraint generation

- To alleviate missing constraints due to loop unrolling, Kint moves constraints inside a loop to the outer scope if possible.
- for(i = 0; i < n; ++i)
- a[i] = ...;
- constraint in the loop：i <s 0
- constraint out of the loop：n <s 0

The design of Kint-Limitations

- Kint only support C
- Kint will miss conversion errors that are not caught by existing invari-ants
- Kint analyzes loops by unrolling them once
- if the solver times out, Kint may miss errors corresponding to the queried constraints

Main Content

- What is Kint ?
- The features of Kint
- The design of Kint
- **The evaluation of Kint**
- NaN integer semantics

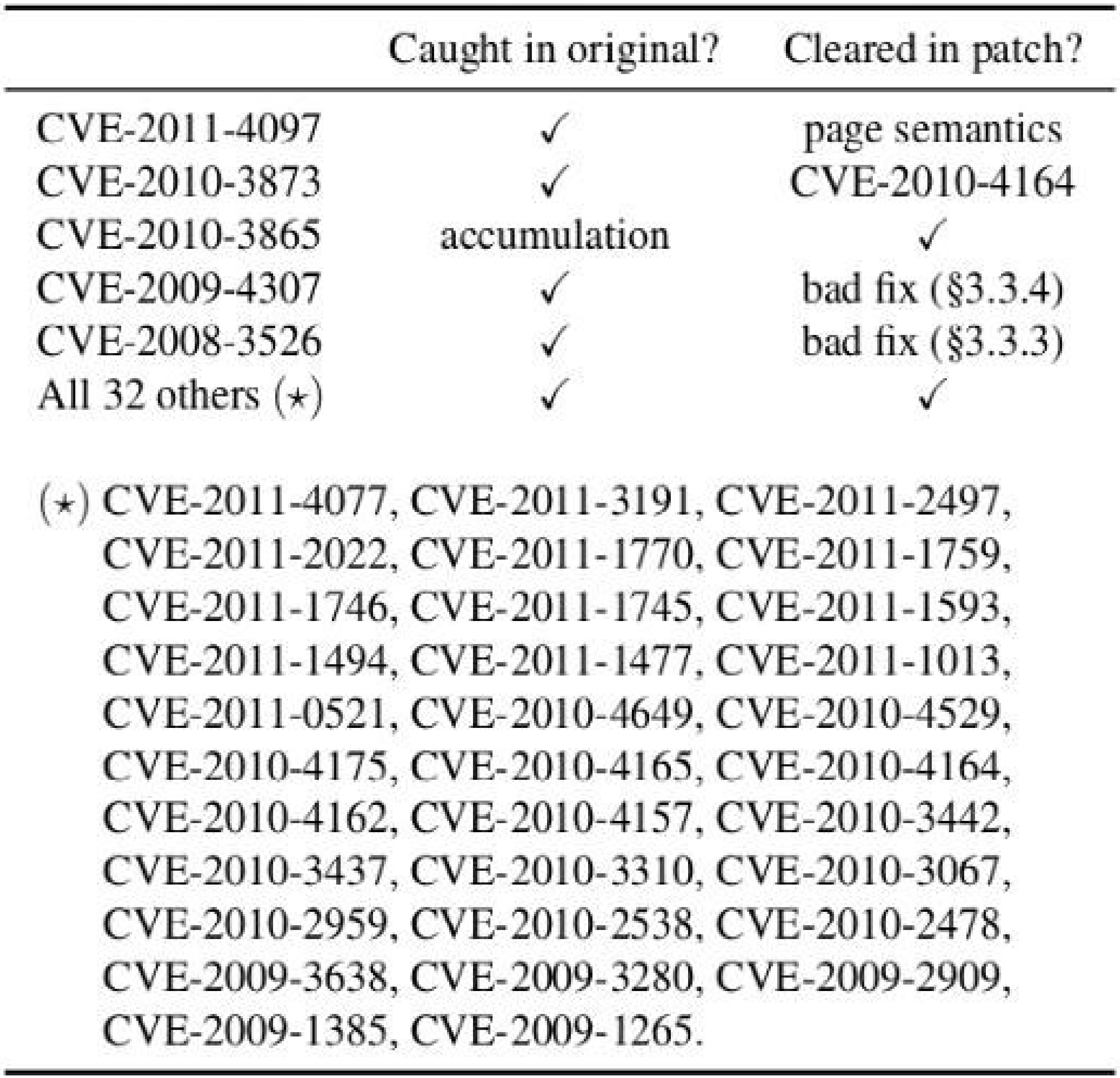
Evaluation of Kint

- find new bugs
- Completeness
- False errors
- Performance

Evaluation of Kint-find new bugs

- 2011.11-2012.4
- Linux kernel: 105
- Lighttpd: 1
- OpenSSH: 5

Evaluation of Kint-Completeness



	Caught in original?	Cleared in patch?
CVE-2011-4097	✓	page semantics
CVE-2010-3873	✓	CVE-2010-4164
CVE-2010-3865	accumulation	✓
CVE-2009-4307	✓	bad fix (§3.3.4)
CVE-2008-3526	✓	bad fix (§3.3.3)
All 32 others (⋆)	✓	✓

(⋆) CVE-2011-4077, CVE-2011-3191, CVE-2011-2497, CVE-2011-2022, CVE-2011-1770, CVE-2011-1759, CVE-2011-1746, CVE-2011-1745, CVE-2011-1593, CVE-2011-1494, CVE-2011-1477, CVE-2011-1013, CVE-2011-0521, CVE-2010-4649, CVE-2010-4529, CVE-2010-4175, CVE-2010-4165, CVE-2010-4164, CVE-2010-4162, CVE-2010-4157, CVE-2010-3442, CVE-2010-3437, CVE-2010-3310, CVE-2010-3067, CVE-2010-2959, CVE-2010-2538, CVE-2010-2478, CVE-2009-3638, CVE-2009-3280, CVE-2009-2909, CVE-2009-1385, CVE-2009-1265.

Evaluation of Kint-False positives

- Three experiments：
- CVE experiment
- Whole-kernel report analysis
- Single module analysis

Evaluation of Kint-Performance

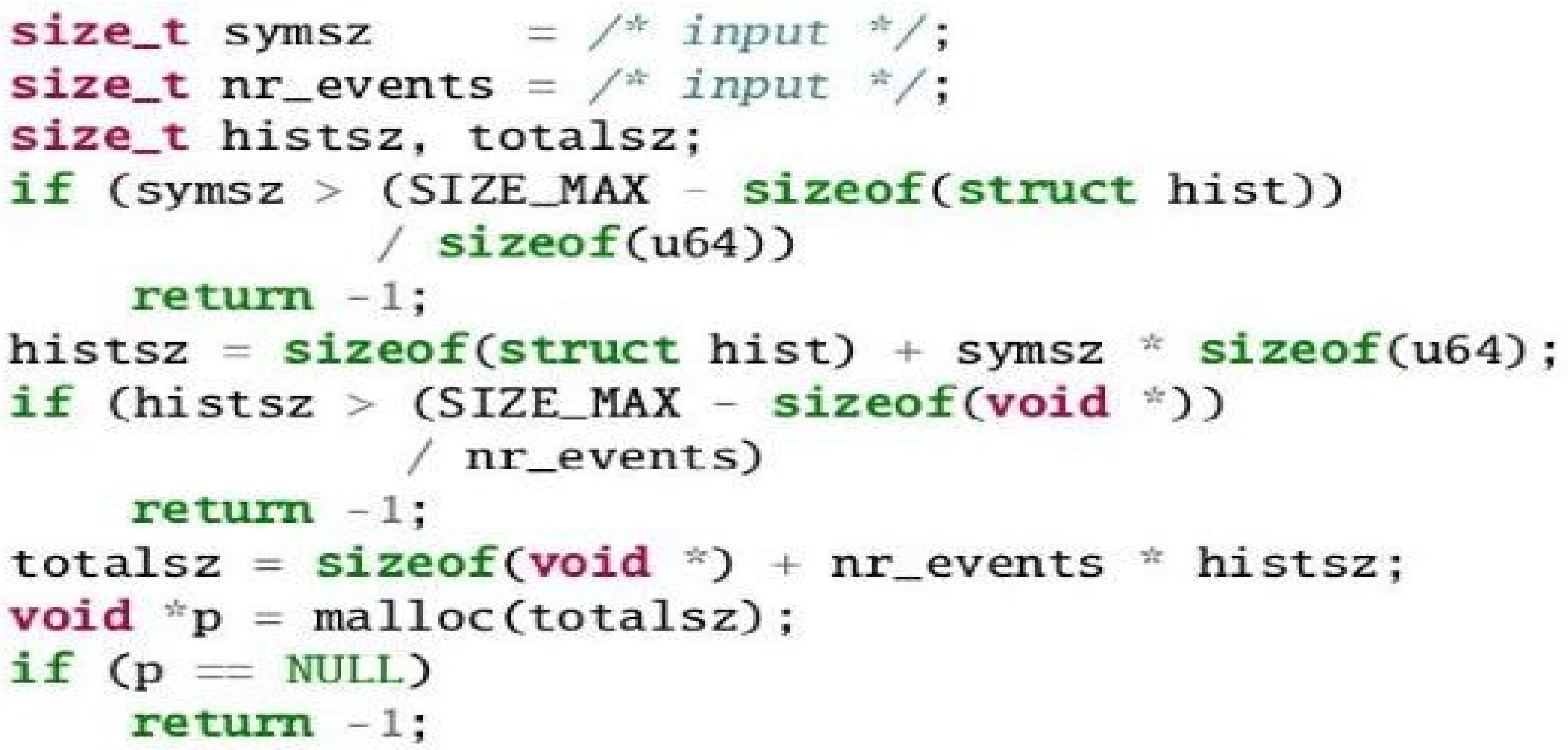
- Kint analyzed 8,916 files within roughly 160 minutes: 33 minutes for compila-tion using Clang, 87 minutes for range and taint analyses, and 37 minutes for generating constraints and solving 420,742 queries, of which 3,944 (0.94%) queries timed out.

Main Content

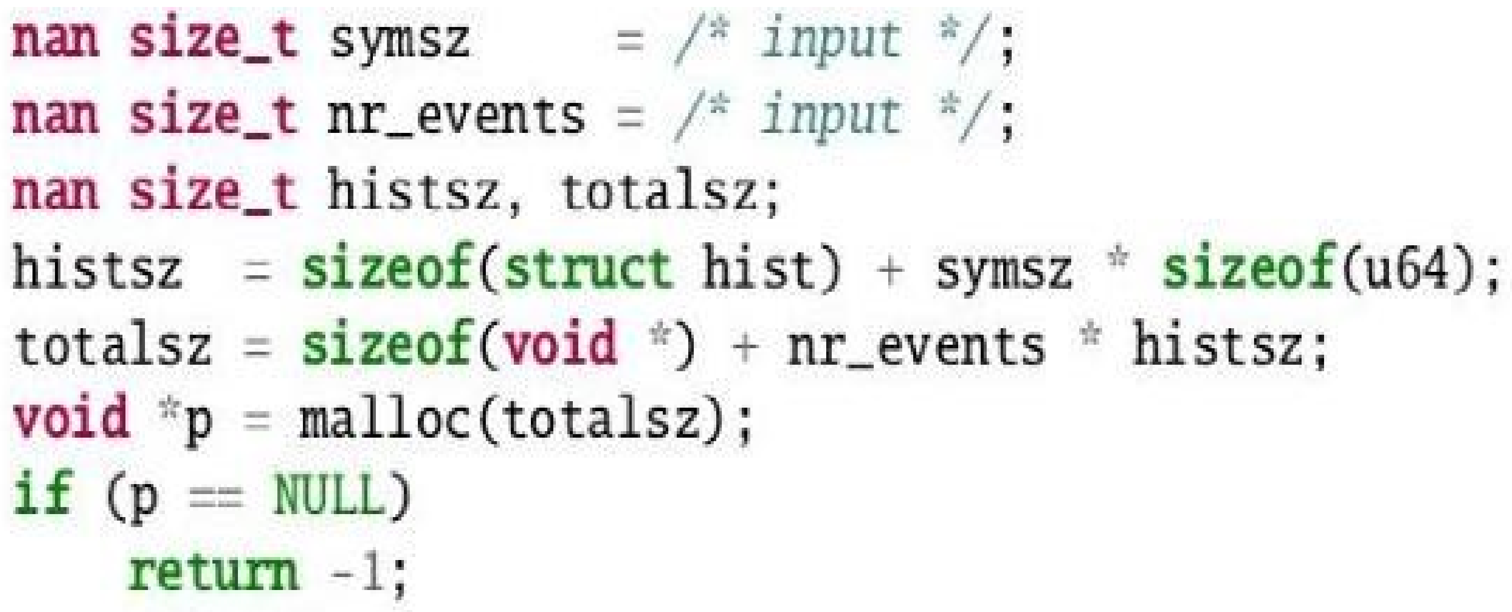
- What is Kint ?
- The features of Kint
- The design of Kint
- The evaluation of Kint
- **NaN integer semantics**

NaN integer semantics

- NaN - not-a-number
- NaN state



```
size_t symsz     = /* input */;
size_t nr_events = /* input */;
size_t histsz, totalsz;
if (symsz > (SIZE_MAX - sizeof(struct hist))
            / sizeof(u64))
    return -1;
histsz = sizeof(struct hist) + symsz * sizeof(u64);
if (histsz > (SIZE_MAX - sizeof(void *))
             / nr_events)
    return -1;
totalsz = sizeof(void *) + nr_events * histsz;
void *p = malloc(totalsz);
if (p == NULL)
    return -1;
```

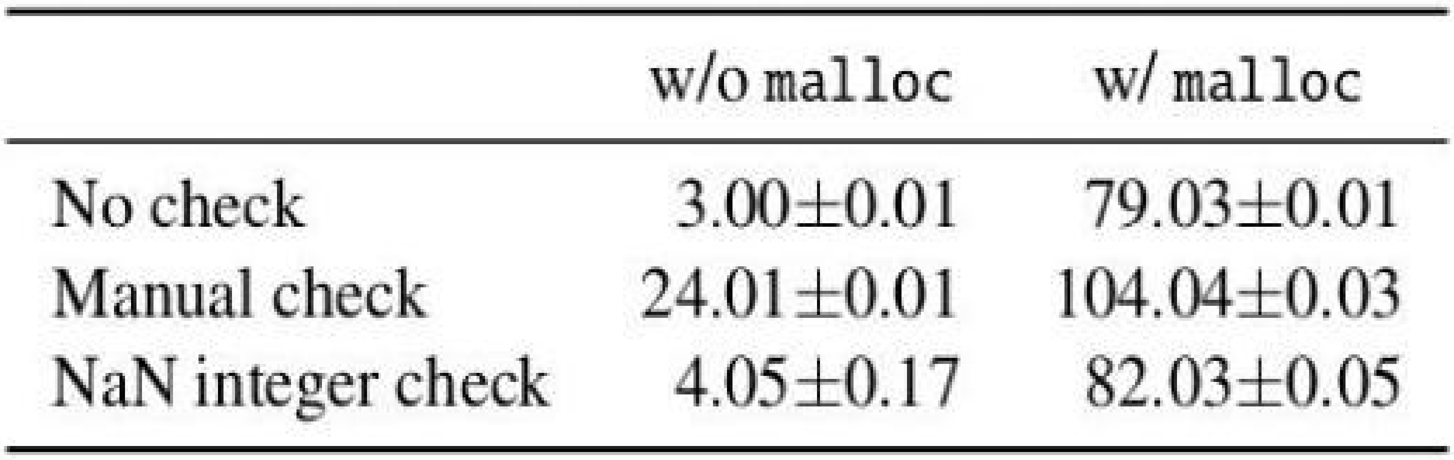


```
nan size_t symsz     = /* input */;
nan size_t nr_events = /* input */;
nan size_t histsz, totalsz;
histsz  = sizeof(struct hist) + symsz * sizeof(u64);
totalsz = sizeof(void *) + nr_events * histsz;
void *p = malloc(totalsz);
if (p == NULL)
    return -1;
```

NaN integer semantics

- void *malloc(nan size_t size)
- {
- if (isnan(size))
- return NULL;
- return libc_malloc((size_t) size);
- }

NaN integer semantics



	w/o `malloc`	w/ `malloc`
No check	3.00±0.01	79.03±0.01
Manual check	24.01±0.01	104.04±0.03
NaN integer check	4.05±0.17	82.03±0.05