Effect-Based Analysis of C Programs

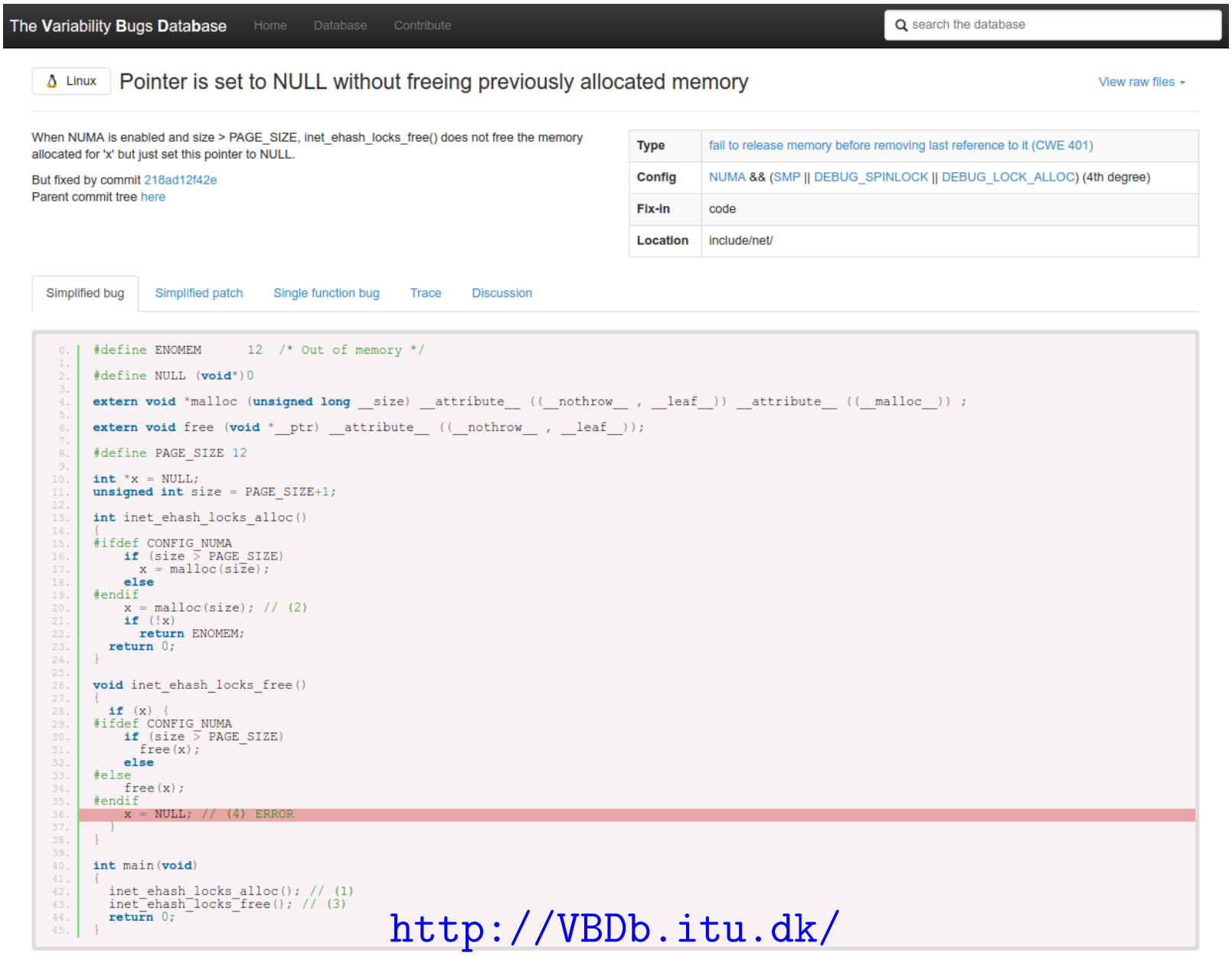
Iago Abal Rivas

iago@itu.dk

April 7, 2016

IT UNIVERSITY OF COPENHAGEN

Know your enemy



E.g. memory leak (single function)

```
1 void foo(void) {
2     int *p = malloc(sizeof(int));
3     *p = 42;
4     printf("%d", *p);
●5    return;
6 }
```

E.g. memory leak (cross function)

```
1 void foo(void) {
2     int *p = malloc(sizeof(int));
3     *p = 42;
4     f(p); // void f(int *x) { printf("%d", *x); }
●5    return;
6 }
```

E.g. memory leak (effect abstraction)

```
1 void foo(void) {
2     alloc(*p);              /* malloc(sizeof(int)) */
3     read(p), write(*p);     /* *p = 42 */
4     read(p,*p), out(*p);    /* f(p) */
●5    return;
6 }
```

E.g. double lock (fixed by d7e9711)

```
893 /* This routine is guarded by dqonoff_mutex mutex */
894 static void add_dquot_ref(struct super_block *sb, int type)
895 {
896         struct inode *inode, *old_inode = NULL;
897 #ifdef CONFIG_QUOTA_DEBUG
898         int reserved = 0;
899 #endif
900
901         spin_lock(&inode_sb_list_lock);
902         list_for_each_entry(inode, &sb->s_inodes, i_sb_list) {
903                 spin_lock(&inode->i_lock);
904                 if ((inode->i_state & (I_FREEING|I_WILL_FREE|I_NEW)) ||
905                     !atomic_read(&inode->i_writecount) ||
906                     !dqinit_needed(inode, type)) {
907                         spin_unlock(&inode->i_lock);
908                         continue;
909                 }
910 #ifdef CONFIG_QUOTA_DEBUG
911                 if (unlikely(inode_get_rsv_space(inode) > 0))
912                         reserved = 1;
913 #endif
914                 __iget(inode);
915                 spin_unlock(&inode->i_lock);
916                 spin_unlock(&inode_sb_list_lock);
917
918                 iput(old_inode);
919                 __dquot_initialize(inode, type);
920
921                 /*
922                  * We hold a reference to 'inode' so it couldn't have been
923                  * removed from s_inodes list while we dropped the
924                  * inode_sb_list_lock We cannot iput the inode now as we can be
925                  * holding the last reference and we cannot iput it under
926                  * inode_sb_list_lock. So we keep the reference and iput it
927                  * later.
928                  */
929                 old_inode = inode;
930                 spin_lock(&inode_sb_list_lock);
931         }
932         spin_unlock(&inode_sb_list_lock);
933         iput(old_inode);
934
935 #ifdef CONFIG_QUOTA_DEBUG
936         if (reserved) {
937                 quota_error(sb, "Writes happened before quota was turned on "
938                         "thus quota information is probably inconsistent. "
939                         "Please run quotacheck(8)");
940         }
941 #endif
942 }
```

E.g. double lock (fixed by d7e9711)

```
893 /* This routine is guarded by dqonoff_mutex mutex */
894 static void add_dquot_ref(struct super_block *sb, int type)
895 {
896         struct inode *inode, *old_inode = NULL;
897 #ifdef CONFIG_QUOTA_DEBUG
898         int reserved = 0;
899 #endif
900
901         spin_lock(&inode_sb_list_lock);
902         list_for_each_entry(inode, &sb->s_inodes, i_sb_list) {
903                 spin_lock(&inode->i_lock);
904                 if ((inode->i_state & (I_FREEING|I_WILL_FREE|I_NEW)) ||
905                     !atomic_read(&inode->i_writecount) ||
906                     !dqinit_needed(inode, type)) {
907                         spin_unlock(&inode->i_lock);
908                         continue;
909                 }
910 #ifdef CONFIG_QUOTA_DEBUG
911                 if (unlikely(inode_get_rsv_space(inode) > 0))
912                         reserved = 1;
913 #endif
914                 __iget(inode);
915                 spin_unlock(&inode->i_lock);
916                 spin_unlock(&inode_sb_list_lock);
917
918                 iput(old_inode);
919                 __dquot_initialize(inode, type);
920
921                 /*
922                  * We hold a reference to 'inode' so it couldn't have been
923                  * removed from s_inodes list while we dropped the
924                  * inode_sb_list_lock We cannot iput the inode now as we can be
925                  * holding the last reference and we cannot iput it under
926                  * inode_sb_list_lock. So we keep the reference and iput it
927                  * later.
928                  */
929                 old_inode = inode;
930                 spin_lock(&inode_sb_list_lock);
931         }
```

E.g. double lock (fixed by d7e9711)

```
893 /* This routine is guarded by dqonoff_mutex mutex */
894 static void add_dquot_ref(struct super_block *sb, int type)
895 {
896         struct inode *inode, *old_inode = NULL;
897 #ifdef CONFIG_QUOTA_DEBUG
898         int reserved = 0;
899 #endif
900
901         spin_lock(&inode_sb_list_lock);
902         list_for_each_entry(inode, &sb->s_inodes, i_sb_list) {
903                 spin_lock(&inode->i_lock);
904                 if ((inode->i_state & (I_FREEING|I_WILL_FREE|I_NEW)) ||
905                     !atomic_read(&inode->i_writecount) ||
906                     !dqinit_needed(inode, type)) {
907                         spin_unlock(&inode->i_lock);
908                         continue;
909                 }
910 #ifdef CONFIG_QUOTA_DEBUG
911                 if (unlikely(inode_get_rsv_space(inode) > 0))
912                         reserve
913 #endif
914                 __iget(inode);
915                 spin_unlock(&in
916                 spin_unlock(&in
917
918                 iput(old_inode)
919                 __dquot_initial
920
921                 /*
922                  * We hold a re
923                  * removed from
924                  * inode_sb_lis
925                  * holding the last reference and we cannot iput it under
926                  * inode_sb_list_lock. So we keep the reference and iput it
927                  * later.
928                  */
929                 old_inode = inode;
930                 spin_lock(&inode_sb_list_lock);
931         }
```

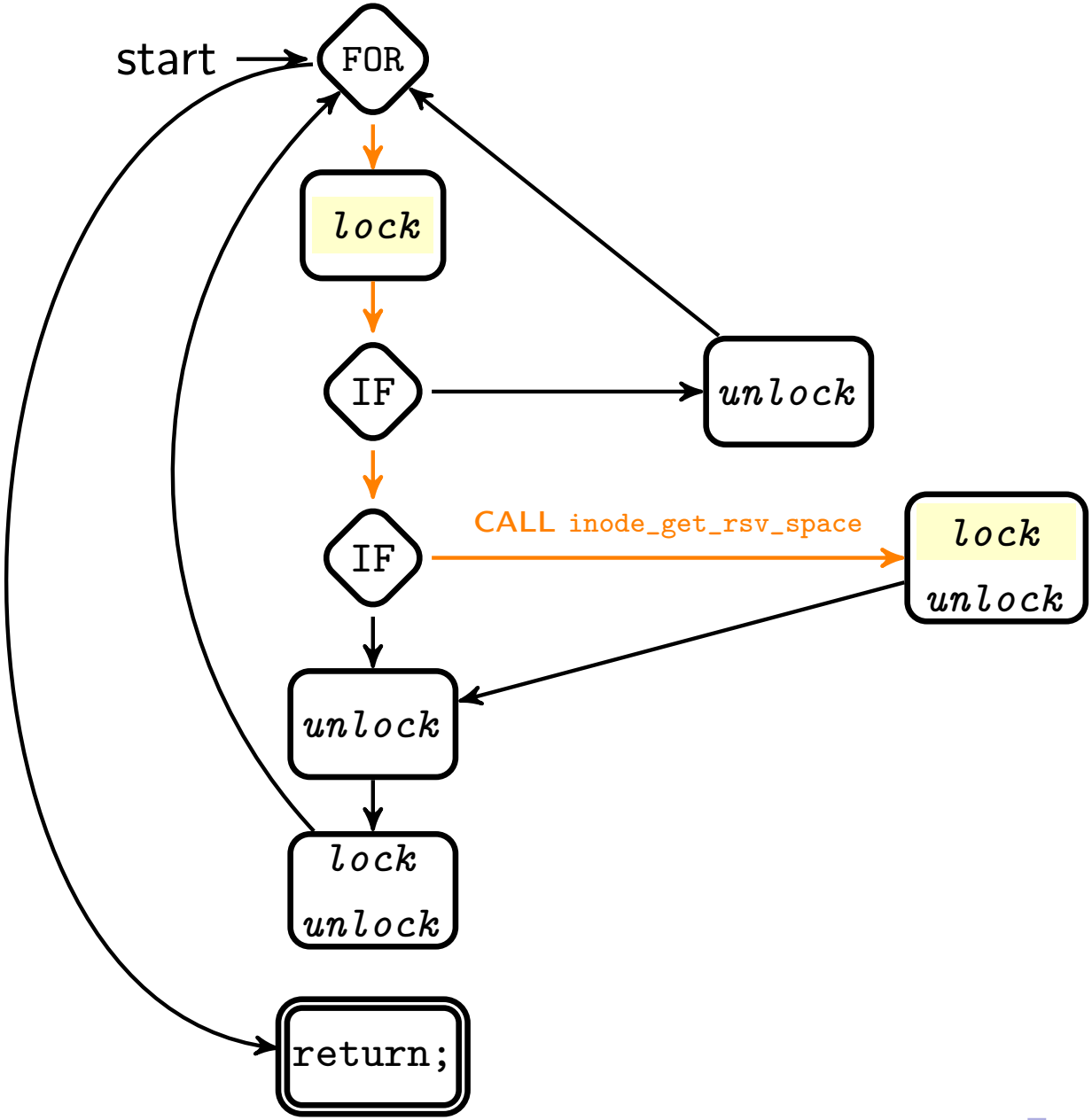
```
1517 static qsize_t inode_get_rsv_space(struct inode *inode)
1518 {
1519         qsize_t ret;
1520
1521         if (!inode->i_sb->dq_op->get_reserved_space)
1522                 return 0;
1523         spin_lock(&inode->i_lock);
1524         ret = *inode_reserved_space(inode);
1525         spin_unlock(&inode->i_lock);
1526         return ret;
1527 }
```

E.g. double lock (fixed by d7e9711)

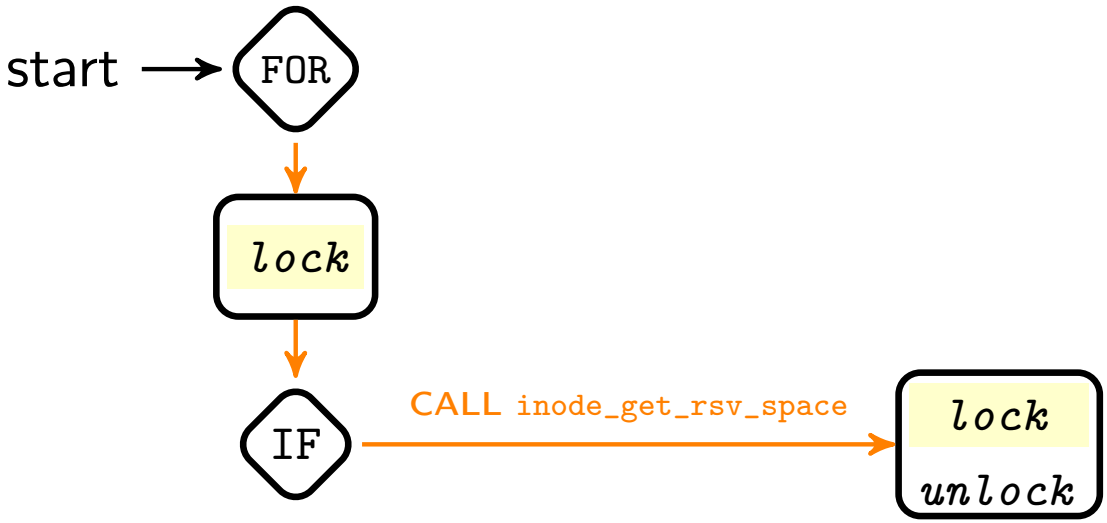
```
893 /* This routine is guarded by dqonoff_mutex mutex */
894 static void add_dquot_ref(struct super_block *sb, int type)
895 {
896         struct inode *inode, *old_inode = NULL;
897 #ifdef CONFIG_QUOTA_DEBUG
898         int reserved = 0;
899 #endif
900
901         spin_lock(&inode_sb_list_lock);
902         list_for_each_entry(inode, &sb->s_inodes, i_sb_list) {
903                 spin_lock(&inode->i_lock);
904                 if ((inode->i_state & (I_FREEING|I_WILL_FREE|I_NEW)) ||
905                     !atomic_read(&inode->i_writecount) ||
906                     !dqinit_needed(inode, type)) {
907                         spin_unlock(&inode->i_lock);
908                         continue;
909                 }
910 #ifdef CONFIG_QUOTA_DEBUG
911                 if (unlikely(inode_get_rsv_space(inode) > 0))
912                         reserve
913 #endif
914                 __iget(inode);
915                 spin_unlock(&in
916                 spin_unlock(&in
917
918                 iput(old_inode)
919                 __dquot_initial
920
921                 /*
922                  * We hold a re
923                  * removed from
924                  * inode_sb_lis
925                  * holding the last reference and we cannot iput it under
926                  * inode_sb_list_lock. So we keep the reference and iput it
927                  * later.
928                  */
929                 old_inode = inode;
930                 spin_lock(&inode_sb_list_lock);
931         }
```

```
1517 static qsize_t inode_get_rsv_space(struct inode *inode)
1518 {
1519         qsize_t ret;
1520
1521         if (!inode->i_sb->dq_op->get_reserved_space)
1522                 return 0;
1523         spin_lock(&inode->i_lock);
1524         ret = *inode_reserved_space(inode);
1525         spin_unlock(&inode->i_lock);
1526         return ret;
1527 }
```

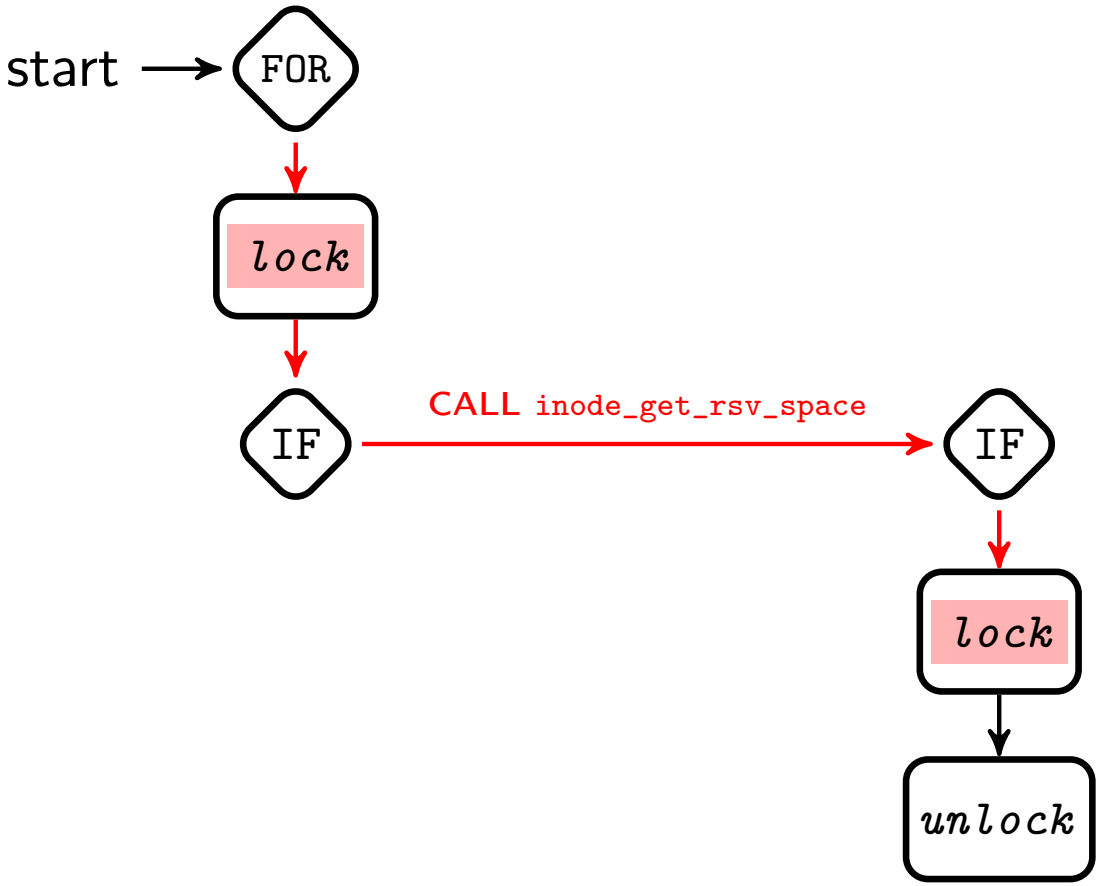
E.g. double lock



E.g. double lock



E.g. double lock



EBA in a nutshell

1. Performs *shape-and-effect* inference of C programs.
2. Decorates programs' CFG with shape and effects info.
3. Model-checks decorated CFG looking for bug patterns.

https://EBA.wikit.itu.dk/

EBA in a nutshell

1. Performs *shape-and-effect* inference of C programs.
2. Decorates programs' CFG with shape and effects info.
3. Model-checks decorated CFG looking for bug patterns.

`https://EBA.wikit.itu.dk/`

- Very early prototype with some limitations.
- Analyzes Linux files in seconds (to minutes)!

Shapes

$$
\begin{array}{llcl}
\textit{r-value shapes} & Z^R & : & \bot \mid \text{ptr } Z^L \mid \zeta \\
\textit{f-value shapes} & Z^F & : & Z^R \mid Z_1^L \times \cdots \times Z_n^L \xrightarrow{\varphi} Z_0^R \\
\textit{l-value shapes} & Z^L & : & \text{ref}_\rho\ Z^F
\end{array}
$$

- C types do not reflect aliasing.
  - E.g. `int x, *p; p = &x;` $\Rightarrow$ `x` and `*p` are aliased.
- C types are *(ab)*used in many ways.
  - E.g. `int data; f((struct inode *)data)`.
- E.g. Given `int x = 42;` then `&x` has shape ptr ref$_\rho$ $\bot$:

Effects

$$\begin{array}{lllll} F_X \ f_X & : & \varepsilon(\overrightarrow{\rho}) & & \\ \text{Effect} \quad \varphi & : & \emptyset \mid \{f_X\} \mid \xi \mid \varphi_1 \cup \varphi_2 & & \end{array}$$

- Effects denote operations on memory regions.
- Given an expression e, what happens when e is evaluated?
- E.g. $lock_\rho$ denotes the acquisition of a lock object at ρ.
- Crucially, a function call can be replaced by its effect signature.
- Flow-insensitive abstraction $\Rightarrow$ *Effect-driven inlining*.

Evaluation

Table 2: Analyzing Linux: double locks DL, return with a lock held LH, enable softirqs with interrupts disabled BI, disable nested softirqs NB. The four last columns count the alarms raised in each subsystem

kernel subsystem	files	average time [s]	average mem. [MB]	DL	LH	BI	NB
arch	1367	5.71	229	0	1	0	0
crypto	116	1.48	212	0	0	0	0
drivers	8,598	22.93	346	7	6	1	2
fs	877	9.12	260	4	10	0	0
init	9	19.03	351	0	0	0	0
ipc	10	24.51	461	0	1	0	0
kernel	221	15.23	335	1	1	1	0
lib	213	1.23	241	0	0	0	0
mm	82	14.31	320	0	1	0	0
net	1215	18.18	372	0	0	0	5
security	76	7.54	272	0	0	0	0
sound	584	24.02	381	0	0	0	0
total	13,368	19.05	329	12	20	2	7

Evaluation

Table 3: The reported alarms. FP-a: false positives caused by aliasing imprecision. FP-p: false positives caused by path correlations. For true positives, we count how many span multiple functions (*cross fun*).

checker	*alarms*	*FP-a*	*FP-p*	*cross fun*
DL	12	9	3	0
LH	20	0	4	3
BI	2	0	0	2
NB	7	0	0	7
sum	41	9	7	12

Demo

Thank you