

Effect-Based Analysis of C Programs

Iago Abal Rivas

iago@itu.dk

April 7, 2016

IT UNIVERSITY OF COPENHAGEN

Know your enemy



Pointer is set to NULL without freeing previously allocated memory

[View raw files](#) +

When NUMA is enabled and size > PAGE_SIZE, inet_ehash_locks_free() does not free the memory allocated for 'x' but just set this pointer to NULL.

But fixed by commit [21bad12f42e](#)

Parent commit tree [here](#)

Type	fail to release memory before removing last reference to it (CWE 401)
Config	NUMA && (SMP DEBUG_SPINLOCK DEBUG_LOCK_ALLOC) (4th degree)
Fix-in	code
Location	include/net/

[Simplified bug](#)[Simplified patch](#)[Single function bug](#)[Trace](#)[Discussion](#)

```
01. #define ENOMEM 12 /* Out of memory */
02.
03. #define NULL (void*)0
04.
05. extern void *malloc (unsigned long __size) __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__malloc__));
06. extern void free (void * __ptr) __attribute__ ((__nothrow__, __leaf__));
07.
08. #define PAGE_SIZE 12
09.
10. int *x = NULL;
11. unsigned int size = PAGE_SIZE+1;
12.
13. int inet_ehash_locks_alloc()
14. {
15. #ifdef CONFIG_NUMA
16.     if (size > PAGE_SIZE)
17.         x = malloc(size);
18.     else
19. #endif
20.         x = malloc(size); // (2)
21.     if (!x)
22.         return ENOMEM;
23.     return 0;
24. }
25.
26. void inet_ehash_locks_free()
27. {
28.     if (x) {
29. #ifdef CONFIG_NUMA
30.         if (size > PAGE_SIZE)
31.             free(x);
32.         else
33. #else
34.             free(x);
35. #endif
36.         x = NULL; // (4) ERROR
37.     }
38. }
39.
40. int main(void)
41. {
42.     inet_ehash_locks_alloc(); // (1)
43.     inet_ehash_locks_free(); // (3)
44.     return 0;
45. }
```

<http://VBDb.itu.dk/>

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     }
932     spin_unlock(&inode_sb_list_lock);
```



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->dqonoff_reserved > 0))
912             reserve;
913 #endif
914         iget(inode);
915         spin_unlock(&inode->i_lock);
916         spin_unlock(&inode_sb_list_lock);
917         iput(old_inode);
918         dquot_initial;
919         /*
920          * We hold a reference to old_inode because it was
921          * removed from the list. We still hold a reference to
922          * it because we are holding the last reference to it
923          * under the inode_sb_list_lock. So we keep the reference
924          * and iput it later.
925          */
926         old_inode = inode;
927         spin_lock(&inode_sb_list_lock);
928     }
929     spin_unlock(&inode_sb_list_lock);
```

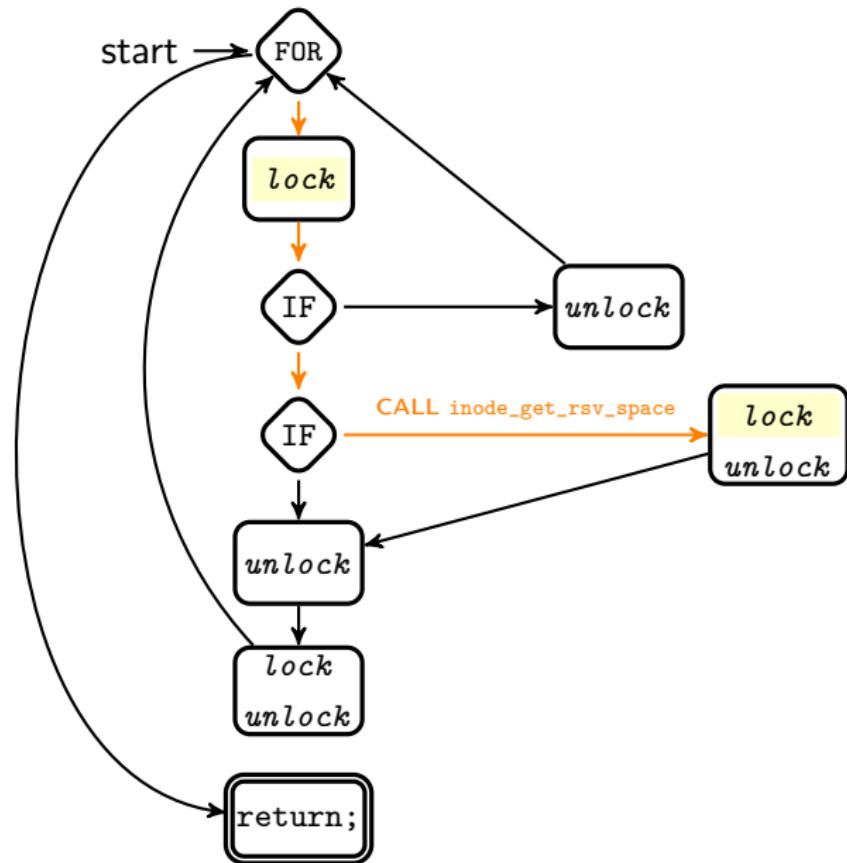


E.g. double lock (fixed by d7e9711)

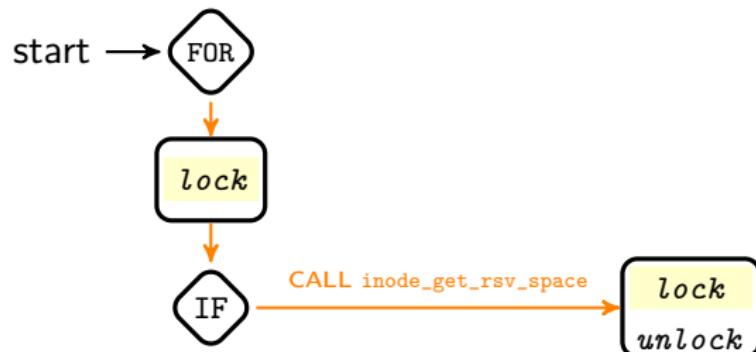
```
893 /* This routine is guarded by dqonoff_mutex mutex */
894 static void add_dquot_ref(struct super_block *sb, int type)
895 {
896     struct inode *inode, *old_inode = NULL;
897 #ifdef CONFIG_QUOTA_DEBUG
898     int reserved = 0;
899 #endif
900
901     spin_lock(&inode_sb_list_lock);
902     list_for_each_entry(inode, &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->dqonoff_reserved > 0))
912             reserve;
913 #endif
914         iget(inode);
915         spin_unlock(&inode->i_lock);
916         spin_unlock(&inode_sb_list_lock);
917         ifput(old_inode);
918         dquot_initialize(inode);
919         spin_lock(&inode->i_lock);
920         if (inode->dqonoff_reserved > 0)
921             /* We hold a reference to the inode
922              * removed from the list.
923              * inode_sb_list_lock is held.
924              * holding the last reference and we cannot ifput it under
925              * inode_sb_list_lock. So we keep the reference and ifput it
926              * later.
927              */
928             old_inode = inode;
929             spin_lock(&inode_sb_list_lock);
930         spin_unlock(&inode_sb_list_lock);
931     }
932 }
```



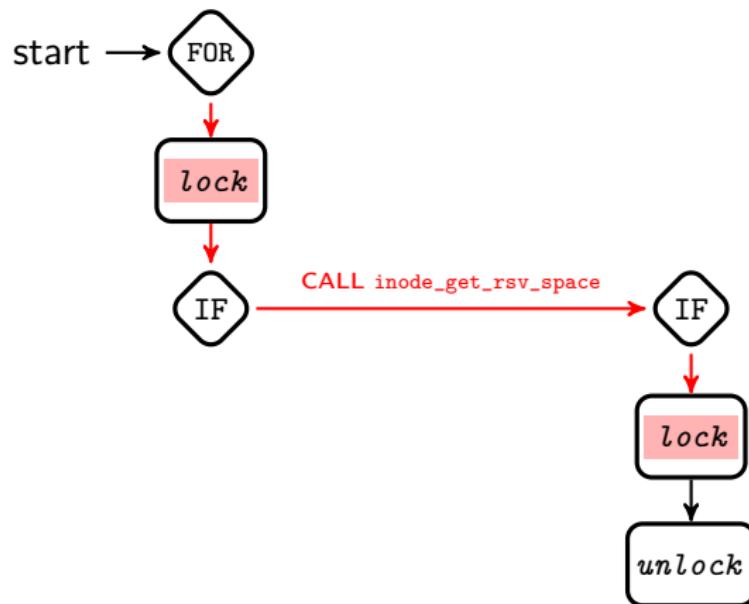
E.g. double lock



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

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

- ▶ C types do not reflect aliasing.
 - ▶ E.g. `int x, *p; p = &x;` ⇒ `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 refρ ⊥`:

0xCAFE:
0xCAFE → 42

Effects

$$\begin{array}{l} Fx \ f_x : \varepsilon(\vec{\rho}) \\ \text{Effect } \varphi : \emptyset \mid \{f_x\} \mid \xi \mid \varphi_1 \cup \varphi_2 \end{array}$$

- ▶ 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*).

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

Demo

Demo

Thank you

