

Correctness analysis

Preliminaries

- True positive (TP): An influential change is correctly classified as influential
 - `x in EIS && x in AIS`
 - Occurs if a **FAILURE** verification result from CBMC is considered sound based on manual inspection.
- True negative (TN): An equivalent change is correctly classified as equivalent
 - `x not in EIS && x not in AIS`
 - Occurs if a **SUCCESS** verification result from CBMC is considered sound based on manual inspection.
- False positive (FP): An equivalent change is incorrectly classified as influential
 - `x in EIS && x not in AIS`
 - Occurs if a **FAILURE** verification result from CBMC is considered unsound based on manual inspection.
- False negative (FN): An influential change is correctly classified as equivalent
 - `x not in EIS && x in AIS`
 - Occurs if a **SUCCESS** verification result from CBMC is considered unsound based on manual inspection.

Note that the CBMC traces do not show states in order of occurrence, the counterexample trace may also differ between repeated invocations since there could exist several counterexamples.

The standard requirement for manually verifying that a verification result is sound is to device an equivalent counterpart to the **FAILURE** example which highlights that the failed verification indeed contains a new execution flow. All of these crafted examples were performed with `--unwind 1` (just like the actual trials).

Note, the harnesses created by EUF are always created inside of a `.harnesses` directory in the old version of the repository. This means that all headers that are included correspond to headers from the old version of the dependency. Any changes to the struct definitions in these headers which may have occurred in an update, are thus not taken into account when declaring the non-deterministic input variables.

Each library will however still have their own canonically defined version of each struct separate from the one imported in the harness. I.e. invocations of a new function will use the struct definition relevant to their own context but the input may have been initialised using a different template.

If not otherwise stated, screenshots with four splits show the influential update to the *left* and the equivalent update to the *right*.

Unless otherwise stated, each function analysis focuses on a specific **SUCCESS** and **FAILURE** result. This is preferable since multiple occurrences of the same result for a function often relate to the same underlying change.

libonig

```
renumber_node_backref()

SUCCESS_UNWIND_FAIL (1)
FAILURE_UNWIND_FAIL (2)

./scripts/analyze_function.sh onig renumber_node_backref 388e 2350
git difftool --no-index ~/.cache/euf/oniguruma-388e/src/regcomp.c \
  ~/.cache/euf/oniguruma-2350/src/regcomp.c
# 1899:1 <-> 1900:1
# Wed Jun 7 14:20:18 2017 -> Tue Jun 20 16:30:21 2017
```

```
# => ~/.cache/euf/oniguruma-388e/.harnesses/rename_node_backref.c (FAILURE_UNWIND_FAIL)
```

```
./scripts/analyze_function.sh onig renumber_node_backref 3b63 3c57  
git difftool --no-index ~/.cache/euf/oniguruma-3b63/src/regcomp.c \  
~/cache/euf/oniguruma-3c57/src/regcomp.c
```

1824:1 <-> 1901:1

Wed May 24 13:43:25 2017 -> Thu Jun 8 13:24:35 2017

```
# => ~/.cache/euf/oniguruma-3b63/.harnesses/rename_node_backref.c (SUCCESS_UNWIND_FAIL)
```

```
static int  
renumber_node_backref(Node* node, GroupNumRemap* map)  
{  
    int i, pos, n, old_num;
```

```

int b->back_num;
BRefNode* bn = BREF(node);

if (!IS_BACKREF_NAME_REF((node)))
    return OGREJECT(NODEDID_BACKREF_OR_CALL_NODE);

old_num = bn->back_num;
if (IS_NULL(bn->back_dynamic))
    hacks = bn->back_static;
else
    hacks = bn->back_dynamic;

for (i = 0; i < pos; i++) {
    n = map[bn->ids[i]].new_val;
    if (n > 0)
        backi[nos] = n;
    nos++;
}
return nos;
}

```

```
/home/jonas/.cache/euf/oniguruma-388e/src/regcomp.  
\\<renumber node backref\\>
```

```
static int  
renumber_node_backref(Node* node, GroupNumRemap* map)  
{  
    int i, pos, n, old_num;
```

```

    int <back>;  

    BrefNodes* bn = BREF_NODE(node);  

  

    if (!NODE_IS_NAME_REF(node))  

        return ONIGERR_NUMBERED_BACKREF_OR_CALL_NOT_ALLOW;  

  

    old_num = bn->back_num;  

    if (IS_NULL(bn->back_dynamic))  

        backs = bn->back_static;  

    else  

        backs = bn->back_dynamic;  

  

    for (i = 0, pos = 0; i < old_num; i++) {  

        if (backs[i] > 0) {  

            backs[pos] = new_val;  

            pos++;  

        }
    }
  

    bn->back_num = pos;
    return 0;
}

```

c 29% 1895 1 N /home/jonas/.cache/euf/oniguruma-2350/src/regcomp.

```
static int  
renumber_node_backref(Node* node, GroupNumRemap* map)  
{  
    int i, pos, n, old_num;
```

```

    int *back_num;
    int *backs;
    BFRNode* bn = NBBFNode();
}

if (!IS_BACRF_NAME_DEF(n))
    return ONTIGER_NUMBERED_BACRF_OR_CALL_NOT_ALLOWED;

old_num = bn->back_num;
if (IS_NULL(bn->back_dynamic))
    backs = bn->back_static;
else
    backs = bn->back_dynamic;

for (i = 0, pos = 0; i < old_num; i++) {
    if (n == backs[i]) new_val;
    if (n > 0)
        backs[pos] = n;
    pos++;
}
}

bn->back_num = pos;
return 0;
}

```

N... <./.cache/euf/oniguruma-3b63/src/regcomp.c c 28%
Δcrenumber

```
static int  
renumber_node_backref(Node* node, GroupNumRemap* map)  
{  
    int i, pos, n, old_num;
```

```

    if (!NOPE_IS_NAME_REF(node))
        return OMEGERR_NOMETHOD_BACKREF_OR_CALL_NOT_ALLOWED;

    old_num = bn->back_num;
    if (IS_NULL(bn->back_dynamic))
        backs = bn->back_static;
    else
        backs = bn->back_dynamic;

    for (i = 0, pos = 0; i < old_num; i++) {
        if (n == backs[i].new_val)
            if (n != 0)
                backs[pos] = n;
            pos++;
    }
}

bn->back_num = pos;
return 0;

```

jonas/.cache/euf/oniguruma-3c57/src/regcomp.c c 29% 189 [1]

The influential and equivalent update both originate from textually identical versions of `renumber_node_backref`. Both updates modify the `if` condition that determines if `ONIGERR_NUMBERED_BACKREF_OR_CALL_NOT_ALLOWED` (-209) is returned. CBMC has found a counterexample where the `2350` revision evaluates the condition as true while the old version evaluates it as false for the same input.

```

    default:
        break;
    }

    return r;
}

static int
renumber_node_backref(Node* node, GroupNumRemap* map)
{
    int i, pos, n, old_num;
    int *backs;
    BRefNode* bn = BREF(node);

    if (!IS_BACKREF_NAME_REF(bn))
        return ONIGERR_NUMBERED_BACKREF_OR_CALL_NOT_ALLOWED;

    old_num = bn->back_num;
    if (IS_NULL(bn->back_dynamic))
        backs = bn->back_static;
    else
        backs = bn->back_dynamic;

    for (i = 0, pos = 0; i < old_num; i++) {
        n = map[backs[i]].new_val;
        if (n > 0) {
            backs[pos] = n;
            pos++;
        }
    }

    bn->back_num = pos;
    return 0;
}

```

```

    default:
        break;
    }

    return r;
}

static int
renumber_node_backref(Node* node, GroupLinkRemap* map)
{
    int i, pos, n, old_num;
    int *backs;
    BRefNode* bn = BREF((node));

    if (!NODE_IS_NAME_REF(node))
        return ONEKEY_NUMBERED_BACKREF_OR_CALL_NOT_ALLOWED;

    old_num = bn->back_num;
    if (IS_NULL(bn->back_dynamic))
        backs = bn->back_static;
    else
        backs = bn->back_dynamic;

    for (i = 0, pos = 0; i < old_num; i++) {
        n = map[banks[i]].new_val;
        if (n > 0)
            backs[pos] = n;
        pos++;
    }

    bn->back_num = pos;
    return 0;
}

/home/jonas/.cache/euf/oniguruma-2350/src/regcomp.c c

```

```
==> State 11637 <=
```

```
State 11637 file /home/jonas/.cache/euf/oniguruma-388e.harnesses/rename_node_backref.c function euf_main line 41 thread 0
```

```
    ret_val=0 (00000000 00000000 00000000 00000000)
```

```
State 11655 file /home/jonas/.cache/euf/oniguruma-388e.harnesses/rename_node_backref.c function euf_main line 41 thread 0
```

```
    ret_val=-209 (11111111 11111111 00101111)
```

```
State 11656 file /home/jonas/.cache/euf/oniguruma-388e.harnesses/rename_node_backref.c function euf_main line 42 thread 0
```

```
    ret=0 (00000000 00000000 00000000 00000000)
```

```
State 11680 file /home/jonas/.cache/euf/oniguruma-388e.harnesses/rename_node_backref.c function euf_main line 42 thread 0
```

```
    ret=0 (00000000 00000000 00000000 00000000)
```

```
State 11648 file regcomp.c function renumber_node_backref_old_b026324c6904b2a line 1903 thread 0
```

```
-----
```

```
    bn=(BRegNode *)NULL (?)
```

```
State 11649 file regcomp.c function renumber_node_backref_old_b026324c6904b2a line 1903 thread 0
```

```
-----
```

```
    bn=(BRegNode *)NULL (?)
```

```
State 11667 file regcomp.c function renumber_node_backref line 1984 thread 0
```

```
-----
```

```
    bn=(BRegnode *)NULL (?)
```

```
State 11668 file regcomp.c function renumber_node_backref line 1904 thread 0
```

```
-----
```

```
    bn=(BRegNode *)NULL (?)
```

```
State 11627 file /home/jonas/.cache/euf/oniguruma-388e.harnesses/rename_node_backref.c function euf_main line 35 thread 0
```

```
-----
```

```
    node=((struct _Node *)NULL) (?)
```

```
State 11641 file /home/jonas/.cache/euf/oniguruma-388e.harnesses/rename_node_backref.c function euf_main line 41 thread 0
```

```
    node=((struct _Node *)NULL) (?)
```

```
State 11658 file /home/jonas/.cache/euf/oniguruma-388e.harnesses/rename_node_backref.c function euf_main line 42 thread 0
```

```
-----
```

```
    node=((Node *)NULL) (?)
```

```
~/Repos/euf [main]
```

Expanding macros shows that the change from `NBREF` \rightarrow `BREF` in the influential case has no effect.

// Same across all versions

```
#define ONIGERR_NUMBERED_BACKREF_OR_CALL_NOT_ALLOWED
```

-209

// 388e, 3b63 (old versions) and 3c57 (new equivalent version)

```
#define NBREF(node) ((node)->u.bref)
```

// 2350 (new influential version)

```
#define BREF_(node) (&((node)->u.bref))
```

The change from IS_BACKREF_NAME_REF → NODE_IS_NAME_REF looks equivalent in both cases on first glance but expanding all macros shows a notable difference between 2350 and the other three revisions.

```
// 388e and 3b63 (old versions)
#define IS_BACKREF_NAME_REF(bn) (((bn)->state & NST_NAME_REF) != 0)
```

// 3c57 (new equivalent)

```
#define NODE_IS_NAME_REF(node) (((node)->u.base.status & NST_NAME_REF) != 0)
```

// 2350 (new influential)

```
#define NODE_IS_NAME_REF(node) ((NODE_STATUS(node) & NST_NAME_REF) != 0)
#define NODE_STATUS(node) (((Node*)node)->u.base.status)
```

The difference which triggers the counterexample lies in the definition of NST_NAME_REF.

```
#define NST_NAMED_GROUP      (1<<10)
#define NST_NAME_REF          (1<<11)
#define NST_IN_REPEAT         (1<<12) /* STK_REPEAT is nested in
/home/jonas/.cache/euf/oniguruma-388e/src/regparse.h  cpp 30%129$9
NST_NAME_REF */

#define NST_NAME_REF          (1<<10)
#define NST_IN_REPEAT         (1<<11) /* STK_REPEAT is nested in
/home/jonas/.cache/euf/oniguruma-2350/src/regparse.h  cpp 32%132$9
NST_NAME_REF */

#define NST_NAMED_GROUP      (1<<10)
#define NST_NAME_REF          (1<<11)
#define NST_IN_REPEAT         (1<<12) /* STK_REPEAT is nested in
/home/jonas/.cache/euf/oniguruma-3c57/src/regparse.h  cpp 15%156$9
NST_NAME_REF */

#define NST_NAMED_GROUP      (1<<10)
#define NST_NAME_REF          (1<<11)
#define NST_IN_REPEAT         (1<<12) /* STK_REPEAT is nested in
/home/jonas/.cache/euf/oniguruma-3c57/src/regparse.h  cpp 15%156$9
NST_NAME_REF */


```

Patching regparse.h in revision 2350 to have the same definition of NST_NAME_REF gives a successful equivalence verification. The reason for CBMC's labeling has thus been pin-pointed and both classifications are thus considered sound.

```
#define NQ_TARGET_IS_EMPTY_REC 3
/* status bits */
#define NST_MIN_FIXED      (1<<0)
#define NST_MAX_FIXED      (1<<1)
#define NST_CLEN_FIXED     (1<<2)
#define NST_MARK1           (1<<3)
#define NST_MARK2           (1<<4)
#define NST_MARK3           (1<<5)
#define NST_MARK4           (1<<6)
#define NST_STOP_BT_SIMPLE_REPEAT (1<<6)
#define NST_RECURRENCE      (1<<7)
#define NST_CALLED          (1<<8)
#define NST_ADDR_FIXED      (1<<9)
#define NST_NAMED_GROUP     (1<<10)
#define NST_NAME_REF         (1<<11)
#define NST_IN_REPEAT        (1<<12) /* STK_REPEAT is nested in
/home/jonas/.cache/euf/oniguruma-388e/src/regparse.h  cpp 30%130$40
N_<onos/.cache/euf/oniguruma-2350/src/regparse.h  cpp 32%132$40
~/cache/euf/oniguruma-2350/src/regparse.h" 404L, 13165B written

#define NQ_BODY_IS_EMPTY_REC 3
/* status bits */
#define NST_MIN_FIXED      (1<<0)
#define NST_MAX_FIXED      (1<<1)
#define NST_CLEN_FIXED     (1<<2)
#define NST_MARK1           (1<<3)
#define NST_MARK2           (1<<4)
#define NST_MARK3           (1<<5)
#define NST_MARK4           (1<<6)
#define NST_STOP_BT_SIMPLE_REPEAT (1<<6)
#define NST_RECURRENCE      (1<<7)
#define NST_CALLED          (1<<8)
#define NST_ADDR_FIXED      (1<<9)
#define NST_NAMED_GROUP     (1<<10)
#define NST_NAME_REF         (1<<11)
#define NST_IN_REPEAT        (1<<12) /* STK_REPEAT is nested in
N_<onos/.cache/euf/oniguruma-2350/src/regparse.h  cpp 32%132$40
~/cache/euf/oniguruma-2350/src/regparse.h" 404L, 13165B written

** Results:
/home/jonas/.cache/euf/oniguruma-388e/harnesses/rename_node_backref.c function euf_main
[euf_main.assertion.1] line 44 Equivalent output: SUCCESS
** 0 of 1 failed (1 iterations)
VERIFICATION SUCCESSFUL
real    0m3.22s
user    0m3.140s
sys     0m0.086s
[✓] Passed equivalence assertion: rename_node_backref: 0:00:08.594658
[>] Change set reduction: 71 -> 70: 0:00:25.773006
[>] Total runtime: 0:00:39.840950
-/Repos/euf [main]
Vel ->
```

Before noticing the difference in NST_NAME_REF, differences in the composition of the Node struct were investigated.

```

typedef struct {
    NodeBase base;
    int type;
    int back_num;
    int back_static[NODE_BACKREFS_SIZE];
    int back_dynamic;
    int nest_level;
} BRefNode;

typedef struct {
    NodeBase base;
}

int type;
struct _Node* target;
int char_len;
} AnchorNode;

typedef struct {
    NodeBase base;
}

struct _Node* car;
struct _Node* cdr;
} ConsAltNode;

typedef struct {
    NodeBase base;
    int type;
    int char_len;
} AnchorNode;

typedef struct {
    NodeBase base;
    int type;
    int status;
    struct _Node* body;
} Node;

int type;
int not;
} CTypeNode;

typedef struct _Node {
    union {
        NodeBase base;
    };
    StrNode str;
    CClassNode cclass;
    QTrMode qtrf;
    EncloseNode enclose;
    BRefNode bref;
    AnchorNode anchor;
    ConsAltNode cons;
    CTypeNode ctype;
    #ifdef USE_SUREXO_CALL
    CallNode call;
    #endif
} u;
} Node;

```

```

typedef struct {
    NodeBase base;
    int type;
    int status;
    struct _Node* body;
} Node;

int type;
int back_num;
int back_static[NODE_BACKREFS_SIZE];
int back_dynamic;
int nest_level;
} BRefNode;

typedef struct {
    NodeBase base;
    int type;
    int status;
    struct _Node* body;
} Node;

int type;
int char_len;
} AnchorNode;

typedef struct {
    NodeBase base;
    int type;
    int status;
    struct _Node* target;
    int char_len;
} AnchorNode;

typedef struct {
    NodeBase base;
    int type;
    int status;
    struct _Node* car;
    struct _Node* cdr;
} ConsAltNode;

typedef struct {
    NodeBase base;
    int type;
    int status;
} AnchorNode;

typedef struct {
    NodeBase base;
    int type;
    int not;
} CTypeNode;

typedef struct _Node {
    union {
        NodeBase base;
    };
    StrNode str;
    CClassNode cclass;
    QTrMode qtrf;
    EncloseNode enclose;
    BRefNode bref;
    AnchorNode anchor;
    ConsAltNode cons;
    CTypeNode ctype;
    #ifdef USE_SUREXO_CALL
    CallNode call;
    #endif
} u;
} Node;

```

```

typedef struct {
    NodeBase base;
    int type;
    int status;
    struct _Node* body;
} Node;

int type;
int back_num;
int back_static[NODE_BACKREFS_SIZE];
int back_dynamic;
int nest_level;
} BRefNode;

typedef struct {
    NodeBase base;
    int type;
    int status;
    struct _Node* target;
    int char_len;
} AnchorNode;

typedef struct {
    NodeBase base;
    int type;
    int status;
    struct _Node* car;
    struct _Node* cdr;
} ConsAltNode;

typedef struct {
    NodeBase base;
    int type;
    int status;
    struct _Node* car;
    struct _Node* cdr;
} ConsAltNode;

typedef struct {
    NodeBase base;
    int type;
    int not;
} CTypeNode;

typedef struct _Node {
    union {
        NodeBase base;
        StrNode str;
        CClassNode cclass;
        QTrMode qtrf;
        EncloseNode enclose;
        BRefNode bref;
        AnchorNode anchor;
        ConsAltNode cons;
        CTypeNode ctype;
        #ifdef USE_SUREXO_CALL
        CallNode call;
        #endif
    };
    BitSet bs;
    BBuf* mbuf; /* multi-byte info or NULL */
} Node;

```

```

typedef long OnigStackIndex;
typedef struct _OnigStackType {
    unsigned int type;
    union {
        struct {
            UChar *pcode; /* byte code position */
            UChar *ptrs; /* string position */
        } t;
        UChar *pcode; /* byte code position */
        UChar *ptrs; /* string position */
    };
} OnigStackType;

```

```

typedef long OnigStackIndex;
typedef struct _OnigStackType {
    unsigned int type;
    union {
        struct {
            UChar *pcode; /* byte code position */
            UChar *ptrs; /* string position */
        } t;
        UChar *pcode; /* byte code position */
        UChar *ptrs; /* string position */
    };
} OnigStackType;

```

Notice how neither of the old versions define the `status` field in `NodeBase`. Both of the new versions define the `status` field (revision `3c57` is textually different by defining the `NodeBase` struct nested within the `Node` struct but this does not affect the compiled code) and rely on it during their invocations of `NODE_IS_NAME_REF`.

Recall that EUF initialises all structs inside of a harness using definitions from the old version of the dependency. This means that the `Node` object passed to the new versions should lack `status` and `body` fields. However, the equivalent result in `3b63 -> 3c57` (and the revised version of `388e -> 2350`) indicate that this did not cause issues for verification.

A partial explanation for this lies in the fact that the `Node` struct uses a `union` for all its fields. Since memory will be allocated based on the largest member in the `union`, the `Node` struct would have valid memory in the locations of `status` and `body` despite not having them explicitly defined.

This line of thinking implies that invalid memory accesses would occur if a field exceeding the maximum size of the original `Node` struct would be defined and used in the new version. At a glance, the change from `EncloseNode` to `EnclosureNode` seems like a possible candidate for this type of scenario.

```

typedef struct {
    NodeBase base;
    int state;
} EncloseNode;

int type;
int regnum;
OnigOptionType option;
struct _Node* target;
AbsAddrType call_addr;

/* for multiple call reference */
Oniglen min_len; /* min length (byte) */
Oniglen max_len; /* max length (byte) */
int char_len; /* character length */
int opt_count; /* referenced count in optimize_node_left() */
} EncloseNode;

/home/jonas/.cache/euf/oniguruma-388e/src/regparse.h  cpp 42% 152 3

```

```

typedef struct {
    NodeBase base_type;
    int status;
    struct _Node* body;
} EncloseNode;

int type;
union {
    struct {
        int regnum;
        AbsAddrType call_addr;
        int entry_count;
        int called_state;
    } a;
    struct {
        OnigOptionType option;
    } o;
};

/* for multiple call reference */
Oniglen min_len; /* min length (byte) */
Oniglen max_len; /* max length (byte) */
int char_len; /* character length */
int opt_count; /* referenced count in optimize_node_left() */
} EncloseNode;

/home/jonas/.cache/euf/oniguruma-2358/src/regparse.h  cpp 38% 155 3

```

```

int type;
int regnum;
OnigOptionType option;
struct _Node* target;
AbsAddrType call_addr;

/* for multiple call reference */
Oniglen min_len; /* min length (byte) */
Oniglen max_len; /* max length (byte) */
int char_len; /* character length */
int opt_count; /* referenced count in optimize_node_left() */
} EncloseNode;

/home/jonas/.cache/euf/oniguruma-3b63/src/regparse.h  cpp 35% 124 3

```

```

int type;
int regnum;
OnigOptionType option;
struct _Node* target;
AbsAddrType call_addr;

/* for multiple call reference */
Oniglen min_len; /* min length (byte) */
Oniglen max_len; /* max length (byte) */
int char_len; /* character length */
int opt_count; /* referenced count in optimize_node_left() */
} EncloseNode;

/home/jonas/.cache/euf/oniguruma-3c57/src/regparse.h  cpp 35% 124 3

```

- Equivalent verdict: Sound
- Influential verdict: Sound

subexp_recursive_check_trav()

```

SUCCESS_UNWIND_FAIL (1)
FAILURE_UNWIND_FAIL (1)

```

```

./scripts/analyze_function.sh onig subexp_recursive_check_trav 5b5b cb4f
git difftool --no-index ~/.cache/euf/oniguruma-5b5b/src/regcomp.c \
    ~/.cache/euf/oniguruma-cb4f/src/regcomp.c
# 2950:1 <-> 2929:1
# Thu Jun 1 13:16:09 2017 -> Mon Jun 12 11:30:51 2017
# => ~/.cache/euf/oniguruma-5b5b/.harnesses/subexp_recursive_check_trav.c (FAILURE_UNWIND_FAIL)
./scripts/analyze_function.sh onig subexp_recursive_check_trav 3b63 3c57
git difftool --no-index ~/.cache/euf/oniguruma-3b63/src/regcomp.c \
    ~/.cache/euf/oniguruma-3c57/src/regcomp.c
# 2950:1 <-> 2954:1
# Wed May 24 13:43:25 2017 -> Thu Jun 8 13:24:35 2017
# => ~/.cache/euf/oniguruma-3b63/.harnesses/subexp_recursive_check_trav.c (SUCCESS_UNWIND_FAIL)

```

```

static int
subexp_recursive_check_trav(Node* node, ScanEnv* env)
{
#define FOUND_CALLED_NODE 1

int type;
int r = 0;

type = NTTYPE(node);
switch (type) {
case NT_LIST:
case NT_ALT:
{
    int ret;
    do {
        ret = subexp_recursive_check_trav(NCAR(node), env);
        if (ret == FOUND_CALLED_NODE) r = FOUND_CALLED_NODE;
        else if (ret < 0) return ret;
    } while (IS_NOT_NULL(node = NCDR(node)));
    break;
}

case NT_QTFR:
r = subexp_recursive_check_trav(NQTR(node)->target, env);
if (NQTR(node)->upper == 0) {
    if (r == FOUND_CALLED_NODE)
        NQTR(node)->is_referred = 1;
}
break;

case NT_ANCHOR:
{
    AnchorNode* an = NANCHOR(node);
    switch (an->type) {
    case ANCHOR_PURE_READ:
    case ANCHOR_PREC_READ_NOT:
    case ANCHOR_LOOK_BEHIND:
    case ANCHOR_LOOK_BEHIND_NOT:
        r = subexp_recursive_check_trav(an->target, env);
        break;
    }
}

case NT_ENCLOSURE:
{
    EncloseNode* en = MENCLOSE(node);

    if (!IS_ENCLOSURE(node)) {
        if (IS_ENCLOSURE_CALLED(en)) {
            NODE_STATUS_SET(node, NST_MARK1);
            SET_ENCLOSURE_STATUS(node, NST_MARK1);
            r = subexp_recursive_check_trav(en->target, env);
            if (r != 0) SET_ENCLOSURE_STATUS(node, NST_RECursion);
            CLEAR_ENCLOSURE_STATUS(node, NST_MARK1);
        }
    }
    r = subexp_recursive_check_trav(en->target, env);
    if (r == FOUND_CALLED_NODE)
        r |= FOUND_CALLED_NODE;
}

break;
default:
break;
}
return r;
}

NORMAL /home/jonas/.cache/euf/oniguruma-5b5b/src/regcomp.c c 46% 2940%1

```



```

static int
subexp_recursive_check_trav(Node* node, ScanEnv* env)
{
#define FOUND_CALLED_NODE 1

int r = 0;

switch (NODE_TYPE(node)) {
case NODE_LIST:
case NODE_ALT:
{
    int ret;
    do {
        ret = subexp_recursive_check_trav(NODE_CAR(node), env);
        if (ret == FOUND_CALLED_NODE) r = FOUND_CALLED_NODE;
        else if (ret < 0) return ret;
    } while (IS_NOT_NULL(node = NODE_CDR(node)));
    break;
}

case NODE_QTFR:
r = subexp_recursive_check_trav(NODE_BODY(node), env);
if (NODE_BODY(node)->upper == 0) {
    if (r == FOUND_CALLED_NODE)
        NODE_BODY(node)->is_referred = 1;
}
break;

case NODE_ANCHOR:
{
    AnchorNode* an = NANCHOR(node);
    switch (an->type) {
    case ANCHOR_PURE_READ:
    case ANCHOR_PREC_READ_NOT:
    case ANCHOR_LOOK_BEHIND:
    case ANCHOR_LOOK_BEHIND_NOT:
        r = subexp_recursive_check_trav(an->target, env);
        break;
    }
}

case NODE_ENCLOSURE:
{
    EncloseNode* en = MENCLOSE(node);

    if (!IS_ENCLOSURE(node)) {
        if (IS_ENCLOSURE_CALLED(en)) {
            NODE_STATUS_SET(node, NST_MARK1);
            SET_ENCLOSURE_STATUS(node, NST_MARK1);
            r = subexp_recursive_check_trav(NODE_BODY(node), env);
            if (r != 0) SET_ENCLOSURE_STATUS(node, NST_RECursion);
            CLEAR_ENCLOSURE_STATUS(node, NST_MARK1);
        }
    }
    r = subexp_recursive_check_trav(NODE_BODY(node), env);
    if (r == FOUND_CALLED_NODE)
        r |= FOUND_CALLED_NODE;
}

break;
default:
break;
}
return r;
}

NORMAL /home/jonas/.cache/euf/oniguruma-cb4f/src/regcomp.c c 46% 2928%1

```



```

static int
subexp_recursive_check_trav(Node* node, ScanEnv* env)
{
#define FOUND_CALLED_NODE 1

int type;
int r = 0;

type = NTTYPE(node);
switch (type) {
case NT_LIST:
case NT_ALT:
{
    int ret;
    do {
        ret = subexp_recursive_check_trav(NCAR(node), env);
        if (ret == FOUND_CALLED_NODE) r = FOUND_CALLED_NODE;
        else if (ret < 0) return ret;
    } while (IS_NOT_NULL(node = NCDR(node)));
    break;
}

case NT_QTFR:
r = subexp_recursive_check_trav(NQTR(node)->target, env);
if (NQTR(node)->upper == 0) {
    if (r == FOUND_CALLED_NODE)
        NQTR(node)->is_referred = 1;
}
break;

case NT_ANCHOR:
{
    AnchorNode* an = NANCHOR(node);
    switch (an->type) {
    case ANCHOR_PURE_READ:
    case ANCHOR_PREC_READ_NOT:
    case ANCHOR_LOOK_BEHIND:
    case ANCHOR_LOOK_BEHIND_NOT:
        r = subexp_recursive_check_trav(NODE_BODY(node), env);
        break;
    }
}

case NT_ENCLOSURE:
{
    EncloseNode* en = MENCLOSE(node);

    if (!IS_ENCLOSURE(node)) {
        if (IS_ENCLOSURE_CALLED(en)) {
            NODE_STATUS_SET(node, NST_MARK1);
            SET_ENCLOSURE_STATUS(node, NST_MARK1);
            r = subexp_recursive_check_trav(NODE_BODY(node), env);
            if (r != 0) SET_ENCLOSURE_STATUS(node, NST_RECursion);
            CLEAR_ENCLOSURE_STATUS(node, NST_MARK1);
        }
    }
    r = subexp_recursive_check_trav(NODE_BODY(node), env);
    if (r == FOUND_CALLED_NODE)
        r |= FOUND_CALLED_NODE;
}

break;
default:
break;
}
return r;
}

NORMAL /home/jonas/.cache/euf/oniguruma-3c57/src/regcomp.c c 46% 2940%2

```

Both changes originate from the same base version. The influential case presents a counterexample where the old version returns `r=0` while the new version returns `r=1`.

The `r` variable is assigned with recursive output from the function and with 1 (through logical OR with `FOUND_CALL_NODE=1`). Both updates modify the two lines preceding `r |= FOUND_CALLED_NODE;` in the same manner.

Note that revision `cb4f` is the only one that modifies the names of constants in each `case`. One would assume that `NT_ENCLOSURE` and `NODE_ENCLOSURE` have the same value but the `cb4f` update has actually set `NODE_ENCLOSURE = 5`, violating this assumption since `NT_ENCLOSURE=4`.

With this in mind, it is fairly clear why the verification fails. Given the same `Node` object, a type of 5 will pick the `ENCLOSURE` case in the new version and instead of `NT_ENCLOSURE` the old version will pick the `NT_QTFR = 5` branch. The old branch does not contain the `r |= FOUND_CALLED_NODE;` statement and a failed verification is attained.

Modifying the old `case` statements to be inline with the new values for each symbol yields a successful verification, highlighting this as the root cause for the failed verification.

```
./c/a/s/segcomp -r ./c/e/o/s/segcomp.c

static int
subexp_recursive_check_trav(Node* node, ScanEnv* env)
{
#define FOUND_CALLED_NODE 1

    int type;
    int r = 0;

    type = NTYPE(node);
    switch (type) {
    case 7:
    case 8:
        {
            int ret;
            do {
                ret = subexp_recursive_check_trav(NEWC(node), env);
                if (ret == FOUND_CALLED_NODE) r = FOUND_CALLED_NODE;
                else if (ret < 0) return ret;
            } while (!IS_NOT_NULL(node = NCDR(node)));
            break;
        }
    case 4:
        r = subexp_recursive_check_trav(NOTR(node)->target, env);
        if (!NTR(node)) r = -1;
        if ((NTR(node))->refered == 0)
            NOTR(node)->is_referred = 1;
        break;
    case 6:
        {
            AnchorNode* an = NANCHOR(node);
            switch (an->type) {
            case ANCHOR_PREC_READ:
            case ANCHOR_PREC_READ_NOT:
            case ANCHOR_LOOK_BEHIND:
            case ANCHOR_LOOK_BEHIND_NOT:
                r = subexp_recursive_check_trav(an->target, env);
                break;
            }
        }
        break;
    case 5:
        {
            EncloseNode* en = NENCLOSE(node);

            if (!IS_ENCLOSE_RECURSION(en)) {
                if (!IS_ENCLOSE_CALLED(en)) {
                    SET_ENCLOSE_STATUS(node, NST_MARKS);
                    r = subexp_recursive_check(en->target);
                    if (r != 0) SET_ENCLOSE_STATUS(node, NST_RECURSION);
                    CLEAR_ENCLOSE_STATUS(node, NST_MARKS);
                }
            }
            r = subexp_recursive_check_trav(en->target, env);
            if (!IS_ENCLOSE_CALLED(en))
                r |= FOUND_CALLED_NODE;
        }
        break;
    default:
        break;
    }
    return r;
}

static int
subexp_recursive_check_trav(Node* node, ScanEnv* env)
{
#define FOUND_CALLED_NODE 1

    int r = 0;

    switch (NODE_TYPE(node)) {
    case NODE_LIST: // 7
    case NODE_ALT: // 8
        {
            int ret;
            do {
                ret = subexp_recursive_check_trav(NODE_CAR(node), env);
                if (ret == FOUND_CALLED_NODE) r = FOUND_CALLED_NODE;
                else if (ret < 0) return ret;
            } while (IS_NOT_NULL(node = NODE_CDR(node)));
            break;
        }
    case NODE_QTR: // 4
        r = subexp_recursive_check_trav(NODE_BODY(node), env);
        if (QTR(node)->refered == 0)
            if ((QTR(node))->refered == 1)
                QTR(node)->is_referred = 1;
        break;
    case NODE_ANCHOR: // 6
        {
            AnchorNode* an = ANCHOR(node);
            if (ANCHOR_HAS_BODY(an))
                r = subexp_recursive_check_trav(NODE_ANCHOR_BODY(an), env);
        }
        break;
    case NODE_ENCLOSURE: // 5
        if (!NODE_IS_RECURSION(node)) {
            if (!NODE_IS_CALLED(node)) {
                NODE_STATUS_ADD(node, NST_MARKS);
                if (r != 0) SET_ENCLOSE_STATUS(node, NST_RECURSION);
                NODE_STATUS_REMOVE(node, NST_MARKS);
            }
        }
        r = subexp_recursive_check_trav(en->target, env);
        if (!NODE_IS_CALLED(node))
            r |= FOUND_CALLED_NODE;
    }
    break;
    default:
        break;
    }
    return r;
}

#endif /* _REPOS_EUF */


```

- Equivalent verdict: Sound
 - Influential verdict: Sound

onig_get_start_of_callout_args()

FAILURE (4)
SUCCESS (1)

```
./scripts/analyze_function.sh onig onig_get_start_of_callout_args 0b5512c 23b2  
git difftool --no-index ~/.cache/euf/oniguruma-0b5512c/src/regexec.c \  
~/cache/euf/oniguruma-23b2/src/regexec.c  
# 4891:1 <-> 5104:1  
# Tue Feb 13 13:26:46 2018 -> Mon Feb 26 11:04:07 2018  
# => ~/.cache/euf/oniguruma-0b55/.harnesses/onig_get_start_of_callout_args.c (FAILURE)
```

```
./scripts/analyze_function.sh onig onig_get_start_of_callout_args 32c2 5f2b  
git difftool --no-index ~/.cache/euf/oniguruma-32c2/src/regexec.c \
```

```
~/.cache/euf/oniguruma-5f2b/src/regexec.c  
# 4925:1 <-> 4939:1
```

```
# Tue Feb 13 14:02:35 2018 -> Wed Feb 21 17:36:04 2018
# => ./cache/euf/oniguruma-32c2/.harnesses/onig_get_start_of_callout_args.c (SUCCESS)
```

```
# Comparing the influential (_right_ as it is newer) and equivalent new states
./scripts/analyze_function.sh onig onig_get_start_of_callout_args 5f2b 23b2
git difftool --no-index ~./cache/euf/oniguruma-5f2b/src/regexec.c \
~./cache/euf/oniguruma-23b2/src/regexec.c
```

```
extern const UChar* onig_get_start_of_callout_args(OnigCalloutArgs* args)
{
    CalloutArgs* a = (CalloutArgs*)args;
    return a->start;
}

extern const UChar* onig_get_current_of_callout_args(OnigCalloutArgs* args)
{
    CalloutArgs* a = (CalloutArgs*)args;
    return a->current;
}

extern OnigRegex* onig_get_regex_of_callout_args(OnigCalloutArgs* args)
{
    CalloutArgs* a = (CalloutArgs*)args;
    return a->regex;
}

extern unsigned long onig_get_retry_counter_of_callout_args(OnigCalloutArgs* args)
{
    return args->retry_in_match_counter;
}
```

... (repeated for other functions like onig_get_right_range_of_callout_args, onig_get_current_of_callout_args, onig_get_regex_of_callout_args, and onig_get_retry_counter_of_callout_args)

The influential and equivalent cases are textually identical in this scenario and thus also originate from the same base state. The only change is the removal of a pointer cast from `OnigCalloutArgs` to `CalloutArgs`. In both of the new revisions, the definition of `CalloutArgs` has been replaced with a struct definition for `OnigCalloutArgsStruct` (which `OnigCalloutArgs` is a `typedef` for). The influential update differs in that it has removed the `content` and `content_end` fields while adding a new field, `match` in this struct.

```
typedef struct {
    enum OnigCalloutIn in;
    enum OnigCalloutOf of;
    int num_id; /* name id or ONIG_NO_NAME_ID */
    const OnigUChar* content;
    const OnigUChar* content_end;
    OnigRegex* regex;
    const OnigUChar* subject;
    const OnigUChar* subject_end;
    const OnigUChar* start;
    const OnigUChar* right_range;
    const OnigUChar* current; // current matching position
    unsigned long retry_in_match_counter;
} /* invisible to users */

StackType* stk_base;
StackType* stk_top;
StackIndex* mem_start_stk;
StackIndex* mem_end_stk;
} CalloutArgs;

#endif
```

... (repeated for other functions like OnigCalloutArgsStruct, OnigCalloutIn, OnigCalloutOf, and OnigCalloutArgs)

Since the `start` field is placed after the new content fields, it will be accessed at a different offset in `0b55` compared to `23b2`. The internal format of `OnigCalloutArgs` (i.e. the format in each version of `libonig`) will dictate what offsets are used to retrieve data. The inconsistent return value thus arises from the new version attempting to use the indices of the new version of `OnigCalloutArgs` upon an object initialised using the old format. The fact that a successful verification is attained when patching the new version with the old fields supports this line of reasoning.

While the reason behind the FAILURE case can be soundly explained, it constitutes a FP from a practical point of view since the function itself has not changed in an influential way. Had the new version been given a `OnigCalloutArgs` struct matching its own internal representation, the behaviour would remain the same. The current design of EUF does not have an interface which supports several definitions of the same struct while also keeping all overlapping fields equal (through assumptions), something akin to this would be required to resolve FPs of this type.

- **Equivalent verdict:** Sound
 - **Influential verdict:** Unsound

`onig_get_current_of_callout_args()` and `onig_get_regex_of_callout_args()`

FAILURE (4+4)
SUCCESS (1+1)

```
./scripts/analyze_function.sh onig onig_get_current_of_callout_args 0b5512c 23b2
./scripts/analyze_function.sh onig_get_regex_of_callout_args 0b5512c 23b2
git difftool --no-index ~/.cache/euf/oniguruma-0b55/src/regexec.c ~/.cache/euf/oniguruma-23b2/src/regexec.c
# 4898:1 <-> 5116:1
# Tue Feb 13 13:26:46 2018 -> Mon Feb 26 11:04:07 2018
# => ~/.cache/euf/oniguruma-0b55/.harnesses/onig_get_current_of_callout_args.c (FAILURE)
```

```

# => ~/.cache/euf/oniguruma-0b55/.harnesses/onig_get_regex_of_callout_args.c (FAILURE)

./scripts/analyze_function.sh onig onig_get_current_of_callout_args 32c2 5f2b
./scripts/analyze_function.sh onig_get_regex_of_callout_args 32c2 5f2b
git difftool --no-index ~/.cache/euf/oniguruma-32c2/src/regexec.c ~/.cache/euf/oniguruma-5f2b/src/regexec.c
# 4932:1 <-> 4951:1
# Tue Feb 13 14:02:35 2018 -> Wed Feb 21 17:36:04 2018
# => ~/.cache/euf/oniguruma-32c2/.harnesses/onig_get_current_of_callout_args.c (SUCCESS)
# => ~/.cache/euf/oniguruma-32c2/.harnesses/onig_get_regex_of_callout_args.c (SUCCESS)

```

The same analysis given for `onig_get_start_of_callout_args()` applies for both updates to these functions.

```

)
extern const UChar*
onig_get_current_of_callout_args(OnigCalloutArgs* args)
{
    CalloutArgs* a = (CalloutArgs*)args;
    return a->current;
}

extern OnigRegex
onig_get_regex_of_callout_args(OnigCalloutArgs* args)
{
    CalloutArgs* a = (CalloutArgs*)args;
    return a->regex;
}

extern unsigned long
onig_get_retry_counter_of_callout_args(OnigCalloutArgs* args)
{
    return args->retry_in_match_counter;
}

extern int
onig_get_content_of_callout_args(OnigCalloutArgs* args)
{
    return args->content;
}

```

File paths: /home/jonas/.cache/euf/oniguruma-0b5512c/src/regexec.c [1], /home/jonas/.cache/euf/oniguruma-23b2/src/regexec.c [1], /home/jonas/.cache/euf/oniguruma-32c2/src/regexec.c [1], /home/jonas/.cache/euf/oniguruma-5f2b/src/regexec.c [1]

- Equivalent verdict: Sound
- Influential verdict: Unsound

`onig_get_content_of_callout_args()` and `onig_get_content_end_of_callout_args()`

FAILURE (1+1)
SUCCESS (1+1)

```

./scripts/analyze_function.sh onig onig_get_content_of_callout_args b1cf 61ca9c7
./scripts/analyze_function.sh onig onig_get_content_end_of_callout_args b1cf 61ca9c7
git difftool --no-index ~/.cache/euf/oniguruma-b1cf/src/regexec.c \
    ~/.cache/euf/oniguruma-61ca9c7/src/regexec.c
# 4928:1 <-> 4903:1
# Thu Feb 15 16:34:51 2018 -> Thu Feb 22 13:57:46 2018
# => ~/.cache/euf/oniguruma-b1cf/.harnesses/onig_get_content_of_callout_args.c (FAILURE)

./scripts/analyze_function.sh onig onig_get_content_of_callout_args 32c2 5f2b
./scripts/analyze_function.sh onig onig_get_content_end_of_callout_args 32c2 5f2b
git difftool --no-index ~/.cache/euf/oniguruma-32c2/src/regexec.c \
    ~/.cache/euf/oniguruma-5f2b/src/regexec.c
# 4911:1 <-> 4927:1
# Tue Feb 13 14:02:35 2018 -> Wed Feb 21 17:36:04 2018
# => ~/.cache/euf/oniguruma-32c2/.harnesses/onig_get_content_of_callout_args.c (SUCCESS)

```

```

extern const UChar*
onig_get_content_of_callout_args(OnigCalloutArgs* args)
{
    return args->content;
}

extern const UChar*
onig_get_content_and_of_callout_args(OnigCalloutArgs* args)
{
    return args->content_end;
}

/home/jonas/.cache/euf/oniguruma-b1cf/src/regexec.c  c  89%  

:4928

```

```

extern const UChar*
onig_get_content_of_callout_args(OnigCalloutArgs* args)
{
    int num;
    CalloutListEntry* e;

    num = args->num;
    e = onig_reg_callout_list_at(args->regex, num);
    if (IS_NUL(e)) return 0;
    if (e->of == ONIG_CALLOUT_OF_NAME) return 0;

    return e->u.content.start;
}

extern const UChar*
onig_get_content_and_of_callout_args(OnigCalloutArgs* args)
{
    int num;
    CalloutListEntry* e;

    num = args->num;
    e = onig_reg_callout_list_at(args->regex, num);
    if (IS_NUL(e)) return 0;
    if (e->of == ONIG_CALLOUT_OF_NAME) return 0;

    return e->u.content.end;
}

/home/jonas/.cache/euf/oniguruma-61ca9c7/src/regexec.c  c  88%  

:4525

```

```

extern enum OnigCalloutOf
onig_get_callout_of_of_callout_args(OnigCalloutArgs* args)
{
    CalloutArgs* a = (CalloutArgs*) args;
    return a->of;
}

extern int
onig_get_name_id_of_callout_args(OnigCalloutArgs* args)
{
    CalloutArgs* a = (CalloutArgs*) args;
    return a->name_id;
}

extern const UChar*
onig_get_content_of_callout_args(OnigCalloutArgs* args)
{
    int num;
    CalloutArgs* a = (CalloutArgs*) args;
    return a->content;
}

extern const UChar*
onig_get_content_and_of_callout_args(OnigCalloutArgs* args)
{
    int num;
    CalloutArgs* a = (CalloutArgs*) args;
    return a->content_end;
}

/home/jonas/.cache/euf/oniguruma-32c2/src/regexec.c  c  90%  

:4911

```

```

extern OnigCalloutOf
onig_get_callout_of_of_callout_args(OnigCalloutArgs* args)
{
    return args->of;
}

extern int
onig_get_name_id_of_callout_args(OnigCalloutArgs* args)
{
    return args->name_id;
}

extern const UChar*
onig_get_content_of_callout_args(OnigCalloutArgs* args)
{
    return args->content;
}

extern const UChar*
onig_get_content_and_of_callout_args(OnigCalloutArgs* args)
{
    return args->content_end;
}

/home/jonas/.cache/euf/oniguruma-5f2b/src/regexec.c  c  90%  

:4621

```

The equivalent updates for both functions follow the same format as that described for `onig_get_start_of_callout_args()`. The new equivalent revision, *5f2b* and the old influential revision, *b1cf*, share the same structure of `OnigCalloutArgsStruct` and the functions in question.

Both influential changes return a field with a similar name and presumably the same internal purpose, either `content.start` or `content.end`, from a `CalloutListEntry` object which is derived from the passed argument (`OnigCalloutArgs`). The `CalloutListEntry` is identically defined in both new revisions but is only utilised in *61ca*.

The influential update has removed the `content` and `content_end` fields from its `OnigCalloutArgs` struct and while input/output behaviour could feasibly have been maintained from an architectural point of view, proving this with CBMC would as described in the analysis of `onig_get_start_of_callout_args()` require initialisation of two different types of `OnigCalloutArgs` variables with an assumed equality in every shared field.

```

#ifndef USE_CALLOUT
struct OnigCalloutArgsStruct {
    enum OnigCalloutIn;
    enum OnigCalloutOf;
    int name_id; /* name id or ONIG_NO_NAME_ID */
    int num;
    const OnigUChar* content;
    const OnigUChar* content_end;
    OnigRegex* regex;
    const OnigUChar* subject;
    const OnigUChar* subject_end;
    const OnigUChar* start;
    const OnigUChar* right_range;
    const OnigUChar* current; // current matching position
    unsigned long retry_in_match_counter;
};

/* invisible to users */
StackType* stk_base;
StackType* stk;
StackIndex* mem_start_stk;
StackIndex* mem_end_stk;
};

#endif

```

```

#ifndef USE_CALLOUT
struct OnigCalloutArgsStruct {
    OnigCalloutIn;
    OnigCalloutOf;
    int name_id; /* name id or ONIG_NO_NAME_ID */
    int num;
    const OnigUChar* content;
    const OnigUChar* content_end;
    OnigRegex* regex;
    const OnigUChar* subject;
    const OnigUChar* subject_end;
    const OnigUChar* start;
    const OnigUChar* right_range;
    const OnigUChar* current; // current matching position
    unsigned long retry_in_match_counter;
};

/* invisible to users */
StackType* stk_base;
StackType* stk;
StackIndex* mem_start_stk;
StackIndex* mem_end_stk;
};

#endif

```

```

#ifndef USE_CALLOUT
struct OnigCalloutArgsStruct {
    OnigCalloutIn;
    OnigCalloutOf;
    int name_id; /* name id or ONIG_NO_NAME_ID */
    int num;
    const OnigUChar* content;
    const OnigUChar* content_end;
    OnigRegex* regex;
    const OnigUChar* subject;
    const OnigUChar* subject_end;
    const OnigUChar* start;
    const OnigUChar* right_range;
    const OnigUChar* current; // current matching position
    unsigned long retry_in_match_counter;
};

/* invisible to users */
StackType* stk_base;
StackType* stk;
StackIndex* mem_start_stk;
StackIndex* mem_end_stk;
};

#endif

```

```

#ifndef USE_CALLOUT
typedef enum {
    CALLOUT_TYPE_SINGLE      = 0,
    CALLOUT_TYPE_START_CALL  = 1,
    CALLOUT_TYPE_BOTH_CALL   = 2,
    CALLOUT_TYPE_START_MARK_END_CALL = 3
} CalloutType;

```

```

typedef struct {
    int Flag;
    OnigCalloutOf of;
    int in;
    int name_id;
    CalloutType type;
    union {
        struct {
            UChar start;
            UChar end;
        } content;
        struct {
            int max_num;
            int passed_num;
            OnigType types[ONIG_CALLOUT_MAX_ARG_NUM];
            OnigValue vals[ONIG_CALLOUT_MAX_ARG_NUM];
        } arg;
    } u;
} CalloutListEntry;
#endif

#ifndef USE_CALLOUT
typedef enum {
    CALLOUT_TYPE_SINGLE      = 0,
    CALLOUT_TYPE_START_CALL  = 1,
    CALLOUT_TYPE_BOTH_CALL   = 2,
    CALLOUT_TYPE_START_MARK_END_CALL = 3,
} CalloutType;

```

```

typedef struct {
    int Flag;
    OnigCalloutOf of;
    int in;
    int name_id;
    CalloutType type;
    union {
        struct {
            int start;
            int end;
        } content;
        struct {
            int max_num;
            int passed_num;
            OnigType types[ONIG_CALLOUT_MAX_ARG_NUM];
            OnigValue vals[ONIG_CALLOUT_MAX_ARG_NUM];
        } arg;
    } u;
} CalloutListEntry;
#endif

```

Restoring `OnigCalloutArgsStruct` to its original state (as done for `onig_get_start_of_callout_args()`) does not yield a successful verification in this scenario since the return value is also dependent on the new struct, `CalloutListEntry`.

```

args.name_id  = (name_id);\nargs.num      = num;\nargs.content  = cstart;\nargs.content_end = cend;\nargs.regex    = reg;\nargs.subject  = str;\nargs.subject_end = end;\nargs.start    = sstart;\nargs.right_range = right_range;\nargs.current  = $;\n+-- 2 times: args.retry_in_match_counter = retry_in_match_counter;\nargs.mem_start_stk = mem_start_stk;\nargs.mem_end_stk = mem_end_stk;\nresult = (func)(args, user);\n} while (0)

#define RETRACTION_CALLOUT(func, aof, name_id, num, cstart, cend,\nint result,\nOnigCalloutArgs args)\nCALLOUT_BODY(func, ONIG_CALLOUT_IN RETRACTION, aof, name_id, num\nswitch (result) {\ncase ONIG_CALLOUT_FAIL:\ngoto fail;\nbreak;\ncase ONIG_CALLOUT_SUCCESS:\nbreak;\n+-- 95 lines: case ONIG_CALLOUT_ABORT:\n} u;\n} StackType;

#ifndef USE_CALLOUT\n
struct OnigCalloutArgsStruct {\n    enum OnigCalloutIn in;\n    enum OnigCalloutOf of;\n    int name_id; /* name id or ONIG_NO_NAME_ID */\n    int num;\n    const OnigUChar* content;\n    const OnigUChar* subject;\n    const OnigUChar* subject_end;\n    OnigRegex regex;\n    const OnigUChar* subject;\n+-- 493 lines: const OnigUChar* subject_end;\n    else if (k->type == STK_RETURN)\n        level++;\n    k--;\n}\n} while (0)

#define STACK_PUSH_CALLOUT_CODE(enum, xcontent, xcontent_end) do {\nSTACK_ENSURE(1);\nstk->type = STK_CALLOUT;\nstk->zid = ONIG_NO_NAME_ID;\nstk->u.callout.num = (enum);\nstk->u.callout.content = (xcontent);\nstk->u.callout.content_end = (xcontent_end);\nSTACK_INC;\n} while (0)

#define STACK_PUSH_CALLOUT_NAME(aiid, anum, xcontent, xcontent_end) do {\nSTACK_ENSURE(1);\nstk->type = STK_CALLOUT;\nstk->zid = (aiid);\nstk->u.callout.num = (anum);\nstk->u.callout.content = (xcontent);\nstk->u.callout.content_end = (xcontent_end);\nSTACK_INC;\n} while (0)

#define STACK_PUSH_CALLOUT_NAME(aiid, anum) do {\nSTACK_ENSURE(1);\nstk->type = STK_CALLOUT;\nstk->zid = (aiid);\nstk->u.callout.num = (anum);\n\nSTACK_INC;\n} while (0)

#define STACK_PUSH_CALLOUT_CODE(aiid, anum, xcontent, xcontent_end) do {\nSTACK_ENSURE(1);\nstk->type = STK_CALLOUT;\nstk->zid = (aiid);\nstk->u.callout.num = (anum);\nstk->u.callout.content = (xcontent);\nstk->u.callout.content_end = (xcontent_end);\nSTACK_INC;\n} while (0)

```

```

args.name_id  = (name_id);\nargs.num      = num;\nargs.content  = cstart;\nargs.content_end = cend;\nargs.regex    = reg;\nargs.subject  = str;\nargs.subject_end = end;\nargs.start    = sstart;\nargs.right_range = right_range;\nargs.current  = $;\n+-- 2 times: args.retry_in_match_counter = retry_in_match_counter;\nargs.mem_start_stk = mem_start_stk;\nargs.mem_end_stk = mem_end_stk;\nresult = (func)(args, user);\n} while (0)

#define RETRACTION_CALLOUT(func, aof, name_id, num, user) do {\nint result;\nOnigCalloutArgs args;\nCALLOUT_BODY(func, ONIG_CALLOUT_IN RETRACTION, aof, name_id, num\nswitch (result) {\ncase ONIG_CALLOUT_FAIL:\ngoto fail;\nbreak;\ncase ONIG_CALLOUT_SUCCESS:\nbreak;\n+-- 95 lines: case ONIG_CALLOUT_ABORT:\n} u;\n} StackType;

#ifndef USE_CALLOUT\n
struct OnigCalloutArgsStruct {\n    OnigCalloutIn in;\n    OnigCalloutOf of;\n    int name_id; /* name id or ONIG_NO_NAME_ID */\n    int num;\n    const OnigUChar* content;\n    const OnigUChar* subject;\n    const OnigUChar* subject_end;\n    OnigRegex regex;\n    const OnigUChar* subject;\n+-- 493 lines: const OnigUChar* subject_end;\n    else if (k->type == STK_RETURN)\n        level++;\n    k--;\n}\n} while (0)

#define STACK_PUSH_CALLOUT_CODE(enum) do {\nSTACK_ENSURE(1);\nstk->type = STK_CALLOUT;\nstk->zid = ONIG_NO_NAME_ID;\nstk->u.callout.num = (enum);\n\nSTACK_INC;\n} while (0)

#define STACK_PUSH_CALLOUT_NAME(aiid, anum) do {\nSTACK_ENSURE(1);\nstk->type = STK_CALLOUT;\nstk->zid = (aiid);\nstk->u.callout.num = (anum);\n\nSTACK_INC;\n} while (0)

#define STACK_PUSH_CALLOUT_CODE(aiid, anum, xcontent, xcontent_end) do {\nSTACK_ENSURE(1);\nstk->type = STK_CALLOUT;\nstk->zid = (aiid);\nstk->u.callout.num = (anum);\nstk->u.callout.content = (xcontent);\nstk->u.callout.content_end = (xcontent_end);\n\nSTACK_INC;\n} while (0)

```

Passing problem to propositional reduction
converting SSA
Runtime Convert SSA: 0.2296s
Running propositional reduction
Post-processing
Runtime Post-process: 3.4515e-05s
Solving with MiniSAT 2.2.1 with simplifier
4564 variables, 5279 clauses
SAT checker: instance is SATISFIABLE
Runtime Solver: 0.00261285s
Runtime decision procedure: 0.232952s
Running decision procedure: 0.232952s
Building proof trace
** Results:
/home/jonas/.cache/euf/oniguruma-b1cf/.harnesses/onig_get_content_of_callout_args.c function euf_main [euf_main.assertion.1] line 41 Equivalent output: FAILURE
Trace for euf_main.assertion.1:
Violated property:
file /home/jonas/.cache/euf/oniguruma-b1cf/.harnesses/onig_get_content_of_callout_args.c function euf_main line 41 thread 0
Equivalent output:
ret.old == ret
ret.old == ret
** 1 of 1 failed (2 iterations)
VERIFICATION FAILED
real 0m.058s
user 0m.951s
sys 0m.045s
[] Rejected equivalence assertion: onig_get_content_of_callout_args: 0:00:11.643538
[] Change set reduction: 24 -> 24: 0:00:33.382810
[] Total runtime: 0:00:50.3875489
[] make clean
[] ./configure
[] make -j 15
[] make clean
[] ./configure
[] make -j 15
==> TRACE <= State 29104 file /home/jonas/.cache/euf/oniguruma-b1cf/.harnesses/onig_get_content_of_callout_args.c function euf_main line 38 thread 0
ret.old==((const OnigUChar *)NULL) (?)
State 29112 file /home/jonas/.cache/euf/oniguruma-b1cf/.harnesses/onig_get_content_of_callout_args.c function euf_main line 38 thread 0
ret.old==((const OnigUChar *)NULL) + 1099511627776L (?)
State 29113 file /home/jonas/.cache/euf/oniguruma-b1cf/.harnesses/onig_get_content_of_callout_args.c function euf_main line 39 thread 0
ret==((const OnigUChar *)NULL) (?)
State 29140 file /home/jonas/.cache/euf/oniguruma-b1cf/.harnesses/onig_get_content_of_callout_args.c function euf_main line 39 thread 0
ret==((const OnigUChar *)NULL) (?)
State 29067 file /home/jonas/.cache/euf/oniguruma-b1cf/.harnesses/onig_get_content_of_callout_args.c function euf_main line 34 thread 0
args==((OnigCalloutArgs *)NULL) (?)
State 29188 file /home/jonas/.cache/euf/oniguruma-b1cf/.harnesses/onig_get_content_of_callout_args.c function euf_main line 38 thread 0
args==((OnigCalloutArgs *)NULL) (?)
State 29117 file /home/jonas/.cache/euf/oniguruma-b1cf/.harnesses/onig_get_content_of_callout_args.c function euf_main line 39 thread 0
args==((OnigCalloutArgs *)NULL) (?)
State 29119 file regexec.c function onig_get_content_of_callout_args line 4908 thread 0
e==((CalloutListEntry *)NULL) (?)
State 29134 file regexec.c function onig_get_content_of_callout_args line 4911 thread 0
e==((CalloutListEntry *)NULL) + 1L (?)
State 29079 file /home/jonas/.cache/euf/oniguruma-b1cf/.harnesses/onig_get_content_of_callout_args.c function euf_main line 35 thread 0
return_value.nondet_OnigCalloutArgsStruct.start==((const OnigUChar *)NULL) (?)
State 29096 file /home/jonas/.cache/euf/oniguruma-b1cf/.harnesses/onig_get_content_of_callout_args.c function euf_main line 35 thread 0
args\$object.start==((const OnigUChar *)NULL) (?)
~>pos\$euf [main]
vel=>

- **Equivalent verdict:** Sound
- **Influential verdict:** Unsound

libusb

```
parse_endpoint()
  SUCCESS_UNWIND_FAIL (4)
  FAILURE_UNWIND_FAIL (13)

./scripts/analyze_function.sh usb parse_endpoint 5c89 42f9
git difftool --no-index ~/.cache/euf/libusb-5c89/libusb(descriptor.c \
  ~/.cache/euf/libusb-42f9/libusb(descriptor.c
# 83:12 <-> 83:12
# Tue Jun 15 22:33:45 2021 -> Wed Jul 21 11:47:25 2021
# => ~/.cache/euf/libusb-5c89/.harnesses/parse_endpoint.c (SUCCESS_UNWIND_FAIL)

./scripts/analyze_function.sh usb parse_endpoint ba6b d424
git difftool --no-index ~/.cache/euf/libusb-ba6b/libusb(descriptor.c \
  ~/.cache/euf/libusb-d424/libusb(descriptor.c
# 83:12 <-> 101:12
# Mon Aug 10 19:19:21 2020 -> Mon Aug 17 14:15:10 2020
# => ~/.cache/euf/libusb-ba6b/.harnesses/parse_endpoint.c (FAILURE_UNWIND_FAIL)
```

The equivalent update originates from a modification to the `usbi_dbg` prototype and does not affect the return value in any way, the classification is thus considered sound.

```

static int parse_endpoint(struct libusb_context *ctx,
    struct libusb_endpoint_descriptor *endpoint, const uint8_t *buffer, int size)
{
    const struct usb1_descriptor_header *header;
    const uint8_t *begin;
    void *extra;
    int parsed = 0;
    int len;

    if (size < DESC_HEADER_LENGTH) {
        usbi_err(ctx, "short endpoint descriptor read %d/%d",
            size, DESC_HEADER_LENGTH);
        return LIBUSB_ERROR_IO;
    }

    header = (const struct usb1_descriptor_header *)buffer;
    if (header->bDescriptorType != LIBUSB_DT_ENDPOINT) {
        usbi_err(ctx, "unexpected descriptor 0x%02x (expected 0x%02x)",
            header->bDescriptorType, LIBUSB_DT_ENDPOINT);
        return parsed;
    }

    else if (header->bLength < LIBUSB_DT_ENDPOINT_SIZE) {
        usbi_err(ctx, "invalid endpoint bLength (%u)", header->bLength);
        return LIBUSB_ERROR_IO;
    }

    else if (header->bLength > size) {
        usbi_warn(ctx, "short endpoint descriptor read %d/%u",
            size, header->bLength);
        return parsed;
    }

    if (header->bLength > LIBUSB_DT_ENDPOINT_AUDIO_SIZE)
        parse_descriptor(buffer, "bbbbbb", endpoint);
    else
        parse_descriptor(buffer, "bbbbwb", endpoint);

    buffer += header->bLength;
    size -= header->bLength;
    parsed += header->bLength;

    /* Skip over the rest of the Class Specific or Vendor Specific */
    /* descriptors */
    begin = buffer;
    while (size >= DESC_HEADER_LENGTH) {
        header = (const struct usb1_descriptor_header *)buffer;
        if (header->bDescriptorType == DESC_HEADER_LENGTH) {
            usbi_err(ctx, "invalid extra ep desc len (%u)",
                header->bLength);
            return LIBUSB_ERROR_IO;
        } else if (header->bLength > size) {
            usbi_warn(ctx, "short extra ep desc read %d/%u",
                size, header->bLength);
            return parsed;
        }

        /* If we find another "proper" descriptor then we're done */
        if (header->bDescriptorType == LIBUSB_DT_INTERFACE ||
            header->bDescriptorType == LIBUSB_DT_CONFIG ||
            header->bDescriptorType == LIBUSB_DT_DEVICE)
            break;
    }

    usbi_dbg(ctx, "skipping descriptor 0x%02x", header->bDescriptorType);
    buffer += header->bLength;
    size -= header->bLength;
    parsed += header->bLength;
}

NORMAL /home/jonas/.cache/euf/libusb-5c89/libusbdescriptor.c c 7%|82%1 /home/jonas/.cache/euf/libusb-42f9/libusbdescriptor.c c 7%|82%1

static int parse_endpoint(struct libusb_context *ctx,
    struct libusb_endpoint_descriptor *endpoint, const uint8_t *buffer, int size)
{
    const struct usb1_descriptor_header *header;
    const uint8_t *begin;
    void *extra;
    int parsed = 0;
    int len;

    if (size < DESC_HEADER_LENGTH) {
        usbi_err(ctx, "short endpoint descriptor read %d/%d",
            size, DESC_HEADER_LENGTH);
        return LIBUSB_ERROR_IO;
    }

    header = (const struct usb1_descriptor_header *)buffer;
    if (header->bDescriptorType != LIBUSB_DT_ENDPOINT) {
        usbi_err(ctx, "unexpected descriptor 0x%02x (expected 0x%02x)",
            header->bDescriptorType, LIBUSB_DT_ENDPOINT);
        return LIBUSB_ERROR_IO;
    }

    if (header->bLength > LIBUSB_DT_ENDPOINT_SIZE) {
        usbi_err(ctx, "invalid endpoint bLength (%u)", header->bLength);
        return LIBUSB_ERROR_IO;
    }

    else if (header->bLength > size) {
        usbi_warn(ctx, "short endpoint descriptor read %d/%u",
            size, header->bLength);
        return parsed;
    }

    if (header->bLength >= LIBUSB_DT_ENDPOINT_AUDIO_SIZE)
        parse_descriptor(buffer, "bbbbbb", endpoint);
    else
        parse_descriptor(buffer, "bbbbwb", endpoint);

    buffer += header->bLength;
    size -= header->bLength;
    parsed += header->bLength;

    /* Skip over the rest of the Class Specific or Vendor Specific */
    /* descriptors */
    begin = buffer;
    while (size >= DESC_HEADER_LENGTH) {
        header = (const struct usb1_descriptor_header *)buffer;
        if (header->bDescriptorType == DESC_HEADER_LENGTH) {
            usbi_err(ctx, "invalid extra ep desc len (%u)",
                header->bLength);
            return LIBUSB_ERROR_IO;
        } else if (header->bLength > size) {
            usbi_warn(ctx, "short extra ep desc read %d/%u",
                size, header->bLength);
            return parsed;
        }

        /* If we find another "proper" descriptor then we're done */
        if (header->bDescriptorType == LIBUSB_DT_INTERFACE ||
            header->bDescriptorType == LIBUSB_DT_CONFIG ||
            header->bDescriptorType == LIBUSB_DT_DEVICE)
            break;
    }

    usbi_dbg(ctx, "skipping descriptor 0x%02x", header->bDescriptorType);
    buffer += header->bLength;
    size -= header->bLength;
    parsed += header->bLength;
}

> Building GOTO bin library: /home/jonas/.cache/euf/libusb-42f9
> make clean
> ./configure
> make -j 15
> Building GOTO bin library: /home/jonas/.cache/euf/libusb-5c89
> make Clean
> ./configure
> make -j 15
> Starting change set reduction...
> Generating GOTO bin library: /home/jonas/.cache/euf/libusb-5c89 parse_endpoint(); parse_endpoint_old_id.c (23/71)
[✓] Identity verification successful: parse_endpoint: 0:00:01.566375
[✓] (ID) Starting CMC analysis for libusb(descriptor.c:83:12:parse endpoint()); parse_endpoint_id.c (23/71)
[✓] Identity verification successful: parse_endpoint_id: 0:00:01.566198
[✓] Starting CMC analysis for libusb(descriptor.c:83:12:parse endpoint()): parse_endpoint.c (23/71)
> VCCS-EXIST true
CMC version: 5.0.0.0 () 64-bit x86_64 linux
Reading ODO program from file ./runner
Generating GOTO Program
Adding CPROVER Library (x86_64)
using old value in module parse_endpoint file <built-in-additions> line 20
36028797018963968u6l
ignoring new value in module <built-in-library> file <built-in-additions> line 20
25175988313685244u0
using old value in module parse_endpoint file <built-in-additions> line 20
36028797018963968u6l
ignoring new value in module <built-in-library> file <built-in-additions> line 20
25175988313685244u0
Normal Function Pointers and Virtual Functions
Generic Property Instrumentation
Running with 12 object files, 52 offset bits (user-specified)
Starting Bounded Model Checking
**** WARNING: no body for function clock_gettime; assigning non-deterministic values to any pointer arguments
**** WARNING: no body for function clock_gettime; assigning non-deterministic values to any pointer arguments
**** WARNING: no body for function pthread_self; assigning non-deterministic values to any pointer arguments
**** WARNING: no body for function snprintf; assigning non-deterministic values to any pointer arguments
**** WARNING: no body for function vsprintf; assigning non-deterministic values to any pointer arguments
Runtime Sym: 0.0859115s
size of program expression: 4309 steps
size of program expression: 4309 steps
size of program expression: 4309 steps
Generated 1 VCC(s), 1 remaining after simplification
Runtime Postprocess Equation: 0.000223891s
Passing problem to propositional reduction
converting SSA
Runtime Convert SSA: 0.235523s
Running propositional reduction
Post-processing
Runtime Post-process: 0.00031424s
Solving with MiniSAT 2.2.3 with simplifier
496682 variables, 100 clauses
SAT solver instance is UNSATISFIABLE
Runtime Solver: 0.55375s
Runtime decision procedure: 0.788795s

** Results:
/home/jonas/.cache/euf/libusb-5c89/harnesses/parse_endpoint.c function euf_main
[euf_main.assertion.1] line 56 Equivalent output: SUCCESS

** 0 of 1 failed (1 iterations)
VERIFICATION SUCCESSFUL

real    0m1.055s
user   0m 0.996s
sys    0m 0.069s
[✓] Passed equivalence assertion: parse_endpoint: 0:00:01.566192
[✓] Change set reduction: 71 > 70 : 0:00:04.702574
Total runtime: 0:00:12.277649
~/repos/euf [main]
vol

```

The influential update has several modifications but notably moves the `else` block that is evaluated for `bLength < LIBUSB_DT_ENDPOINT_SIZE` (`bLength < 7`) higher up in the function and converts it to an `else if` statement.

The counterexample trace shows that for `size=2` (a size not below `DESC_HEADER_LENGTH`) the old version takes the `else if (header->bLength > size)` branch, returning the value of `parsed`, 0, while the new version takes the `else if (header->bLength < LIBUSB_DT_ENDPOINT_SIZE)` branch, returning `LIBUSB_ERROR_IO` (-1).

CBMC has thus found a sound counter-example where a value of `header->bLength` in the range `[size, LIBUSB_DT_ENDPOINT_SIZE]` will trigger different behaviour across the two versions. A complete list of the states for `bLength` was not attainable from CBMC but the trace should theoretically show a value in this prescribed range.

```

static int parse_endpoint(struct libusb_context *ctx,
    struct libusb_endpoint_descriptor *endpoint, unsigned char *buffer, int size)
{
    struct usbi_descriptor_header *header;
    unsigned char *extra;
    unsigned char *begin;
    int parsed = 0;
    int len;

    if (size < DESC_HEADER_LENGTH) {
        usbi_err(ctx, "short endpoint descriptor read %d/%d",
            size, DESC_HEADER_LENGTH);
        return LIBUSB_ERROR_IO;
    }

    header = (struct usbi_descriptor_header *)buffer;
    if (header->bDescriptorType != LIBUSB_DT_ENDPOINT) {
        usbi_err(ctx, "unexpected descriptor %x (expected %x)",
            header->bDescriptorType, LIBUSB_DT_ENDPOINT);
        return parsed;
    }

    else if (header->bLength > size) {
        usbi_warn(ctx, "short endpoint descriptor read %d/%d",
            size, header->bLength);
        return parsed;
    }

    if (header->bLength >= LIBUSB_DT_ENDPOINT_AUDIO_SIZE)
        parse_descriptor(buffer, "bbbbwbb", endpoint);
    else if (header->bLength >= LIBUSB_DT_ENDPOINT_SIZE)
        parse_descriptor(buffer, "bbbbw", endpoint);
    else {
        usbi_err(ctx, "invalid endpoint bLength (%d)", header->bLength);
        return LIBUSB_ERROR_IO;
    }

    buffer += header->bLength;
    size -= header->bLength;
    parsed += header->bLength;

    /* Skip over the rest of the Class Specific or Vendor Specific */
    /* descriptors */
    begin = buffer;
    while (size >= DESC_HEADER_LENGTH) {
        22 lines: header = (struct usbi_descriptor_header *)buffer;.....
    }

    /* Copy any unknown descriptors into a storage area for drivers */
    /* to later parse */
    len = (int)(buffer - begin);
    if (len <= 0)
        return parsed;

    extra = malloc((size_t)len);
    if (!extra)
        return LIBUSB_ERROR_NO_MEM;

    memcpy(extra, begin, len);
    endpoint->extra = extra;
    endpoint->extra_length = len;

    return parsed;
}

NORMAL ʃ euf-libusb  c libusb-bab/libusb/descriptor.c  c utf-8[unix] 14%165/1112= 11 /home/jonas/.cache/euf/libusb-d424/libusb/descriptor.c  c utf-8[unix] 15%183/1157= 11
-/..cache/euf/libusb-bab/libusb/descriptor.c" 1112l, 33353B written

static int parse_endpoint(struct libusb_context *ctx,
    struct libusb_endpoint_descriptor *endpoint, const uint8_t *buffer, int size)
{
    const struct usbi_descriptor_header *header;
    const uint8_t *begin;
    void *extra;
    int parsed = 0;
    int len;

    if (size < DESC_HEADER_LENGTH) {
        usbi_err(ctx, "short endpoint descriptor read %d/%d",
            size, DESC_HEADER_LENGTH);
        return LIBUSB_ERROR_IO;
    }

    header = (const struct usbi_descriptor_header *)buffer;
    if (header->bDescriptorType != LIBUSB_DT_ENDPOINT) {
        usbi_err(ctx, "unexpected descriptor %x (expected %x)",
            header->bDescriptorType, LIBUSB_DT_ENDPOINT);
        return parsed;
    }

    else if (header->bLength < LIBUSB_DT_ENDPOINT_SIZE) {
        usbi_err(ctx, "invalid endpoint bLength (%u)", header->bLength);
        return LIBUSB_ERROR_IO;
    } else if (header->bLength > size) {
        usbi_warn(ctx, "short endpoint descriptor read %d/%d",
            size, header->bLength);
        return parsed;
    }

    if (header->bLength >= LIBUSB_DT_ENDPOINT_AUDIO_SIZE)
        parse_descriptor(buffer, "bbbbwbb", endpoint);
    else if (header->bLength >= LIBUSB_DT_ENDPOINT_SIZE)
        parse_descriptor(buffer, "bbbbw", endpoint);
    else {
        usbi_err(ctx, "invalid endpoint bLength (%d)", header->bLength);
        return LIBUSB_ERROR_IO;
    }

    buffer += header->bLength;
    size -= header->bLength;
    parsed += header->bLength;

    /* Skip over the rest of the Class Specific or Vendor Specific */
    /* descriptors */
    begin = buffer;
    while (size >= DESC_HEADER_LENGTH) {
        22 lines: header = (const struct usbi_descriptor_header *)buffer;.....
    }

    /* Copy any unknown descriptors into a storage area for drivers */
    /* to later parse */
    len = (int)(buffer - begin);
    if (len <= 0)
        return parsed;

    extra = malloc((size_t)len);
    if (!extra)
        return LIBUSB_ERROR_NO_MEM;

    memcpy(extra, begin, len);
    endpoint->extra = extra;
    endpoint->extra_length = len;

    return parsed;
}

file /home/jonas/.cache/euf/libusb-bab/harnesses/parse_endpoint.c function euf_main line 61 thread 0
Equivalent output
=====
** 1 of 1 failed (2 iterations)
VERIFICATION FAILED
real   0m1.397s
user   0m1.317s
sys    0m0.075s
[*] Rejecting equivalence assertion: parse_endpoint: 0:00:01.869136
    Change set reduction: 37 -> 37: 0:00:05.604094
    Total runtime: 0:00:12.980278
    make clean
    ./configure
    make -j 15
    make clean
    ./configure
    make -j 15
==> TRACE <=
State 443 file /home/jonas/.cache/euf/libusb-bab/harnesses/parse_endpoint.c function euf_main line 58 thread 0
=====
ret_old= (00000000 00000000 00000000 00000000)
State 528 file /home/jonas/.cache/euf/libusb-bab/harnesses/parse_endpoint.c function euf_main line 58 thread 0
=====
ret_old= (00000000 00000000 00000000 00000000)
State 529 file /home/jonas/.cache/euf/libusb-bab/harnesses/parse_endpoint.c function euf_main line 59 thread 0
=====
ret= (00000000 00000000 00000000 00000000)
State 612 file /home/jonas/.cache/euf/libusb-bab/harnesses/parse_endpoint.c function euf_main line 59 thread 0
=====
ret=1 (11111111 11111111 11111111 11111111)
State 439 file /home/jonas/.cache/euf/libusb-bab/harnesses/parse_endpoint.c function euf_main line 51 thread 0
=====
size=0 (00000000 00000000 00000000 00000000)
State 442 file /home/jonas/.cache/euf/libusb-bab/harnesses/parse_endpoint.c function euf_main line 51 thread 0
=====
size=2 (00000000 00000000 00000000 00000010)
State 450 file /home/jonas/.cache/euf/libusb-bab/harnesses/parse_endpoint.c function euf_main line 58 thread 0
=====
size=2 (00000000 00000000 00000000 00000010)
State 536 file /home/jonas/.cache/euf/libusb-bab/harnesses/parse_endpoint.c function euf_main line 59 thread 0
=====
size=2 (00000000 00000000 00000000 00000010)
State 454 file descriptor.c function parse_endpoint.old_b026324c6904b2a line 89 thread 0
=====
parsed=0 (00000000 00000000 00000000 00000000)
State 455 file descriptor.c function parse_endpoint.old_b026324c6904b2a line 89 thread 0
=====
parsed=0 (00000000 00000000 00000000 00000000)
State 540 file descriptor.c function parse_endpoint line 107 thread 0
=====
parsed=0 (00000000 00000000 00000000 00000000)
State 541 file descriptor.c function parse_endpoint line 107 thread 0
=====
parsed=0 (00000000 00000000 00000000 00000000)
State 411 file /home/jonas/.cache/euf/libusb-bab/harnesses/parse_endpoint.c function euf_main line 48 thread 0
=====
return_value_nondet.libusb_endpoint_descriptor.bLength=0 (00000000)
State 540 file descriptor.c function parse_endpoint line 107 thread 0
=====
endpointObj.bLength=0 (00000000)
State 552 file descriptor.c function parse_endpoint line 122 thread 0
=====
format="invalid endpoint bLength (%u)" (?)
State 563 file core.c function usbi_log line 2656 thread 0
=====
format="invalid endpoint bLength (%u)" (?)
State 563 file core.c function usbi_log line 2656 thread 0
=====
vet=>

```

Removing the (header->bLength < LIBUSB_DT_ENDPOINT_SIZE) branch from the new version yields a successful verification.

This seems like a FN since if `bLength < 7` were to be true for the old version, it would return -1 while the new version would take the `bLength < 9` branch and continue execution. Replacing the `parse_descriptor(buffer, "bbbbwb", endpoint)` call with `return -99` expectedly creates a failed verification, highlighting that the `bLength < 9` block in the new version is indeed evaluated together with the old `bLength < 7` block for at least one trace.

```

static int parse_endpoint(struct libusb_context *ctx,
    struct libusb_endpoint_descriptor *endpoint, unsigned char *buffer, int size)
{
    struct usb_descriptor_header *header;
    unsigned char *extra;
    unsigned char *begin;
    int parsed = 0;
    int len;

    if (size < DESC_HEADER_LENGTH) {
        usbi_err(ctx, "short endpoint descriptor read %d/%d",
            size, DESC_HEADER_LENGTH);
        return LIBUSB_ERROR_IO;
    }

    header = (struct usb_descriptor_header *)buffer;
    if (header->bDescriptorType != LIBUSB_DT_ENDPOINT) {
        usbi_err(ctx, "unexpected descriptor %x (expected %x)",
            header->bDescriptorType, LIBUSB_DT_ENDPOINT);
        return parsed;
    } else if (header->bLength > size) {
        usbi_err(ctx, "short endpoint descriptor read %d/%d",
            size, header->bLength);
        return parsed;
    }

    if (header->bLength > LIBUSB_DT_ENDPOINT_AUDIO_SIZE)
        parse_descriptor(buffer, "bbbbbbbb", endpoint);
    else if (header->bLength > LIBUSB_DT_ENDPOINT_SIZE)
        parse_descriptor(buffer, "bbbbb", endpoint);
    ~ else // bLength < 7
    usbi_err(ctx, "invalid endpoint length (%d)", header->bLength);
    return LIBUSB_ERROR_IO;
}

buffer += header->bLength;
size -= header->bLength;
parsed += header->bLength;

/* Skip over the rest of the Class Specific or Vendor Specific */
/* descriptors */
begin = buffer;
while (size >= DESC_HEADER_LENGTH) {
    ~~~~ 22 times: header = (struct usb_descriptor_header *)buffer;.....
}

/* Copy any unknown descriptors into a storage area for drivers */
/* to later parse */
len = (int)(buffer - begin);
if (len <= 0)
    return parsed;

extra = malloc((size_t)len);
if (!extra)
    return LIBUSB_ERROR_NO_MEM;

memcpy(extra, begin, len);
endpoint->extra = extra;
endpoint->extra_length = len;

return parsed;
}

static void clear_interface(struct libusb_interface *usb_interface)
{
    int i;
    int j;

    if (usb_interface->altsetting) {
        ~~~~ 13 lines: for (i = 0; i < usb_interface->num_altsetting; i++) {.....}
        free(void *)usb_interface->altsetting;
        usb_interface->altsetting = NULL;
    }
}

usbi_dbg("skipping descriptor %02x", header->bDescriptorType);
buffer += header->bLength;
size -= header->bLength;
parsed += header->bLength;

/* Copy any unknown descriptors into a storage area for drivers */
/* to later parse */
len = (int)(buffer - begin);
if (len <= 0)
    return parsed;

extra = malloc((size_t)len);
NORMAL ¶ euf-dk2.c /libusb-dk2/libusb/descriptor.c c utf-8[unix] 9%111/1112=§15
~-.cache/euf/libusb-dk2/libusb/descriptor.c" 1154L, 35052B written

```

Runtime Solver: 0.654468s
Runtime decision procedure: 0.94344s
Building error trace
no Roots
/home/jonas/.cache/euf/libusb-bab6/harnesses/parse_endpoint.c function euf_main
[euf_main.assertion.1] line 61 Equivalent output: FAILURE
Trace for euf_main.assertion.1:
Violated property:
file /home/jonas/.cache/euf/libusb-bab6/harnesses/parse_endpoint.c function euf_main line 61 thread 0
path /home/jonas/.cache/euf/libusb-bab6/harnesses/parse_endpoint.c
ret_old == ret
** 1 of 1 failed (2 iterations)
VERIFICATION FAILED
real 0m1.229s
user 0m0.008s
sys 0m0.054s
[X] Rejected equivalence assertion: parse_endpoint: 0:00:01.717922
> Change set reduction: 37 -> 37: 0:00:05.104273
Total runtime: 0:00:12.604283
> make clean
> ./configure
> make -j 15
make clean
> ./configure
> make -j 15
==> TRACE ==
State 443 file /home/jonas/.cache/euf/libusb-bab6/harnesses/parse_endpoint.c function euf_main line 58 thread 0
ret_old= (00000000 00000000 00000000 00000000)
State 507 file /home/jonas/.cache/euf/libusb-bab6/harnesses/parse_endpoint.c function euf_main line 58 thread 0
ret_old= (11111111 11111111 11111111 11111111)
State 508 file /home/jonas/.cache/euf/libusb-bab6/harnesses/parse_endpoint.c function euf_main line 59 thread 0
ret=0 (00000000 00000000 00000000 00000000)
State 531 file /home/jonas/.cache/euf/libusb-bab6/harnesses/parse_endpoint.c function euf_main line 59 thread 0
ret=99 (11111111 11111111 11111111 11001100)
State 439 file /home/jonas/.cache/euf/libusb-bab6.harnesses/parse_endpoint.c function euf_main line 51 thread 0
size=0 (00000000 00000000 00000000 00000000)
State 442 file /home/jonas/.cache/euf/libusb-bab6.harnesses/parse_endpoint.c function euf_main line 51 thread 0
size=11 (00000000 00000000 00000000 00001011)
State 450 file /home/jonas/.cache/euf/libusb-bab6.harnesses/parse_endpoint.c function euf_main line 58 thread 0
size=11 (00000000 00000000 00000000 00001011)
State 515 file /home/jonas/.cache/euf/libusb-bab6.harnesses/parse_endpoint.c function euf_main line 59 thread 0
size=11 (00000000 00000000 00000000 00001011)
State 454 file descriptor.c function parse_endpoint.old_b026324c6904b2a line 89 thread 0
parsed=0 (00000000 00000000 00000000 00000000)
State 455 file descriptor.c function parse_endpoint.old_b026324c6904b2a line 89 thread 0
parsed=0 (00000000 00000000 00000000 00000000)
State 519 file descriptor.c function parse_endpoint line 107 thread 0
parsed=0 (00000000 00000000 00000000 00000000)
State 520 file descriptor.c function parse_endpoint line 107 thread 0
parsed=0 (00000000 00000000 00000000 00000000)
State 411 file /home/jonas/.cache/euf/libusb-bab6.harnesses/parse_endpoint.c function euf_main line 48 thread 0
return_value_mondet.libusb.endpoint.descriptor.blengt=0 (00000000)
State 423 file /home/jonas/.cache/euf/libusb-bab6.harnesses/parse_endpoint.c function euf_main line 48 thread 0
endpointObj.blengt=0 (00000000)
State 468 file descriptor.c function parse_endpoint.old_b026324c6904b2a line 113 thread 0
format="invalid endpoint blength (%d)" (?)
State 470 file core.c function usbi_log.old_b026324c6904b2a line 2693 thread 0
format="invalid endpoint blength (%d)" (?)
format="invalid endpoint blength (%d)" (?)
~Deeps/euf [main]
val ~>

However, placing the `return -99` statement outside of the `else` block yields an equivalent result. For this classification to be valid would infer that no trace in the new version actually reaches the inserted `return` statement, which should not be possible unless one of the three earlier `return` statements are taken.

```

static int parse_endpoint(struct libusb_context *ctx,
    struct libusb_endpoint_descriptor *endpoint, unsigned char *buffer, int size)
{
    struct usbi_descriptor_header *header;
    usbi_err(ctx, "short endpoint descriptor read %d/%d",
        size, DESC_HEADER_LENGTH);
    int parsed = 0;
    int len;

    if (size < DESC_HEADER_LENGTH) {
        usbi_err(ctx, "short endpoint descriptor read %d/%d",
            size, DESC_HEADER_LENGTH);
        return LIBUSB_ERROR_IO;
    }

    header = (struct usbi_descriptor_header *)buffer;
    if (header->bDescriptorType != LIBUSB_DT_ENDPOINT) {
        usbi_err(ctx, "unexpected descriptor %x (expected %x)",
            header->bDescriptorType, LIBUSB_DT_ENDPOINT);
        return parsed;
    } else if (header->bLength > size) {
        usbi_warn(ctx, "short endpoint descriptor read %d/%d",
            size, header->bLength);
        return parsed;
    }

    if (header->bLength >= LIBUSB_DT_ENDPOINT_AUDIO_SIZE)
        parse_descriptor(buffer, "bbbbbbbb", endpoint);
    else if (header->bLength >= LIBUSB_DT_ENDPOINT_SIZE)
        parse_descriptor(buffer, "bbbbbb", endpoint);
    else if (header->bLength >= LIBUSB_DT_CONFIG_SIZE)
        parse_descriptor(buffer, "bbbbb", endpoint);
    else if (header->bLength >= LIBUSB_DT_DEVICE_SIZE)
        parse_descriptor(buffer, "bb", endpoint);

    usbi_err(ctx, "invalid endpoint length (%d)", header->bLength);
    return LIBUSB_ERROR_IO;
}

buffer += header->bLength;
size -= header->bLength;
parsed += header->bLength;

/* Skip over the rest of the Class Specific or Vendor Specific */
/* descriptors */
while (size >= DESC_HEADER_LENGTH) {
    header = (struct usbi_descriptor_header *)buffer;
    if (header->bLength < DESC_HEADER_LENGTH) {
        usbi_err(ctx, "invalid extra ep desc len (%d)",
            size);
        return LIBUSB_ERROR_IO;
    } else if (header->bLength > size) {
        usbi_warn(ctx, "short extra ep desc read %d/%d",
            size, header->bLength);
        return parsed;
    }

    /* If we find another "proper" descriptor then we're done */
    if (header->bDescriptorType == LIBUSB_DT_ENDPOINT ||
        (header->bDescriptorType == LIBUSB_DT_CONFIG) ||
        (header->bDescriptorType == LIBUSB_DT_CONFIG) ||
        (header->bDescriptorType == LIBUSB_DT_DEVICE))
        break;

    usbi_dbg("skipping descriptor %x", header->bDescriptorType);
    buffer += header->bLength;
    size -= header->bLength;
    parsed += header->bLength;
}

/* Copy any unknown descriptors into a storage area for drivers */
/* home/jonas/.cache/euf/libusb-bab/libusb/descriptor.c c utf-8[unix] 8%194/1113=§13
/*-./.cache/euf/libusb-d424/libusb/descriptor.c 1155L_351000 written

```

(+) Enumerating global symbols...
 (+) The following static functions overlap with fields in globally defined structs and will be excluded from CBMC analysis:
 ('release_interface', 'claim_interface', 'handle_events')
 (+) Global symbol enumeration: 0:0:01:053543
 (+) Total functions: 1360
 (+) Built-in library directory: /home/jonas/.cache/euf/libusb-d424
 (+) Building GOT0 bin library: /home/jonas/.cache/euf/libusb-bab0
 (+) Starting change set reduction...
 (+) (ID _old_) Starting CBMC analysis for libusb/descriptor.c:83:12;parse_endpoint(): parse_endpoint_old_id.c (3/37)
 (+) Identity verification successful: parse_endpoint: 0:0:01:86698
 (+) (ID _new_) Starting CBMC analysis for libusb/descriptor.c:83:12;parse_endpoint(): parse_endpoint.c (3/37)
 (+) Identity verification successful: parse_endpoint: 0:0:01:519340
 (+) Starting CBMC analysis for libusb/descriptor.c:83:12;parse_endpoint(): parse_endpoint.c (3/37)
 (+) VCCS_EXIT: true
 CBMC version 5.2.0 () 64-bit x86_64 Linux
 Running on a target program file ./runner
 Generating CPROVER library (86 64)
 using old values in module parse_endpoint file <built-in-additions> line 20
 36028797189339680L
 ignoring new values in module <built-in-library> file <built-in-additions> line 20
 223179981368250L
 using old values in module parse_endpoint file <built-in-additions> line 20
 36028797189339680L
 ignoring new values in module <built-in-library> file <built-in-additions> line 20
 223179981368250L
 Running on function pointers and virtual functions
 Generic Pointer Instrumentation
 Running with 12 object bits, 52 offset bits (user-specified)
 Starting Bounded Model Checking
 **** WARNING: no body for function clock_gettime; assigning non-deterministic values to any pointer arguments
 **** WARNING: no body for function syscalls; assigning non-deterministic values to any pointer arguments
 **** WARNING: no body for function pthread_self; assigning non-deterministic values to any pointer arguments
 **** WARNING: no body for function snprintf; assigning non-deterministic values to any pointer arguments
 Runtime Symex: 0.077394s
 Runtime Postprocess: 0.00019782s
 Passing problem to propositional reduction
 Conversion: 0.00019782s
 Runtime Convert SSA: 0.28725s
 Running propositional reduction
 Post-processing
 Runtime Post-process: 0.00247501s
 Solving with MinSAT+CNF simplifier
 SAT Solver: 0.64385s
 Runtime decision procedure: 0.931199s
 ** Results:
 /home/jonas/.cache/euf/libusb-bab/harnesses/parse_endpoint.c function euf_main
 [euf_main.assertion.1] line 61 Equivalent output: SUCCESS
 ** 0 of 1 failed (1 iterations)
 VERIFICATION SUCCESSFUL
 real 0m1.192s
 user 0m0.011s
 sys 0m0.055s
 (+) Passed equivalence assertion: parse_endpoint: 0:0:01:666832
 (+) Change set reduction: 37 -> 36: 0:0:05:055236
 (+) Total runtime: 0:00:12.418773
 (+) build command:
 (+) make -j 15
 (+) make clean
 (+) ./configure
 (+) make -j 15
 =>> TRACE <=>
 ~/Repos/euf [main]
 val ~>

The `parse_descriptor` function does not contain any obvious behaviour that could prevent `return -99` from still being executed but replacing the call with a `usbi_err` call (essentially a NOOP) triggers the expected counter example.

Note how the counter example trace has a large value set for `size`, this indirectly implies that `bLength` has a value exceeding `size` since the third return statement in the new version was not taken. This in turn means that the `usbi_err` call should not be evaluated, making the reason as to why the modification triggers a counter example even less clear.

```

static void clear_endpoint(struct libusb_endpoint_descriptor *endpoint)
{
    free((void *)endpoint->extra);
}

static int parse_endpoint(struct libusb_context *ctx,
    struct libusb_endpoint_descriptor *endpoint, unsigned char *buffer, int size)
{
    struct usbi_descriptor_header *header;
    unsigned char *extra;
    unsigned char *begin;
    int parsed = 0;
    int len;

    if (size < DESC_HEADER_LENGTH) {
        usbi_err(ctx, "short endpoint descriptor read %d/%d",
            size, DESC_HEADER_LENGTH);
        return LIBUSB_ERROR_IO;
    }

    header = (struct usbi_descriptor_header *)buffer;
    if (header->bDescriptorType != LIBUSB_DT_ENDPOINT) {
        usbi_err(ctx, "unexpected descriptor %x (expected %x)",
            header->bDescriptorType, LIBUSB_DT_ENDPOINT);
        return parsed;
    } else if (header->bLength > size) {
        usbi_warn(ctx, "short endpoint descriptor read %d/%d",
            size, header->bLength);
        return parsed;
    }

    if (header->bLength >= LIBUSB_DT_ENDPOINT_AUDIO_SIZE)
        parse_descriptor(buffer, "bbbbbbb", endpoint);
    else if (header->bLength >= LIBUSB_DT_ENDPOINT_SIZE)
        parse_descriptor(buffer, "bbbbb", endpoint);
    else if (header->bLength < 7)
        usbi_err(ctx, "invalid endpoint bLength (%d)", header->bLength);
    return LIBUSB_ERROR_IO;
}

buffer += header->bLength;
size -= header->bLength;
parsed += header->bLength;

/* Skip over the rest of the Class Specific or Vendor Specific */
/* descriptors */
begin = buffer;
while (size >= DESC_HEADER_LENGTH) {
    header = (struct usbi_descriptor_header *)buffer;
    if (header->bLength > DESC_HEADER_LENGTH) {
        usbi_err(ctx, "invalid extra ep desc len (%d)",
            header->bLength);
        return LIBUSB_ERROR_IO;
    } else if (header->bLength > size) {
        usbi_warn(ctx, "short extra ep desc read %d/%d",
            size, header->bLength);
        return parsed;
    }

    /* If we find another "proper" descriptor then we're done */
    if ((header->bDescriptorType == LIBUSB_DT_ENDPOINT) ||
        (header->bDescriptorType == LIBUSB_DT_INTERFACE) ||
        (header->bDescriptorType == LIBUSB_DT_CONFIG) ||
        (header->bDescriptorType == LIBUSB_DT_DEVICE))
        break;
}

usbi_dbg("skipping descriptor %x", header->bDescriptorType);
buffer += header->bLength;

```

/home/jonas/.cache/euf/libusb-bab6/libusb/descriptor.c c 10%112/1112=§13
NORMAL § eu->d424 <libusb-d424/libusb/descriptor.c c utf-8[unix] 11%131/1155=§12
~/cache/euf/libusb-d424/libusb/descriptor.c* 1155L, 358798 written

```

static void clear_endpoint(struct libusb_endpoint_descriptor *endpoint)
{
    free((void *)endpoint->extra);
}

static int parse_endpoint(struct libusb_context *ctx,
    struct libusb_endpoint_descriptor *endpoint, const uint8_t *buffer, int size)
{
    const struct usbi_descriptor_header *header;
    void *extra;
    int parsed = 0;
    int len;

    if (size < DESC_HEADER_LENGTH) {
        usbi_err(ctx, "short endpoint descriptor read %d/%d",
            size, DESC_HEADER_LENGTH);
        return LIBUSB_ERROR_IO;
    }

    header = (const struct usbi_descriptor_header *)buffer;
    if (header->bDescriptorType != LIBUSB_DT_ENDPOINT) {
        usbi_err(ctx, "unexpected descriptor %0x (expected %0x)",
            header->bDescriptorType, LIBUSB_DT_ENDPOINT);
        return parsed;
    } else if (header->bLength > size) {
        usbi_warn(ctx, "short endpoint descriptor read %d/%d",
            size, header->bLength);
        return parsed;
    }

    if (header->bLength >= LIBUSB_DT_ENDPOINT_AUDIO_SIZE)
        parse_descriptor(buffer, "bbbbbbb", endpoint);
    else if (header->bLength < 9)
        parse_descriptor(buffer, "bbbbb", endpoint);
    else if (header->bLength == 9)
        parse_descriptor(buffer, "00000000", endpoint);
    else
        return -99;
}

buffer += header->bLength;
size -= header->bLength;
parsed += header->bLength;

/* Skip over the rest of the Class Specific or Vendor Specific */
/* descriptors */
begin = buffer;
while (size >= DESC_HEADER_LENGTH) {
    header = (const struct usbi_descriptor_header *)buffer;
    if (header->bLength > DESC_HEADER_LENGTH) {
        usbi_err(ctx, "invalid extra ep desc len (%u)",
            header->bLength);
        return LIBUSB_ERROR_IO;
    } else if (header->bLength > size) {
        usbi_warn(ctx, "short extra ep desc read %d/%d",
            size, header->bLength);
        return parsed;
    }

    /* If we find another "proper" descriptor then we're done */
    if ((header->bDescriptorType == LIBUSB_DT_ENDPOINT) ||
        (header->bDescriptorType == LIBUSB_DT_INTERFACE) ||
        (header->bDescriptorType == LIBUSB_DT_CONFIG) ||
        (header->bDescriptorType == LIBUSB_DT_DEVICE))
        break;
}

usbi_dbg("skipping descriptor %0x", header->bDescriptorType);
buffer += header->bLength;

```

Equivalent output
verdict: VERIFICATION FAILED
** 1 of 1 failed (2 iterations)
VERIFICATION FAILED
real 0m1.371s
user 0m1.312s
sys 0m0.064s
make clean
make -j 15
make clean
make
make -j 15
=> TRACE <=
State 443 file /home/jonas/.cache/euf/libusb-bab6/.harnesses/parse_endpoint.c function euf_main line 54 t
hread 0
ret_0ld=0 (00000000 00000000 00000000 00000000)
State 507 file /home/jonas/.cache/euf/libusb-bab6/.harnesses/parse_endpoint.c function euf_main line 54 t
hread 0
ret_0ld=-1 (11111111 11111111 11111111 11111111)
State 508 file /home/jonas/.cache/euf/libusb-bab6/.harnesses/parse_endpoint.c function euf_main line 55 t
hread 0
ret_0= (00000000 00000000 00000000 00000000)
State 569 file /home/jonas/.cache/euf/libusb-bab6/.harnesses/parse_endpoint.c function euf_main line 55 t
hread 0
ret_99=-99 (11111111 11111111 11111111 10011101)
State 501 file /home/jonas/.cache/euf/libusb-bab6/.harnesses/parse_endpoint.c function euf_main line 51 t
hread 0
size_0= (00000000 00000000 00000000 00000000)
State 442 file /home/jonas/.cache/euf/libusb-bab6/.harnesses/parse_endpoint.c function euf_main line 51 t
hread 0
size_13=217730 (00001000 00000000 00000000 00000010)
State 450 file /home/jonas/.cache/euf/libusb-bab6/.harnesses/parse_endpoint.c function euf_main line 54 t
hread 0
size_13=217730 (00001000 00000000 00000000 00000010)
State 451 file /home/jonas/.cache/euf/libusb-bab6/.harnesses/parse_endpoint.c function euf_main line 54 t
hread 0
size_13=217730 (00001000 00000000 00000000 00000010)
State 515 file /home/jonas/.cache/euf/libusb-bab6/.harnesses/parse_endpoint.c function euf_main line 55 t
hread 0
size_13=217730 (00001000 00000000 00000000 00000010)
State 454 file descriptor.c function parse_endpoint_old_b026324c6904b2a line 89 thread 0
parsed_0= (00000000 00000000 00000000 00000000)
State 455 file descriptor.c function parse_endpoint_old_b026324c6904b2a line 89 thread 0
parsed_0= (00000000 00000000 00000000 00000000)
State 519 file descriptor.c function parse_endpoint line 107 thread 0
parsed_0= (00000000 00000000 00000000 00000000)
State 520 file descriptor.c function parse_endpoint line 107 thread 0
parsed_0= (00000000 00000000 00000000 00000000)
State 411 file /home/jonas/.cache/euf/libusb-bab6/.harnesses/parse_endpoint.c function euf_main line 48 t
hread 0
return_value nondet libusb endpoint descriptor.blength=0 (00000000)
State 423 file /home/jonas/.cache/euf/libusb-bab6/.harnesses/parse_endpoint.c function euf_main line 48 t
hread 0
endpoint\$object.blength=0 (00000000)
-/Repos/euf [main]
vel ->

As previously shown, the *bab6* -> *d424* update contains a valid counter-example and thereby does not constitute a FP per say. However, modifying the update shows cracks in the coverage of CBMC for this function, highlighting the fact that several forms of similar influential updates could be incorrectly labeled and create FPs.

- **Equivalent verdict:** Inconclusive
- **Influential verdict:** Sound