Boy Who Cried Wolf

Many FPs decrease trust in the analyzer
Vicious Circle

- Low trust
- Small attention budget
- Poor understanding

In the analyzer: To understand the report

This must be a false positive
Break the Vicious Circle via Better Reporting

In the analyzer:
- Low trust

To understand the report:
- Small attention budget

Vicious circle:
- Poor understanding
- Good report

Good understanding:
- Higher trust

This must be a false positive
Interactive Reports: Understandable & Complete
Issues by Complexity
Simple Issues: Single Source Location

```c
int fixme(bool flag) {
    int arr[2] = {{0} = 3, {1} = 2};
    int x = 14e300;
    x = (x - arr[0]);
    if (flag)
        return [1] {
            if (flag)
                return (int *)x - (int.*)arr;
        }
    }
```
Simple Issues: Plain-Text

6 warnings generated.
/home/arseniy/proj/euollvm-talk/primloc-only.cpp:2:17: warning: array designators are a C99 extension [clang-diagnostic-c99-designator]
    int arr[2] = [{0} = 3, [1] = 2];

/home/arseniy/proj/euollvm-talk/primloc-only.cpp:3:11: warning: implicit conversion of out of range value from 'double' to 'int' is undefined [clang-diagnostic-literal-conversion]
    int x = 14e300;

/home/arseniy/proj/euollvm-talk/primloc-only.cpp:4:8: warning: Although the value stored to 'x' is used in the enclosing expression, the value is never actually read [clang-analyzer-deadcode.DeadStores]
    x = (x += arr[0]);

/home/arseniy/proj/euollvm-talk/primloc-only.cpp:8:16: warning: cast to 'int *' from smaller integer type 'int' [clang-diagnostic-int-to-pointer-cast]
    return (int *)x - (int *)arr;

/home/arseniy/proj/euollvm-talk/primloc-only.cpp:9:5: warning: non-void lambda does not return a value in all control paths [clang-diagnostic-return-type]
}();

/home/arseniy/proj/euollvm-talk/primloc-only.cpp:10:1: warning: non-void function does not return a value in all control paths [clang-diagnostic-return-type]
Simple Issues: GUI

```c++
int fixme(bool flag) {
    int x = 1e300;
    x = (x += arr[0]);
    if (flag)
        return [=] {
            if (flag)
                return (int *)x - (int *)arr;
        }();
}
```
Involved Issues: Multiple Locations

```cpp
#include <vector>
#include <string>

void myvec() {
    std::vector<std::string> vs{1, 2};
}
```

```
many-seconds-lirrelevant.cpp:5:28: error: no matching constructor for initialization of `std::vector<std::string>`
std::vector<std::string> vs{1, 2};
^
/usr/bin/../lib/gcc/x86_64-linux-gnu/11/../../../../include/c++/11/bits/stl_vector.h:585:7: note: candidate constructor not viable: requires 3 arguments, but 2 were provided
vector(vector&& __rv, const allocator_type& __a) noexcept
```
Multiple Locations: GUI

"InheritableAttr", and remove the ones you manually duplicated.

A Code Smell

Add a using-declaration to this derived class to inherit the constructors of "InheritableParamAttr", and remove the ones you manually duplicated.

A Code Smell

Add a using-declaration to this derived class to inherit the constructors of "StmtIteratorImpl", and remove the ones you manually duplicated.

A Code Smell

The derived class

Removable constructor

Matching constructor in base class

Removable constructor

Matching constructor in base class

Removable constructor

Matching constructor in base class

Do not read
Path-Sensitive Issues: Flow

```c
int getInt();
int *getPtr();

void fixme() {
    int x = getInt();
    int y = getInt();
    int *p = getPtr();
    if (x < y) {
        if (p)
            x = 8;
        if (x == 3)
            y = *p;
    }
}
```

- **Flow 1**
  1. 'p' initialized here [7, 2]
  2. Assuming 'x' is < 'y' [8, 6]
  3. Taking true branch [8, 2]
  4. Assuming 'p' is null [9, 8]
  5. Taking false branch [9, 4]
  6. Assuming 'x' is equal to 3 [11, 8]
  7. Taking true branch [11, 4]
  8. Dereference of null pointer (loaded from variable 'p') [...]

Dereference of null pointer (loaded from variable 'p')
Path-Sensitive Issues: Flow

```c
int getInt();
int *getPtr();

void fixme() {
    int x = getInt();
    int y = getInt();
    int *p = getPtr();

    if (x < y) {
        if (p) {
            x = 8;
            if (x == 3) {
                y = *p;
            }
        }
    }
    y = *p;
}
```
Null pointer passed as read buffer "ntdev.Buffer" in call to "wcsncasecmp"

Parameter values should be appropriate cpp:S3807

Bug  Critical  Open  Not assigned  5min effort  0 comments

Where is the issue?  Why is this an issue?

```
263     ans.Buffer[ans.MaximumLength - 1] = '\0';
264     got_one = true;
265     /* Special case for local disks: It's most feasible if the
266        DOS device name reflects the DOS drive, so we check for this
267        explicitly and only return prematurely if so. */
268     if (ntdev.Length < wcslen(HARDDISK_PREFIX))
269     || wcscasecmp(ntdev.Buffer, HARDDISK_PREFIX, 0) != 0
270     && odl->ObjectName.Length == 2 * sizeof (wchar_t))
271     {
272         if (trailing)
273         {
274             /* If there's a trailing path, it's a perfectly valid
275                DOS pathname without the \ prefix. Unless it's
276                longer than MAX_PATH - 1 in which case it needs
277                the \ prefix. */
278             /*
279                 
280                 */
```

SonarQube™ technology is powered by SonarSource SA
Data Center Edition - Version 9.9 (build 65466) - LGPL v3 - Community - Documentation - Plugins - Web API
Null pointer passed as read buffer "ntdev.Buffer" in call to "wcsncasecmp"

Parameter values should be appropriate: cpp:S3807

Bug - Critical - Open - Not assigned - 5min effort - 0 comments

Where is the issue?

Why is this an issue?

```c
/* If the comparison succeeds, the name of the directory entry is a valid DOS device name, if prepended with "\". Return that valid DOS path. */

wchar_t *trailing = NULL;

if (3) { ntdev.Length > tgtdev.Length
  trailing = ntdev.Buffer + tgtdev.Length / sizeof (wchar_t);
  ULONG len = RtlUnicodeStringToAnsiSize (&odl->ObjectName);
  if (5) { trailing
    len += wcsntombs (NULL, trailing, 0);
    free (ret);
    ret = (char *) malloc (len + 4);
    strcpy (ret, "\\\\\\\n");
    ans.Length = 0;
    ans.MaximumLength = len;
    ans.Buffer = ret + 4;
    RtlUnicodeStringToAnsiString (&ans, &odl->ObjectName, FALSE);
  } if (4) { trailing
    my_wcsntombs (ans.Buffer + ans.Length, trailing,
```
DFG
Choose Your Own Adventure

167 You find a secret door which opens into the bend where two passageways meet. To the north a short passage runs into a dead end, and to the east, the passageway reaches a crossroads. If you will step through this secret door into the passageway, turn to 187. If you decide against going through the secret door, close it and return down the passage to the crossroads – turn to 339.

168 You open the door to a large room. A large chair behind a solid-looking table suggests to you that someone, or something, of rank uses this room. A chest in the centre catches your eye. In a corner of the room stands a man-sized creature with a warty face, standing over a smaller creature of similar race. With the whip in his hand, the Orc Chieftain has been beating his servant, who is whimpering beneath him. Will you:

- Back them both? Turn to 372
- Spring at the Chieftain in the hope that his servant will aid you? Turn to 65
- Leave the room and head back for the junction? Turn to 293

169 With the whip in his hand, the Orc Chieftain has been beating his servant, who is whimpering beneath him.
Data Flow Based Interactive Report
Null pointer passed as read buffer "ntdev.Buffer" in call to "wcsncasecmp".

### Relevant values:

This buffer is null:
- Use: ntdev.Buffer
- Def: ntdev.Buffer

```c
continue;
if (tgtdev.Length /* There's actually a symlink pointing to an empty string: \\
GLOBALROOT -> ** */
&& RtlEqualUnicodePathPrefix (&ntdev, &tgtdev, TRUE))
{
    /* If the comparison succeeds, the name of the directory entry is a valid DOS device name, if prepended with "\\.\". Return that valid DOS path. */
    wchar_t *trailing = NULL;
    if (ntdev.Length > tgtdev.Length)
        trailing = ntdev.Buffer + tgtdev.Length / sizeof (WCHAR);
    ULONG len = RtlUnicodeStringToAnsiSize (odi->ObjectName);
    if (trailing)
        len += my wcstombs (NULL, trailing, 0);
    free (ret);
    ret = (char *) malloc (len + 4);
    strcp y (ret, "\\.\\.\\");
    ans.Length = 0;
    ans.MaximumLength = len;
    ans.Buffer = ret + 4;
    RtlUnicodeStringToAnsiString (&ans, &odi->ObjectName, FALSE);
    if (trailing)
        my wcstombs (ans.Buffer + ans.Length, trailing,
                    ans.MaximumLength - ans.Length);
    ans.Buffer[ans.MaximumLength - 1] = '\0';
    got_one = true;
    /* Special case for local disks: It's most feasible if the DOS device name reflects the DOS drive, so we check for this explicitly and only return prematurely if so. */
    if (ntdev.Length < wcslen (HARDDISK PREFIX)
        || wcsncasecmp (ntdev.Buffer, HARDISK PREFIX, 8) != 0
        || (odi->ObjectName.Length == 2 * sizeof (WCHAR)
        && odi->ObjectName.Buffer[1] == L':'))
    {
        if (trailing)
        {
            /* If there's a trailing path, it's a perfectly valid DOS pathname without the \\. prefix. Unless it's longer than MAX_PATH - 1 in which case it needs the \\ prefix. */
            if ((len = strlen (ret + 4)) >= MAX_PATH)
                ret[2] = '?'s;
            else
                remove (ret, ret + 4, strlen (ret + 4) + 1);
        }
    }
```
Null pointer passed as read buffer "ntdev.Buffer" in call to "wcsncasecmp".

Exploded points:
This buffer is null:
Use: ntdev.Buffer
Def: ntdev.Buffer

ntdev.Buffer

Relevant values:
"ntdev.Buffer".
Use: ntdev.Buffer
Def: NtOpenSymbolicLinkObject (&link, SYMBOLIC LINK_QUERY, &ntobj)

"ntdev".
Use: ntdev
Def: ntdev

continue;
if (tgtdev.Length /* There's actually a symlink pointing to an empty string: "\??\GLOBALROOT -> "$" */
 && RtlEqualUnicodePathPrefix (&ntdev, &tgtdev, TRUE))
{
 /* If the comparison succeeds, the name of the directory entry is a valid DOS device name, if prepended with "\\\". Return that valid DOS path. */
 wchar_t *trailing = NULL;
 if (ntdev.Length > tgtdev.Length)
 trailing = ntdev.Buffer + tgtdev.Length / sizeof (WCHAR);
 ULONG len = RtlUnicodeStringToAnsiSize (Sodi->ObjectName);
 if (trailing)
   len += _my_wcstombs (trailing); trailing, 0);
 free (ret);
 ret = (char *) malloc (len + 4);
 strcpy (ret, "\\\");  
 ans.Length = 0;
 ans.MaximumLength = len;
 ans.Buffer = ret + 4;
 RtlUnicodeStringToAnsiString (&ans, Sodi->ObjectName, FALSE);
 if (trailing)
   _my_wcstombs (ans.Buffer + ans.Length, trailing, 
 ans.MaximumLength - ans.Length);
 ans.Buffer[ans.MaximumLength - 1] = '\0';
 got_one = true;
 /* Special case for local disks: It's most feasible if the
 DOS device name reflects the DOS drive, so we check for this
 explicitly and only return prematurely if so. */
 if (ntdev.Length < wcslen (HARDDISK PREFIX) 
 || wcsncasecmp (ntdev.Buffer, HARDDISK PREFIX, 8) != 0
 || (odi->ObjectName.Length == 2 * sizeof (WCHAR)
  &\d.i->ObjectName.Buffer[1] == L':'))
{
 if (trailing)
{
 /* If there's a trailing path, it's a perfectly valid
 DOS pathname without the \. prefix. Unless it's longer than MAX_PATH - 1 in which case it needs
 the \ prefix. */
 if ((len = strlen (ret + 4)) >= MAX_PATH)
    ret[2] = '?';
 else
    remove (ret, ret + 4, strlen (ret + 4) + 1);
get_device_name (char *path)
{
  UNICODE_STRING ntdev, tgtdev, ntdevdir;
  ANSI_STRING ans;
  OBJECT_ATTRIBUTES ntobj;
 NTSTATUS status;
  HANDLE lnk, dir;
  bool got_one = false;
  char *ret = strdup (path);
  PDIRECTORY_BASIC_INFORMATION odi = (PDIRECTORY_BASIC_INFORMATION)
    alloc (4096);
  BOOLEAN restart;
  ULONG cont;
  if (!strncasecmp (path, GLOBALROOT_PREFIX "\", sizeof (GLOBALROOT_PREFIX)))
    path += strlen (GLOBALROOT_PREFIX) - 1;
  if (strncasecmp (path, "\Device\", 8))
    return ret;
  if (!RtlAllocateUnicodeString (&ntdev, 65534))
    return ret;
  if (!RtlAllocateUnicodeString (&tgtdev, 65534))
    return ret;
  RtlInitAnsiString (&ans, path);
  RtlAnsiStringToUnicodeString (&ntdev, &ans, FALSE);
  /* First check if the given device name is a symbolic link itself. If so, query it and use the new name as actual device name to search for it in the DOS device name directory. If not, just use the incoming device name. */
  InitializeObjectAttributes (&ntobj, &ntdev, OBJ_CASE_INSENSITIVE, NULL, NULL);
  status = NtOpenSymbolicLinkObject (&lnk, SYMBOLIC_LINK_QUERY, &ntobj);
  if (NT_SUCCESS (status))
    {
      status = NtQuerySymbolicLinkObject (lnk, &tgtdev, NULL);
      NtClose (lnk);
      if (!NT_SUCCESS (status))
        goto out;
      RtlCopyUnicodeString (&ntdev, &tgtdev);
      }
  else if (status != STATUS_OBJECT_TYPE_MISMATCH
            & status != STATUS_OBJECT_PATH_SYNTAX_BAD)
    goto out;
  for (int i = 0; i < 2; ++i)
Null pointer passed as read buffer "ntdev.Buffer" in call to "wcsncasecmp".

**Explored points:**

*This buffer is null:*

**Use:** ntdev.Buffer  
**Def:** ntdev.Buffer

```
"ntdev.Buffer":  
Use: ntdev.Buffer  
Def: ntdev.Buffer  
```

**Invalidation**  
Def: NtOpenSymbolicLinkObject  
&lnk, SYMBOLIC_LINK_QUERY, &ntobj)

**trailing**

"ntdev.Buffer" is null

**Use:** trailing  
**Def:** trailing = ntdev.Buffer + tgtdev.Length / sizeof (WCHAR)

**Relevant values:**

ntdev.Buffer  
tgtdev.Length

```c
for (restart = TRUE, cont = 0;
    NT_SUCCESS (NtQueryDirectoryObject (dir, odi, 4096, TRUE,
                                          restart, &cont, NULL));
    restart = FALSE)
{
    /* For each entry check if it's a symbolic link. */
    InitializeObjectAttributes (&ntobj, &odi->ObjectName,
                               OBJ_CASE_INSENSITIVE, dir, NULL);
    status = NtOpenSymbolicLinkObject (&lnk, SYMBOLIC_LINK_QUERY, &ntobj);
    if (!NT_SUCCESS (status))
        continue;
    tgtdev.Length = 0;
    tgtdev.MaximumLength = 512;
    /* If so, query it and compare the target of the symlink with the */
    /* incoming device name. */
    status = NtQuerySymbolicLinkObject (lnk, &tgtdev, NULL);
    NtClose (lnk);
    if (!NT_SUCCESS (status))
        continue;
    if (tgtdev.Length  
        /* There's actually a symlink pointing to an */
        /* empty string: \??GLOBALROOT -> "" */
        &RtlEqualUnicodePathPrefix (&ntdev, &tgtdev, TRUE))
    {
        /* If the comparison succeeds, the name of the directory entry is */
        /* a valid DOS device name, if prepended with "\\\". Return that */
        /* valid DOS path. */
        wchar_t *trailing = NULL;
        if (ntdev.Length > tgtdev.Length)
            trailing = ntdev.Buffer + tgtdev.Length / sizeof (WCHAR);
        ULONG len = RtlUnicodeStringToAnsiSize (&odi->ObjectName);
        if (trailing)
            len += wcsnctombs (NULL, trailing, 0);
        free (ret);
        ret = (char *) malloc (len + 4);
        strcpy (ret, "\\\\\"");
        ans.Length = 0;
        ans.MaximumLength = len;
        ans.Buffer = ret + 4;
        RtlUnicodeStringToAnsiString (&ans, &odi->ObjectName, FALSE);
        if (trailing)
            wcsnctombs (ans.Buffer + ans.Length, trailing,
                        wcsnctombs (ans.Buffer + ans.Length, trailing,);
Conclusion

- Clarity is as important as precision
- GUI overcomes limitations of plain-text
- GUI capabilities are underutilized
- The completeness vs brevity tradeoff is rudimentary