support@viva64.com Contact Us
Русский English
Collected Errors:
9574
Checked Projects:
245
42 tips on C++:
Read

PVS-Studio

Static Code Analyzer for C, C++ and C#

- Product page
- Documentation
- Troubleshooting FAQ

Download and try Buy

Blog:

- 19.05.2016

  Introduction to Roslyn and its use in program development

  Roslyn is a platform which provides the developer with powerful tools to parse and analyze code. It's not enough just ...

  Read more
- 18.05.2016

  Showing abilities of PVS-Studio analyzer by examples of Microsoft open-source projects

  Microsoft gradually started to open the code of some projects. Our team is very happy about this. We support the ...

  Read more
- 11.05.2016

  Analyzing Firebird 3.0

  A new version of Firebird DBMS was released not so long ago. This release was one of the most significant ...

  Read more

Featured:

- 11.04.2016

  An always up-to-date list of articles describing errors that we find in open source projects with PVS-Studio

analyzer

It contains articles describing the errors that were discovered by analyzing different open-source projects.

Read more

- 05.01.2015

  Readers' FAQ on Articles about PVS-Studio, 2015

  In the comments to our articles, readers would often ask the same questions. We decided to make a FAQ to ...

  Read more

- 12.03.2014

  How we compared code analyzers: CppCat, Cppcheck, PVS-Studio, and Visual Studio

  We have carried out a thorough comparison of four analyzers for C/C++ code: CppCat, Cppcheck, PVS-Studio, and Visual Studio's built-in ...

  Read more

Follow our CTO:

Tweets by @Code_Analysis

Andrey Karpov Retweeted

'No Bugs' Hare
@NoBugsHare

I strongly prefer my own IDL with my own IDL compiler and my own (bit-oriented) encoding!
ow.ly/glzG300fQNe

20 May

Andrey Karpov
@Code_Analysis

- Home
- Blog
- Analyzing Firebird 3.0

Analyzing Firebird 3.0

11.05.2016 Pavel Belikov

A new version of Firebird DBMS was released not so long ago. This release was one of the most significant in the project's history, as it marked substantial revision of the architecture, addition of multithreading support, and performance improvements. Such a significant update was a good occasion for us to scan Firebird one more time with PVS-Studio static code analyzer.

Introduction

Firebird is a cross-platform open-source database management system written in C++ that runs on Microsoft Windows, Linux, Mac OS X, and many Unix-like operating systems. It can be used and distributed for free. To learn more about Firebird, welcome to the official site.

We have already scanned Firebird with our analyzer before. The previous report can be found in the article "A Spin-off: Firebird Checked by PVS-Studio". For this analysis, we took the project code from GitHub, the master branch. The building process is described in detail in the article at the project website. We analyzed the source files in PVS-Studio Standalone, version 6.03, using the Compiler Monitoring mechanism, which allows you to scan projects without integrating the tool into the build system. The log file generated by the analyzer can be viewed both in the Standalone version and in Visual Studio.

Typos

```
void advance_to_start()
{
  ....
  if (!isalpha(c) && c != '_' && c != '.' && c != '_')
    syntax_error(lineno, line, cptr);
  ....
}
```

PVS-Studio diagnostic message: V501 There are identical sub-expressions 'c != '_'' to the left and to the right of the '&&' operator. reader.c 1203

The analyzer detected a logical expression with two identical subexpressions c != '_'. The last condition contains a typo and should actually compare the c variable with some other character. In other functions nearby, the variable is tested for the '$' character, so it should probably be used in our example as well:

```
if (!isalpha(c) && c != '_' && c != '.' && c != '$')
```

Another mistake resulting from the programmer's inattention:

```
int put_message(....)
{
  if (newlen <= MAX_UCHAR)
    {
    put(tdgbl, attribute);
    put(tdgbl, (UCHAR) newlen);
  }
  else if (newlen <= MAX_USHORT)
  {
    if (!attribute2)
      BURP_error(314, "");
    ....
  }
  else
    BURP_error(315, "");
  ....
}
```

PVS-Studio diagnostic messages:

- V601 The string literal is implicitly cast to the bool type. Inspect the second argument. backup.cpp 6113
- V601 The string literal is implicitly cast to the bool type. Inspect the second argument. backup.cpp 6120

Here we deal with a wrong call to the BURP_error function. This is how the function is declared:

```
void BURP_error(USHORT errcode, bool abort,
     const MsgFormat::SafeArg& arg = MsgFormat::SafeArg());

void BURP_error(USHORT errcode, bool abort, const char* str);
```

The second argument is a boolean value and the third one is a string. In our example, however, the string literal is passed as the second argument and is, therefore, cast to true. The function call should be rewritten in the following way: BURP_error(315, true, "") or BURP_error(315, false, "").

However, there are cases when only the project authors can tell if there is an error or not.

```
void IDX_create_index(....)
{
  ....
  index_fast_load ifl_data;
  ....
  if (!ifl_data.ifl_duplicates)
    scb->sort(tdbb);

  if (!ifl_data.ifl_duplicates)
    BTR_create(tdbb, creation, selectivity);

  ....
}
```

PVS-Studio diagnostic message: V581 The conditional expressions of the 'if' operators situated alongside each other are identical. Check lines: 506, 509. idx.cpp 509

This example deals with two blocks of code that check the same condition in succession. There might be a typo in one of them, or this issue has to do with copying or deleting some code fragments. In any case, this code looks strange.

In the next example we'll discuss an issue that deals with pointers.

```
static void string_to_datetime(....)
{
  ....

  const char* p = NULL;
  const char* const end = p + length;

  ....

  while (p < end)
  {
    if (*p != ' ' && *p != '\t' && p != 0)
    {
      CVT_conversion_error(desc, err);
      return;
    }
    ++p;
  }

  ....
}
```

PVS-Studio diagnostic message: V713 The pointer p was utilized in the logical expression before it was verified against nullptr in the same logical expression. cvt.cpp 702

In the condition, the p variable is compared with nullptr right after dereferencing. It may indicate that some other condition should have been used instead of this check, or that this check is just not necessary.

Earlier in the code, a similar fragment can be found:

```
while (++p < end)
{
  if (*p != ' ' && *p != '\t' && *p != 0)
    CVT_conversion_error(desc, err);
}
```

To avoid errors like this, use appropriate literals when comparing with zero: '\0' for type char, 0 for numbers, and nullptr for pointers. Sticking to this rule will help you avoid lots of silly errors like that.

Unsafe use of memcmp

```
SSHORT TextType::compare(ULONG len1, const UCHAR* str1,
                         ULONG len2, const UCHAR* str2)
{
  ....

  SSHORT cmp = memcmp(str1, str2, MIN(len1, len2));

  if (cmp == 0)
    cmp = (len1 < len2 ? -1 : (len1 > len2 ? 1 : 0));

  return cmp;
}
```

PVS-Studio diagnostic message: V642 Saving the 'memcmp' function result inside the 'short' type variable is inappropriate. The significant bits could be lost breaking the program's logic. texttype.cpp 3

The memcmp function returns the following values:

- < 0 if str1 is less than str2
- 0 if str1 equals str2
- > 0 if str1 is greater than str2

The function does not guarantee to return exact values when the strings are not equal, so storing the result in a variable of size less than that of type int may lead to losing the most significant bits and distorting the execution logic.

Extra checks

```
void Trigger::compile(thread_db* tdbb)
{
  SET_TDBB(tdbb);

  Database* dbb = tdbb->getDatabase();
  Jrd::Attachment* const att = tdbb->getAttachment();

  if (extTrigger)
    return;

  if (!statement /*&& !compile_in_progress*/)
  {
    if (statement)
      return;

    ....
  }
}
```

PVS-Studio diagnostic message: V637 Two opposite conditions were encountered. The second condition is always false. Check lines: 778, 780. jrd.cpp 778

The analyzer detected checks of two opposite conditions. The second condition seems to be no longer necessary since the first one was changed at some point in the past, so it can be deleted, although it is completely up to the author to make this decision.

The following code fragment is another example of strange branching.

```
static void asgn_from( ref* reference, int column)
{
  TEXT variable[MAX_REF_SIZE];
  TEXT temp[MAX_REF_SIZE];

  for (; reference; reference = reference->ref_next)
  {
    const gpre_fld* field = reference->ref_field;
    ....

    if (!field || field->fld_dtype == dtype_text)
      ....
    else if (!field || field->fld_dtype == dtype_cstring)
      ....
    else
      ....
  }
}
```

PVS-Studio diagnostic message: V560 A part of conditional expression is always false: !field. int_cxx.cpp 217

If the field pointer is non-null, the code will never reach the condition in the else if branch. Either this check is redundant or there should be some other comparison instead of it. It's not clear, whether this condition contradicts the execution logic.

In addition to these examples, a number of redundant checks were found in logical expressions.

```
bool XnetServerEndPoint::server_init(USHORT flag)
{
  ....

  xnet_connect_mutex = CreateMutex(ISC_get_security_desc(),
                         FALSE, name_buffer);
  if (!xnet_connect_mutex ||
          (xnet_connect_mutex && ERRNO == ERROR_ALREADY_EXISTS))
  {
    system_error::raise(ERR_STR("CreateMutex"));
  }

  ....
}
```

PVS-Studio diagnostic message: V728 An excessive check can be simplified. The '||' operator is surrounded by opposite expressions '!xnet_connect_mutex' and 'xnet_connect_mutex'. xnet.cpp 2231

The check if (!xnet_connect_mutex || (xnet_connect_mutex && ERRNO == ERROR_ALREADY_EXISTS)) can be simplified to if (!xnet_connect_mutex || ERRNO == ERROR_ALREADY_EXISTS). The correctness of such transformation can be easily proved with the truth table.

Unsafe comparison of an unsigned variable

```
static bool write_page(thread_db* tdbb, BufferDesc* bdb, ....)
{
  ....
  if (bdb->bdb_page.getPageNum() >= 0)
  ....
}
```

PVS-Studio diagnostic message: V547 Expression 'bdb->bdb_page.getPageNum() >= 0' is always true. Unsigned type value is always >= 0. cch.cpp 4827

The bdb->bdb_page.getPageNum() >= 0 condition will always be true, as the function returns an unsigned value. This error probably has to do with an incorrect check of the value. Based on other similar comparisons in the project, I think the code should actually look like this:

```
if (bdb->bdb_page.getPageNum() != 0)
```

Null pointer dereferencing

```
static bool initializeFastMutex(FAST_MUTEX* lpMutex,
  LPSECURITY_ATTRIBUTES lpAttributes, BOOL bInitialState,
  LPCSTR lpName)
{
  if (pid == 0)
    pid = GetCurrentProcessId();
  LPCSTR name = lpName;

  if (strlen(lpName) + strlen(FAST_MUTEX_EVT_NAME) - 2
                                                   >= MAXPATHLEN)
  {
    SetLastError(ERROR_FILENAME_EXCED_RANGE);
    return false;
  }

  setupMutex(lpMutex);

  char sz[MAXPATHLEN];
  if (lpName)
  ....
}
```

PVS-Studio diagnostic message: V595 The 'lpName' pointer was utilized before it was verified against nullptr. Check lines: 2814, 2824. isc_sync.cpp 2814

Warning V595 is the most common among the projects scanned by PVS-Studio, and Firebird is no exception. In total, the analyzer found 30 issues triggering this diagnostic.

In this example, the call strlen(lpName) precedes a pointer check for nullptr, thus leading to undefined behavior when trying to pass a null pointer to the function. The pointer-dereferencing operation is hidden inside the call to strlen, which makes it difficult to find the error without a static analyzer.

Testing for nullptr after new

```
rem_port* XnetServerEndPoint::get_server_port(....)
{
  ....
  XCC xcc = FB_NEW struct xcc(this);

  try {

    ....
  }
  catch (const Exception&)
  {
    if (port)
      cleanup_port(port);
    else if (xcc)
      cleanup_comm(xcc);

    throw;
  }

  return port;
}
```

PVS-Studio diagnostic message: V668 There is no sense in testing the 'xcc' pointer against null, as the memory was allocated using the 'new' operator. The exception will be generated in the case of memory allocation error. xnet.cpp 2533

The analyzer warns us that the new operator cannot return nullptr - one must use a try-catch block or new (std::nothrow). However, this example is a bit more complicated. The programmer uses macro FB_NEW to allocate memory. This macro is declared in the file alloc.h:

```
#ifdef USE_SYSTEM_NEW
#define OOM_EXCEPTION std::bad_alloc
```

```
#else
#define OOM_EXCEPTION Firebird::BadAlloc
#endif

#define FB_NEW new(__FILE__, __LINE__)

inline void* operator new(size_t s ALLOC_PARAMS)
throw (OOM_EXCEPTION)
{
  return MemoryPool::globalAlloc(s ALLOC_PASS_ARGS);
}
```

I can't say for sure if this particular example is incorrect, as it uses a non-standard allocator; but the presence of throw (std::bad_alloc) in the operator declaration makes this check quite suspicious.

Unsafe use of realloc

```
int mputchar(struct mstring *s, int ch)
{
  if (!s || !s->base) return ch;
  if (s->ptr == s->end) {
    int len = s->end - s->base;
    if ((s->base = realloc(s->base, len+len+TAIL))) {
      s->ptr = s->base + len;
      s->end = s->base + len+len+TAIL; }
    else {
      s->ptr = s->end = 0;
      return ch; } }
  *s->ptr++ = ch;
  return ch;
}
```

PVS-Studio diagnostic message: V701 realloc() possible leak: when realloc() fails in allocating memory, original pointer 's->base' is lost. Consider assigning realloc() to a temporary pointer. mstring.c 42

What is bad about expressions of the ptr = realloc(ptr, size) pattern is that the pointer to the memory block will be lost when realloc returns nullptr. To avoid it, one needs to save the result returned by realloc in a temporary variable and then assign this value to ptr after comparing it with nullptr.

```
temp_ptr = realloc(ptr, new_size);
if (temp_ptr == nullptr) {
  //handle exception
} else {
  ptr = temp_ptr;
}
```

Unused enum values in switch

```
template <typename CharType>
LikeEvaluator<CharType>::LikeEvaluator(....)
{
  ....
  PatternItem *item = patternItems.begin();
  ....
  switch (item->type)
  {
  case piSkipFixed:
  case piSkipMore:
    patternItems.grow(patternItems.getCount() + 1);
    item = patternItems.end() - 1;
    // Note: fall into
    case piNone:
      item->type = piEscapedString;
      item->str.data = const_cast<CharType*>
                        (pattern_str + pattern_pos - 2);
      item->str.length = 1;
      break;
    case piSearch:
      item->type = piEscapedString;
      // Note: fall into
    case piEscapedString:
      item->str.length++;
      break;
  }
  ....
}
```

PVS-Studio diagnostic message: V719 The switch statement does not cover all values of the 'PatternItemType' enum: piDirectMatch. evl_string.h 324

Not all enum values were used in the switch statement; the default block is absent, too. This example seems to lack the code that handles the piDirectMatch element. Other similar issues:

- V719 The switch statement does not cover all values of the 'PatternItemType' enum: piDirectMatch, piSkipMore. evl_string.h 351
- V719 The switch statement does not cover all values of the 'PatternItemType' enum: piDirectMatch. evl_string.h 368
- V719 The switch statement does not cover all values of the 'PatternItemType' enum: piDirectMatch. evl_string.h 387

Buffer overflow

```
const int GDS_NAME_LEN = 32;
....
bool get_function(BurpGlobals* tdgbl)
{
  ....
  struct isc_844_struct {
    ....
    short isc_870; /* gds__null_flag */
    ....
    char  isc_874 [125]; /* RDB$PACKAGE_NAME */
    ....
  } isc_844;

  att_type attribute;
  TEXT    temp[GDS_NAME_LEN * 2];
  ....
  SSHORT prefixLen = 0;
  if (!/*X.RDB$PACKAGE_NAME.NULL*/
       isc_844.isc_870)
  {
    prefixLen = static_cast<SSHORT>(strlen(/*X.RDB$PACKAGE_NAME*/
                                           isc_844.isc_874));
    memcpy(temp, /*X.RDB$PACKAGE_NAME*/
                 isc_844.isc_874, prefixLen);
    temp[prefixLen++] = '.';
  }
  ....

}
```

PVS-Studio diagnostic message: V557 Array overrun is possible. The value of 'prefixLen ++' index could reach 124. restore.cpp 10040

The size of the buffer isc_844.isc_874 is 125; therefore, the largest value possible of strlen(isc_844.isc_874) is 124. The size of temp is 64, which is less than that value. Writing at this index may cause a buffer overflow. A safer way is to allocate a larger storage for the temp variable.

Shifting negative numbers

```
static ISC_STATUS stuff_literal(gen_t* gen, SLONG literal)
{
  ....

  if (literal >= -32768 && literal <= 32767)
    return stuff_args(gen, 3, isc_sdl_short_integer, literal,
                      literal >> 8);

  ....

}
```

PVS-Studio diagnostic message: V610 Unspecified behavior. Check the shift operator '>>'. The left operand is negative ('literal' = [-32768..32767]). array.cpp 848

The code contains a right-shift operation on a negative number. As the C++ standard states, such an operation leads to undefined behavior, i.e. it may produce different results on different compilers and platforms. The code should be rewritten as follows:

```
if (literal >= -32768 && literal <= 32767)
  return stuff_args(gen, 3, isc_sdl_short_integer, literal,
                    (ULONG)literal >> 8);
```

Another fragment triggering this warning:

V610 Unspecified behavior. Check the shift operator '>>'. The left operand is negative ('i64value' = [-2147483648..2147483647]). exprnodes.cpp 6382

Variable redefinition

```
THREAD_ENTRY_DECLARE Service::run(THREAD_ENTRY_PARAM arg)
{
  int exit_code = -1;
  try
  {
    Service* svc = (Service*)arg;
    RefPtr<SvcMutex> ref(svc->svc_existence);
    int exit_code = svc->svc_service_run->serv_thd(svc);

    svc->started();
    svc->svc_sem_full.release();
    svc->finish(SVC_finished);
  }
  catch (const Exception& ex)
  {
    // Not much we can do here
    iscLogException("Exception in Service::run():", ex);
  }

  return (THREAD_ENTRY_RETURN)(IPTR) exit_code;
}
```

PVS-Studio diagnostic message: V561 It's probably better to assign value to 'exit_code' variable than to declare it anew. Previous declaration: svc.cpp, line 1893. svc.cpp 1898

In this example, the exit_code variable is redefined instead of being assigned a value. Variable redefinition hides the previously declared variable from the scope and makes the function always return an incorrect value, which is -1.

Fixed code:

```
THREAD_ENTRY_DECLARE Service::run(THREAD_ENTRY_PARAM arg)
{
  int exit_code = -1;
  try
  {
    Service* svc = (Service*)arg;
    RefPtr<SvcMutex> ref(svc->svc_existence);
    exit_code = svc->svc_service_run->serv_thd(svc);

    svc->started();
    svc->svc_sem_full.release();
    svc->finish(SVC_finished);
  }
  catch (const Exception& ex)
  {
    // Not much we can do here
    iscLogException("Exception in Service::run():", ex);
  }

  return (THREAD_ENTRY_RETURN)(IPTR) exit_code;
}
```

Conclusion

As the new analysis shows, the project developers have fixed most of the issues found during the previous analysis, so those bugs are no longer there, which is a good sign that the compiler did a good job. However, using the analyzer regularly could help achieve even better results because that way it allows catching bugs at earlier stages. Incremental analysis and compatibility with any build system allow integrating the analyzer easily into your project. Using static analysis helps save plenty of time and catch errors that are difficult to detect by means of debugging or dynamic analysis.

g 6 | 0 | f 16 | 1 | 51

Next Previous

Our Customers

NAT GAMES

ESPRIT®
The Right Choice

- Centrify®

- IBM®

- VISPIRON
HIGH ENGINEERING

- TERADYNE

- AWR | A National Instruments Company™

-

MERGE

-

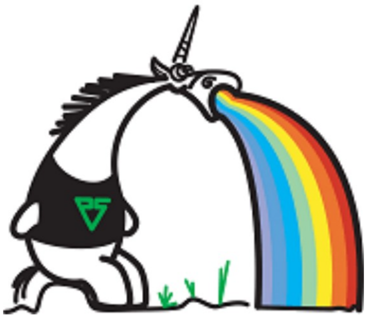
CSIR
our future through science

-

roxar
MAXIMUM RESERVOIR PERFORMANCE

-

‹ ›



We develop the static code analyzer PVS-Studio for C, C++ and C# code. This tool has managed to catch bugs in Chromium, Qt, Clang, etc. Check your code too.
support@viva64.com Contact Us

PVS-Studio

- Download
- Product page
- Documentation
- Messages
- Troubleshooting

Buy

- Buy PVS-Studio
- Site License
- Licensing FAQ

Our Advances

- Checked projects
- Detected errors
- Customers

Interesting

- Blog
- Articles
- C++ quiz
- Merchandise
- 64-bit lessons
- Knowledge base
- Terminology

Company

- About Us
- Jobs
- News
- Team
- Address
- Contact Us
- Sitemap

✓ Orphus Ctrl+Enter