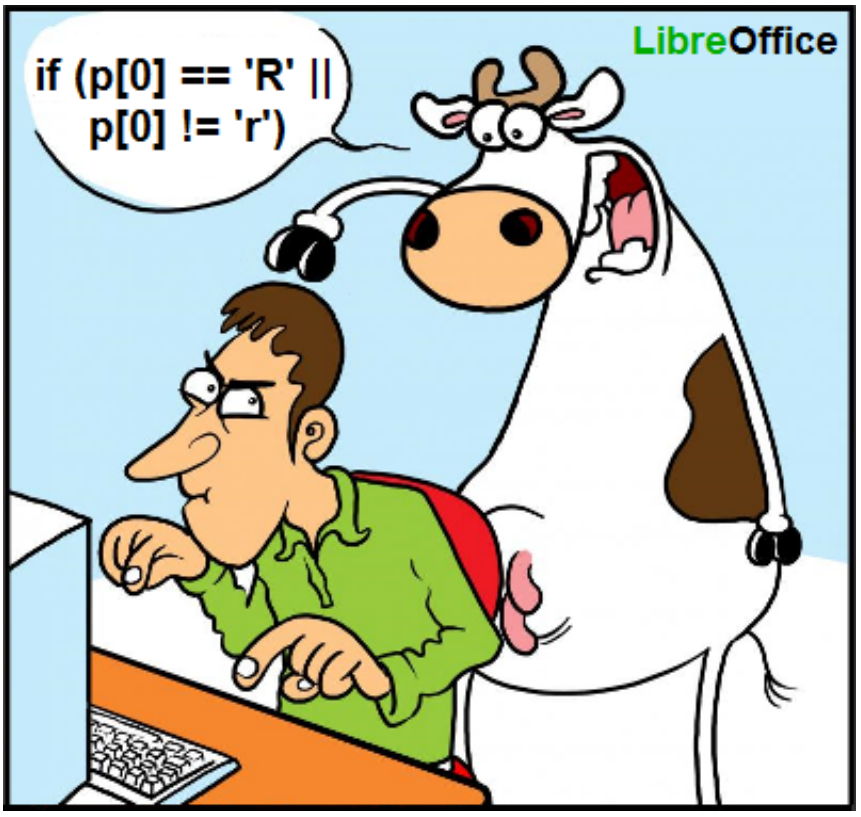
Проверка проекта LibreOffice

01.03.2015 Андрей Карпов

Предлагаем читателю очередную статью о проверке известного open-source проекта. В этот раз мы проверили проект LibreOffice, представляющий собой офисный пакет. В его разработке принимает участие более чем 480 программистов. Код оказался весьма качественным и регулярно проверяемым статическим анализатором Coverity. Но, как и в любом другом большом проекте, были найдены новые ошибки и недочеты, о которых мы и расскажем в статье. Для разнообразия, в этот раз нас будут сопровождать не единороги, а коровы.



LibreOffice — мощный офисный пакет, полностью совместимый с 32/64-битными системами. Переведён более чем на 30 языков мира. Поддерживает большинство популярных операционных систем, включая GNU/Linux, Microsoft Windows и Mac OS X.

LibreOffice бесплатен и имеет открытый исходный код. Написан на языках: Java, Python, C++. Анализу была подвергнута та часть проекта, которая написана на C++ (и немножко на C, C++/CLI). Version: 4.5.0.0.alpha0+ (Git revision: 368367).

Анализ был выполнен с помощью статического анализатора кода PVS-Studio.

Рассмотрим, какие ошибки были найдены, и что можно с ними сделать. Хочу отметить, что некоторые ошибки могут оказаться не ошибками. Я не знаком с кодом и мог принять за ошибку вполне корректный код. Однако, раз этот код сбил с толку анализатор и меня, значит что-то не так. Этот код пахнет, и хорошо провести его рефакторинг, чтобы снизить вероятность его неправильного понимания в процессе развитии проекта.

Опечатки

Любой код не обходится без опечаток. Многие, конечно, находятся и исправляются на этапе тестирования, но некоторые остаются жить внутри программ на долгое время. Как правило они находятся в редко используемых функциях или не оказывают сильного влияние на работу программы.

Например, встретилась вот такая функция сравнения, которая работает только на одну треть:

```
class SvgGradientEntry
{
  ....
```

```
  bool operator==(const SvgGradientEntry& rCompare) const
  {
    return (getOffset() == rCompare.getOffset()
            && getColor() == getColor()
            && getOpacity() == getOpacity());
  }
  ....
}
```

Предупреждение PVS-Studio: V501 There are identical sub-expressions to the left and to the right of the '==' operator: getColor() == getColor() svggradientprimitive2d.hxx 61

Наверное, это ошибка не приносит много вреда. Возможно, этот оператор '==' вообще не используется. Однако, раз анализатор смог найти эту ошибку, он сможет найти и более серьезные сразу после написания нового кода. Поэтому основная ценность статического анализа не в разовых запусках, а в регулярном использовании.

Как можно было бы попробовать избежать такой ошибки? Не знаю. Возможно, если приучить себя более тщательно выравнивать блоки однотипного кода, то ошибка была бы более заметна. Например, функцию можно оформить так:

```
bool operator==(const SvgGradientEntry& rCompare) const
{
  return    getOffset()  == rCompare.getOffset()
         && getColor()   == getColor()
         && getOpacity() == getOpacity();
}
```

Теперь стало более заметно, что в правом столбце не хватает "rCompare". Хотя если честно, заметно не очень сильно. Может и не помочь. Человеку свойственно ошибаться. И поэтому статический анализатор бывает хорошим помощником.

А вот пример, где опечатку явно можно было избежать. Кто-то неудачно написал код для обмена значений между двумя переменными.


```
void TabBar::ImplGetColors(....)
{
  ....
  aTempColor = rFaceTextColor;
  rFaceTextColor = rSelectTextColor;
  rSelectTextColor = rFaceTextColor;
  ....
}
```

Предупреждение PVS-Studio: V587 An odd sequence of assignments of this kind: A = B; B = A;. Check lines: 565, 566. tabbar.cxx 566

В последней строке вместо 'rFaceTextColor' следовало использовать 'aTempColor'.

Не нужно было писать код для обмена значений "вручную". Было бы проще и надёжней воспользоваться стандартной функцией std::swap():

```
swap(rFaceTextColor, rSelectTextColor);
```

Продолжим. От следующей ошибки думаю защититься невозможно. Опечатка в чистом виде:

```
void SAL_CALL Theme::disposing (void)
{
  ChangeListeners aListeners;
  maChangeListeners.swap(aListeners);

  const lang::EventObject aEvent (static_cast<XWeak*>(this));
```

```
  for (ChangeListeners::const_iterator
           iContainer(maChangeListeners.begin()),
           iContainerEnd(maChangeListeners.end());
       iContainerEnd!=iContainerEnd;
       ++iContainerEnd)
  {
    ....
  }
}
```

Предупреждение PVS-Studio: V501 There are identical sub-expressions to the left and to the right of the '!=' operator: iContainerEnd != iContainerEnd theme.cxx 439

Цикл выполняться не будет, так как условие "iContainerEnd!=iContainerEnd" всегда ложно. Подвело схожие названия итераторов. На самом деле, нужно было написать: "iContainer!=iContainerEnd". Кстати, здесь кажется есть ещё одна ошибка. Странно, что увеличивается итератор "iContainerEnd".

Рассмотрим другой неудачный цикл:

```
static void lcl_FillSubRegionList(....)
{
  ....
  for( IDocumentMarkAccess::const_iterator_t
      ppMark = pMarkAccess->getBookmarksBegin();     <<<<----
      ppMark != pMarkAccess->getBookmarksBegin();    <<<<----
      ++ppMark)
  {
    const ::sw::mark::IMark* pBkmk = ppMark->get();
    if( pBkmk->IsExpanded() )
      rSubRegions.InsertEntry( pBkmk->GetName() );
  }
}
```

Предупреждение PVS-Studio: V625 Consider inspecting the 'for' operator. Initial and final values of the iterator are the same. uiregionsw.cxx 120

Цикл выполняться не будет. В условии итератор 'ppMark' нужно сравнивать с 'pMarkAccess->getBookmarksEnd()'. Идей, как защититься от такой ошибки с помощью правил написания кода, у меня нет. Просто опечатка.

Кстати, иногда ошибка есть, но она никак не влияет на правильное выполнение программы. Сейчас продемонстрирую одну из таких опечаток:

```
bool PolyPolygonEditor::DeletePoints(....)
{
  bool bPolyPolyChanged = false;
```

```
  std::set< sal_uInt16 >::const_reverse_iterator
    aIter;( rAbsPoints.rbegin() );
  for( aIter = rAbsPoints.rbegin();
       aIter != rAbsPoints.rend(); ++aIter )
  ....
}
```

Предупреждение PVS-Studio: V530 The return value of function 'rbegin' is required to be utilized. polypolygoneditor.cxx 38

Ошибка вот здесь: aIter;(rAbsPoints.rbegin());

Хотели инициализировать итератор. Но случайно вклинилась точка с запятой. Итератор остался неинициализированным. А выражение "(rAbsPoints.rbegin());" болтается в воздухе и ничего не делает.

Спасает ситуацию то, что в цикле итератор всё-таки инициализируется нужным значением. В общем ошибки нет, но лишнее выражение лучше убрать. Кстати, этот цикл был размножен с помощью Copy-Paste, поэтому стоит заглянуть ещё в строку 69 и 129 в этом же файле.

Напоследок опечатка в конструкторе класса:

```
XMLTransformerOOoEventMap_Impl::XMLTransformerOOoEventMap_Impl(
        XMLTransformerEventMapEntry *pInit,
        XMLTransformerEventMapEntry *pInit2 )
{
  if( pInit )
    AddMap( pInit );
  if( pInit )
    AddMap( pInit2 );
}
```

Предупреждение PVS-Studio: V581 The conditional expressions of the 'if' operators situated alongside each other are identical. Check lines: 77, 79. eventoootcontext.cxx 79

Второй оператор 'if' должен проверять указатель 'pInit2'.

Может быть так и задумано, но очень подозрительно

Есть несколько фрагментов кода, которые кажется содержат опечатки. Но я не уверен. Возможно, так и задумано.

```
class VCL_DLLPUBLIC MouseSettings
{
  ....
  long GetStartDragWidth() const;
  long GetStartDragHeight() const;
```

```
  ....
}

bool ImplHandleMouseEvent( .... )
{
  ....
  long nDragW  = rMSettings.GetStartDragWidth();
  long nDragH  = rMSettings.GetStartDragWidth();
  ....
}
```

Предупреждение: V656 Variables 'nDragW', 'nDragH' are initialized through the call to the same function. It's probably an error or un-optimized code. Consider inspecting the 'rMSettings.GetStartDragWidth()' expression. Check lines: 471, 472. winproc.cxx 472

Не понятно, должны инициализироваться переменные nDragW и nDragH одним и тем же значением, или нет. Если да, то не хватает комментария. Или лучше было бы явно написать:

```
long nDragW  = rMSettings.GetStartDragWidth();
long nDragH  = nDragW;
```

Похожая ситуация:

```
void Edit::ImplDelete(....)
{
  ....
  maSelection.Min() = aSelection.Min();
  maSelection.Max() = aSelection.Min();
  ....
}
```

V656 Variables 'maSelection.Min()', 'maSelection.Max()' are initialized through the call to the same function. It's probably an error or un-optimized code. Consider inspecting the 'aSelection.Min()' expression. Check lines: 756, 757. edit.cxx 757

Для тех, кто работает с проектом здесь всё сразу понятно. Я не работаю, и поэтому не знаю, есть здесь ошибка или нет.

И последний случай. В классе есть три функции:

- GetVRP()
- GetVPN()
- GetVUV()

Однако, вот здесь, для инициализации константы 'aVPN' используется функция GetVRP().

```
void ViewContactOfE3dScene::createViewInformation3D(....)
{
```

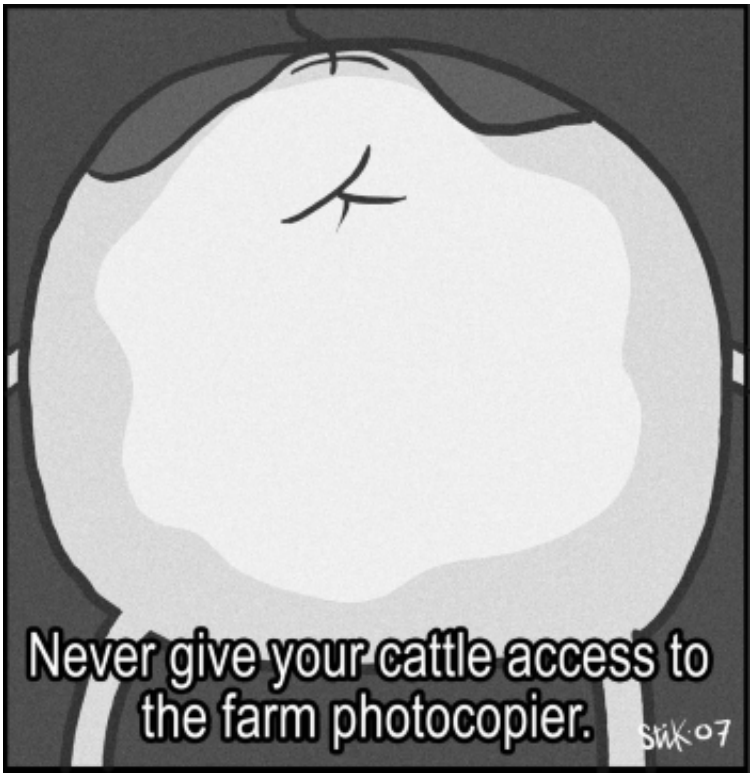
```
  ....
  const basegfx::B3DPoint aVRP(rSceneCamera.GetVRP());
  const basegfx::B3DVector aVPN(rSceneCamera.GetVRP());  <<<---
  const basegfx::B3DVector aVUV(rSceneCamera.GetVUV());
  ....
}
```

Предупреждение PVS-Studio: V656 Variables 'aVRP', 'aVPN' are initialized through the call to the same function. It's probably an error or un-optimized code. Consider inspecting the 'rSceneCamera.GetVRP()' expression. Check lines: 177, 178. viewcontactofe3dscene.cxx 178

Анализатор выдал ещё одно предупреждение V656. Я почти уверен, что там настоящая ошибка. Но приводить код не буду, так как он громоздкий. Прошу разработчиков посмотреть вот сюда:

- V656 Variables 'oNumOffset1', 'oNumOffset2' are initialized through the call to the same function. It's probably an error or un-optimized code. Check lines: 68, 69. findattr.cxx 69

Copy-Paste



Вынужден признать, что без Copy-Paste программирование временами будет крайне утомительным и скучным занятием. Без Ctrl-C, Ctrl-V программировать не получится, как бы не хотелось запретить эти сочетания клавиш. Поэтому не буду призывать не копировать код. Но призываю всех: копируя код, будьте аккуратны и бдительны!

```
uno::Sequence< OUString >
SwXTextTable::getSupportedServiceNames(void)
{
```

```
  uno::Sequence< OUString > aRet(4);
  OUString* pArr = aRet.getArray();
  pArr[0] = "com.sun.star.document.LinkTarget";
  pArr[1] = "com.sun.star.text.TextTable";
  pArr[2] = "com.sun.star.text.TextContent";
  pArr[2] = "com.sun.star.text.TextSortable";
  return aRet;
}
```

Предупреждение PVS-Studio: V519 The 'pArr[2]' variable is assigned values twice successively. Perhaps this is a mistake. Check lines: 3735, 3736. unotbl.cxx 3736

Классический эффект последней строки. Прочти уверен, что последняя строчка была получена из предпоследней. Заменили "Content" на "Sortable", а про индекс '2' забыли.

Очень похожий случай:

```
Sequence<OUString> FirebirdDriver::getSupportedServiceNames_Static()
{
  Sequence< OUString > aSNS( 2 );
  aSNS[0] = "com.sun.star.sdbc.Driver";
  aSNS[0] = "com.sun.star.sdbcx.Driver";
  return aSNS;
}
```

Предупреждение PVS-Studio: V519 The 'aSNS[0]' variable is assigned values twice successively. Perhaps this is a mistake. Check lines: 137, 138. driver.cxx 138

Но самое ужасное, что иногда благодаря Copy-Paste ошибки размножаются. Продемонстрирую это на примере. К сожалению, код будет несколько тяжёл для чтения. Потерпите.

Итак, в программе есть вот такая функция:

```
static bool GetPropertyValue(
  ::com::sun::star::uno::Any& rAny,
  const ::com::sun::star::uno::Reference<
                  ::com::sun::star::beans::XPropertySet > &,
  const OUString& rPropertyName,
  bool bTestPropertyAvailability = false );
```

Обратите внимание, что последний аргумент 'bTestPropertyAvailability' является необязательным.

Ещё надо сказать, что такое 'sal_True':

```
#define sal_True ((sal_Bool)1)
```

Теперь собственно ошибка. Посмотрите, как вызывается функция GetPropertyValue():

```
sal_Int32 PPTWriterBase::GetLayoutOffset(....) const
{
  ::com::sun::star::uno::Any aAny;
  sal_Int32 nLayout = 20;
  if ( GetPropertyValue(
          aAny, rXPropSet, OUString( "Layout" ) ), sal_True )
    aAny >>= nLayout;

  DBG(printf("GetLayoutOffset %" SAL_PRIdINT32 "\n", nLayout));
  return nLayout;
}
```

Предупреждение PVS-Studio: V639 Consider inspecting the expression for 'GetPropertyValue' function call. It is possible that one of the closing ')' brackets was positioned incorrectly. pptx-epptbase.cxx 442

Если присмотреться, то выяснится, что одна из закрывающихся круглых скобок стоит не на своём месте. В результате, функция GetPropertyValue() в качестве последнего аргумента получает не 'sal_True', а значение по умолчанию (равное 'false').

Но это только пол беды. Дополнительно испортилась работа оператора 'if'. Условие выглядит так:

```
if (foo(), sal_True)
```

Оператор запятая возвращает свою правую часть. В результате условие всегда истинно.

Ошибка в этом коде не связана с Copy-Paste. Обыкновенная опечатка. Не там поставлена скобка. Бывает.

Печально то, что эта ошибка была размножена по другим участкам программы. Если в одном месте ошибку исправят, то высока вероятность, что в остальных местах оно останется.

Copy-Paste привёл к появлению этой ошибки ещё в 9 местах:

- epptso.cxx 993
- epptso.cxx 3677
- pptx-text.cxx 518
- pptx-text.cxx 524
- pptx-text.cxx 546
- pptx-text.cxx 560
- pptx-text.cxx 566
- pptx-text.cxx 584
- pptx-text.cxx 590

В заключении раздела 3 последних некритических предупреждения. Просто одна лишняя проверка:

```
#define CHECK_N_TRANSLATE( name ) \
  else if (sServiceName == SERVICE_PERSISTENT_COMPONENT_##name) \
    sToWriteServiceName = SERVICE_##name

void OElementExport::exportServiceNameAttribute()
{
  ....
  CHECK_N_TRANSLATE( FORM );      <<<<----
  CHECK_N_TRANSLATE( FORM );      <<<<----
  CHECK_N_TRANSLATE( LISTBOX );
  CHECK_N_TRANSLATE( COMBOBOX );
  CHECK_N_TRANSLATE( RADIOBUTTON );
  CHECK_N_TRANSLATE( GROUPBOX );
  CHECK_N_TRANSLATE( FIXEDTEXT );
  CHECK_N_TRANSLATE( COMMANDBUTTON );
  ....
}
```

Предупреждение PVS-Studio: V517 The use of 'if (A) {...} else if (A) {...}' pattern was detected. There is a probability of logical error presence. Check lines: 177, 178. elementexport.cxx 177

Ничего страшного, но недочёт. Ещё две лишние проверки можно найти здесь:

- querydesignview.cxx 3484
- querydesignview.cxx 3486

Храброе использование функции realloc()

Функция realloc() используется столь явно небезопасно, что я не рискую назвать это ошибкой. Видимо, это сознательное решение авторов. Раз не удалось выделить память, используя malloc()/realloc(), то пусть программа лучше сразу упадёт. Нечего "брыкаться". Все равно, если программа продолжит работать, вряд ли что из этого хорошего выйдет. Но не честно засчитать выданные анализатором сообщения за ложные срабатывания. Поэтому рассмотрим, что не понравилось анализатору.

Для примера изучим реализацию функции add() в классе FastAttributeList:

```
void FastAttributeList::add(sal_Int32 nToken,
  const sal_Char* pValue, size_t nValueLength )
{
  maAttributeTokens.push_back( nToken );
  sal_Int32 nWritePosition = maAttributeValues.back();
  maAttributeValues.push_back( maAttributeValues.back() +
                               nValueLength + 1 );
  if (maAttributeValues.back() > mnChunkLength)
  {
     mnChunkLength = maAttributeValues.back();
     mpChunk = (sal_Char *) realloc( mpChunk, mnChunkLength );
```

```
  }
  strncpy(mpChunk + nWritePosition, pValue, nValueLength);
  mpChunk[nWritePosition + nValueLength] = '\0';
}
```

Предупреждение PVS-Studio: V701 realloc() possible leak: when realloc() fails in allocating memory, original pointer 'mpChunk' is lost. Consider assigning realloc() to a temporary pointer. fastattribs.cxx 88

Главная беда этого кода в том, что не проверяется результат работы функции realloc(). Безусловно, ситуация, когда не удастся выделить память весьма редка. Но представим - это случилось. Тогда realloc() возвращает NULL. Далее возникнет аварийная ситуация, когда функция strncpy() начнёт копировать данные не пойми куда:

```
   mpChunk = (sal_Char *) realloc( mpChunk, mnChunkLength );
}
strncpy(mpChunk + nWritePosition, pValue, nValueLength);
```

Но анализатору не нравится другое. Предположим, что есть какой-то обработчик ошибок. И программа продолжит своё выполнение. Вот только возникает memory leak. В переменную mpChunk будет записан 0, и освободить память уже невозможно. Поясню этот паттерн ошибки чуть подробнее. Многие не задумываются и неправильно используют realloc().

Рассмотрим искусственный пример кода:

```
char *p = (char *)malloc(10);
....
p = (char *)realloc(p, 10000);
```

Если не удастся выделить память, то переменная 'p' будет "испорчена". Теперь нет никакой возможности освободить память, указатель на которую хранился в 'p'.

В таком виде ошибка очевидна. Но на практике такой код встречается достаточно часто. Анализатор выдаёт ещё 8 предупреждений на эту тему, но рассматривать их смысла нет. Все равно LibreOffice считает, что память можно выделить всегда.

Ошибки в логике

Встретилось ряд забавных ошибок в условиях. Причины, видимо, разные: невнимательность, опечатки, недостаточное знание языка.


```
void ScPivotLayoutTreeListData::PushDataFieldNames(....)
{
  ....
  ScDPLabelData* pLabelData = mpParent->GetLabelData(nColumn);

  if (pLabelData == NULL && pLabelData->maName.isEmpty())
    continue;
  ....
}
```

Предупреждение PVS-Studio: V522 Dereferencing of the null pointer 'pLabelData' might take place. Check the logical condition. pivotlayouttreelistdata.cxx 157

Логическая ошибка в условии. Если указатель нулевой, то мы его разыменуем. Как я понимаю, здесь следовало использовать оператор ||.

Аналогичная ошибка:

```
void grabFocusFromLimitBox( OQueryController& _rController )
{
  ....
  vcl::Window* pWindow = VCLUnoHelper::GetWindow( xWindow );
  if( pWindow || pWindow->HasChildPathFocus() )
  {
    pWindow->GrabFocusToDocument();
  }
  ....
}
```

Предупреждение PVS-Studio: V522 Dereferencing of the null pointer 'pWindow' might take place. Check the logical condition. querycontroller.cxx 293

Здесь наоборот, вместо '||' следовало написать '&&'.

Теперь чуть более сложное условие:

```
enum SbxDataType {
  SbxEMPTY    =  0,
  SbxNULL     =  1,
  ....
};

void SbModule::GetCodeCompleteDataFromParse(CodeCompleteDataCache& aCache)
{
  ....
  if( (pSymDef->GetType() != SbxEMPTY) ||
      (pSymDef->GetType() != SbxNULL) )
  ....
}
```

Предупреждение PVS-Studio: V547 Expression is always true. Probably the '&&' operator should be used here. sbxmod.cxx 1777

Для простоты перепишу выражение:

```
if (type != 0 || type != 1)
```

Условие всегда истинно.

Две похожих ошибки можно найти здесь:

- V547 Expression is always true. Probably the '&&' operator should be used here. sbxmod.cxx 1785
- V547 Expression is always false. Probably the '||' operator should be used here. xmlstylesexporthelper.cxx 223

Встретилось два места, где условие является избыточным. Я думаю это ошибки:

```
sal_uInt16 ScRange::ParseCols(....)
{
  ....
  const sal_Unicode* p = rStr.getStr();
  ....
  case formula::FormulaGrammar::CONV_XL_R1C1:
    if ((p[0] == 'C' || p[0] != 'c') &&
        NULL != (p = lcl_r1c1_get_col(
                            p, rDetails, &aStart, &ignored )))
```

```
    {
  ....
}
```

Предупреждение PVS-Studio: V590 Consider inspecting the 'p[0] == 'C' || p[0] != 'c'' expression. The expression is excessive or contains a misprint. address.cxx 1593

Условие (p[0] == 'C' || p[0] != 'c') можно сократить до (p[0] != 'c'). Уверен, что это ошибка и условие должно быть таким: (p[0] == 'C' || p[0] == 'c').

Идентичная ошибку можно найти в этом же файле чуть ниже:

- V590 Consider inspecting the 'p[0] == 'R' || p[0] != 'r'' expression. The expression is excessive or contains a misprint. address.cxx 1652

Пожалуй, к ошибкам в логике можно отнести ситуацию, когда указатель в начале разыменовывается, а потом только проверяется на равенство нулю. Это весьмараспространенная ошибка во всех программах. Обычно она возникает из-за невнимательности в процессе рефакторинга кода.

Типичный пример:

```
IMPL_LINK(....)
{
  ....
  SystemWindow *pSysWin = pWindow->GetSystemWindow();
  MenuBar      *pMBar   = pSysWin->GetMenuBar();
  if ( pSysWin && pMBar )
  {
    AddMenuBarIcon( pSysWin, true );
  }
  ....
}
```

Предупреждение PVS-Studio: V595 The 'pSysWin' pointer was utilized before it was verified against nullptr. Check lines: 738, 739. updatecheckui.cxx 738

Указатель 'pSysWin' разыменовывается в выражении 'pSysWin->GetMenuBar()'. Затем он проверяется на равенство нулю.

Предлагаю создателям LibreOffice также обратить внимание вот на эти места: LibreOffice-V595.txt.

И последняя, на этот раз более запутанная ситуация. Если устали, то можно пропустить это место. Рассмотрим обыкновенно перечисление:

```
enum BRC_Sides
{
    WW8_TOP = 0, WW8_LEFT = 1, WW8_BOT = 2,
    WW8_RIGHT = 3, WW8_BETW = 4
```

```
};
```

Обратите внимание, именованные константы не являются степенью двойки. Это просто числа. В том числе там есть 0.

А работают с этими константами, как будто они представляют степень двойки. Пытаются по маске выбрать и проверить отдельные биты:

```
void SwWW8ImplReader::Read_Border(....)
{
  ....
  if ((nBorder & WW8_LEFT)==WW8_LEFT)
    aBox.SetDistance(
      (sal_uInt16)aInnerDist.Left(), BOX_LINE_LEFT );

  if ((nBorder & WW8_TOP)==WW8_TOP)
    aBox.SetDistance(
      (sal_uInt16)aInnerDist.Top(), BOX_LINE_TOP );

  if ((nBorder & WW8_RIGHT)==WW8_RIGHT)
    aBox.SetDistance(
      (sal_uInt16)aInnerDist.Right(), BOX_LINE_RIGHT );

  if ((nBorder & WW8_BOT)==WW8_BOT)
    aBox.SetDistance(
      (sal_uInt16)aInnerDist.Bottom(), BOX_LINE_BOTTOM );
  ....
}
```

Предупреждение PVS-Studio: V616 The 'WW8_TOP' named constant with the value of 0 is used in the bitwise operation. ww8par6.cxx 4742

Это неправильные действия. Например, условие ((nBorder & WW8_TOP)==WW8_TOP) всегда истинно. Для пояснения подставлю числа: ((nBorder & 0)==0).

Неправильно сработает проверка и на WW8_LEFT, если в переменной nBorder будет значение WW8_RIGHT, равное 3. Подставляем ((3 & 1) == 1). Получается, что WW8_RIGHT примем за WW8_LEFT.

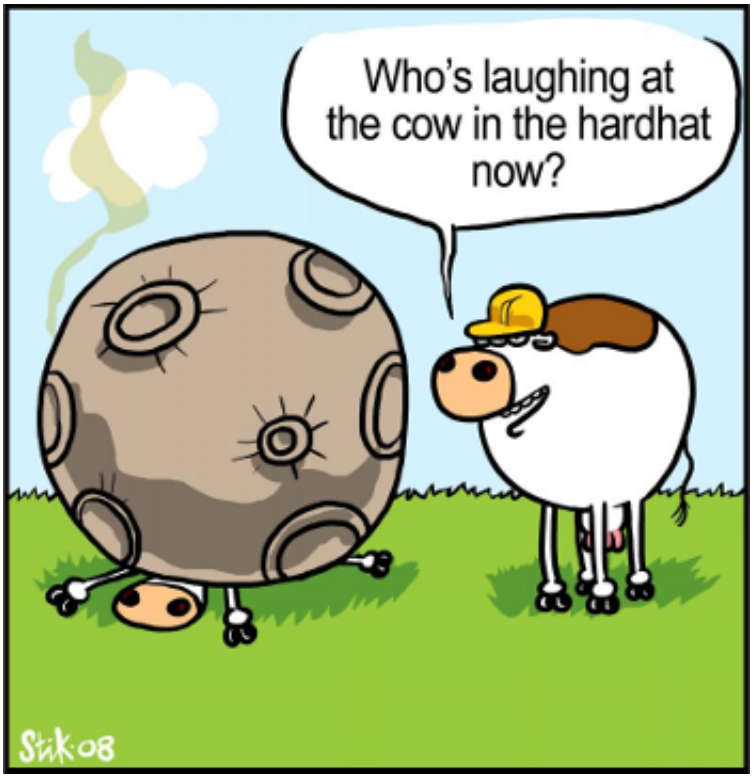
Скелет в шкафу

Анализатор время от времени обнаруживает аномальные места в коде. Это не ошибки, а хитрая задумка. Трогать их смысла нет, но посмотреть может быть интересно. Вот один из таких случаев, где анализатору не понравился аргумент функции free():

```
/* This operator is supposed to be unimplemented, but that now leads
 * to compilation and/or linking errors with MSVC2008. (Don't know
 * about MSVC2010.) As it can be left unimplemented just fine with
```

```
 * gcc, presumably it is never called. So do implement it then to
 * avoid the compilation and/or linking errors, but make it crash
 * intentionally if called.
 */
void SimpleReferenceObject::operator delete[](void * /* pPtr */)
{
  free(NULL);
}
```

Техника безопасности



Анализатор выявил ряд моментов, которые делают код программы опасным. Опасности разнообразны по своей природе, но я решил их собрать в один раздел.

```
void writeError( const char* errstr )
{
  FILE* ferr = getErrorFile( 1 );
  if ( ferr != NULL )
  {
    fprintf( ferr, errstr );
    fflush( ferr );
  }
}
```

Предупреждение PVS-Studio: V618 It's dangerous to call the 'fprintf' function in such a manner, as the line being passed could contain format specification. The example of the safe code: printf("%s",

str); unoapploader.c 405

Если в строке 'errstr' встретятся управляющие символы, то произойти может всё, что угодно. Программа может упасть, может записать в файл мусор или произойдёт что-то ещё (подробности).

Правильно будет написать так:

```
fprintf( ferr, "%s", errstr );
```

Ещё два места, где неправильно используется функция printf():

- climaker_app.cxx 261
- climaker_app.cxx 313

Теперь про опасное использование dynamic_cast.

```
virtual ~LazyFieldmarkDeleter()
{
  dynamic_cast<Fieldmark&>
    (*m_pFieldmark.get()).ReleaseDoc(m_pDoc);
}
```

Предупреждение PVS-Studio: V509 The 'dynamic_cast<T&>' operator should be located inside the try..catch block, as it could potentially generate an exception. Raising exception inside the destructor is illegal. docbm.cxx 846

При работе с ссылками, если преобразование невозможно выполнить, оператор dynamic_cast генерирует исключение std::bad_cast.

Если в программе возникает исключение, начинается свертывание стека, в ходе которого объекты разрушаются путем вызова деструкторов. Если деструктор объекта, разрушаемого при свертывании стека, бросает еще одно исключение, и это исключение покидает деструктор, библиотека C++ немедленно аварийно завершает программу, вызывая функцию terminate(). Из этого следует, что деструкторы никогда не должны распространять исключения. Исключение, брошенное внутри деструктора, должно быть обработано внутри того же деструктора.

По этой же причине опасно в деструкторы вызывать оператор new. Этот оператор при нехватке памяти сгенерирует исключение std::bad_alloc. Хорошим тоном будет обернуть его в блок try-catch.

Пример опасного кода:

```
WinMtfOutput::~WinMtfOutput()
{
  mpGDIMetaFile->AddAction( new MetaPopAction() );
  ....
}
```

Предупреждения PVS-Studio: V509 The 'new' operator should be located inside the try..catch block, as it could potentially generate an exception. Raising exception inside the destructor is illegal. winmtf.cxx 852

Прочие опасные действия в деструкторе:

- V509 The 'dynamic_cast<T&>' operator should be located inside the try..catch block, as it could potentially generate an exception. Raising exception inside the destructor is illegal. ndtxt.cxx 4886
- V509 The 'new' operator should be located inside the try..catch block, as it could potentially generate an exception. Raising exception inside the destructor is illegal. export.cxx 279
- V509 The 'new' operator should be located inside the try..catch block, as it could potentially generate an exception. Raising exception inside the destructor is illegal. getfilenamewrapper.cxx 73
- V509 The 'new' operator should be located inside the try..catch block, as it could potentially generate an exception. Raising exception inside the destructor is illegal. e3dsceneupdater.cxx 80
- V509 The 'new' operator should be located inside the try..catch block, as it could potentially generate an exception. Raising exception inside the destructor is illegal. accmap.cxx 1683
- V509 The 'new' operator should be located inside the try..catch block, as it could potentially generate an exception. Raising exception inside the destructor is illegal. frmtool.cxx 938

Кстати, раз пошла речь про operator new, то отмечу опасность вот такого кода:

```
extern "C" oslFileHandle
SAL_CALL osl_createFileHandleFromOSHandle(
  HANDLE     hFile,
  sal_uInt32 uFlags)
{
  if ( !IsValidHandle(hFile) )
      return 0; // EINVAL

  FileHandle_Impl * pImpl = new FileHandle_Impl(hFile);
  if (pImpl == 0)
  {
    // cleanup and fail
    (void) ::CloseHandle(hFile);
    return 0; // ENOMEM
  }
  ....
}
```

Предупреждение PVS-Studio: V668 There is no sense in testing the 'pImpl' pointer against null, as the memory was allocated using the 'new' operator. The exception will be generated in the case of memory allocation error. file.cxx 663

Оператор 'new' при нехватке памяти генерирует исключение. Таим образом, проверять указатель, который вернул оператор не имеет смысла. Он всегда не равен 0. При нехватке памяти не будет вызвана функция CloseHandle():

```
FileHandle_Impl * pImpl = new FileHandle_Impl(hFile);
if (pImpl == 0)
{
  // cleanup and fail
  (void) ::CloseHandle(hFile);
  return 0; // ENOMEM
}
```

Я могу ошибаться. Я не знаю проект LibreOffice. Возможно разработчики используют специальный вариант библиотек, в которых оператор 'new' не кидает исключение, а возвращает nullptr. Если это так, то прошу просто не обращать внимание на предупреждения с номером V668. Их можно отключить, чтобы они не мешались.

Если оператор new кидает исключение, то рекомендую просмотреть следующие 126 сообщений:LibreOffice-V668.txt.

Следующая опасность кроется в реализации одной из функций DllMain:

```
BOOL WINAPI DllMain( HINSTANCE hinstDLL,
                     DWORD fdwReason, LPVOID lpvReserved )
{
  ....
  CreateThread( NULL, 0, ParentMonitorThreadProc,
                (LPVOID)dwParentProcessId, 0, &dwThreadId );
  ....
}
```

Предупреждение PVS-Studio: V718 The 'CreateThread' function should not be called from 'DllMain' function. dllentry.c 308

Внутри функции DllMain() нельзя вызывать многие функции, так как это может привести к зависанию приложения или иным ошибкам. Именно к таким функциям относится CreateThread().

Ситуация с DllMain хорошо описана в статье на сайте MSDN: Dynamic-Link Library Best Practices.

Этот код может работать, но он опасен и когда-нибудь может подвести.

Встретилась ситуация, где функция wcsncpy() может привести к переполнению буфера:

```
typedef struct {
  ....
  WCHAR wszTitle[MAX_COLUMN_NAME_LEN];
  WCHAR wszDescription[MAX_COLUMN_DESC_LEN];
} SHCOLUMNINFO, *LPSHCOLUMNINFO;

HRESULT STDMETHODCALLTYPE CColumnInfo::GetColumnInfo(
  DWORD dwIndex, SHCOLUMNINFO *psci)
```

```
{
  ....
  wcsncpy(psci->wszTitle,
          ColumnInfoTable[dwIndex].wszTitle,
          (sizeof(psci->wszTitle) - 1));
  return S_OK;
}
```

Предупреждение PVS-Studio: V512 A call of the 'wcsncpy' function will lead to overflow of the buffer 'psci->wszTitle'. columninfo.cxx 129

Выражение (sizeof(psci->wszTitle) - 1) неверно. Забыли поделить на размер одного символа:

```
(sizeof(psci->wszTitle) / sizeof(psci->wszTitle[0]) - 1)
```

Последний тип ошибки, который мы рассмотрим в этом разделе, являются неработающие вызовы функций memset(). Пример:

```
static void __rtl_digest_updateMD2 (DigestContextMD2 *ctx)
{
  ....
  sal_uInt32 state[48];
  ....
  memset (state, 0, 48 * sizeof(sal_uInt32));
}
```

Предупреждение PVS-Studio: V597 The compiler could delete the 'memset' function call, which is used to flush 'state' buffer. The RtlSecureZeroMemory() function should be used to erase the private data. digest.cxx 337

Я уже много раз писал про этот вид ошибки. Поэтому опишу её буквально парой слов, а подробности можно узнать, перейдя по предложенным ссылкам.

Компилятор в праве удалить вызов функции memset(), если после её вызова обнулённая память никак не используется. Именно это здесь и произойдёт. В результате в памяти останутся какие-то приватные данные.

Подробности:

1. V597. The compiler could delete the 'memset' function call, which is used to flush 'Foo' buffer.
2. Перезаписывать память - зачем?
3. Zero and forget -- caveats of zeroing memory in C .

Прочие места, где не чистятся приватные данные: LibreOffice-V597.txt.

Всякое разное

```
Guess::Guess()
```

```
{
  language_str = DEFAULT_LANGUAGE;
  country_str = DEFAULT_COUNTRY;
  encoding_str = DEFAULT_ENCODING;
}

Guess::Guess(const char * guess_str)
{
  Guess();
  ....
}
```

Предупреждение PVS-Studio: V603 The object was created but it is not being used. If you wish to call constructor, 'this->Guess::Guess(....)' should be used. guess.cxx 56

Программист, написавший этот код, недостаточно хорошо знает язык Си++. Он хотел вызывать один конструктор из другого. Но, на самом деле, он создал временный неименованный объект. Из-за ошибки некоторые поля класса так остаются неинициализированными. Подробности.

Ещё один неудачный конструктор: camera3d.cxx 46

```
sal_uInt32 readIdent(....)
{
  size_t nItems = rStrings.size();
  const sal_Char** pStrings = new const sal_Char*[ nItems+1 ];
  ....
  delete pStrings;
  return nRet;
}
```

Предупреждение PVS-Studio: V611 The memory was allocated using 'new T[]' operator but was released using the 'delete' operator. Consider inspecting this code. It's probably better to use 'delete [] pStrings;'. profile.hxx 103

Правильно будет: delete [] pStrings;.

Ещё одно предупреждение про неправильное освобождение памяти:

- V611 The memory was allocated using 'new T[]' operator but was released using the 'delete' operator. Consider inspecting this code. It's probably better to use 'delete [] pStrings;'. profile.hxx 134

```
static const int kConventionShift = 16;
static const int kFlagMask = ~((~int(0)) << kConventionShift);
```

Предупреждение PVS-Studio: V610 Undefined behavior. Check the shift operator '<<'. The left operand '(~int (0))' is negative. grammar.hxx 56

Имеет место неопределённое поведение из-за сдвига отрицательного значения (подробности).

```
sal_Int32 GetMRest() const {return m_nRest;}

OUString LwpBulletStyleMgr::RegisterBulletStyle(....)
{
  ....
  if (pIndent->GetMRest() > 0.001)
  ....
}
```

Предупреждение PVS-Studio: V674 The '0.001' literal of the 'double' type is compared to a value of the 'long' type. Consider inspecting the 'pIndent->GetMRest() > 0.001' expression. lwpbulletstylemgr.cxx 177

Что-то здесь не так. Нет смысла сравнивать целочисленное значение с 0.001.

Неприятная путаница с типом возвращаемого значения:

```
BOOL SHGetSpecialFolderPath(
  HWND hwndOwner,
  _Out_  LPTSTR lpszPath,
  _In_   int csidl,
  _In_   BOOL fCreate
);

#define FAILED(hr) (((HRESULT)(hr)) < 0)

OUString UpdateCheckConfig::getDesktopDirectory()
{
  ....
  if( ! FAILED( SHGetSpecialFolderPathW( .... ) ) )
  ....
}
```

Предупреждение PVS-Studio: V716 Suspicious type conversion: BOOL -> HRESULT. updatecheckconfig.cxx 193

Программист решил, что SHGetSpecialFolderPath() возвращает тип HRESULT. Но, на самом деле, функция возвращает BOOL. Чтобы исправить код, следует убрать из условия макрос FAILED.

Ещё одна такая ошибка: updatecheckconfig.cxx 222

А вот здесь наоборот не хватает макроса FAILED. Так проверять статус типа HRESULT нельзя:

```
bool UniscribeLayout::LayoutText( ImplLayoutArgs& rArgs )
{
  ....
  HRESULT nRC = ScriptItemize(....);
```

```
  if( !nRC ) // break loop when everything is correctly itemized
    break;
  ....
}
```

Предупреждение PVS-Studio: V545 Such conditional expression of 'if' operator is incorrect for the HRESULT type value 'nRC'. The SUCCEEDED or FAILED macro should be used instead. winlayout.cxx 1115

Думаю, здесь следует заменить запятую на точку с запятой:

```
void Reader::ClearTemplate()
{
    if( pTemplate )
    {
        if( 0 == pTemplate->release() )
            delete pTemplate,
        pTemplate = 0;
    }
}
```

Предупреждение PVS-Studio: V626 Consider checking for misprints. It's possible that ',' should be replaced by ';'. shellio.cxx 549

Неинтересная мелочь:

```
void TabBar::ImplInit( WinBits nWinStyle )
{
  ....
  mbMirrored = false;
  mbMirrored = false;
  ....
}
```

Предупреждение PVS-Studio: V519 The 'mbMirrored' variable is assigned values twice successively. Perhaps this is a mistake. Check lines: 415, 416. tabbar.cxx 416

И здесь ещё: V519 The 'aParam.mpPreviewFontSet' variable is assigned values twice successively. Perhaps this is a mistake. Check lines: 4561, 4562. output2.cxx 4562

Неправильно магическая константа, обозначающая длину строки:

```
static bool CallRsc2(....)
{
  ....
  if( !rsc_strnicmp( ....,  "-fp=", 4 ) ||
      !rsc_strnicmp( ...., "-fo=", 4 ) ||
      !rsc_strnicmp( ...., "-presponse", 9 ) ||   <<<<----
```

```
      !rsc_strnicmp( ...., "-rc", 3 ) ||
      !rsc_stricmp( ...., "-+" ) ||
      !rsc_stricmp( ...., "-br" ) ||
      !rsc_stricmp( ...., "-bz" ) ||
      !rsc_stricmp( ...., "-r" ) ||
      ( '-' != *.... ) )
  ....
}
```

Предупреждение PVS-Studio: V666 Consider inspecting third argument of the function 'rsc_strnicmp'. It is possible that the value does not correspond with the length of a string which was passed with the second argument. start.cxx 179

Длина строки "-presponse" 10, а не 9 символов.

Странный 'break' внутри цикла:

```
OUString getExtensionFolder(....)
{
  ....
  while (xResultSet->next())
  {
    title = Reference<sdbc::XRow>(
      xResultSet, UNO_QUERY_THROW )->getString(1 /* Title */ ) ;
    break;
  }
  return title;
}
```

Предупреждение PVS-Studio: V612 An unconditional 'break' within a loop. dp_manager.cxx 100

Ещё три странных цикла:

- V612 An unconditional 'break' within a loop. svdfppt.cxx 3260
- V612 An unconditional 'break' within a loop. svdfppt.cxx 3311
- V612 An unconditional 'break' within a loop. personalization.cxx 454

Маловероятное разыменование нулевого указателя:

```
BSTR PromptNew(long hWnd)
{
  ....
  ADOConnection* piTmpConnection = NULL;

  ::CoInitialize( NULL );

  hr = CoCreateInstance(
                CLSID_DataLinks,
                NULL,
```

```
                CLSCTX_INPROC_SERVER,
                IID_IDataSourceLocator,
                (void**)&dlPrompt
                );
  if( FAILED( hr ) )
  {
    piTmpConnection->Release();
    dlPrompt->Release( );
    return connstr;
  }
  ....
}
```

Предупреждение PVS-Studio: V522 Dereferencing of the null pointer 'piTmpConnection' might take place. adodatalinks.cxx 84

Если вдруг функция CoCreateInstance() вернёт статус ошибки, то произойдёт разыменование указателя 'piTmpConnection' который равен NULL.

Микрооптимизации

Статический анализатор ни в коей мере не может быть заменой инструментам профилирования. Только профилировщик может показать, где стоит оптимизировать вашу программу.

Тем не менее статический анализатор может показать места в коде, которые можно легко улучшить. Не обязательно, что от этого программа будет работать быстрее. Но хуже точно не будет. Пожалуй, речь идёт скорее о хорошем стиле кодирования.

Посмотрим, какие рекомендации выдаёт PVS-Studio касательно микрооптимизаций.

Передача объектов по ссылке

Если объект, переданный в функцию не изменяется, то эстетично будет передать его по ссылке, а не значению. Конечно, это относится не ко всем объектам. Однако, если, например, мы имеем дело со строками, то нет смысла зря выделять память и копировать содержимое строки.

Пример:

```
string getexe(string exename, bool maybeempty) {
  char* cmdbuf;
  size_t cmdlen;
  _dupenv_s(&cmdbuf, &cmdlen, exename.c_str());
  if(!cmdbuf) {
    if (maybeempty) {
      return string();
    }
    cout << "Error " << exename << " not defined. "
      "Did you forget to source the environment?" << endl;
    exit(1);
  }
  string command(cmdbuf);
  free(cmdbuf);
  return command;
}
```

Объект 'exename' используется только для чтения. Поэтому анализатор сообщает: V813 Decreased performance. The 'exename' argument should probably be rendered as a constant reference. wrapper.cxx 18

Объявление функции можно изменить следующим образом:

```
string getexe(const string &exename, bool maybeempty)
```

Передача сложных объектов по константной ссылке обычно более эффективна и позволяет избежать проблемы "срезки". Для тех, кто недостаточно хорошо понимает суть, предлагаю обратиться к 20 правилу "Предпочитайте передачу по ссылке на const передаче по значению" из книги:

Мэйерс С. "Эффективное использование C++. 55 верных способов улучшить структуру и код ваших программ" - М.: ДМК Пресс, 2006. - 300 с.: ил. ISBN 5-94074-304-8

Ещё одной родственной диагностикой является V801. Всего, анализатор выдал 465 предупреждений, где на его взгляд можно передавать объект по ссылке: LibreOffice-V801-V813.txt.

Использование префиксного инкремента

Для итераторов операция префиксного инкремента немного быстрее. Более подробно про это можно прочитать в "Правило 6. Различайте префиксную форму операторов инкремента и декремента" из книги:

Мейерс С. Наиболее эффективное использование C++. 35 новых рекомендаций по улучшению ваших программ и проектов: Пер. с англ. - М.: ДМК Пресс, 2000. - 304 с.: ил. (Серия "Для программистов"). ISBN 5-94074-033-2. ББК 32.973.26-018.1.

Рекомендация может показаться надуманной и, что на практике между 'A++'и '++A' никакой разницы нет. Я изучил этот вопрос, провёл эксперименты и считаю, что эту рекомендацию следует применять (подробности).

Пример кода:

```
typename InterfaceMap::iterator find(const key &rKey) const
{
  typename InterfaceMap::iterator iter = m_pMap->begin();
  typename InterfaceMap::iterator end = m_pMap->end();

  while( iter != end )
  {
    equalImpl equal;
    if( equal( iter->first, rKey ) )
      break;
    iter++;
  }
  return iter;
}
```

Предупреждение PVS_Studio: V803 Decreased performance. In case 'iter' is iterator it's more effective to use prefix form of increment. Replace iterator++ with ++iterator. interfacecontainer.h 405

Выражение "iter++" стоит заменить на "++iter". Не знаю, посчитают ли разработчики рациональным потратить на это время. Если решат, что стоит поправить, то вот ещё 257 мест, где можно заменить постфиксный инкремент на префиксный: LibreOffice-V803.txt.

Проверка, что строка пустая

Чтобы узнать, что строка пустая, необязательно вычислять её длину. Пример неэффективного кода:

```
BOOL GetMsiProp(....)
{
  ....
  char* buff = reinterpret_cast<char*>( malloc( nbytes ) );
  ....
  return ( strlen(buff) > 0 );
}
```

Предупреждение PVS-Studio: V805 Decreased performance. It is inefficient to identify an empty string by using 'strlen(str) > 0' construct. A more efficient way is to check: str[0] != '\0'. sellang.cxx 49

Неэффективность в том, что нужно перебрать все символы в строке, пока не встретитсятерминальный ноль. Но достаточно проверить только один байт:

```
return buff[0] != '\0';
```

Такой код не очень красив, поэтому лучше будет завести специальную функцию:

```
inline bool IsEmptyStr(const char *s)
{
  return s == nullptr || s[0] == '\0';
}
```

Здесь появилась лишняя проверка на равенства указателя нулю. Мне это не очень нравится и можно подумать над другими вариантами. Но всё равно, эта функция будет работать быстрее чем strlen().

Другие неэффективные проверки: LibreOffice-V805.txt.

Прочее

Есть ещё несколько предупреждений анализатора, которые могут показаться интересными:LibreOffice-V804_V811.txt.

Количество ложных срабатываний

В статье я упомянул 240 предупреждений, которые мне показались интересными. Всего анализатор выдал около 1500 предупреждений общего плана (GA) 1 и 2 уровня. Это значит, что анализатор выдаёт много ложных срабатываний? Нет. Большинство предупреждений вполне по делу, но говорить про них в статье нет никакого смысла.

Время от времени мы получаем от наших пользователей положительные отзывы, в которых они говорят: "Анализатор PVS-Studio выдаёт мало ложных срабатываний, что очень удобно". Мы тоже считаем, что ложных срабатываний мало. Но как же так? В статье рассказано только о 16% сообщений. Что всё остальное? Это ложные срабатывания?

Конечно, есть ложные срабатывания. От этого никуда не денешься. Для их подавления есть рядмеханизмов. Однако, большая часть предупреждений хотя и не выявило ошибку, но указала на код, который плохо пахнет. Попробую пояснить на примерах.

Анализатор выдал **206** предупреждений V690 о том, что в классе реализован конструктор копирования, но не реализован оператор присваивания. Вот один из таких классов:

```
class RegistryTypeReader
{
public:
  ....
  inline RegistryTypeReader(const RegistryTypeReader& toCopy);
  ....
};

inline RegistryTypeReader::RegistryTypeReader(const RegistryTypeReader& toCopy)
  : m_pApi(toCopy.m_pApi)
  , m_hImpl(toCopy.m_hImpl)
  { m_pApi->acquire(m_hImpl); }
```

С большой вероятностью никакой ошибки нет. Скорее всего, operator = не используется во всех 206 классах. А вдруг используется?

Здесь программисту надо сделать выбор.

Если он считает, что код опасен, то он должен реализовать оператор присваивания или запретить его. Если на его взгляд опасности нет, то он может отключить диагностику V690, и список подозрительных мест сразу похудеет на 206 предупреждений.

Другой пример. Ранее в статье упоминался следующий подозрительный фрагмент:

```
if( pInit )
  AddMap( pInit );
if( pInit )
  AddMap( pInit2 );
```

Он выявлен с помощью диагностики V581. Но, если честно, я смотрел предупреждения V581 очень поверхностно и мог что-то пропустить. Дело в том, что их ещё 70 штук. И анализатор не виноват. Откуда ему знать, зачем писать вот так:

```
static bool lcl_parseDate(....)
{
  bool bSuccess = true;
```

```
  ....
  if (bSuccess)
  {
    ++nPos;
  }

  if (bSuccess)
  {
    bSuccess =
      readDateTimeComponent(string, nPos, nDay, 2, true);
  ....
}
```

Два раза проверяется 'bSuccess'. Вдруг второй раз следует проверить другую переменную?

Что делать с такими 70 предупреждениями вновь должен решить программист. Если он любит делать несколько одинаковых проверок, чтобы выделить какие-то логические блоки, то анализатор конечно не прав. Нужно отключить диагностику V581 и сразу исчезнут 70 предупреждений.

Если программист не столь уверен в себе, то ему придётся что-то предпринять. Можно отрефакторить код:

```
static bool lcl_parseDate(....)
{
  bool bSuccess = true;
  ....
  if (bSuccess)
  {
    ++nPos;
    bSuccess =
      readDateTimeComponent(string, nPos, nDay, 2, true);
  ....
}
```

Самое главное, что нет серьезной проблемы с ложными срабатываниями. Если человеку группа предупреждений кажется бессмысленной для его проекта, он просто отключает их и очень сильно сокращает количество предупреждений, которое ему следует изучать. Если, на его взгляд, код следует посмотреть и поправить, то это никакие не ложные срабатывания, а самые настоящие полезные предупреждения.

Примечание. Можно начать использовать анализатор, не просматривая сотни или тысячи предупреждений. Можно воспользоваться новым механизмом разметки сообщений. Надо скрыть все предупреждения, которые есть и смотреть только на сообщения, которые будут появляются в новом коде. А к ошибкам в старом коде можно будет вернуться в более свободный от срочных дел момент времени.

Заключение

Хотя как всегда в моей статье перечислена масса ошибок, недочетов и ляпов, код LibreOffice весьма качественный. Да и регулярное использование Coverity говорит о серьёзном подходе к разработке. Для проекта такого объёма ошибок весьма мало.

Что хотел сказать этой статьёй? Да в общем то ничего. Немного рекламы и не более того. Используйте статический анализатор PVS-Studio регулярно и будете находить множество ляпов на самых ранних этапах.

Я подобен корове на последней картинке. Пришёл, навалил кучу ошибок и убежал. А авторам LibreOffice их теперь разгребать. Прошу прощения. Уж такова моя работа.