Detected C-sharp Issues

Contents

Articles

Article Licenses

Detected C-sharp Issues

Note: To download all of these pages from the Wiki as a PDF, go to the book page [1].

Issue code	Description
→ CS.ASSIGN.SELF	Assignment of expression to itself
→ CS.CMP.VAL.NULL	Possible comparing value type expression with 'null'
→ CS.CONSTCOND.DO	'do' controlling expression is always true or always false
→ CS.CONSTCOND.IF	'if' controlling expression is always true or always false
→ CS.CONSTCOND.SWITCH	'switch' selector expression is constant
→ CS.CONSTCOND.TERNARY	Controlling condition in conditional expression is always true or always false
→ CS.CONSTCOND.WHILE	'while' controlling expression is always true or always false
→ CS.CTOR.VIRTUAL	Virtual member call in constructor
→ CS.EMPTY.CATCH	Empty catch clause
→ CS.FLOAT.EQCHECK	Equality check on floating point type
→ CS.FRACTION.LOSS	Possible loss of fraction
→ CS.HIDDEN.MEMBER.LOCAL.CLASS	Member is hidden by a local variable
→ CS.HIDDEN.MEMBER.LOCAL.STRUCT	Member is hidden by a local variable
→ CS.HIDDEN.MEMBER.PARAM.CLASS	Member is hidden by a parameter
→ CS.HIDDEN.MEMBER.PARAM.STRUCT	Member is hidden by a parameter
→ CS.IFACE.EMPTY	Empty interface
→ CS.LOOP.STR.CONCAT	String concatenation in a loop
→ CS.NRE.CHECK.CALL.MIGHT	Object reference may be passed to function that can dereference it after it was positively checked for null
→ CS.NRE.CHECK.CALL.MUST	Object reference will be passed to function that may dereference it after it was positively checked for null
→ CS.NRE.CHECK.MIGHT	Object reference may be dereferenced after it was positively checked for null
→ CS.NRE.CHECK.MUST	Object reference will be dereferenced after it was positively checked for null
→ CS.NRE.CONST.CALL	null is passed to function that can dereference it
→ CS.NRE.CONST.DEREF	null is dereferenced
→ CS.NRE.FUNC.CALL.MIGHT	Result of function that may return null may be passed to another function that may dereference it
→ CS.NRE.FUNC.CALL.MUST	Result of function that may return null will be passed to another function that may dereference it
→ CS.NRE.FUNC.MIGHT	Result of function that can return null may be dereferenced
→ CS.NRE.FUNC.MUST	Result of function that may return null will be dereferenced
→ CS.NRE.GEN.CALL.MIGHT	Null object reference may be passed to function that may dereference it
→ CS.NRE.GEN.CALL.MUST	Null object reference will be passed to function that may dereference it
→ CS.NRE.GEN.MIGHT	Null object reference may be dereferenced
→ CS.NRE.GEN.MUST	Null object reference will be dereferenced
→ CS.OVRD.EQUALS	Overriding 'Equals' operator on a reference type
→ CS.RLK	Resource leak

→ CS.RNRE	Suspicious dereference before null check
→ CS.UNCHECKED.CAST	Object with type Object is incorrectly cast to another object
→ CS.UNCHECKED.LOOPITER.CAST	Object with type Object is incorrectly cast to another object in a loop iterator
→ CS.WRONG.CAST	Object is incorrectly cast to another object
→ CS.WRONG.CAST.MIGHT	Object may be incorrectly cast to another object
→ CS.WRONGSIG.CMPTO	Wrong signature of 'CompareTo' method
→ CS.WRONGUSE.REFEQ	Improper usage of Object.ReferenceEquals

References

[1] http://www.klocwork.com/products/documentation/Insight-9.1/Insight-9.1:Books/Detected_C-sharp_Issues

Checkers:CS.ASSIGN.SELF

A language entity is assigned to itself.

Vulnerability and risk

Self assignment does not have any effect. Even if this is not an error on its own, it may indicate a larger error in the code.

Example 1

```
1  class Foo {
2      struct Boo {
3          public int x;
4      }
5
6      void Assigner() {
7          decimal d = 0;
8          d = d;              // defect
9          Boo boo1, boo2;
10         boo1.x = boo2.x;    // OK
11         boo2.x = boo2.x;    // defect
12     }
13 }
```

Checkers:CS.CMP.VAL.NULL

Entity of a type parameter without reference constraints in a generic type is compared with 'null'.

Vulnerability and risk

If a type parameter in a generic type has no reference contraints, it can be substituted with a value type. Comparing value type entities with 'null' always returns false and is therefore useless.

Example 1

```
1  namespace Namespace {
2      class Foo<T1> where T1: class {
3          T1 t;
4          bool checkT() {
5              return (t == null);  // OK - cannot be a value type
6          }
7      }
8      class Bar<T2> where T2: struct {
9          T2 t;
10         bool setT(T2 arg) {
11             if (arg != null)     // defect
12                 t = arg;
13         }
14     }
15 }
```

Checkers:CS.CONSTCOND.DO

The condition of a 'do' statement is always true or always false.

Example 1

```
1  class Increaser {
2      void Increase() {
3          int x = 3;
4          do {
5              x++;
6          } while (3 < 10);  // defect - the condition is constant
7          do {
8              x--;
9          } while(false);    // Ok - typical usage of 'do' constructs
when a user to organize a code block
10         do {
11             return;
12         } while(true);     // Ok - typical usage of 'do' constructs
when a user to organize an infinite loop
13     }
14 }
```

Checkers:CS.CONSTCOND.IF

The condition of an 'if' statement is always true or always false.

Example 1

```
1  class Thing {
2      void DoAction() {
3          if (sizeof(char) < 2)   // defect - the condition is
constant
4          {
5              /* ... */
6          }
7      }
8  }
```

Checkers:CS.CONSTCOND.TERNARY

The condition of a conditional expression is always true or always false.

Example 1

```
1  class IntSize {
2      void GetIntSize() {
3          return (sizeof(int) > 4 ? sizeof(int) : 4);  // defect -
the condition is constant
4      }
5  }
```

Checkers:CS.CONSTCOND.SWITCH

The condition of a 'switch' statement is constant.

Example 1

```
1  class Foo {
2      void Method() {
3          switch(3 + 2)   // defect - the condition is constant
4          {
5              /* ... */
6          }
7      }
8  }
```

Checkers:CS.CONSTCOND.WHILE

The condition of a 'while' statement is always true or always false.

Example 1

```
1  class Something {
2      void DoSomething() {
3          while (false) {    // defect
4              /* ... */
5          }
6          while (true) {     // Ok - typical usage of 'do' constructs
when a user to organize an infinite loop
7              /* ... */
8          }
9      }
10 }
```

Checkers:CS.CTOR.VIRTUAL

Constructor calls a virtual method defined in its class

Vulnerability and risk

When a virtual method is called, the actual type that executes the method is not selected until run time. When a constructor calls a virtual method, it is possible that the constructor for the instance that invokes the method has not yet executed.

Example 1

```
1  namespace NameSpace {
2      class BadlyConstructedType {
3          public BadlyConstructedType() {
4              DoBusiness();         // defect - call to a virtual
method
5          }
6          public virtual void DoBusiness() {
7              // doing business...
8          }
9      }
10
11     public class DerivedType : BadlyConstructedType {
12         public DerivedType () {}
13         public override void DoBusiness() {
14             // this method is may be called when the corresponding
object is not constructed...
15         }
16     }
17 }
```

Checkers:CS.EMPTY.CATCH

Nothing is written in a catch block. If you catch an exception, it is better to process it than ignore it.

Example 1

```
1  class FileHandler {
2      public void Open(String name) {
3          try {
4              // opening file ...
5          } catch (FileNotFoundException e) {   // defect - no
statements in the 'catch' clause
6          }
7      }
8  }
```

Language: **English**

Checkers:CS.FLOAT.EQCHECK

Two float or double values are compared using equality operators (==, !=).

Vulnerability and risk

Avoid equality checks on floating point types because of the possible inaccuracy of floating point calculations. The example below can lead to an infinite loop because x1 + 700 times ((x2 - x1) / 700) is not equal to x2, due to inaccuracy.

Example 1

```
1  class Math {
2      public static double integral(MyFunction f, double x1, double
x2) {
3          double x = x1;
4          double result = 0;
5          double step = (x2 - x1) / 700;
6          while (x != x2) {                    // defect, (x <= x2)
should be used instead
7              result = result + f.valueFor(x) * step;
8              x = x + step;
9          }
10         return result;
11     }
12 }
```

Checkers:CS.FRACTION.LOSS

Possible loss of fraction when dividing integral values and assigning result to a floating point entity.

Vulnerability and risk

When two integral values are divided, the result is also truncated to an integral value (with loss of fraction portion). When the result is assigned to a floating point variable, the intent is most probably to get a real number without fraction loss.

Example 1

```
1  class Foo {
2      int Divider(int a, long b) {
3          decimal d = a / b;   // defect
4          float f;
5          f = b % 2;           // defect
6          d = a / f;           // OK - one of the operands is not
integral
7      }
8  }
```

Checkers:CS.HIDDEN.MEMBER.LOCAL.CLASS

Class data member is hidden by a local variable.

Vulnerability and risk

When local variables hide members of containing classes, those members become accessible only through 'this' link. This behaviour is most likely not intended.

Example 1

```
1  class Zoo {
2      private int monkeys;
3      void KillMonkeys () {
4          bool monkeys = false;  // defect - local variable hides a
class member
5      }
6  }
```

Checkers:CS.HIDDEN.MEMBER.LOCAL.STRUCT

Struct data member is hidden by a local variable.

Vulnerability and risk

When local variables hide members of containing structures, those members become accessible only through 'this' link. This behaviour is most likely not intended.

Example 1

```
1  struct Zoo {
2      private int bears;
3      void KillBears () {
4          bool bears = false;  // defect - local variable hides a
struct member
5      }
6  }
```

Checkers:CS.HIDDEN.MEMBER.PARAM.CLASS

Class data member is hidden by a function parameter.

Vulnerability and risk

When function parameters hide members of containing classes, those members become accessible only through 'this' link. This behaviour is most likely not intended.

Example 1

```
1  class Zoo {
2      private int deers;
3      int numberOfSpecies;
4      void LodgeDeers (bool deers) {  // defect - method parameter
hides a class member
5          if (deers)
6              numberOfSpecies++;
7      }
8  }
```

Checkers:CS.HIDDEN.MEMBER.PARAM.STRUCT

Struct data member is hidden by a function parameter.

Vulnerability and risk

When function parameters hide members of containing structures, those members become accessible only through 'this' link. This behaviour is most likely not intended.

Example 1

```
1  struct Zoo {
2      private int snakes;
3      int numberOfSpecies;
4      void LodgeSnakes (bool snakes) {  // defect - method parameter
hides a struct member
5          if (snakes)
6              numberOfSpecies++;
7      }
8  }
```

Checkers:CS.IFACE.EMPTY

The interface does not declare any members or extend two or more other interfaces. Interfaces define members that provide a behavior or usage contract. If an empty interface extends two or more other interfaces, it combines their contracts into one. If an empty interface extends the only other interface, it does not define a contract that can be implemented, and is therefore useless.

Example 1

```
1  namespace NameSpace {
2     public interface IBadInterface  {   // defect
3     }
4     public interface IGoodInterface  {  // Ok
5         void Method();
6     }
7     public interface IOnlyParentInterface : IGoodInterface {
                  // defect
8     }
9     public interface ITwoParentsInterface : IGoodInterface,
IBadInterface {   // Ok
10    }
11 }
```

Checkers:CS.LOOP.STR.CONCAT

Using string concatenation in a loop wastes time and memory. Use StringBuilder instead.

Example 1

```
1  class Foo {
2      public string Build(int n, string x) {
3          string res = "";
4          for (int i = 0; i < n; i++) {
5              res += x;       // defect
6          }
7          return res;
8      }
9  }
```

Checkers:CS.NRE.CHECK.CALL.MIGHT

An object reference value from a path where it is positively checked for null might be passed to a function that can dereference it without checking for null.

Vulnerability and risk

Dereferencing a null object reference is a critical runtime problem that will crash the application on some operating systems and throw a runtime exception on others.

Example 1

```
1  public class A {
2    public void abc() {}
3     public void foo(A a) {
4      if (flag2)
5        return;
6      a.abc();
7     }
8
9    public A boo() {
10     if (flag3)
11       return new A();
12     return null;
13   }
14
15    public void var() {
16     A a = new A();
17     if (a != null) {
18       DoSomething();
19     }
20     if (flag) {
21       foo(a);
22     }
23    }
24
25    private void DoSomething() {}
26
27    private bool flag;
28    private bool flag2;
29    private bool flag3;
30 }
```

Klocwork produces an issue report (CS.NRE.CHECK.CALL.MIGHT) at line 21 for variable 'a'. Variable 'a' is compared with null value at line 17, and therefore may still be expected to be null if it is passed as argument 1 to function 'foo' at line 21, which may dereference it.

Checkers:CS.NRE.CHECK.CALL.MUST

An object reference value that is positively checked for null will be dereferenced either explicitly, or through a call to a function that can dereference it, without checking for null.

Vulnerability and risk

Dereferencing a null object reference is a critical runtime problem that will crash the application on some operating systems and throw a runtime exception on others.

Example 1

```
1  public class A {
2    public void abc() {}
3     public void foo(A a) {
4      if (flag2)
5        return;
6      a.abc();
7     }
8
9    public A boo() {
10     if (flag3)
11       return new A();
12     return null;
13   }
14
15    public void var() {
16     A a = new A();
17     if (a != null) {
18       DoSomething();
19     }
20     foo(a);
21    }
22
23    private void DoSomething() {}
24
25    private bool flag;
26    private bool flag2;
27    private bool flag3;
28 }
```

Klocwork produces an issue report (CS.NRE.CHECK.CALL.MUST) at line 20 for variable 'a'. Variable 'a' is compared with null value at line 17, and therefore can be expected to be null when it is passed as argument 1 to function 'foo' at line 20, which may dereference it.

Checkers:CS.NRE.CHECK.MIGHT

An object reference value from a path where it is positively checked for null might be dereferenced either explicitly, or through a call to a function that can dereference it, without checking for null.

Vulnerability and risk

Dereferencing a null object reference is a critical runtime problem that will crash the application on some operating systems and throw a runtime exception on others.

Example 1

```
1                   public class A {
2                       public void foo() {
3                           A a = new A();
4                           if (a == null)
5                               if (flag)
6                                   a.foo();
7                       }
8                       private bool flag;
9                   }
```

Klocwork produces an issue report (CS.NRE.CHECK.MIGHT) at line 6 for variable 'a'. Variable 'a' is compared with 0 value at line 4, and therefore may still be expected to be null when it is dereferenced at line 6 after flag check at line 5.

Checkers:CS.NRE.CHECK.MUST

An object reference value that is positively checked for null will be dereferenced either explicitly, or through a call to a function that can dereference it, without checking for null.

Vulnerability and risk

Dereferencing a null object reference is a critical runtime problem that will crash the application on some operating systems and throw a runtime exception on others.

Example 1

```
1                   public class A {
2                       public void foo() {
3                           A a = new A();
4                           if (a == null)
5                               a.foo();
6                       }
7                   }
```

Klocwork produces an issue report (CS.NRE.CHECK.MUST) at line 5 for variable 'a'. Variable 'a' is compared with null value at line 4, and therefore can be expected to be null when it is dereferenced at line 5.

Checkers:CS.NRE.CONST.CALL

A null object reference constant value is passed to a function that may dereference it without checking for null.

Vulnerability and risk

Dereferencing a null object reference is a critical runtime problem that will crash the application on some operating systems and throw a runtime exception on others.

Example 1

```
1                       class Param {
2                           public int par1() {
3                               return 0;
4                           }
5                       }
6                       class NPD3 {
7                           public void foo() {
8                               foo2(null);
9                           }
10                          public void foo2(Param param) {
11                              if (flag)
12                                  return;
13                              param.par1();
14                          }
15                          private bool flag;
16                      }
```

Klocwork produces an issue report (CS.NRE.CONST.CALL) at line 8. Constant null-pointer value is passed as argument 1 to function 'foo2' at line 8, which may dereference it.

Checkers:CS.NRE.CONST.DEREF

A null object reference constant is dereferenced either explicitly or through a call to a function that can dereference it.

Vulnerability and risk

Dereferencing a null object reference is a critical runtime problem that will crash the application on some operating systems and throw a runtime exception on others.

Example 1

```
1                   class Param {
2                       public int par1() {
3                           return 0;
4                       }
5                   }
6                   class NPD3 {
7                       public void foo() {
8                           foo2(null);
9                       }
10                      public void foo2(Param param) {
11                          param.par1();
12                      }
13                  }
```

Klocwork produces an issue report (CS.NRE.CONST.DEREF) at line 8. Constant null pointer is dereferenced by passing argument 1 to function 'foo2' at line 8.

Checkers:CS.NRE.FUNC.CALL.MIGHT

An object reference value from a call to a function that might return null might be passed to a function that might dereference it, without checking for null.

Vulnerability and risk

Dereferencing a null object reference is a critical runtime problem that will crash the application on some operating systems and throw a runtime exception on others.

Example 1

```
1                   public class A {
2                       public void abc() {}
3                       public void foo(A a) {
4                           if (flag2)
5                               return;
6                           a.abc();
7                       }
8
9                       public A boo() {
10                          if (flag3)
11                              return new A();
12                          return null;
13                      }
14
15                      public void var() {
16                          A a = boo();
17                          if (flag)
18                              foo(a);
19                      }
20
21                      private bool flag;
22                      private bool flag2;
23                      private bool flag3;
24                  }
```

Klocwork produces an issue report (CS.NRE.FUNC.CALL.MIGHT) at line 19 for variable 'a'. Variable 'a' is assigned to a value which might be null, and which comes from a call to function 'boo' at line 17. This variable may still be null when it is passed as argument 1 to function 'foo' at line 19, which may dereference it.

Checkers:CS.NRE.FUNC.CALL.MUST

An object reference value from a call to a function that might return null will be passed to a function that might dereference it without checking for null.

Vulnerability and risk

Dereferencing a null object reference is a critical runtime problem that will crash the application on some operating systems and throw a runtime exception on others.

Example 1

```
1  public class A {
2     public void abc() {}
3     public void foo(A a) {
4       if (flag2)
5         return;
6       a.abc();
7     }
8
9     public A boo() {
10      if (flag3)
11        return new A();
12      return null;
13    }
14
15    public void var() {
16      A a = boo();
17      foo(a);
18    }
19
20    private bool flag2;
21    private bool flag3;
22  }
```

Klocwork produces an issue report (CS.NRE.FUNC.CALL.MUST) at line 17 for variable 'a'. Variable 'a' is assigned to a value which might be null, and which comes from a call to function 'boo' at line 16. This variable will be passed as argument 1 to function 'foo' at line 17, which may dereference it.

Checkers:CS.NRE.FUNC.MIGHT

An object reference value from a call to a function that might return null might be dereferenced either explicitly or through a call to a function that can dereference it, without checking for null.

Vulnerability and risk

Dereferencing a null object reference is a critical runtime problem that will crash the application on some operating systems and throw a runtime exception on others.

Example 1

```
1                   public class A {
2                       public A foo() {
3                           if (flag)
4                               return null;
5                           return new A();
6                       }
7
8                       public void var() {
9                           A a = foo();
10                          if (flag)
11                              a.foo();
12                      }
13
14                      private bool flag;
15                  }
```

Klocwork produces an issue report (CS.NRE.FUNC.MIGHT) at line 11 for variable 'a'. If Variable 'a' is assigned to a value which might be null and which comes from a call to function 'foo' at line 9, it may still be null when it is dereferenced at line 11.

Checkers:CS.NRE.FUNC.MUST

An object reference value from a call to a function that might return null will be dereferenced either explicitly, or through a call to a function that can dereference it, without checking for null.

Vulnerability and risk

Dereferencing a null object reference is a critical runtime problem that will crash the application on some operation systems or throw a runtime exception on others.

Example 1

```
1                   public class A {
2                       public A foo() {
3                           if (flag)
4                               return null;
5                           return new A();
6                       }
7
8                       public void var() {
9                           A a = foo();
10                          a.foo();
11                      }
12
13                      private bool flag;
14                  }
```

Klocwork produces an issue report (CS.NRE.FUNC.MUST) at line 10 for variable 'a'. Variable 'a' is assigned to a value which might be null and which comes from a call to function 'foo' at line 9, and will be dereferenced at line 10.

Checkers:CS.NRE.GEN.CALL.MIGHT

An object reference value from local assignment of a null-constant or from a call to a function that will return null might be passed to a function that might dereference it, without checking for null.

Vulnerability and risk

Dereferencing a null object reference is a critical runtime problem that will crash the application on some operating systems and throw a runtime exception on others.

Example 1

```
1  public class A {
2    public void abc() {}
3     public void foo(A a) {
4      if (flag2)
5        return;
6      a.abc();
7     }
8
9     public void var() {
10     A a = null;
11     if (flag)
12       foo(a);
13    }
14
15    private bool flag;
16    private bool flag2;
17 }
```

Klocwork produces an issue report (CS.NRE.GEN.CALL.MIGHT) at line 12 for variable 'a'. Variable 'a' is explicitly assigned to null value at line 10. It may be passed as argument 1 to function 'foo' at line 12, which may dereference it.

Checkers:CS.NRE.GEN.CALL.MUST

An object reference from local assignment of a null-constant or from a call to a function that will return null will be passed to a function that can dereference it, without checking for null.

Vulnerability and risk

Dereferencing a null object reference is a critical runtime problem that will crash the application on some operating systems and throw a runtime exception on others.

Example 1

```
1  public class A {
2    public void abc() {}
3     public void foo(A a) {
4      if (flag)
5        return;
6      a.abc();
7     }
8
9     public void var() {
10     A a = null;
11     foo(a);
12    }
13
14    private bool flag;
15 }
```

Klocwork produces an issue report (CS.NRE.GEN.CALL.MUST) at line 11 for variable 'a'. Variable 'a' is explicitly assigned to null value at line 10 and will be passed as argument 1 to function 'foo' at line 11, which may dereference it.

Checkers:CS.NRE.GEN.MIGHT

An object reference value from a local assignment of a null-constant, or from a call to a function that will return null, might be dereferenced either explicitly, or through a call to a function that will dereference it, without checking for null.

Vulnerability and risk

Dereferencing a null object reference is a critical runtime problem that will crash the application on some operating systems and throw a runtime exception on others.

Example 1

```
1                   public class A {
2                       public void foo() {
3                           A c = null;
4                           if (flag)
5                               c.foo();
6                       }
7
8                       private bool flag;
9                   }
```

Null-source: explicit assignmentNull-dereferencing: explicitKlocwork produces an issue report (CS.NRE.GEN.MIGHT) at line 5 for variable 'c'. If flag is greater than 0, 'c' will be dereferenced at line 6.

Checkers:CS.NRE.GEN.MUST

An object reference value from either a local assignment of a null-constant, or a call to function that will return null, will be dereferenced either explicitly or through a call to a function that may dereference it, without checking for null.

Vulnerability and risk

Dereferencing a null object reference is a critical runtime problem that will crash the application on some operating systems and throw a runtime exception on others.

Example 1

```
1               public class A {
2                   public void foo() {
3                       A c = null;
4                       c.foo();
5                   }
6               }
```

Null-source: explicit assignmentNull-dereferencing: explicitKlocwork produces an issue report (CS.NRE.GEN.MUST) at line 4 for variable 'c'. Variable 'c' is explicitly assigned to null value at line 3 and will be dereferenced at line 4.

Checkers:CS.OVRD.EQUALS

A public or nested public reference type overloads the equality operator (Equals(object)).

Vulnerability and risk

For reference types, the default implementation of the equality operator is almost always correct. By default, two references are equal only if they point to the same object.

Example 1

```
1  public class Foo {
2      public bool Equals(object o) {     // defect
3          return true;
4      }
5
6      private class InnerClass {
7          public bool Equals(object o) { // OK - not a public class
8              return true;
9          }
10     }
11
12     public struct InnerStruct {
13         public bool Equals(object o) { // OK - not a reference type
14             return true;
15         }
16     }
17 }
```

Language: **English**

Checkers:CS.RLK

References to an object of type which implements 'IDisposable' interface are lost, but 'Dispose' method is not called.

Vulnerability and risk

Managed classes use the IDisposable interface to allow their consumers to release potentially expensive resources before the garbage collector finalizes the object.

Example 1

```
1  using System;
2  using System.IO;
3
4  namespace LeakExample
5  {
6      class Test
7      {
8          public String Run(String name)
9          {
10             StreamReader reader = new StreamReader(name);
11             String result = reader.ReadToEnd();
12             reader.Close();
13             return result;
14         }
15     }
16 }
```

Klocwork produces an issue report on the example. StreamReader class implements IDisposable interface, the implementation of Close method calls Dispose method. An object of 'StreamReader' type is allocated and its reference is assigned to 'reader' variable. If the call to 'ReadToEnd' throws an exception, control is transferred out of method 'Run', variable 'reader' goes out of scope, object referenced by it becomes lost, but related resources are not disposed.

Security Guidelines

- CWE-404:Improper Resource shutdown or release [1]

Language: **English**

References

[1] http://cwe.mitre.org/data/definitions/404.html

Checkers:CS.RNRE

This warning is reported in situations where a null object reference is dereferenced, then compared with null, and there are no object reference changes on the trace between dereferencing and checking. So, it is very likely that at dereferencing, the object reference might be null, or the null check is improper.

Vulnerability and risk

Identifies one of three things:

- A pointer that can be null by design was dereferenced without a proper check, and this will lead to a runtime error.
- A condition is written incorrectly and, therefore, the code will not work as intended.
- There is a redundant check; unnecessary code will be generated.

Example 1

```
1                   public class A {
2                       public void foo() {
3                           A a = null;
4                           a.foo();
5                           if (a == null)
6                               a = new A();
7                       }
8                   }
```

Klocwork produces an issue report (CS.RNRE) at line 4 for variable 'a'. It's dereferenced at line 4 by calling method 'foo' and then, at line 5, 'a' is checked with null in if statement.

Checkers:CS.UNCHECKED.CAST

This warning is reported in situations when one object, with type Object, is cast to another object with the possibility of lost data or even program failure.

Vulnerability and risk

Either data may be lost, or the program may fail.

Example 1

```
1                   using System;
2                   public class A {
3                      public int a;
4                   }
5                   public class ClassCastTests {
6                       public void foo() {
7                           A a;
8                           Object o = new object();
9                           a = (A)o;
10                      }
11                  }
```

Object o of class Object and object a of class A are declared on lines 5-6. Then, on line 7, Object is cast to A, which is invalid.

Checkers:CS.UNCHECKED.LOOPITER.CAST

This warning is reported in loop iterators when one object, with type Object, is cast to another object with the possibility of lost data or even program failure.

Vulnerability and risk

Either data may be lost, or the program may fail.

Example 1

```
1  using System.Collections;
2  class Boo {
3     public int a;
4  }
5  class ForEachTest {
6      static void Main(string[] args) {
7          ArrayList o = new ArrayList();
8          foreach (Boo i in o) {}
9      }
10 }
```

Object o of class ArrayList is declared on line 19. Then, on line 20, in loop iterator Object can be cast to Boo, which is invalid.

Checkers:CS.WRONG.CAST

This warning is reported in situations when one object is cast to another object with the possibility of lost data or even program failure.

Vulnerability and risk

Either data may be lost, or the program may fail.

Example 1

```
1                   public class Object1 : Object2 {
2                      public int a;
3                   }
4                   public class Object2 {
5                      public int a;
6                   }
7                   public class ClassCastTests {
8                       public void foo() {
9                           Object1 o1;
10                          Object2 o2 = new Object2();
11                          o1 = (Object1)o2;
12                      }
13                  }
```

Object o1 of class Object1 and object o2 of class Object2 are declared on lines 5-6. Then, on line 7, Object2 is cast to Object1, which is invalid.

Checkers:CS.WRONG.CAST.MIGHT

This warning is reported in situations when one object can be cast to another object with the possibility of lost data or even program failure.

Vulnerability and risk

Either data may be lost, or the program may fail.

Example 1

```
1                   public class Object1 : Object2 {
2                      public int a;
3                   }
4                   public class Object2 {
5                      public int b;
6                   }
7                   public class ClassCastTests {
8                       public void foo() {
9                           Object1 o1;
10                          Object2 o2 = new Object2();
11                          if (flag)
12                              o1 = (Object1)o2;
13                      }
14                      private bool flag;
15                  }
```

Object o1 of class Object1 and object o2 of class Object2 are declared on lines 5-6. Then, on line 8, Object2 can be cast to Object1, depending on flag on line 7, which is invalid.

Checkers:CS.WRONGSIG.CMPTO

Method 'compareTo' declared with signature different from 'public int compareTo(Object)'.

Vulnerability and risk

The intent was probably to implement the interface method of IComparable interface, but since this method has a different signature it is not the same method and will not be called when comparator is used.

Example 1

```
1  class Foo {
2      String name;
3      int compareTo(MyClass a) {
4          return name.compareTo(a.name);  // defect
5      }
6  }
```

Checkers:CS.WRONGUSE.REFEQ

The method 'Object.ReferenceEquals' is always false because it is called with a value type. Such checks are therefore not useful.

Example 1

```
1  namespace NameSpace {
2      struct S {
3      }
4      class A {
5      }
6
7      class Processor {
8          public bool CompareArgs(S s, A a, Object o) {
9              if (System.Object.ReferenceEquals(s, a)) // defect
10                 return false;
11             if (System.Object.ReferenceEquals(o, s)) // defect
12                 return false;
13             if (System.Object.ReferenceEquals(a, o)) // OK
14                 return false;
15             return true;
16         }
17     }
18 }
```

License

Klocwork Inc.
Toll-free telephone (North America): 1-866-556-2967
E-mail: sales@klocwork.com or support@klocwork.com
Web site: http://www.klocwork.com
In the U.S.:
8 New England Executive Park, Suite 180 Burlington, Massachusetts 01803 USA
In Canada:
30 Edgewater Street, Suite 114 Ottawa, Ontario Canada K2L 1V8