

CERT Verifier

Чутков Ростислав Игоревич,
444 группа

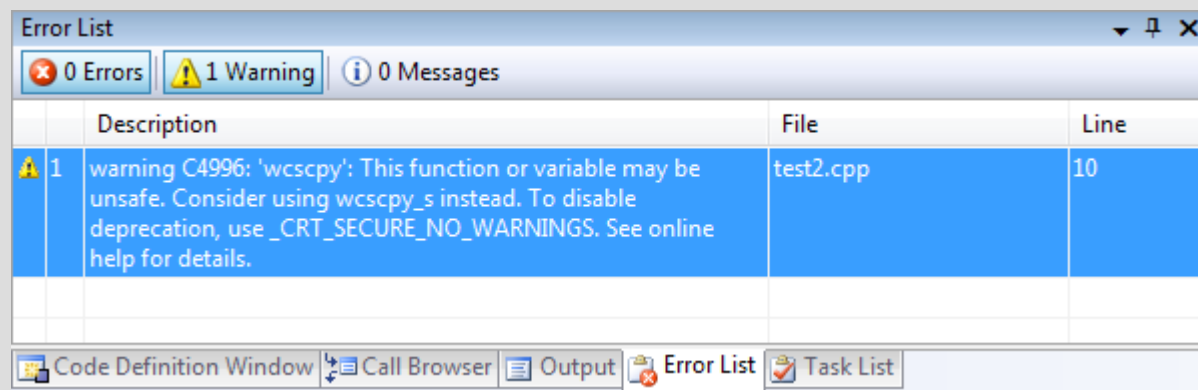
руководитель:
Штукенберг Дмитрий Григорьевич

На пути к совершенному коду

- Visual Studio 6.0
 - Performance & type conversion warnings
- Visual Studio 2003
 - Secure CRT
- Visual Studio 2005
 - Secure STL
- Visual Studio 2008
 - Strong deprecation

Integrated code verifier

- ✓ Высокая скорость работы
- ✓ Удобство навигации
- ✓ Обновляющийся стандарт безопасности



Integrated code verifier

- x Анализирует построчно
 - x Нет проверки большей части проблем
 - x Легко обмануться
 - ♦ Было
 - `wchar_t` a[20];
 - `wcscmp(a,L"TEST1234567890abcdef");`
 - ♦ Стало:
 - `wchar_t` a[20];
 - `wcscmp_s(a,10,L"TEST1234567890abcdef");`
- `pragma warning ( disable : 4199 )`

CERT index

Browser address bar: <https://www.securecoding.cert.org/confluence/pages/viewpage.action;jsessionId=E2B75B2EE7397B768DD26F>

Page Title: C++ Secure Coding Practices

CERT C++ Secure Coding Standard

Added by [Confluence Administrator](#), last edited by [David Svoboda](#) on Oct 03, 2008 ([view change](#))

Labels: (None)

C++ inherits a host of opportunities for type violations from C and adds a few of its own.
– Bjarne Stroustrup, "A rationale for semantically enhanced library languages"

- [00. Introduction](#)
- [01. Preprocessor \(PRE\)](#)
- [02. Declarations and Initialization \(DCL\)](#)
- [03. Expressions \(EXP\)](#)
- [04. Integers \(INT\)](#)
- [05. Floating Point Arithmetic \(FLP\)](#)
- [06. Arrays \(ARR\)](#)
- [07. Characters and Strings \(STR\)](#)
- [08. Memory Management \(MEM\)](#)
- [09. Input Output \(FIO\)](#)
- [10. Environment \(ENV\)](#)
- [11. Signals \(SIG\)](#)
- [12. Error Handling \(ERR\)](#)
- [13. Object Orientation \(OBJ\)](#)
- [14. Templates and the STL \(TPL\)](#)
- [49. Miscellaneous \(MSC\)](#)

Standards
[Overview](#)
[C Language](#)
[C++](#)
[Java](#)

CERT Websites
[CERT](#)
[Secure Coding](#)
[Tech Tips](#)

CERT Employment Opportunities

Related Sites
[US-CERT](#)
[www.us-cert.gov](#)
[Carnegie Mellon](#)

THE CERT C SECURE CODING STANDARD
ROBERT C. SEACORD

Secure Coding in C and C++
Robert C. Seacord

Готово

www.securecoding.cert.org

CERT rules

https://www.securecoding.cert.org/confluence/display/cplusplus/99.+The+Void+(VOID) ☆ Google ABP

Children | [Hide Children](#) | [View in Hierarchy](#)

- [ARR00-A. Avoid using the sizeof operator to determine the size of an array](#)
- [ARR30-C. Guarantee that array indices are within the valid range](#)
- [ARR31-C. Use consistent array notation across all source files](#)
- [BSC00-A. Prefer basic_string to the use of null-terminated byte strings](#)
- [DAN30-C. Do not refer to an object outside of its lifetime](#)
- [DAN31-A. Set pointers to deleted objects to NULL \(DRAFT\)](#)
- [DAN33-C. Do not use invalid iterators](#)
- [DAN34-C. Do not dereference invalid pointers](#)
- [DCL00-A. Declare immutable values using const or enum](#)
- [DCL03-A. Place const as the rightmost declaration specifier](#)
- [DCL05-A. Ensure derived classes properly conform to an "is-a" relationship with their base classes](#)
- [DCL07-CPP. Include the appropriate type information in function declarators](#)
- [DCL10-CPP. Maintain the contract between the writer and caller of variadic functions](#)
- [DCL11-CPP. Ensure type consistency when using variadic functions](#)
- [DCL12-CPP. Implement abstract data types using opaque types](#)
- [DCL31-C. Avoid self initialization](#)
- [DCL31-CPP. Declare identifiers before using them](#)
- [DCL32-C. Avoid runtime static initialization of objects with external linkage](#)
- [DCL32-CPP. Guarantee that mutually visible identifiers are unique](#)
- [DCL33-CPP. Ensure that restrict-qualified source and destination pointers in function arguments do not reference overlapping objects](#)
- [DCL35-CPP. Do not invoke a function using a type that does not match the function definition](#)
- [EXP01-A. The second operands of the logical AND and OR operators should not contain side effects](#)
- [EXP06-A. Use parentheses for precedence of operation](#)
- [EXP07-A. Operands to the sizeof operator should not contain side effects](#)
- [EXP08-A. A switch statement should have a default clause unless every enumeration value is tested](#)
- [EXP30-C. Do not cast away const](#)
- [EXP32-C. Do not access a volatile object through a non-volatile reference](#)
- [EXP33-C. Do not reference uninitialized variables](#)
- [EXP34-C. Do not depend on order of evaluation between sequence points](#)
- [EXP35-C. Ensure that the right hand operand of a shift operation is within range](#)
- [EXP35-CPP. Do not access or modify an array in the result of a function call after a subsequent sequence point](#)
- [EXP37-C. Avoid side effects in assertions](#)
- [EXP38-CPP. Do not call offsetof\(\) on bit-field members or invalid types](#)
- [EXP39-C. Don't bitwise copy class objects](#)
- [FIO01-A. Do not make assumptions about open\(\) and file creation](#)
- [FLP00-A. Consider avoiding floating point numbers when precise computation is needed](#)
- [FLP01-C. Do not use floating point variables as loop counters](#)

www.securecoding.cert.org

Примеры правил

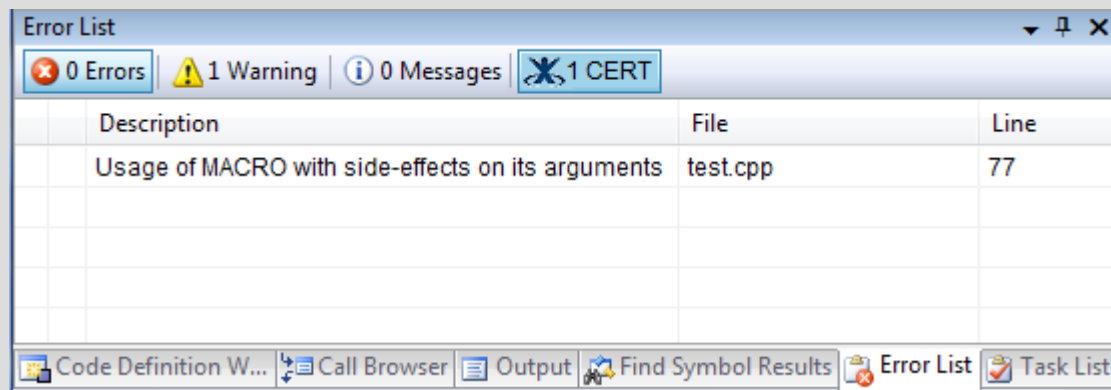
- **MSC07-CPP**. Detect and remove dead code
- **EXP05-CPP**. Do not use C-style casts
- **FLP01-CPP**. Take care in rearranging floating point expressions
- **ARR30-CPP**. Guarantee that array indexes are within the valid range
- **ARR37-CPP**. Do not add or subtract an integer to a pointer to a non-array object
- **EXP12-CPP**. Do not ignore values returned by functions

Основная цель работы

- Создание средства, упрощающего доведение проекта до CERT-compliant состояния

Интеграция с VS

- Необходимо добавить еще одну секцию в контейнере Error List
- Проверка будет происходить на этапе компиляции
- При щелчке по ошибке будет произведен переход к соответствующему месту исходного кода

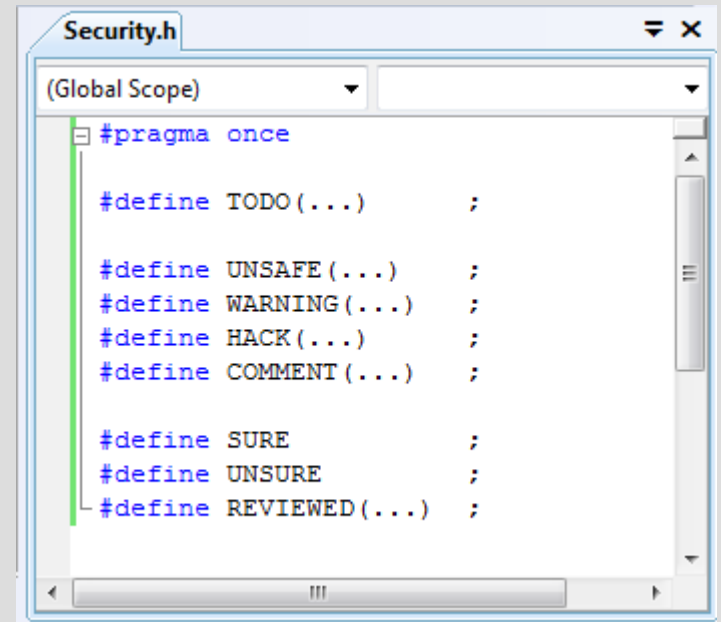


Реакция программиста

- Переписать подозрительный фрагмент кода
- Указать verifier'у, что ошибки здесь нет
 - TODO
 - SURE
 - REVIEWED
 - COMMENT
- Или ошибка есть, но будет исправлена ПОТОМ
 - UNSAFE
 - WARNING

Security.h helper

- Программа с комментариями компилируется
 - Указания verifier'у никак не влияют на сборку программ
- ✓ Возможность использовать Find All References из контекстного меню Visual Studio.



The screenshot shows a Visual Studio window titled "Security.h". The editor displays the following C code:

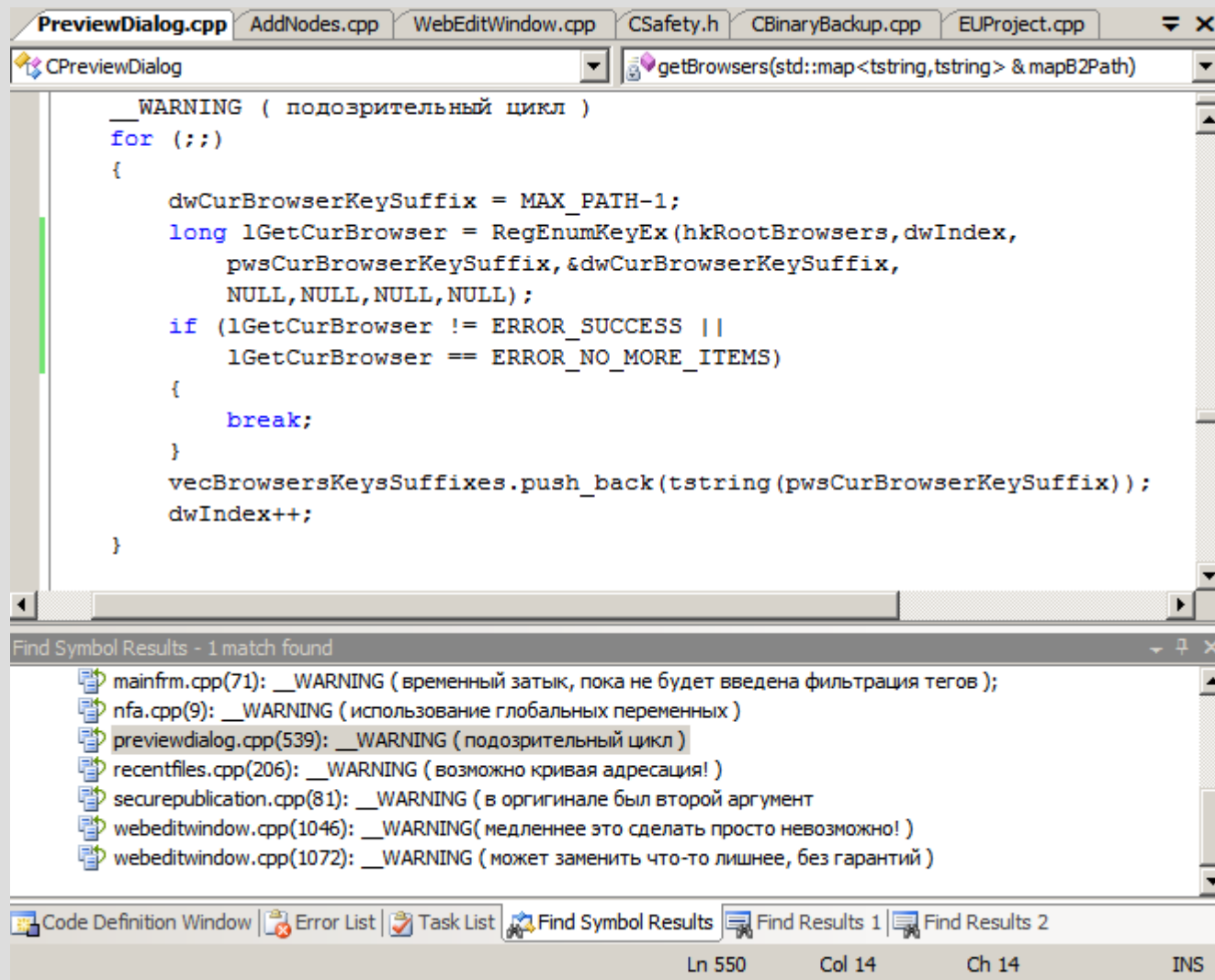
```
(Global Scope)
#pragma once

#define TODO(...) ;

#define UNSAFE(...) ;
#define WARNING(...) ;
#define HACK(...) ;
#define COMMENT(...) ;

#define SURE ;
#define UNSURE ;
#define REVIEWED(...) ;
```

Использование Security.h

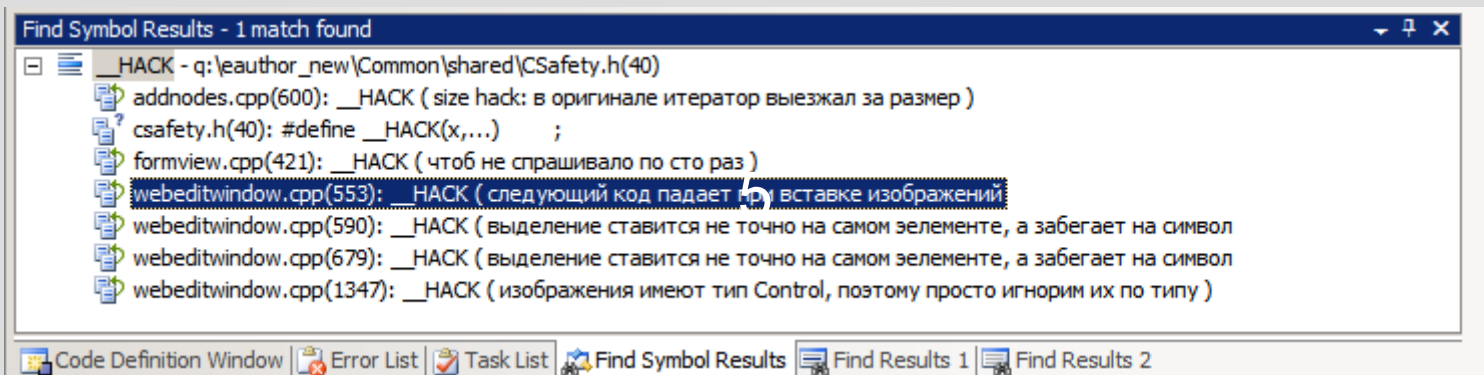


Помечаем
цикл с
“некорректным”
условием для
дальнейшего
переписывания.

Снизу видим список
всех аналогичных
проблем,
используемый для
навигации.

Контроль качества кода

- Прочитав комментарии к проблемным местам кода можно сделать вывод о состоянии продукта
 - Данную процедуру может выполнять *any technical person*, не обязательно программист C++



Контроль соответствий

```
➤ COMMENT ( заполняем начальную матрицу | 3b37d2aa1 )
➤ {
➤     p._11 = 1; p._12 = 0; p._13 = 0; p._14 = 0;
➤     p._21 = 0; p._22 = -1; p._23 = 0; p._24 = 0;
➤     p._31 = 1; p._32 = 0; p._33 = 1; p._34 = 0;
➤     p._41 = 400; p._42 = 300; p._43 = 0; p._44 = 1;
➤ }
```

- 3b37d2aa1 – контрольная сумма по коду и комментарию
- Генерируется по нажатию Ctrl-D в пределах блока
- Несоответствие CRC – тоже Warning

Существующие программы

- **Flawfinder**
 - works by using a built-in database of C/C++ functions with well-known problems
- **c++check**
 - checks memory leaks, mismatching allocation-deallocation, buffer overrun
- **Oink**
 - checks static assertions about C++ programs that can be expressed as a polymorphic dataflow of type qualifiers
- **Dehydra** (scriptable engine for code analysis)
- **Edoc++** (checking through documenting)