ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р

(проект, окончательная редакция)

Защита информации

РАЗРАБОТКА БЕЗОПАСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Безопасный компилятор языков C/C++. Общие требования

Information protection. Secure software development. Safe C/C++ compiler. General requirements

Настоящий проект стандарта не подлежит применению до его утверждения

Москва
Стандартинформ
202X

Предисловие

1 РАЗРАБОТАН Федеральным государственным бюджетным учреждением науки Институт системного программирования имени В.П. Иванникова Российской академии наук (ИСП РАН).

2 ВНЕСЕН Техническим комитетом по стандартизации ТК 362 «Защита информации»

3 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ приказом Федерального агентства по техническому регулированию и метрологии от « » ____ 202X г. №

4 ВВЕДЕН ВПЕРВЫЕ

Правила применения настоящего стандарта установлены в части 1 статьи 16 Федерального закона от 29 июня 2015 г. № 162-ФЗ «О стандартизации в Российской Федерации». Информация об изменениях к настоящему стандарту публикуется в ежегодном (по состоянию на 1 января текущего года) информационном указателе «Национальные стандарты», а официальный текст изменений и поправок – в ежемесячном информационном указателе «Национальные стандарты». В случае пересмотра (замены) или отмены настоящего стандарта соответствующее уведомление будет опубликовано в ближайшем выпуске ежемесячного информационного указателя «Национальные стандарты». Соответствующая информация, уведомление и тексты размещаются также в информационной системе общего пользования – на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет (www.gost.ru)

Содержание

Введение

Уязвимости в исходном и бинарном коде программного обеспечения являются причиной реализации угроз безопасности информации и, следовательно, фактором риска. Такие уязвимости могут появляться не только в результате ошибок, содержащихся в программах, но и в результате процедуры сборки программы, т.е. преобразования исходного кода программы в бинарный код. В частности, уязвимости могут появляться в программе в результате оптимизаций кода, выполняемых компилятором.

Для программ на языках С и С++ существенная часть уязвимостей, проявляющихся при сборке программы, возникает из-за использования в ее коде конструкций, поведение которых не регламентируется стандартом языка (т.н. неопределенное поведение или зависящее от реализации поведение). Современные компиляторы выполняют оптимизацию программы, строго полагаясь на стандарт языка, и если программа допускает нарушения стандарта, в ходе ее оптимизации компилятор может использовать неверное для этой программы предположение о ее семантике. В результате в бинарном коде такой программы могут появляться уязвимости, которых не было в исходной программе. На практике конструкции с неопределенным поведением встречаются в исходном коде программ, как системных, так и прикладных. Это создаёт угрозу безопасности информации для программного обеспечения, так как оптимизирующий компилятор может преобразовать такие конструкции в исходном коде в бинарный код, содержащий уязвимости. Указанные проблемы могут быть решены использованием компилятора, не вносящего в программу уязвимости при выполнении оптимизаций – безопасного компилятора.

Целью применения безопасного компилятора является уменьшение количества уязвимостей, которые могут быть внесены инструментами компиляции при сборке программы в ее бинарный код. Такие уязвимости не присутствуют в исходном коде программы и возникают из-за использования современными средствами сборки всех возможностей языков С и С++.

Настоящий стандарт направлен на достижение целей, связанных с предотвращением появления уязвимостей в программах, и содержит общие требования к безопасному компилятору программ на языках С и С++ (стандарты ISO/IEC 9899 и ISO/IEC 14882 соответственно), задача которого является не вносить в бинарный код программы ошибки, которых не было в исходном коде программы, которые могут появиться в ходе компиляции, в том числе в ходе выполнения оптимизаций кода программы. Стандарт

уточняет требования к мерам по разработке безопасного программного обеспечения, реализуемые при выполнении конструирования и комплексирования программного обеспечения, вводя дополнительные требования к используемым инструментальным средствам (безопасному компилятору).

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Защита информации

РАЗРАБОТКА БЕЗОПАСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Безопасный компилятор языков C/C++. Общие требования

Information protection. Secure software development. Safe C/C++ compiler. General requirements

Дата введения - _____

1 Область применения

Настоящий стандарт устанавливает общие требования к безопасному компилятору программ на языках C и C++ (стандарты ISO/IEC 9899 и ISO/IEC 14882 соответственно). Целью работы безопасного компилятора является не вносить в бинарный код программы ошибки, которых не было в исходном коде программы и которые могут появиться в ходе компиляции, в том числе в ходе выполнения оптимизаций кода программы. Стандарт задает требования к динамической компоновке и загрузке программ, выполнение которых необходимо для поддержки ряда возможностей безопасного компилятора. Стандарт уточняет требования к мерам по разработке безопасного программного обеспечения, реализуемые при выполнении конструирования и комплексирования программного обеспечения, в части требований к используемым инструментальным средствам (безопасному компилятору). Стандарт определяет требования к функциям безопасного компилятора и задает нефункциональные требования к безопасному компилятору, задает требования к методике проверки требований к безопасному компилятору.

Настоящий стандарт предназначен:

- для разработчиков компиляторов;
- для разработки безопасного программного обеспечения при выборе и оценке средств компиляции.

2 Нормативные ссылки

В настоящем стандарте использованы нормативные ссылки на следующие стандарты:

ГОСТ 19781-90 Обеспечение систем обработки информации программное. Термины и определения.

ГОСТ Р 56939-2016 Защита информации. Разработка безопасного программного обеспечения. Общие требования.

ГОСТ Р 58412-2019 Защита информации. Разработка безопасного программного обеспечения. Угрозы безопасности информации при разработке программного обеспечения.

П р и м е ч а н и е — При пользовании настоящим стандартом целесообразно проверить действие ссылочных стандартов в информационной системе общего пользования – на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет или по ежегодному информационному указателю «Национальные стандарты», который опубликован по состоянию на 1 января текущего года, и по выпускам ежемесячного информационного указателя «Национальные стандарты» за текущий год. Если заменен ссылочный стандарт, на который дана недатированная ссылка, то рекомендуется использовать действующую версию этого документа с учетом всех внесенных в данную версию изменений. Если заменен ссылочный стандарт, на который дана датированная ссылка, то рекомендуется использовать версию этого документа с указанным выше годом утверждения (принятия). Если после утверждения настоящего стандарта в ссылочный стандарт, на который дана датированная ссылка, внесено изменение, затрагивающее положение, на которое дана ссылка, то это положение рекомендуется применять без учета данного изменения. Если ссылочный стандарт отменен без замены, то положение, в котором дана ссылка на него, рекомендуется применять в части, не затрагивающей эту ссылку.

3 Термины и определения

В настоящем стандарте применены следующие термины с соответствующими определениями:

3.1

программа: Данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определенного алгоритма.

[ГОСТ 19781-90, таблица 1, пункт 1]

3.2

программное обеспечение: Совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ.

[ГОСТ 19781-90, таблица 1, пункт 2]

3.3 **разработчик программного обеспечения:** Организация, которая выполняет разработку программного обеспечения (в том числе анализ требований, проектирование, приемочные испытания) в ходе всего жизненного цикла программного обеспечения.

Примечание — Адаптировано из ГОСТ Р ИСО/МЭК 12207-2010.

3.4 **компилятор:** Инструментальное средство, выполняющее компиляцию.

Примечание — Адаптировано из ГОСТ 19781-90.

3.5 **компиляция (сборка программы):** Трансляция программы из ее исходного кода на языке программирования высокого уровня в бинарный код машинного языка.

Примечание — Адаптировано из ГОСТ 19781-90.

3.6 **безопасный компилятор:** Компилятор, в ходе компиляции не вносящий в бинарный код программы ошибки, которых не было в исходном коде программы.

Примечание — Безопасный компилятор не удаляет ошибки, имеющиеся в исходном коде программы, а пытается избежать таких преобразований кода, в результате которых после окончания компиляции в результирующем бинарном коде программы появляются ошибки, которых не было в исходном коде. Безопасный компилятор может добавлять в ходе компиляции бинарный код, который обеспечивает защиту от выполнения неопределенных конструкций в ходе выполнения программы.

П р и м е ч а н и е — Общеупотребительна практика называть компилятором C/C++ командную оболочку, выполняющую последовательность согласованных трансляций: препроцессирование, собственно компиляцию, ассемблирование, статическую компоновку. В связи с этим в настоящем стандарте при необходимости, кроме требований к безопасному компилятору, приводятся требования к соответствующим инструментальным средствам системы программирования языков C/C++.

3.7 **неопределенная конструкция:** Элемент программы на языке высокого уровня (выражение, оператор и другие), поведение которого в ходе выполнения программы не определено стандартами соответствующих языков.

П р и м е ч а н и е — Использование неопределенных конструкций в исходном коде программы является ошибочным, и при выполнении оптимизац

ионных преобразований компиляторы вправе полагаться на отсутствие в исходном коде программы конструкций с неопределенным поведением. В случае наличия таких конструкций в бинарный код программы компилятором может быть внесена ошибка. На практике в программах на языках C/C++ неопределенные конструкции встречаются часто.

3.8 **побочный эффект:** Изменение состояния среды выполнения программы (изменение ячейки памяти, файла, вызов функции, выполняющей эти действия, и другие).

П р и м е ч а н и е — Принято считать, что при выполнении программы сохранение ею результатов ее работы является побочным эффектом.

3.9 **датчик срабатывания ошибок:** Конструкция исходного или бинарного кода, которая вставляется в программу при ее сборке из исходного кода, интерпретации или двоичной трансляции и которая при обнаружении определенной ошибки в ходе выполнения программы вызывает ее аварийный останов и/или выдачу диагностической информации.

3.10 **санитайзер:** Компонент компилятора, встраивающий датчики срабатывания ошибок в генерируемый код во время компиляции программы.

3.11

исходный код программы: Программа в текстовом виде на каком-либо языке программирования.

[ГОСТ Р 58412–2019, п. 3.2]

3.12

инструментальное средство: Программа, используемая как средство разработки, тестирования, анализа, производства или модификации других программ или документов на них.

[ГОСТ Р 58412 – 2019, пункт 3.1]

3.13 **автоматическая память:** Память, хранящая локальные переменные и аргументы функции программы только во время выполнения этой функции.

3.14 **статическая память:** Память, хранящая глобальные переменные, константы, функции программы в течение всего времени выполнения программы.

3.15 **динамическая компоновка:** Процесс формирования окончательной бинарной программы для исполнения, выполняемый динамическим загрузчиком во время загрузки исполняемого файла или работы программы.

3.16 **статическая компоновка:** Процесс формирования бинарной программы (загрузочного модуля) из одного или нескольких объектных модулей путем разрешения ссылок между модулями и, при необходимости, настройки адресов.

Примечание — Адаптировано из ГОСТ 19781-90.

3.17 **исполняемый файл:** Бинарный файл, содержащий исполняемый модуль программы, который может быть запущен в среде выполнения программы.

3.18 **стандартная библиотека языка программирования:** Набор констант, переменных, процедур, классов и т.п., которые присутствуют во всех реализациях языка программирования.

Примечание — Состав стандартной библиотеки и поведение ее частей для языков C/C++ зафиксированы в стандартах соответствующих языков.

4 Общие требования к безопасному компилятору

4.1 Использование безопасного компилятора позволяет не вносить в бинарный код программы ошибки, которых не было в исходном коде программы, тем самым уменьшается вероятность реализации угроз внедрения уязвимостей программы из-за неверного использования инструментальных средств при разработке программного обеспечения (ГОСТ Р 58412-2019). Безопасный компилятор не выполняет оптимизационные преобразования программы, если обнаруживает неопределенные конструкции, и не полагается на отсутствие таких конструкций в программе. Безопасный компилятор также оповещает пользователя об обнаруженных

неопределенных конструкциях, внедряет в исполняемую программу код для динамического контроля таких конструкций (в случае их обнаружения в ходе работы программы происходит ее аварийное завершение), обеспечивает журналирование процесса трансляции для построения соответствия между исходным кодом программы и сгенерированным компилятором бинарным кодом.

4.2 Безопасный компилятор компилирует исходный код программы на языке высокого уровня в семантически эквивалентный бинарный код. В ходе компиляции безопасный компилятор выполняет преобразования исходного и бинарного кода для улучшения характеристик получаемого бинарного кода − в том числе скорости, размера и обеспечения безопасного функционирования кода.

4.3 Во время компиляции при выполнении преобразований исходного и бинарного кода безопасный компилятор должен избегать использования свойств неопределенных конструкций, а также оповещать пользователя о наличии таких конструкций в исходном коде программы и обеспечивать защиту от выполнения неопределенных конструкций во время работы программы.

4.4 В состав безопасного компилятора должны быть включены:

а) компоненты безопасных преобразований исходного и машинного кода, не использующие свойства неопределенных конструкций;

б) компоненты управления конфигурацией процесса трансляции;

в) компоненты оповещения пользователя о наличии в исходном коде неопределенных конструкций;

г) компоненты преобразования исходного кода с целью добавления в программу машинного кода, обеспечивающего защиту от потенциально ошибочного выполнения неопределенных конструкций во время работы программы (далее – динамический контроль неопределенных конструкций);

д) компоненты управления распределением автоматической и статической памяти (далее – управление распределением памяти).

4.5. Требования настоящего стандарта формулируются в отношении компилятора. Тем не менее, система программирования языков C/C++ предполагает выполнение согласованной последовательности этапов трансляции: препроцессирования, собственно компиляции, ассемблирования и статической компоновки. В связи с этим требования настоящего стандарта при технологической необходимости (в части компонентов управления распределением автоматической и статической памяти) распространяются на всю совокупность программных инструментов,

обеспечивающих трансляцию исходных текстов C/C++ в исполняемый код, включая используемые библиотеки поддержки времени выполнения.

4.6. Для соответствия требованиям настоящего стандарта разработчик ПО должен использовать для компиляции программ на языках C/C++ безопасный компилятор для этих языков.

Соответствие требованиям настоящего стандарта достигается:

а) разработкой нового компилятора;

б) доработкой уже существующего компилятора;

в) должной настройкой конфигурации уже существующего компилятора при компиляции программ.

4.7 Для дифференциации требований к безопасному компилятору выделяются три класса безопасного компилятора. Самый низкий класс – третий, самый высокий – первый. Классы безопасного компилятора определяют баланс между безопасностью генерируемого кода (меньшим количеством остающихся в коде уязвимостей либо большей защитой от имеющихся уязвимостей) и производительностью генерируемого кода (более высокие классы требуют реализации большего количества функций безопасного компилятора и ведут к большему замедлению получающейся программы). Допустимые замедления программы для каждого класса компилятора приведены в 5.1.

Примечание — В Приложении А дается перечень опций компилятора GCC, использование которых позволяет отвечать требованиям третьего класса (за исключением 5.2.1 г), д); 5.2.5, 5.2.6 б)).

4.8. Для соответствия требованиям настоящего стандарта в части реализации функций безопасности безопасный компилятор программ на языках C/C++ должен соответствовать требованиям раздела 5 настоящего стандарта.

4.9 Для соответствия требованиям настоящего стандарта в части допустимых замедлений получаемых программ безопасный компилятор программ на языках C/C++ должен соответствовать требованиям раздела 6 настоящего стандарта.

4.10. Проверка соответствия безопасного компилятора требованиям разделов 5 и 6 проводится согласно требованиям, приведенным в разделе 7.

5 Требования к функциям безопасного компилятора

5.1 Безопасный компилятор соответствующего класса должен реализовывать функции безопасности, приведенные в таблице 1 для каждого класса.

Таблица 1 – Функции безопасного компилятора

Наименование функции безопасности	Класс компилятора		
	3	2	1
1. Безопасные преобразования исходного и машинного кода.	+	+	=
2. Оповещение пользователя в ходе сборки программы о наличии в исходном коде неопределенных конструкций	+	+	=
3. Динамический контроль неопределенных конструкций	–	–	+
4. Управление распределением автоматической и статической памяти	–	–	+
5. Управление конфигурацией процесса трансляции	+	+	+
6. Журналирование процесса трансляции	+	=	=
7. Управление механизмами повышения защищенности	+	=	=

Обозначение «+» в строке функции безопасности указывает на наличие требований, предъявляемых к данной функции безопасности, для соответствующего класса безопасного компилятора.

Обозначение «−» в строке функции безопасности указывает на отсутствие требований, предъявляемых к данной функции безопасности, для соответствующего класса безопасного компилятора.

Обозначение «=» означает, что требования соответствуют требованиям, предъявляемым к предыдущему классу безопасного компилятора.

5.2 Безопасный компилятор 3 класса должен реализовывать следующие функции.

5.2.1 Для реализации функции по безопасным преобразованиям исходного и машинного кода безопасный компилятор при выполнении преобразований должен предполагать следующее:

а) при выполнении операций над целыми числами может возникнуть знаковое целочисленное переполнение во всех случаях, когда с помощью анализа значений переменных и условий выполнения операций не доказано обратное;

б) значения указателей различающихся типов могут совпадать при операциях чтения и записи данных через эти указатели (далее – разыменование указателя);

в) при разыменовании указателя данный указатель может содержать нулевое значение во всех случаях, когда с помощью анализа условий выполнения разыменования указателя не может быть доказано обратное;

г) при выполнении целочисленных операций деления и взятия остатка значение делителя может быть равно нулю, если с помощью анализа возможных значений переменных и условий выполнения операций над целыми числами не доказано обратное;

д) при выполнении операций побитового сдвига значение второго аргумента (величина сдвига) может быть отрицательным либо оказаться больше или равным ширине типа (количеству занимаемых битов) первого аргумента операции (сдвигаемого значения), если с помощью анализа возможных значений переменных и условий выполнения операций над целыми числами не доказано обратное.

5.2.2 Для реализации функции по безопасным преобразованиям исходного и машинного кода безопасный компилятор должен обеспечивать следующие свойства генерируемого кода:

а) при вызове функций стандартной библиотеки языка С (С++) должен быть реализован механизм защиты от переполнения буфера постоянного (заданного константой во время компиляции) размера;

б) в процессе трансляции в машинный код должен быть добавлен механизм обнаружения записи, выходящей за границу кадра стека, кроме случаев, когда применение этого механизма невозможно (например, в коде ядра или загрузчика операционной системы, в коде реализации самого этого механизма);

в) генерируемый код должен быть позиционно-независимым, кроме случаев, когда это не применимо (например, в коде ядра операционных систем). При этом код, который в дальнейшем будет загружаться динамическим загрузчиком должен быть позиционно-независимым как для случая сборки программы для получения динамических библиотек, так и для получения исполняемого файла. Если из-за ограничений операционной системы данное требование выполнить невозможно, генерируемый код должен допускать перебазирование (загрузку по произвольным адресам);

г) вызовы функций стандартной библиотеки fprintf, fwprintf, printf и wprintf не могут быть заменены на эквивалентные им последовательности машинных инструкций (для функций из этого списка требование распространяется также на компиляторы более высоких 2 и 1 классов защиты);

д) вызовы функций стандартной библиотеки memcpy, memmove, memset, strcat, strcpy, strncat, strncpy, snprintf, sprintf, swprintf, wcscat, wcscpy, wcsncat, wcsncpy, wmemcpy, wmemmove и wmemset не могут быть заменены на эквивалентные им последовательности машинных инструкций (данное требование действует только на компиляторы 3 класса и не переносится на более высокие классы, так как во втором классе присутствуют требования, касающихся любых операций записи в память).

Примечание — После встраивания тела функции (последовательности машинных инструкций) в место ее вызова в ходе дальнейших оптимизаций некоторые инструкции могут быть удалены. Безопасный компилятор должен сохранить все инструкции.

5.2.3 Для реализации функции по оповещению пользователя о наличии в исходном коде неопределенных конструкций безопасный компилятор должен в процессе трансляции выдавать предупреждения при наступлении следующих событий:

а) обнаружена переменная, размещенная в автоматической памяти, которая хранится на машинном регистре перед вызовом функции longjmp, и ее значение используется после вызова этой функции путем загрузки из соответствующего регистра;

б) обнаружена операция чтения или записи в массив по адресу, который находится за пределами выделенной памяти для этого массива;

в) обнаружена целочисленная операция деления или взятия остатка, при которой значение делителя равно нулю;

г) обнаружена операция побитового сдвига, в которой второй аргумент операции сдвига (величина сдвига) отрицателен либо больше или равен ширине типа (количеству занимаемых битов) первого аргумента операции (сдвигаемого значения).

5.2.4 Для реализации функции по управлению конфигурацией процесса трансляции безопасный компилятор при наличии небезопасного режима работы должен содержать механизм включения и отключения функциональности, реализующей функции третьего класса.

5.2.5 Для реализации функции по журналированию процесса трансляции безопасный компилятор должен фиксировать в базе данных компиляции для каждой единицы трансляции:

а) рабочую директорию компиляции;

б) имя файла, представляющего единицу трансляции;

в) хэш-код (контрольная сумма) файла, представляющего единицу трансляции;

г) выполняемую команду компиляции целиком или полный список аргументов компиляции;

д) имя выходного файла, получаемого в результате компиляции;

е) хэш-код (контрольная сумма) выходного файла;

ж) в случае включения в единицу трансляции файлов, например, при препроцессировании, имя каждого включаемого файла и его хэш-код.

Для ведения базы данных компиляции должен использоваться текстовый формат обмена данными JSON (RFC 8259), совместимый с форматом базы данных компиляции компилятора clang. Пример записи:

```
[
  { "directory": "/opt/ejudge",
    "command": "/usr/bin/gcc –g -Wall –std=gnu99 –Wno-pointer-sign
-c -o file.o file.c",
    "file": "file.c" },
  ...
]
```

База данных компиляции состоит из массива записей, описывающих трансляцию одной единицы трансляции.

В базу данных компиляции должна вноситься информация, позволяющая идентифицировать программы, задействованные в трансляции исходных текстов в исполняемый код, а также все используемые настройки этих программ и конфигурационные файлы.

Должна поддерживаться возможность по завершении компиляции программы зафиксировать состояние базы данных компиляции, вычислив контрольную сумму содержимого базы данных.

Для подсчета контрольных сумм как единиц трансляции, так и самой базы данных компиляции должна использоваться функция хэширования, определенная межгосударственным стандартом ГОСТ 34.11-2017.

5.2.6 Для реализации функции по управлению механизмами защищенности безопасный компилятор должен дополнительно включать в ходе компиляции, реализованные в нем либо совместно с операционной системой механизмы повышения защищенности кода (помимо обязательных для реализации свойств кодогенерации из 5.2.2 б), в)). Например, такими механизмами являются:

а) механизм рандомизации размещения кода в адресном пространстве;

б) механизм контроля за целостностью потока управления;

в) механизм контроля за целостностью стека и пр.

5.3 Безопасный компилятор 2 класса должен реализовывать следующие функции.

5.3.1 Для реализации функции по безопасным преобразованиям исходного и машинного кода безопасный компилятор должен удовлетворять всем требованиям, установленным 5.2.1, и всем требованиям, установленным 5.2.2, за исключением перечисления д).

Также безопасным компилятором должны обеспечиваться следующие свойства генерируемого кода:

а) при наличии хотя бы одного обращения к участку памяти (чтения), должны быть сохранены (оставлены) все побочные эффекты для всех операций записи этого участка (как предшествующих, так и последующих);

б) все переменные из автоматической памяти, для которых не задано значение инициализации, должны иметь нулевое значение инициализации.

Дополнительно при выполнении преобразований безопасный компилятор должен предполагать следующее:

а) при выполнении операции побитового сдвига в случае, когда значение второго аргумента (величина сдвига) отрицательно либо больше или равно ширине типа (количеству занимаемых битов) первого аргумента операции (сдвигаемого значения), результат вычисления может зависеть от особенностей процессорной архитектуры и не может быть заменен на фиксированное значение;

б) при выполнении операций над векторными машинными регистрами и памятью используемые адреса объектов векторных типов данных в памяти могут иметь произвольные значения (не кратные размеру типов);

в) при выполнении операций сложения и вычитания со значениями адресов (далее – адресная арифметика) результаты операций могут иметь произвольные значения.

5.3.2 Для реализации функции по оповещению пользователя о наличии в исходном коде неопределенных конструкций безопасный компилятор должен в процессе трансляции выдавать ошибку и аварийно остановить выполнение трансляции при наступлении следующих событий:

а) любом из событий, перечисленных в 5.2.3;

б) в исходном коде программы используется функция gets.

5.3.3 Для реализации функции по управлению конфигурацией процесса трансляции безопасный компилятор при наличии менее безопасных режимов работы должен содержать механизм включения и отключения функциональности, реализующей функции второго или одновременно третьего и второго классов.

5.4 Безопасный компилятор 1 класса должен реализовывать следующие функции.

5.4.1 Для реализации функции по безопасным преобразованиям исходного и машинного кода безопасный компилятор должен удовлетворять 5.3.1.

5.4.2 Для реализации функции по оповещению пользователя о наличии в исходном коде неопределенных конструкций безопасный компилятор должен удовлетворять 5.3.2.

5.4.3 Для реализации функции по динамическому контролю неопределенных конструкций безопасный компилятор должен обеспечить добавление в процессе трансляции программы машинного кода, обеспечивающего защиту от ошибочного выполнения неопределенных конструкций во время работы программы. При выполнении неопределенных конструкций работа программы должна быть аварийно остановлена. В случае, когда аварийная остановка является неприемлемой (для некоторых видов системного программного обеспечения, например, ядра операционной системы, гипервизоров 1-го типа и др.), допускается применение иных механизмов защиты. Должны поддерживаться следующие неопределенные конструкции:

а) операция записи значения в переменную булевого типа, которое отлично от лжи и истины;

б) операция преобразования значения переменной вещественного типа с плавающей запятой в переменную другого типа (целочисленного или с плавающей запятой), приводящая к целочисленному или вещественному переполнению;

в) операция целочисленного сдвига, в которой второй аргумент операции сдвига (величина сдвига) отрицателен либо не меньше ширины типа (количества занимаемых битов) первого аргумента операции (сдвигаемого значения);

г) операция со знаковыми целочисленными переменными, результат которой не может быть представлен в результирующей переменной;

д) операция чтения или записи по указателю, значение которого не кратно размеру типа хранящегося по этому адресу объекта;

е) операция чтения или записи по указателю, значение которого равно нулю;

ж) операция чтения или записи в массив по адресу, находящемуся за пределами выделенной для массива памяти, если размер памяти может быть определен в процессе трансляции;

з) операция адресной арифметики, приводящая к целочисленному переполнению;

и) операция вызова по указателю функции, формальный тип которой не соответствует фактическому типу указателя;

к) окончание работы функции, имеющей возвращаемое значение, без выполнения операции возврата;

л) операция вызова встроенной функции компилятора с недопустимыми значениями аргументов;

м) операция вызова встроенных функций компилятора, не подлежащих вызову;

н) операция целочисленного деления или взятия остатка, значения делителя в которой равно нулю;

о) выделение в автоматической памяти массива переменной длины, размер которого не является положительным.

5.4.4 Для реализации функции по управлению распределением памяти безопасный компилятор должен обеспечить возможность при каждом запуске программы на выполнение выполнять уникальное (особое для данного запуска) распределение в памяти машинного кода функций в процессе динамической компоновки программы.

Для реализации данного требования, наряду с его поддержкой в компиляторе, также необходима соответствующая поддержка со стороны динамического загрузчика операционной системы.

При наличии в операционной системе динамического загрузчика данное требование не распространяется на ядро операционной системы, а также системные программы, для запуска которых не используется динамический загрузчик (в том числе на сам динамический загрузчик).

При сборке программ для архитектур, которые не предусматривают наличие динамического загрузчика, а также для операционных систем, в которых невозможна реализация собственного динамического загрузчика, данное требование заменяется требованием статического уникального распределения памяти.

Во всех случаях, когда организация динамического распределения в памяти невозможна, безопасный компилятор должен обеспечить возможность уникального распределения автоматической и статической памяти программы полностью на этапе её компиляции. Компилятор должен поддерживать следующие возможности при управлении распределением:

а) распределение памяти должно конфигурироваться целочисленным параметром (ключом), при этом для одинаковых задаваемых ключей и одного и того же исходного кода компилятор должен создавать одинаковое распределение памяти;

б) возможность уникального распределения локальных переменных функций для каждого процесса трансляции;

в) возможность уникального распределения памяти для машинного кода функций для каждого процесса трансляции;

г) возможность уникального распределения глобальных переменных программного модуля в соответствующей области памяти для каждого процесса трансляции.

5.4.5 Для реализации функции по управлению конфигурацией процесса трансляции безопасный компилятор при наличии менее безопасных режимов работы должен содержать механизм включения и отключения функциональности, реализующей функции первого, первого и второго или всех трех классов.

6 Требования к производительности генерируемого кода

Замедление производительности генерируемого безопасным компилятором кода (времени работы кода) по сравнению с производительностью кода, генерируемого в небезопасном режиме работы этого же компилятора (в котором не реализуются функции 5.1), должно составлять:

а) для 3 класса – не более 20%,

б) для 2 класса – не более 50%,

в) для 1 класса – не более 200%.

7 Проверка выполнения требований к безопасному компилятору

7.1 Соответствие безопасного компилятора требованиям, изложенным в разделах 5 и 6 настоящего стандарта, проверяется с помощью набора квалификационных тестов. Запуск тестов должен осуществляться с помощью автоматизированной системы тестирования.

7.2 Соответствие безопасного компилятора требованиям, изложенным в разделе 6 настоящего стандарта (проверка производительности сгенерированного компилятором кода), проверяется с помощью сравнения производительности сгенерированного машинного кода с производительностью кода, созданного эталонным компилятором в небезопасном режиме работы, при условии включения стандартного режима оптимизаций компилятора (например, для компилятора GCC такой режим соответствует опции –O2).

Для выполнения проверки система тестирования производительности компилирует наборы тестов на производительность два раза: первый раз используется эталонный компилятор, в котором отключены все функции безопасности раздела 5, во второй раз – безопасный компилятор со всеми включенными функциями безопасности. После компиляции оба варианта тестовых программ запускаются на выполнение на одинаковых входных данных на платформе, на которой предполагается функционирование собираемой компилятором программы, и в зависимости от времени выполнения система генерирует интегральную оценку производительности. Разница в оценке производительности должна соответствовать требованиям раздела 6 настоящего стандарта.

7.3 Соответствие безопасного компилятора требованиям раздела 5 настоящего стандарта проверяется с помощью тестов следующих типов:

а) ассемблерные тесты, которые проверяют, что в скомпилированной программе будут находиться или отсутствовать определенные конструкции;

б) тесты времени выполнения, которые проверяют какой-либо результат работы скомпилированной программы (обычно возвращаемое значение или данные стандартного потока вывода);

в) тесты, которые проверяют работу санитайзера: во время работы программы датчики срабатывания ошибок, встроенные санитайзером во время компиляции, должны обнаружить операции с неопределенным поведением и аварийно завершить программу;

г) тесты на наличие предупреждений, которые проверяют, что в процессе компиляции будут выданы требуемые предупреждения.

7.4 Безопасный компилятор можно считать удовлетворяющим выбранному классу, если он проходит все тесты из группы, соответствующей этому классу, а также все тесты из групп, соответствующих менее безопасным классам.

Приложение А

(справочное)

Таблица соответствия функций безопасности компилятора 3 класса и опций компилятора GCC

Номер пункта стандарта	Опция компилятора GCC
5.2.1 а)	-fwrapv -fwrapv-pointer
5.2.1 б)	-fno-strict-aliasing
5.2.1 в)	-fno-delete-null-pointer-checks
5.2.2 а)	-D_FORTIFY_SOURCE=2
5.2.2 б)	-fstack-protector-strong
5.2.2 в)	-fPIE, -fPIC, -fpic
5.2.2 г)	-fno-builtin-fprintf, -fno-builtin-fwprintf, -fno-builtin-printf, -fno-builtin-snprintf, -fno-builtin-sprintf, -fno-builtin-swprintf, -fno-builtin-wprintf
5.2.2 д)	-fno-builtin-memcpy, -fno-builtin-memmove, -fno-builtin-memset, -fno-builtin-strcat, -fno-builtin-strcpy, -fno-builtin-strncat, -fno-builtin-strncpy, -fno-builtin-wcscat, -fno-builtin-scwcpy, -fno-builtin-wcsncat, -fno-builtin-wcsncpy, -fno-builtin-wmemcpy, -fno-builtin-wmemmove, -fno-builtin-wmemset
5.2.3 а)	-Wclobbered
5.2.3 б)	-Warray-bounds
5.2.3 в)	-Wdiv-by-zero
5.2.3 г)	-Wshift-count-negative, -Wshift-count-overflow
5.2.6 а)	-fpic
5.2.6 в)	-fstack-protector

Приложение Б
(справочное)
Пример построения квалификационных тестов для проверки требований раздела 5 стандарта

1) пункт 5.2.1 в) – в приведенном тесте безопасный компилятор не должен из разыменования p->z делать вывод, что p – ненулевой указатель, и удалять вызов функции bar(). Этот вызов должен присутствовать в ассемблерном коде, сгенерированным безопасным компилятором.

```
struct point {
  int x,y,z;
} *p;

extern void ret_point(const char *v);
extern void g(int x);
extern void bar(int x);

int c;
int result = 2;

int f(const char *v) {
  // устанавливает значение p
  ret_point(v);

  c = p->z;
  if (!p) {
    result--;
    bar(result);
    return -1;
  }
  g(c);
  return 0;
}

int main(void) {
  const char s[] = "0123";
  result = result + f(s);
```

```
  return result;
}
```

2) пункт 5.2.1 б) – в приведенном тесте безопасный компилятор из разницы в базовых типах указателей x и y не должен делать вывод, что они указывают на разные участки памяти, и всегда возвращать 0 из функции foo. Тем самым после компиляции указанного теста запуск скомпилированного кода должен выполняться успешно (функция main должна возвращать 0 на архитектуре x86).

```
int foo(int *x, long *y) {
  *x = 0;
  *y = 1;
  return *x;
}

int main(void) {
  long l;
  return 1 - foo((int *)&l, &l);
}
```

3) пункт 5.3.1 а) – в приведенном тесте безопасный компилятор должен сохранить побочные эффекты обоих вызовов memset. Оба вызова должны присутствовать в ассемблерном коде, сгенерированным безопасным компилятором.

```
#include <string.h>

void f(char *);

int main(void) {
  char a[1000];
  f(a);
  memset(a, '*', 1000);
  memset(a, '/', 1000);
  return 0;
}
```

4) пункт 5.3.1 б) – в приведенном тесте безопасный компилятор должен инициализировать нулями массив d на каждой итерации цикла. Это проверяется во внешней функции check_arr, а функции fill и use заполняют и используют массив d.

```
int check_arr(int *d, unsigned n);
void fill(int *d, unsigned n);
void use(int *d, unsigned n);

int main(int argc, char **argv)
{
  unsigned n;
  if (argc == 1)
    argc = 32;
  else
    return 2;
  n = argc;

  while (--argc)
    {
      int d[n];
      if (check_arr(d, argc))
        return 1;
      fill(d, argc);
      use(d, argc);
    }
  return 0;
}
```

5) пункт 5.4.3 ж) – в приведенном тесте безопасный компилятор должен обеспечить обнаружение ошибки выхода за границы выделенной памяти во время выполнения программы для случая, когда размер памяти может быть определен во время компиляции.

```
int foo(int x) {
     return x - 3;
}
```

```
int main(int argc, char **argv) {
     int a[4] = {0, 1, 2, 3};
     return (argc - 1) * a[foo(argc + 1)];
}
```

6) пункт 5.4.3 м) – в приведенном тесте безопасный компилятор должен обеспечить обнаружение ошибки целочисленного деления на ноль во время выполнения программы. Тест запускается на наборах входных данных, включающих деление на 0 и деление INT_MIN на -1.

```
#include <limits.h>

volatile int t;

int main(int argc, char **argv) {
  int x = argc, y = argc - 1;
  // Компилятор не имеет права удалять запись в volatile переменную.  Это делает
  // результат деления "полезным" и не позволяет удалить операцию деления.
  t = x / y;
  x = INT_MIN;
  y = -argc;
  t = x / y;
  return 0;
}
```

Библиография

[1]	ГОСТ 19781-90	Обеспечение систем обработки информации программное. Термины и определения
[2]	ГОСТ Р ИСО/МЭК 12207-2010	Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств
[3]	ГОСТ 34.11-2018	Информационная технология. Криптографическая защита информации. Функция хэширования
[4]	ГОСТ Р 56938-2016	Защита информации. Защита информации при использовании технологий виртуализации. Общие положения