ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р 71207— 2024

Защита информации

РАЗРАБОТКА БЕЗОПАСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Статический анализ программного обеспечения. Общие требования

Издание официальное

Москва
Российский институт стандартизации
2024

Предисловие

1 РАЗРАБОТАН Федеральной службой по техническому и экспортному контролю (ФСТЭК России), Федеральным государственным бюджетным учреждением науки «Институт системного программирования им. 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 18 января 2024 г. № 25-ст

4 ВВЕДЕН ВПЕРВЫЕ

Правила применения настоящего стандарта установлены в статье 26 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.rst.gov.ru)

Содержание

Введение

Уровень безопасности создаваемого программного обеспечения (ПО) связан с наличием уязвимостей и критических ошибок в программном коде. В свою очередь, количество найденных ошибок во многом определяется уровнем используемых в ходе разработки инструментов. Для этого современные инструменты поиска ошибок, такие как статические и динамические анализаторы, обеспечивают определенный набор характеристик анализа: полноту и качество, скорость, удобство для пользователя, применимость к современным языкам программирования и др.

Настоящий стандарт устанавливает общие требования к внедрению и выполнению статического анализа ПО, а также исходные данные, необходимые для его выполнения. Настоящий стандарт устанавливает требования к методам статического анализа, инструментам анализа (статическим анализаторам) и к специалистам, участвующим в анализе. Настоящий стандарт устанавливает методику проверки устанавливаемых требований к инструментам анализа.

Настоящий стандарт входит в комплекс стандартов, направленных на достижение целей, связанных с предотвращением появления и/или устранением уязвимостей программ, содержит общие требования по проведению статического анализа ПО и применяется совместно с ГОСТ Р 56939.

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Защита информации

РАЗРАБОТКА БЕЗОПАСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Статический анализ программного обеспечения. Общие требования

Information protection. Secure software development.
Software static analysis. General requirements

Дата введения — 2024—04—01

1 Область применения

Настоящий стандарт описывает порядок внедрения и выполнения статического анализа, устанавливает требования к его выполнению, классификацию ошибок, находимых статическими анализаторами, а также требования к методам анализа, к инструментам статического анализа, к специалистам, участвующим в выполнении анализа, а также к методике проверки статических анализаторов на соответствие требованиям данного стандарта.

Настоящий стандарт предназначен:

- для разработчиков статических анализаторов;
- для разработчиков средств защиты информации, средств обеспечения безопасности информационных технологий;
- для разработчиков программного обеспечения (ПО).

2 Нормативные ссылки

В настоящем стандарте использованы нормативные ссылки на следующие стандарты:

ГОСТ 19.102 Единая система программной документации. Стадии разработки

ГОСТ 28397 (ИСО 2382-15—85) Языки программирования. Термины и определения

ГОСТ Р 50922 Защита информации. Основные термины и определения

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

ГОСТ Р ИСО/МЭК 12207 Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств

Примечание — 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 стандарта с учетом всех внесенных в данную версию изменений. Если заменен ссылочный стандарт, на который дана датированная ссылка, то рекомендуется использовать версию этого стандарта 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.

Издание официальное

3 Термины, определения и сокращения

3.1 В настоящем стандарте применены термины по ГОСТ Р 50922, ГОСТ 28397, а также следующие термины с соответствующими определениями:

3.1.1 **анализ иерархии классов:** Статический анализ, позволяющий выявить в программах на объектно-ориентированных языках список классов, реализованных в программе, а также отношения наследования между этими классами.

3.1.2 **анализ косвенных вызовов:** Статический анализ, позволяющий в программе, использующей вызовы по указателю или вызовы виртуальных методов, установить множество процедур, которые могут вызываться при выполнении заданного косвенного вызова.

Примечание — Анализ потока управления, анализ потока данных, анализ помеченных данных, анализ псевдонимов и анализ косвенных вызовов в общем случае дают приближенные результаты.

3.1.3 **анализ помеченных данных:** Статический анализ, при котором анализируется течение потока данных от источников до стоков.

Примечания

1 Под источниками понимаются точки программы, в которых данные начинают иметь пометку — некоторое заданное свойство. Под стоками понимаются точки программы, в которых данные перестают иметь пометку.

2 Распространенная цель анализа помеченных данных — показать, что помеченные данные не могут попасть из источников — точек ввода пользователя в стоки — процедуры записи на диск или в сеть. Факт такого попадания означает утечку конфиденциальных данных.

3.1.4 **анализ потока данных:** Статический анализ, при котором определяются свойства обрабатываемых программой данных.

Примечание — Могут определяться возможные значения переменных и констант, точки программы, в которых используются определенные переменные и пр.

3.1.5 **анализ потока управления:** Статический анализ, при котором выделяются процедуры программы, линейные участки кода процедур и условия переходов между этими участками.

3.1.6 **анализ программы на синтаксическом уровне:** Статический анализ, при котором обрабатывается представление программы, полностью отражающее ее синтаксическую структуру, например абстрактное синтаксическое дерево.

3.1.7 **анализ псевдонимов:** Статический анализ, позволяющий установить наличие в программе доступа к одной и той же переменной или функции с помощью различных ссылок (указателей).

3.1.8 **внутреннее представление:** Структуры данных или служебный язык, которые используются статическим анализатором для представления исходного кода программы в ходе анализа.

Примечание — Внутреннее представление может строиться как перед выполнением основного анализа в ходе отдельного, настраиваемого пользователем этапа контролируемой сборки ПО, так и вместе с выполнением статического анализа (например, когда используемый язык программирования не требует выполнения сборки).

3.1.9 **императивный язык программирования:** Язык программирования, в котором используется парадигма программирования, описывающая действия над данными в терминах последовательности команд.

3.1.10 **индуктивная переменная:** Переменная цикла программы, значение которой может быть выражено в виде функции от счетчика цикла или других индуктивных переменных.

Примечание — Индуктивные переменные, как правило, выражаются линейными функциями от счетчика цикла, изменяясь на каждой итерации цикла на фиксированное целое значение.

3.1.11

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

[ГОСТ Р 51904—2002, пункт 3.17]

3.1.12 **контролируемая сборка ПО:** Сборка ПО, при выполнении которой статическим анализатором фиксируются порядок запуска программных средств, их настройки и используемые конфигурации.

3.1.13 **критическая ошибка в программе:** Ошибка, которая может привести к нарушению безопасности обрабатываемой информации.

3.1.14 **ложноотрицательное срабатывание (ошибка второго рода):** Отсутствие предупреждения об ошибке со стороны статического анализатора в ситуации наличия заведомо известной ошибки в программе, которая способна реализоваться в ходе выполнения программы.

3.1.15 **ложноположительное срабатывание (ошибка первого рода):** Выданное статическим анализатором предупреждение, которое указывает на конструкцию или несколько конструкций в программе (возможно, в совокупности с некоторым множеством возможных значений переменных программы), не содержащих ошибку, которая может реализоваться в ходе выполнения программы.

3.1.16 **межмодульные связи:** Совокупность связей по управлению, когда код одного программного модуля передает управление другому модулю, и связей по данным, когда программные модули совместно используют какие-либо данные.

3.1.17 **межмодульный анализ:** Статический анализ, при котором выполняется совместный анализ программы или нескольких программ, состоящих из нескольких программных модулей, и выявляемые свойства программы затрагивают процедуры или переменные из различных модулей.

3.1.18 **межпроцедурный контекстно-чувствительный анализ:** Статический анализ, при котором выявляемые свойства программы учитывают взаимодействие нескольких процедур, в том числе — возникающее в результате выполнения нескольких процедур или вызовов процедурами друг друга, а также контексты их вызова.

Примечания

1 В ходе анализа учитывается контекст вызова процедуры при обработке ее вызова, т. е. сопоставляется информация из вызывающей и вызываемой процедур применительно к каждому месту вызова и его окружению: фактическим параметрам, состоянию глобальных переменных и т. п.

2 Термины 3.1.6, 3.1.17, 3.1.18, 3.1.28, 3.1.32, 3.1.36 определяют различные свойства методов анализа программ, которые могут применяться как при статическом анализе программ, так и для оптимизаций программы в ходе ее компиляции. В настоящем стандарте термины 3.1.6, 3.1.17, 3.1.18, 3.1.28, 3.1.32, 3.1.36 трактуются как свойства методов статического анализа. В свою очередь, термины 3.1.1—3.1.5, 3.1.7 описывают семейства видов анализа.

3.1.19 **модельный вариант (ошибки):** Подтип некоторого типа ошибки в программе, отнесение к которому осуществляется исходя из особенностей реализации ошибки данного типа и особенностей языка программирования.

Примечание — Разбиение типа ошибки на модельные варианты применяется из-за технологических ограничений статического анализа в целях эффективной работы статических анализаторов, сводя задачу поиска ошибки более общего типа к задаче поиска ошибок множества подтипов.

3.1.20 **ошибка в программе:** Конструкция или набор конструкций в исходном коде программы, наличие которой указывает на возможность реализации угроз безопасности информации и/или на ошибку реализованного в программе алгоритма.

Примечания

1 Ошибка в программе может быть уязвимостью программы, программной закладкой, ошибкой реализованного алгоритма, приводить к утечке конфиденциальных данных, отказу программного обеспечения и пр. В настоящем стандарте в задачи статического анализа не входит разграничение ошибок в части последствий, необходимо найти потенциальные места ошибок.

2 При анализе части программы (процедур) с неизвестным контекстом вызова ошибка может реализоваться в контексте, не содержащемся в известной анализатору части. В этом случае судить о ложности срабатывания можно лишь с учетом всей программы.

3.1.21 **поточный вариант (ошибки):** Способ выражения ошибки в исходном коде программы через конкретную структуру потоков управления и данных.

Примечание — Поточные варианты являются общими для всех типов ошибок, так как характеризуют не саму ошибку, а способ ее выражения в исходном коде программы (характерный вид потоков управления и данных).

3.1.22

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

[ГОСТ 19781—90, статья 1]

3.1.23

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

[ГОСТ 19781—90, статья 2]

3.1.24 **путь выполнения:** Последовательность операторов программы или процедуры, которая выполняется при заданных входных данных или параметрах процедуры.

3.1.25 **разработчик программного обеспечения:** Организация, выполняющая работы на всех стадиях разработки программы или ответственная, как минимум, за такие процессы жизненного цикла, как процессы реализации и процессы поддержки программных средств.

Примечание — Стадии разработки — по ГОСТ 19.102, процессы жизненного цикла — по ГОСТ Р ИСО/МЭК 12207.

3.1.26 **сборка ПО:** Процесс построения из исходных модулей ПО программных модулей, готовых к выполнению или интерпретации, и/или библиотек.

3.1.27

сборочная среда: Совокупность программных и аппаратных средств, служб связи, интерфейсов, форматов данных, протоколов, стандартов, обеспечивающих преобразование исходного кода программ в программные пакеты в соответствии с представленными метаданными и с учетом зависимостей программного пакета.

[Адаптировано из ГОСТ Р 54593—2011, пункт 3.13]

3.1.28 **сигнатурный анализ:** Статический анализ, определяющий наличие свойства программы при помощи поиска строк в исходном коде программы по некоторому образцу, в том числе заданному с помощью формального языка поиска, например при помощи регулярных выражений.

3.1.29 **система непрерывной интеграции:** Система, обеспечивающая автоматизированную сборку и тестирование ПО.

Примечание — Частые автоматизированные сборки ПО применяются для скорейшего выявления ошибок и решения проблем, возникающих при объединении изменений кода.

3.1.30 **система сборки ПО:** Совокупность программных средств и конфигурационных файлов, которая позволяет выполнить сборку ПО.

3.1.31 **среда анализа ПО:** Совокупность программных и аппаратных средств, служб связи, интерфейсов, форматов данных, протоколов, стандартов, обеспечивающих статический анализ программы.

3.1.32 **статистический анализ:** Статический анализ, определяющий статистику для некоторого свойства программы, например: насколько часто в программе проверяется возвращаемое значение некоторой функции.

3.1.33 **статический анализ:** Вид работ по инструментальному исследованию программы, основанный на анализе исходных кодов в режиме, не предусматривающем реального выполнения кода, и выполняемый для определения свойств программы.

Примечания

1 В профессиональной терминологии устоялось взаимозаменяемое применение терминов «анализ исходных текстов» и «анализ исходного кода».

2 В частности, статический анализ применяется для выявления ошибок в программе.

3.1.34 **статический анализатор:** Программное инструментальное средство, которое выполняет статический анализ в автоматическом режиме.

3.1.35 **тип ошибки:** Категория ошибок в программе, отражающая их общность в свойствах или характеристиках.

Примечание — Статические анализаторы снабжают предупреждения об ошибке диагностической информацией, в состав которой входит тип ошибки. Применяемая в статических анализаторах типизация разнится, что затрудняет соотнесение предупреждений, полученных от разных анализаторов. Для облегчения работы с предупреждениями в диагностическую информацию добавляют соотнесение ошибки с одной из популярных систем классификации дефектов безопасности, например с перечнем угроз БДУ ФСТЭК России или MITRE CWE.

3.1.36 **чувствительный к путям выполнения анализ:** Статический анализ программы, при котором могут быть определены ее свойства, проявляющиеся лишь на некоторых путях выполнения программы, и условия (или часть условий), при обращении которых в истину выполнение программы пойдет по указанному анализатором пути.

3.2 В настоящем стандарте применены следующие сокращения:

БДУ — банк данных угроз;

ЭВМ — электронная вычислительная машина;

CWE — общий перечень дефектов (недостатков) безопасности (Common Weakness Enumeration);

POSIX — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой (Portable Operating System Interface);

SARIF — формат обмена результатами статического анализа на основе JSON для вывода инструментов статического анализа (Static Analysis Results Interchange Format).

4 Общие положения

4.1 Предотвращение появления и устранение уязвимостей программы может быть достигнуто путем реализации разработчиком ПО мер по разработке безопасного ПО, представленных в ГОСТ Р 56939.

4.2 При создании безопасного ПО разработчик ПО выполняет статический анализ в соответствии с разделами 5—9 в дополнение к положениям ГОСТ Р 56939.

4.3 Меры по разработке безопасного ПО, представленные в настоящем стандарте, выражены в форме требования, рекомендации или допустимого действия, предназначенных для поддержки достижения результатов реализации мер. Для этой цели в настоящем стандарте используют соответствующие глаголы «должен» («обязан»), «следует» и «может», отражающие различия в обязательности между разными формами требований к реализации мер. Глагол «должен» («обязан») применяют для изложения требования, выполнение которого обязательно для соответствия, «следует» — для выражения рекомендации среди других возможностей, «может» — для того, чтобы отразить направление допустимых действий в пределах ограничений настоящего стандарта.

4.4 Для соответствия требованиям настоящего стандарта разработчик ПО должен использовать в ходе разработки статический анализатор или набор статических анализаторов для поиска ошибок. Выявление критических ошибок в программе способствует идентификации уязвимостей, их устранению или разработке компенсирующих мер.

4.5 Статический анализ в рамках жизненного цикла ПО проводят в соответствии с требованиями раздела 5.

4.6 Методы и инструменты статического анализа должны соответствовать требованиям разделов 6—8.

4.7 Требования, предъявляемые к специалистам, проводящим статический анализ, — согласно разделу 9.

4.8 Проверку соответствия статического анализатора требованиям разделов 6—8 проводят согласно методике, приведенной в разделе 10.

4.9 При отсутствии применимых инструментов статического анализа (отсутствует поддержка используемого языка программирования, не поддерживаются используемые средства сборки ПО и т. п.), отвечающих всем требованиям разделов 6—8, следует использовать в жизненном цикле ПО инструменты, наиболее полно выполняющие данные требования. Приоритет следует отдавать инструментам, демонстрирующим лучшие ключевые показатели в соответствии с 8.4.

5 Требования к внедрению и порядку выполнения статического анализа

5.1 Внедрение статического анализа ПО в организации — разработчике ПО состоит из следующих этапов:

а) подготовительный этап:
   1) выбор инструмента статического анализа;
   2) подготовка сборочной среды ПО (для ПО, требующего сборки) и среды анализа ПО;

б) начальный этап:
   1) настройка инструмента статического анализа для данного ПО;
   2) выполнение первичного статического анализа исходного кода ПО;

3) разметка полученных результатов и формирование начальной базы предупреждений о потенциальных ошибках;

в) проведение регулярного статического анализа ПО.

5.2 На подготовительном этапе проведения статического анализа ПО осуществляют выбор инструмента или набора инструментов статического анализа для регулярного проведения анализа всего кода ПО в рамках жизненного цикла ПО в соответствии с требованиями разделов 7 и 8. Для этого должен быть выполнен анализ документации ПО, а также анализ исходного кода ПО для определения языков программирования, на которых написано ПО, параметров среды функционирования ПО. Для ПО, требующего сборки (трансляций кода, компоновки и т. п.), должен быть выполнен анализ документации системы сборки, инструментов сборки ПО и параметров среды их функционирования (операционные системы, процессорные архитектуры).

С учетом этих данных необходимо выбрать один или несколько статических анализаторов, удовлетворяющих требованиям разделов 7 и 8 и обеспечивающих анализ ПО с установленными параметрами.

Статический анализатор (набор анализаторов) должен обеспечивать поиск ошибок в программах, написанных на языках программирования, используемых при разработке ПО. Следует использовать регулярно обновляющиеся статические анализаторы, поддерживающие современные стандарты языков программирования и соответствующие системы сборки.

5.3 Для ПО, требующего сборки, необходимо подготовить сборочную среду ПО и среду анализа ПО. В противном случае, если сборка не предусмотрена (например, ПО состоит исключительно из интерпретируемых скриптов), подготавливают только среду анализа ПО.

Следует использовать среду анализа ПО, позволяющую выполнять статический анализ ПО выбранным инструментом (набором инструментов) анализа с учетом заданных в разделе 5 временных ограничений на периодичность проводимого анализа и устранение выявленных потенциальных ошибок.

Допустимо использовать для сборки и анализа статический анализатор, развернутый в виртуальной инфраструктуре или предоставляемый разработчику ПО в виде сервиса.

5.4 На начальном этапе проведения статического анализа ПО осуществляют:

- настройку инструмента статического анализа для данного ПО;
- выполнение первичного статического анализа исходного кода ПО;
- разметку полученных результатов и формирование начальной базы предупреждений о потенциальных ошибках.

Для ПО, требующего сборки, настройка инструмента анализа заключается в выполнении в подготовленной сборочной среде первичной сборки ПО под контролем статического анализатора и в первичной конфигурации анализатора. В противном случае, если сборка не предусмотрена, действия ограничиваются первичной конфигурацией анализатора.

В ходе конфигурации должны быть выполнены выбор и включение типов предупреждений анализатора, соответствующих списку критических ошибок, приведенных в 6.3. В ходе конфигурации могут быть также включены другие типы предупреждений анализатора, соответствующих другим потенциальным ошибкам, специфичным для анализируемого ПО (дефекты кодирования для языка программирования ПО, принятые стили кодирования и пр.).

Выполнение первичного анализа заключается в запуске анализатора в подготовленной конфигурации и в среде анализа ПО. В результате первичного анализа должен быть получен исходный набор выданных анализатором предупреждений и выполнена настройка параметров анализатора на анализ данного ПО в среде анализа (количество потребляемой памяти, процессорных ядер и т. п.).

Полученный набор предупреждений должен быть размечен согласно 5.5. Результаты разметки должны быть сохранены для возможности сравнения результатов последующих запусков анализатора на данном ПО. Данные результаты формируют начальную базу предупреждений о потенциальных ошибках.

После анализа результатов первичной разметки конфигурация статического анализатора может быть доработана, в частности могут быть включены дополнительные типы предупреждений, выполнена настройка анализатора на используемые в заданном ПО заимствованные компоненты, доработаны и настроены алгоритмы выдачи конкретных типов предупреждений и пр. Кроме того, статический анализатор может быть по-разному настроен для различных компонентов анализируемого ПО, например для тестов искать только критические ошибки.

На основании результатов начального этапа проведения статического анализа ПО может быть принято решение о выборе другого статического анализатора.

5.5 Предупреждения о потенциальных ошибках, полученные в ходе статического анализа ПО, должны быть размечены: просмотрены для экспертного определения их истинности или ложности. Каждое выданное предупреждение о критической ошибке (типы критических ошибок приведены в 6.3—6.5) должно быть отнесено к определенной категории. Категории должны включать: истинные предупреждения, ложные предупреждения, истинные, но не требующие исправлений кода предупреждения. Допускается расширять приведенный перечень категорий.

5.6 Для своевременного выявления и исправления ошибок статический анализ должен регулярно применяться к разрабатываемому ПО. Накопление непроанализированных изменений ухудшает качество проводимой экспертизы результатов анализа, а именно: чаще возникают ошибки, при которых истинное предупреждение неверно классифицируется как ложное срабатывание, а также увеличивается длительность проводимой экспертизы. Затем такое накопление приводит и к усложнению выявленных исправлений ошибок в силу увеличившегося временного промежутка между внесением ошибки и ее исправлением.

Регулярность статического анализа ПО обеспечивается автоматизацией процедуры проведения анализа согласно требованиям 5.6—5.8, например с помощью системы непрерывной интеграции.

Статический анализ ПО в рамках жизненного цикла ПО следует проводить регулярно на этапе конструирования и комплексирования ПО. Статический анализ всего разрабатываемого ПО следует выполнять не реже одного раза в 10 рабочих дней, если за данный период времени исходный код был изменен. Статический анализ добавленных или измененных частей ПО следует выполнять после каждого внесенного изменения.

Результаты каждого проведенного статического анализа должны сохраняться в хранилище результатов анализа.

В случае использования при разработке ПО системы непрерывной интеграции проведение статического анализа должно быть включено в состав автоматически выполняемых проверок кода.

5.7 Процедура выполнения статического анализа ПО состоит из следующих этапов:

а) получение внутреннего представления для статического анализатора по исходному коду всего ПО или его измененных частей. Данный этап выполняют:

1) для ПО, требующего сборки, отдельно в ходе контролируемой сборки ПО в подготовленной сборочной среде;

2) для ПО, не требующего сборки, вместе с последующим этапом;

б) проведение в подготовленной среде анализа ПО следующих видов статического анализа исходного кода ПО:

1) синтаксический анализ исходного кода ПО, в том числе сигнатурный поиск;

2) межпроцедурный контекстно-чувствительный анализ исходного кода ПО.

При необходимости построенное в соответствии с перечислением а) внутреннее представление переносится из сборочной среды в среду анализа ПО;

в) сохранение выданных при анализе предупреждений.

5.8 Выданные статическим анализатором предупреждения должны быть размечены согласно 5.5. Для своевременной и эффективной работы с полученными результатами статического анализа просмотр выданных анализатором предупреждений следует выполнять:

а) при анализе измененных частей ПО — не позже, чем через три рабочих дня после выполнения анализа;

б) при анализе ПО целиком — не позже, чем через 10 рабочих дней после выполнения анализа.

5.9 Выявленные потенциальные ошибки, которые в результате верификации согласно 5.7 были отнесены к истинным, не позже, чем через 10 рабочих дней после выполнения разметки следует исправить либо запланировать сроки их устранения в соответствии с принятыми разработчиком ПО процессами разработки ПО. Если применяется гибкая методология разработки ПО, то план по устранению потенциальной ошибки включается в ближайший цикл разработки. При анализе ПО целиком устранение выявленных потенциальных критических ошибок должно быть выполнено не позднее начала квалификационного тестирования ПО. Для выполненных изменений исходного кода ПО должна быть обеспечена возможность однозначной идентификации изменений, которые исправляют конкретные ошибки.

5.10 Выявленные в ходе статического анализа и устраненные потенциальные ошибки, которые были признаны истинными в ходе разметки предупреждений, можно использовать в ходе динамического анализа на этапе квалификационного тестирования ПО в качестве дополнительной информации

о возможно уязвимых местах программы, если используемые инструменты динамического анализа поддерживают такую возможность.

5.11 Контроль над ходом устранения выявленных потенциальных ошибок следует выполнять не реже одного раза в месяц. Для выполнения контроля следует привлекать специалистов, которые не участвовали в процессе выполнения анализа, экспертного просмотра выданных предупреждений и устранения потенциальных ошибок. Рекомендуется использовать средства разработки (система контроля версий ПО, система отслеживания ошибок ПО), которые позволяют формировать отчет о количестве выявленных и исправленных ошибок в ПО.

5.12 Конфигурация и настройки статического анализатора должны регулярно пересматриваться. Пересмотр следует проводить при изменении архитектуры ПО, добавлении сторонних компонент, выходе новых версий статического анализатора. Для повышения эффективности разработки ПО следует дорабатывать и настраивать алгоритмы выдачи конкретных типов предупреждений, если статический анализатор предоставляет такую возможность.

5.13 В соответствии с требованиями ГОСТ Р 56939 разработчик ПО должен проводить регулярный поиск уязвимостей на этапе эксплуатации жизненного цикла ПО. В состав применяемых инструментальных средств разработчик ПО должен включать статический анализатор. Рекомендуется настраивать конфигурацию статического анализатора и применять специализированные методы статического анализа для более глубокого поиска ошибок. Например, настройка конфигурации может задавать другие пороговые значения для используемых ресурсов, таких как память и вычислительные ядра процессора, менять параметры эвристик применяемых математических моделей. Примером специализированного метода анализа является статический анализ помеченных данных.

6 Классификация критических ошибок, находимых статическими анализаторами

6.1 Требования к статическому анализатору, относящиеся к качеству выполнения анализа и реализованных методов анализа, выражаются с учетом находимых им типов ошибок в программах.

6.2 Ошибки в программах классифицируются на основе типа ошибки и модельных вариантов проявления ошибки (подтипов). Модельные варианты проявления ошибок задают подтипы некоторого типа ошибки, которые удобно искать при помощи статического анализа в том случае, когда точный поиск общего типа ошибки затруднительно выполнить.

Способы выражения модельного варианта ошибки в исходном коде программы задаются поточными вариантами. Поточные варианты возникают из-за свойств потока данных и управления программой и являются общими для всех типов ошибок.

Разнообразие комбинаций модельных и поточных вариантов ошибок необходимо для более точной оценки выполнения требований, предъявляемых статическому анализатору, в частности — выполнения требований 8.4.

6.3 Критические ошибки группируют по типам в зависимости от языка программирования. Для компилируемых языков к типам критических ошибок относят следующие общие типы:

а) ошибки непроверенного использования чувствительных данных (ввода пользователя, файлов, сети и пр.);

б) ошибки целочисленного переполнения и некорректного совместного использования знаковых и беззнаковых чисел;

в) ошибки переполнения буфера (записи или чтения за пределами выделенной для буфера памяти);

г) ошибки некорректного использования системных процедур и интерфейсов, связанных с обеспечением информационной безопасности (шифрования, разграничения доступа и пр.);

д) ошибки при работе с многопоточными примитивами (интерфейсами запуска потоков на выполнение, синхронизации и обмена данными между потоками и пр.).

6.4 В интерпретируемых языках типами критических ошибок являются следующие общие типы:

а) ошибки непроверенного использования чувствительных данных (ввода пользователя, файлов, сети и пр.);

б) ошибки некорректного использования системных процедур и интерфейсов, связанных с обеспечением информационной безопасности (шифрования, разграничения доступа и пр.);

в) ошибки при работе с многопоточными примитивами.

6.5 Для заданного языка программирования типы критических ошибок, приведенные в 6.3 и 6.4, могут дополняться. Для языков C/C++ в дополнение к 6.3 следующие типы ошибок являются критическими:

а) ошибки разыменования нулевого указателя;

б) ошибки деления на ноль;

в) ошибки управления динамической памятью (выделения, освобождения, использования освобожденной памяти);

г) ошибки использования форматной строки;

д) ошибки использования неинициализированных переменных;

е) ошибки утечек памяти, незакрытых файловых дескрипторов и дескрипторов сетевых соединений.

6.6 Модельные варианты для типа ошибки строятся с учетом таких факторов, как язык программирования, его модель памяти, стандартная библиотека языка и т. п. Исходя из этих данных, для типа ошибки выделяются варианты возникновения ошибки в программе на конкретном языке программирования.

6.7 Поточные варианты проявления модельных вариантов ошибки строятся с учетом особенностей организации потоков управления и данных в языке программирования. Для императивных языков программирования задаются следующие общие поточные варианты:

а) варианты потока управления:
   1) прямолинейный участок кода;
   2) условия, зависящие от параметров функции;
   3) циклы с вычислимыми во время компиляции границами;
   4) циклы с линейным изменением индуктивных переменных;
   5) сложные циклы, вложенные циклы.

б) варианты потока данных:
   1) поток данных заключен полностью в одной функции;
   2) данные передаются через массивы;
   3) данные передаются через структуры;
   4) данные передаются через одну или несколько функций.

6.7.1 Дополнительно выделяемые поточные варианты связываются с типом языка программирования (императивный, функциональный, объектно-ориентированный и пр.) и спецификой языка. Например, для языка программирования C дополнительно можно выделить следующие поточные варианты:

а) варианты потока управления:
   1) условия, зависящие от глобальных переменных;
   2) функции, вызываемые по указателю;

б) варианты потока данных:
   1) данные передаются через указатели;
   2) данные передаются через глобальные переменные;
   3) данные передаются через сложные структуры данных (списки, коллекции и т. п.).

6.8 Пример применения классификации критических ошибок, выявляемых статическими анализаторами для ошибки переполнения буфера, приведен в приложении А.

7 Требования к методам анализа, реализованным в статическом анализаторе

7.1 Качество статического анализа определяется тремя характеристиками: долей ложноотрицательных срабатываний, долей ложноположительных срабатываний, скоростью анализа. Указанные характеристики зависят от методов анализа, реализованных в статическом анализаторе. Цель предъявления требований к методам анализа, реализованным в статическом анализаторе, — обеспечить минимальный уровень в части долей пропусков ошибок и ложных срабатываний при сохранении приемлемой скорости анализа.

7.2 Статический анализатор представляет собой один инструмент или набор инструментов, обеспечивающих в совокупности выполнение требований 7.3—7.6. В этом случае для каждого типа ошибки по 7.3 в наборе должны быть инструменты, удовлетворяющие требованиям 7.4—7.6.

7.3 Методы анализа, реализованные в статическом анализаторе, должны обеспечивать поиск критических ошибок, типы которых определяются в соответствии с поддерживаемым языком программирования. Требуемые типы критических ошибок для компилируемых языков — согласно 6.3, кроме

приведенных в перечислении д), для интерпретируемых — согласно 6.4, кроме приведенных в перечислении в), дополнительно для языков C/C++ — согласно 6.5. Методы анализа могут обеспечивать поиск остальных типов критических ошибок по 6.3 и 6.4.

7.4 Статический анализатор должен реализовывать следующие методы анализа: внутрипроцедурный анализ потоков данных и управления; межпроцедурный и межмодульный контекстно-чувствительный анализ потока данных; чувствительный к путям выполнения анализ потоков данных и управления; межпроцедурный и межмодульный контекстно-чувствительный анализ помеченных данных; анализ программы на синтаксическом уровне.

В состав реализуемых методов статического анализа следует включать следующие методы анализа для поиска дополнительных типов ошибок, а также в качестве вспомогательных анализов:

а) сигнатурный поиск;

б) анализ псевдонимов;

в) анализ косвенных вызовов;

г) статистический анализ;

д) анализ иерархии классов.

Максимальная глубина поддерживаемого межпроцедурного анализа (количество вызовов процедур, через которые может передаваться собираемая в ходе анализа информация о потоке данных программы) не должна быть заранее ограничена, но может определяться с учетом доступных для анализа вычислительных ресурсов.

7.5 При анализе ПО, которое требует сборки, должен быть применен вспомогательный анализ для уточнения межмодульных связей, использующий результаты контролируемой сборки ПО. Анализ должен учитывать параметры программных инструментальных средств систем программирования (трансляторов, компиляторов, редакторов связей и т. п.), применяемых в ходе контролируемой сборки ПО, для точного статического установления межмодульных связей.

При анализе ПО, которое не требует сборки, или невозможности выполнить сборку из-за неполного набора исходных кодов ПО, статическое установление межмодульных связей затруднительно. В этом случае в состав методов анализа, реализованных в статическом анализаторе, следует включать методы приближенного восстановления межмодульных связей.

7.6 Если статический анализатор для поиска ошибок, определенных в 6.3, перечисление а), применяет анализ помеченных данных, должна быть предоставлена возможность конфигурации анализа: должны задаваться процедуры-источники и процедуры-стоки чувствительных данных.

8 Требования к инструментам статического анализа

8.1 Целью выставления требований к инструментам статического анализа являются проверка качества выполняемого инструментом анализа, проверка его применимости к современному ПО и проверка возможности его использования в промышленном окружении процесса разработки.

8.2 Статический анализатор должен поддерживать анализ ПО с используемыми заимствованными компонентами целиком, при этом удовлетворяя требованиям 7.3—7.6, 8.3 и 8.4.

8.3 Следует использовать такой статический анализатор (набор анализаторов), чтобы на технических средствах разработчика ПО для поиска критических ошибок обеспечивался полный анализ ПО с используемыми заимствованными компонентами за время, не превышающее двое суток.

8.4 Требования по качеству выполняемого статического анализа предъявляются для критических типов ошибок, приведенных в 6.3—6.5, и проверяются на квалификационном наборе тестов, построенных в соответствии с требованиями раздела 10.

Для данных типов ошибок на наборе тестов, отвечающих требованиям 10.4, статический анализатор должен обеспечивать достижение следующих показателей:

а) долю ошибок первого рода (ложноположительных срабатываний) — не более 50 %;

б) долю ошибок второго рода (ложноотрицательных срабатываний, т. е. пропусков заведомо известных ошибок) — не более 50 %.

Проверки выполнения требований по качеству должны выполняться в порядке, отвечающем требованиям 10.6 в части ложноположительных срабатываний и отвечающем требованиям 10.7 в части ложноотрицательных срабатываний.

8.5 Результаты применения статического анализатора должны содержать следующую информацию:

а) перечень предупреждений о найденных ошибках;

б) описание ошибок;
в) тип ошибок;
г) место в исходном коде программы, где найдены ошибки.

8.6 В ходе разметки, проводимой в соответствии с 5.5, должна быть обеспечена возможность автоматизации разметки выдаваемых предупреждений с целью сокращения времени, затрачиваемого на принятие решения, к какой категории отнести предупреждение. Должна быть обеспечена возможность навигации по коду при работе с предупреждениями. Должно быть обеспечено соотнесение выданных предупреждений с исходным кодом.

Поддержка в принятии решения и автоматизация могут быть обеспечены как средствами из состава статического анализатора, так и сторонними средствами.

Примечание — Поддержка в принятии решения и автоматизация могут быть реализованы графическим интерфейсом анализатора.

8.7 Статический анализатор должен обеспечивать возможность автоматизации действий, приведенных в 5.6, для использования в системах непрерывной интеграции, например с помощью предоставления консольного интерфейса или программного интерфейса статического анализатора.

8.8 Должно быть обеспечено хранение результатов запуска статического анализатора и разметки этих результатов. Каждое выданное анализатором предупреждение должно быть снабжено идентификатором, с помощью которого его можно однозначно найти в результатах анализа. Допускается использовать для хранения результатов сторонние системы (например, базы данных).

8.9 Статический анализатор должен обеспечивать возможность сравнения результатов запусков анализатора для заданного ПО с учетом выполненной пользователем разметки. При сравнении результатов двух запусков должны быть выделены группы предупреждений, присутствующие в результатах обоих запусков; в результатах только одного из запусков. При просмотре пользователем результатов запуска анализатора должна быть обеспечена возможность скрытия предупреждений, размеченных пользователем как ложноположительные срабатывания и присутствующих также в результатах запуска, сравниваемого с данным. Допускается использовать для сравнения результатов сторонние системы.

8.10 Статический анализатор должен содержать в документации описание всех типов ошибок, которые находит анализатор, с указанием для каждого типа ошибки:

а) описания ошибки;
б) возможных причин возникновения;
в) примеров ошибочного кода, для которого выдается предупреждение о данном типе ошибки;
г) примеров или рекомендаций исправления данного типа ошибки.

При описании выявленных типов ошибок следует указывать соответствие типа ошибки в анализаторе одному или нескольким идентификаторам в системе классификации дефектов безопасности MITRE CWE.

8.11 Статический анализатор должен поддерживать выдачу результатов анализа в открытом документированном формате, который может быть прочитан автоматически, для возможности использования анализатора в составе комплексов средств разработки и в системах непрерывной интеграции.

Примечание — Примером открытого стандарта результатов статического анализа является формат SARIF.

8.12 Статический анализатор может поддерживать возможность доработки реализованных алгоритмов (правил) поиска конкретных типов ошибок, а также разработки и добавления алгоритмов поиска новых типов ошибок пользователем анализатора.

9 Требования к специалистам, проводящим статический анализ

9.1 Целью предъявления требований к специалистам, проводящим статический анализ, является обеспечение процесса проведения статического анализа согласно требованиям раздела 5.

9.2 Для обеспечения процессов проведения статического анализа согласно требованиям 5.3 и 5.6 привлекаются специалисты, совмещающие компетенции по разработке ПО с компетенциями по информационно-технологическому обслуживанию (системному администрированию). Следует привлекать специалистов, обладающих следующими знаниями и умениями:

- знание основ и технологий программной инженерии, жизненного цикла ПО;
- знание архитектуры ЭВМ, устройства современных технических средств, применяемых при создании программно-аппаратных средств;

- знание основных сетевых протоколов и сервисов;
- умение оценивать временные и вычислительные ресурсы, необходимые для проведения статического анализа.

9.3 Для обеспечения процессов проведения статического анализа согласно требованиям 5.2, 5.4, 5.7—5.12 привлекаются специалисты, совмещающие компетенции по разработке ПО с компетенциями в области обеспечения кибербезопасности. Следует привлекать специалистов, обладающих следующими знаниями и умениями:

- знание основ безопасности ПО и систем;
- знание базовых алгоритмов и структур данных, базовых положений теории сложности алгоритмов;
- знание языков программирования, использованных при реализации анализируемого ПО;
- знание основ и технологий программной инженерии, жизненного цикла ПО;
- знание базовых принципов устройства операционных систем, основных положений набора стандартов POSIX, если анализируется ПО, рассчитанное на работу под управлением POSIX-совместимых операционных систем, или аналогов этих стандартов для иных случаев;
- знание архитектуры ЭВМ, устройства современных технических средств, применяемых при создании программно-аппаратных средств;
- знание основных сетевых протоколов и сервисов;
- умение выполнять анализ выданных анализатором предупреждений вручную, для выявленных ошибок оценивать их влияние на безопасность ПО;
- умение выбирать статические анализаторы для заданного ПО с учетом технических ограничений реализации и фундаментальных ограничений, обусловленных свойствами реализуемых в статическом анализаторе моделей, алгоритмов и методов.

Следует привлекать специалистов, имеющих опыт проведения статического анализа либо прошедших программы повышения квалификации по работе со статическими анализаторами.

10 Методика проверки требований к статическому анализатору

10.1 Методика проверки требований к статическому анализатору должна включать проверку на соответствие требованиям раздела 7 и 8.4—8.12.

10.1.1 Проверку на соответствие требованиям раздела 7 и 8.5—8.12 выполняют путем анализа документации статических анализаторов и иных реализующих требования программ и последующего анализа результатов их запусков в соответствующих рабочих окружениях (успешен/не успешен). Проверку на полноту реализованных методов и ошибок анализа согласно 7.3—7.6 дополнительно выполняют с помощью набора квалификационных тестов согласно 10.1.2.

10.1.2 Проверка на соответствие требованиям 8.4 должна выполняться с помощью набора квалификационных тестов, удовлетворяющего требованиям 10.2, для каждого языка программирования, поддерживаемого статическим анализатором.

Если для языка программирования отсутствует доступный набор тестов, в полной мере соответствующий требованиям 10.2, допускается использовать набор тестов, частично соответствующий требованиям 10.2.

Примечание — Примером открытого набора тестов, частично соответствующего требованиям 10.2, перечисления а) и б), является Juliet Test Suite.

10.2 В состав набора квалификационных тестов должны входить:

а) квалификационные тесты небольшого размера, содержащие ошибки, которые должны быть найдены анализатором, для всех типов критических ошибок, отвечающих требованиям 6.3 и 6.4, с учетом специфики анализируемого языка. Тесты должны покрывать модельные варианты соответствующих типов ошибок в соответствии с 6.3 и 6.4, в поточных вариантах в соответствии с 6.7 и дополнительно в поточных вариантах, специфичных для анализируемого языка программирования. В тестах следует учитывать целевую платформу выполнения в случае, если она влияет на тестируемый тип ошибки. Допускается создание теста, проверяющего несколько поточных вариантов одновременно. Тесты предназначены для оценки числа ошибок второго рода (ложноотрицательных срабатываний) и оценки полноты поддерживаемых типов ошибок;

б) квалификационные тесты небольшого размера, для которых статическим анализатором не должно быть выдано ложноположительных срабатываний (ошибок первого рода). Тесты должны по-

крывать все типы ошибок в соответствии с 6.3 и 6.4 и специфичные для анализируемого языка программирования типы ошибок. Тесты должны покрывать модельные варианты соответствующих типов ошибок в соответствии с 6.3 и 6.4, в поточных вариантах в соответствии с 6.7 и дополнительно в поточных вариантах, специфичных для анализируемого языка программирования. Допускается создание теста, проверяющего несколько поточных вариантов одновременно. Тесты предназначены для оценки полноты реализованных методов анализа, которые позволяют находить ошибки из тестов в соответствии с перечислением а) данного подраздела, при этом не допуская ошибок первого рода на тестах данного раздела;

в) квалификационные тесты-программы различных размеров, на базе программ с открытым исходным кодом, с заранее известными ошибками. Тесты используются для оценки качества реализованных методов анализа в условиях реальных программ большого размера. В составе набора должны присутствовать тесты, представляющие следующие диапазоны размеров ПО: до 100 тыс. строк кода, от 100 тыс. до 1 млн строк кода, от 1 млн до 10 млн строк кода, более 10 млн строк кода.

Состав набора квалификационных тестов следует регулярно пересматривать.

Пример построения квалификационных тестов для ошибки переполнения буфера с использованием ряда поточных вариантов согласно 10.2, перечисление а), представлен в приложении Б.

10.3 Набор квалификационных тестов должен содержать описание ошибок, в котором указывается размещение ошибочной конструкции или набора конструкций в исходном коде тестовой программы, тип ошибки. Описание ошибок используется для подсчета числа ложноположительных и ложноотрицательных срабатываний.

10.4 Долю ложноположительных срабатываний рассчитывают по формуле

$$100\ \% \cdot FPW/TW, \tag{1}$$

где FPW — число предупреждений об обнаруженных ошибках, отнесенных к ложным срабатываниям;

TW — число всех предупреждений об ошибках, выданных статическим анализатором.

Показатель доли ложноположительных срабатываний, заданный требованиями 8.4, перечисление а), должен независимо рассчитываться и оцениваться для двух групп тестов: квалификационных тестов небольшого размера, отвечающих требованиям 10.2, перечисления а) и б), квалификационных тестов различных размеров, отвечающих требованиям 10.2, перечисление в).

10.5 Долю ложноотрицательных срабатываний (пропусков) рассчитывают по формуле

$$100\ \% \cdot FN/TB, \tag{2}$$

где FN — число известных ошибок, которые не были обнаружены;

TB — суммарное число ошибок в тестовых программах, намеренно заложенных и непреднамеренно внесенных в код и обнаруженных в ходе анализа.

Примечание — Из-за наличия в составе набора тестов на базе реальных программ возможны ситуации выявления ранее неизвестных ошибок, которые были допущены разработчиками программы, на базе которой разрабатывался тест.

Показатель доли ложноотрицательных срабатываний, заданный требованиями 8.4, перечисление б), должен независимо рассчитываться и оцениваться для двух групп тестов: квалификационных тестов небольшого размера, отвечающих требованиям 10.2, перечисления а) и б), квалификационных тестов различных размеров, отвечающих требованиям 10.2, перечисление в).

10.6 Для типов критических ошибок подсчет долей ложных срабатываний и пропусков должен быть выполнен с полной разметкой предупреждений.

Доли пропусков и ложных срабатываний для остальных типов ошибок в системе типизации оцениваемого статического анализатора могут подсчитываться на основе репрезентативной выборки предупреждений. Предупреждения для просмотра должны выбираться случайным образом. Для каждого типа ошибки формируется первичная выборка, которая должна состоять не менее чем из 10 предупреждений. Если все предупреждения первичной выборки оказываются ложными, дальнейшего расширения выборки не происходит. При выявлении подтвержденных ошибок количество размечаемых предупреждений должно быть увеличено до значения, не менее чем в 10 раз превышающего количество выявленных подтвержденных ошибок в первичной выборке.

10.7 Для оценки выполнимости требований 8.3 (скорости работы статического анализатора) следует применять набор квалификационных тестов, удовлетворяющий требованиям 10.2.

Приложение А
(справочное)

Пример применения классификации критических ошибок, выявляемых статическими анализаторами, для ошибки переполнения буфера в языке С

А.1 Ошибка переполнения буфера в языке С характеризуется следующими элементами: буфером, его размером, индексом некорректного обращения к буферу. Исходя из устройства языка С, его модели памяти и стандартной библиотеки, можно выделить следующие особенности:

- буфер выделен в автоматической, статической или динамической памяти;
- размер буфера является константой (т. е. известен на момент компиляции) или вычисляется в ходе работы программы;
- доступ к буферу осуществляется непосредственно (операцией индексации) или через функции стандартной библиотеки (memcpy, memmove и т. п.);
- буфер является обычным массивом или содержит строку языка С;
- размер или индекс буфера получен от пользователя.

А.2 Исходя из вышеприведенных особенностей, можно привести следующую классификацию (типизацию) для ошибки переполнения буфера:

а) общий тип ошибки: ошибка переполнения буфера;

б) модельные варианты:

1) буфер константного размера, выделенный в статической или автоматической памяти, доступ с переполнением выполняется индексацией;

2) буфер константного размера, выделенный в статической или автоматической памяти, доступ с переполнением выполняется в библиотечной функции;

3) буфер константного размера, выделенный в статической или автоматической памяти, доступ с переполнением выполняется индексацией, при этом индекс буфера получен от пользователя;

4) буфер неизвестного при компиляции размера, выделенный в динамической или автоматической памяти, доступ с переполнением выполняется индексацией;

5) буфер неизвестного при компиляции размера, выделенный в динамической или автоматической памяти, доступ с переполнением выполняется в библиотечной функции;

6) буфер неизвестного при компиляции размера, выделенный в динамической или автоматической памяти, доступ с переполнением выполняется индексацией, при этом размер или индекс буфера получены от пользователя;

7) буфер неизвестного при компиляции размера, выделенный в динамической или автоматической памяти, доступ с переполнением выполняется в библиотечной функции, при этом размер или индекс буфера получены от пользователя;

8) варианты, приведенные в перечислениях 1) — 7), в которых буфер является строкой, что дополнительно требует отслеживания ее длины;

9) варианты, приведенные в перечислениях 1) — 7), в которых буфер является строкой, при этом длина строки получена от пользователя;

в) язык программирования: С и С++;

г) связь с классификаторами MITRE CWE: CWE-119, 120, 786, 787, 788, 806.

Приложение Б
(справочное)

Примеры построения квалификационных тестов для ошибки переполнения буфера с использованием поточных вариантов в языке С

Пример 1.

Модельный вариант: А.2, перечисление б)1), поточный вариант: 6.7, перечисления а)2), б)1).

```
#define SIZE 10
int buf[SIZE];
void test(unsigned idx) {
  int sum = 0;
  if (idx < SIZE) {
    sum += buf[idx];
    idx = idx + 1;
  }
  sum += buf[idx]; // ошибка
}
```

Пример 2.

Модельный вариант: А.2, перечисление б)1), поточный вариант: 6.7, перечисления а)3), б)1).

```
#define SIZE 10
int buf[SIZE];
int check (int i);
void test(void) {
  int idx = SIZE;
  for (int i = 0; i < SIZE; i++)
    if (check (i)) { // if может никогда не сработать
      idx = i;
      break;
    }
  buf[idx]++; // ошибка
}
```

Пример 3.

Модельный вариант: А.2, перечисление б)1), поточный вариант: 6.7, перечисления а)1), б)4) и 6.7.1, перечисление б)1).

```
#define SIZE 10
int buf[SIZE];
void access (int* buffer) {
  buffer[SIZE]++; // ошибка при вызове из test1
}
void test1() {
  access (buf);
}
```

Пример 4.

Модельный вариант: А.2, перечисление б)1), поточный вариант: 6.7, перечисление б)4), и 6.7.1, перечисления а)1), б)2.

```
#define SIZE 10
int buf[SIZE];
int status;
int getIndex() {
  if (status)
    return 7; // правильный индекс
  else
   return SIZE; // слишком большой индекс
}
int test2() {
  int i = getIndex();
  buf[i]++; // ошибка
}
```

Пример 5.

Модельный вариант: А.2, перечисление б)2) (вариант со строкой), поточный вариант: 6.7, перечисления а)2), б)1).

```
#include <string.h>
int foo (int cond, int n) {
  char s[100], dst[10]= "";
  int x = 0;
  if (cond)
    strcpy(s, "very very long string ");
  if (n > 15)
    x = n;
  strncat(dst, s, n); // ошибка
  return x;
}
```

Пример 6.

Модельный вариант: А.2, перечисление б)2) (вариант со строкой), поточный вариант: 6.7, перечисления а)3), б)1).

```
#include <string.h>
int check (char *str, int i);
int foo (int cond, int n) {
  char s[100];
  int idx = 100;
  char str[] = "very very long string ";
  int len = strlen(str);
  strcpy(s, str);
  for (int i = 0; i <= len; i++)
    if (check (str, i)) { // if может никогда не сработать
      idx = i;
      break;
    }
  return s[idx]; // ошибка
}
```

УДК 004:006.354 ОКС 35.020

Ключевые слова: защита информации, безопасное программное обеспечение, статический анализ, статический анализатор, ошибка в программе, уязвимость программного обеспечения

Редактор *Е.В. Якубова*
Технический редактор *В.Н. Прусакова*
Корректор *Л.С. Лысенко*
Компьютерная верстка *И.А. Налейкиной*

Сдано в набор 22.01.2024. Подписано в печать 13.02.2024. Формат 60×84⅛. Гарнитура Ариал.
Усл. печ. л. 2,32. Уч.-изд. л. 1,86.

Подготовлено на основе электронной версии, предоставленной разработчиком стандарта

Создано в единичном исполнении в ФГБУ «Институт стандартизации»
для комплектования Федерального информационного фонда стандартов,
117418 Москва, Нахимовский пр-т, д. 31, к. 2.
www.gostinfo.ru info@gostinfo.ru