Устранение дефектов безопасности с CERT C

Устранение небезопасных методов кодирования и неопределенных поведений, которые могут привести к уязвимостям и ненадежным приложениям

LDRA Software Technology

экспонента
ЦЕНТР ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И МОДЕЛИРОВАНИЯ

Введение

Стандарт кодирования CERT C состоит из набора руководств, призванных помочь в разработке безопасных и надежных систем. Он был разработан командой Computer Emergency Response Team (CERT) Института программного обеспечения (SEI) с участием многих отраслевых экспертов[1]. SEI является финансируемым федеральным правительством пионером в области практик программирования, и расположен в Университете Карнеги-Меллона и поддерживается Министерством обороны США.

Подразделение SEI CERT было сформировано в ответ на инцидент с червем Морриса 1988 года и сосредоточено на повышении безопасности и устойчивости компьютерных систем и сетей. CERT пользуется большим уважением за свой опыт и тесно сотрудничает с организациями из частного и государственного секторов, включая Cisco, Oracle и Департамент внутренней безопасности США. Идея стандарта безопасного кодирования для языка C родилась в 2003 году в ходе обсуждений CERT с комитетом ISO / IEC JTC1 / SC22 / WG 14 (Wg14), отвечающим за стандарт языка C.

В этом документе разъясняется, как инструменты LDRA можно использовать для проверки и обеспечения соответствия стандарту CERT C.

Одним из ключевых отличий между инструментами анализа качества программного обеспечения является то, что некоторые выполняют динамический анализ, в то время как другие выполняют статический анализ. Статический анализ выполняется без фактического запуска программного обеспечения, тогда как динамический анализ включает в себя запуск всего программного обеспечения или его компонент. Инструменты, используемые для проверки правил кодирования, связаны с исходным кодом, а не с его исполнением, и поэтому используют методы статического анализа.

CERT C

Основная цель развития - обеспечение того, что стандарт кодирования CERT C всегда отражает современные передовые методы с учетом надежности и безопасности. Эти передовые методы определяются с помощью оценки риска, которая измеряет:

- Вероятность нарушения правила, вызывающего дефект;
- Критичность дефекта;
- Цену исправления дефекта.

Некоторые стандарты включают в себя правила стиля кодирования для обеспечения удобочитаемости и сопровождаемости, обеспечивая такие правила, как предписанный отступ и выравнивание скобок. Эти соображения выходят за рамки CERT C, который рассматривает стиль кодирования, только там, где он может привести к дефектам. Например, строчная буква «l» легко ошибочно принять за число «1» во многих шрифтах, поэтому правило CERT C DCL16-C. «Использовать « L », а не« l », чтобы указать длинное значение», предназначено для решения этой проблемы.

[1] https://www.securecoding.cert.org/confluence/display/c/Acknowledgments

Стандарт кодирования CERT C подразделяет свои руководящие принципы на «правила» и «рекомендации». Для определения принципа, как "правила", требуется выполнить три условия.

- Нарушение руководства может привести к дефекту, который может отрицательно повлиять на надежность или безопасность системы, например, путем введения недостатка безопасности, приводящего к уязвимости;
- В руководстве не полагаются аннотации или предположения исходного кода;
- Соответствие руководству может быть определено посредством автоматического анализа (как статического, так и динамического), формальных методов или ручных методов контроля.

Руководящие принципы определяются как "рекомендации", когда они не отвечают одному или нескольким из этих условий, но по-прежнему могут повысить безопасность, надежность или безопасность программных систем.

Помимо различия между правилами и рекомендациями, CERT C делится на 18 категорий. Эти категории варьируются от рекомендаций, относящихся к препроцессору, и характеристик языка, таких как целые числа и плавающие точки, до соображений API, POSIX и Windows. Очевидно, что рекомендации CERT C предназначены для использования выборочно - правила, касающиеся POSIX или операционных систем Windows, не имеют значения, например, при работе с низкоуровневым приложением.

Легко определить источники вдохновения авторов среди руководящих принципов. Например, хотя стандарты языка C[2] широко используются в определении того, как должен себя вести скомпилированный C-код, они ни в коем случае не являются исчерпывающими. Одна из проблем, связанных с разработкой защищенного кода на языке C, связана с проблемой оставшегося «неопределенного» или «неуказанного» поведения, и в стандарте CERT C существует ряд рекомендаций, для его учета.

Другая группа руководящих принципов была взята из работы корпорации MITRE (MITRE), которая сосредоточена на ряде взаимосвязанных областей, включая кибербезопасность[3].CERT использует опыт MITRE, цитируя и ссылаясь на записи MITRE CVE и CWE в вики CERT.

Проверка соответствия стандарту CERT C Secure Coding

Инструменты статического анализа LDRA содержат надмножество правил и рекомендаций, связанных с CERT C и другими стандартами, и сочетают синтаксический анализ с моделями потоков данных и управления. Такой подход позволяет использовать два уровня абстракции, гарантируя, что проверки выполняются с учетом не только опасных деталей языка C, но также и в контексте программы в целом.

Инструменты реализуют несколько этапов анализа, чтобы применить каталог различных проверок, подмножество которых применяется для подтверждения соответствия рекомендациям CERT C.

Основная фаза статического анализа

Во время основной фазы статического анализа код анализируется, чтобы позволить инструменту проверить проблемные объявления переменных, использование макросов, использование вызова функции, а также локальные потоки управления данных.

Данные уровня выражений

Правила CERT C, касающиеся использования данных уровня выражений, проверяются на этом этапе. Например, ARR37-C «Не суммируйте или не вычитайте целое число с указателем на объект без массива» (рис. 1) и EXP45-C «Не выполнять назначения в операторах выбора».

[2]http://www.open-std.org/jtc1/sc22/wg14/www/standards

[3]https://www.mitre.org/capabilities/cybersecurity/overview?category=all

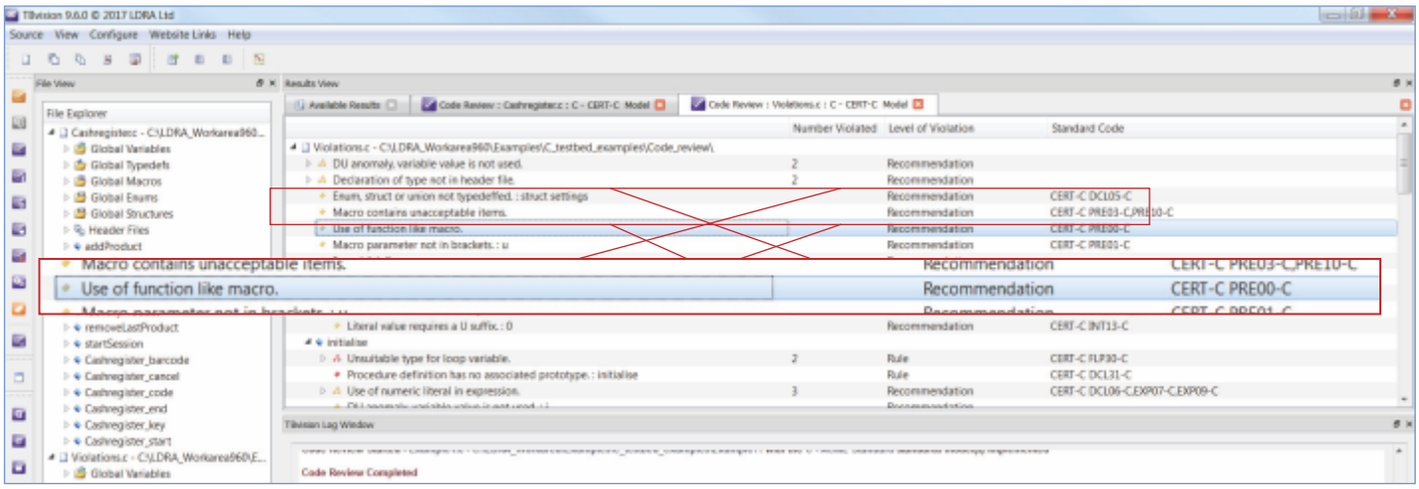


Рисунок 1: Сообщение о нарушении CERT C ARR37-C — «Не добавляйте или не вычитайте целое число в указатель на объект без массива»

Поток управления

Помимо сообщений по ошибкам синтаксиса, на этапе основного статического анализа также устанавливается понимание путей потока управления в коде. Это предоставляет информацию инструменту не только для представления этих путей потока в графической форме, но также для создания переформатированной версии кода, в которой каждая команда занимает одну строку.

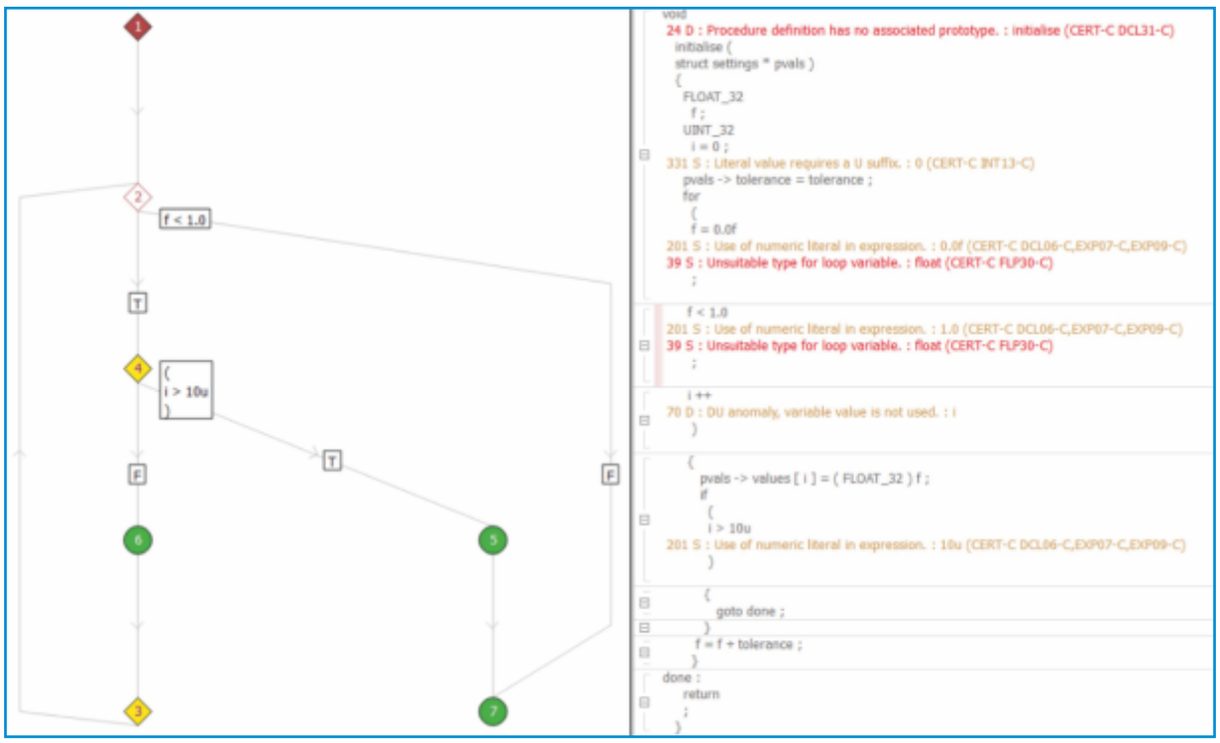


Рисунок 2: Представление потока управления в графической форме вместе с переформатированным исходным кодом

Данные уровня выражений

Эта вторая фаза статического анализа объединяет информацию, относящуюся к потоку данных, дополняет переформатированный исходный код и поток управления для завершения трехмерной модели кода. Комбинация этих элементов обеспечивает основу для проверки использования переменных в сложных потоках управления и в контексте CERT C для проверки нарушения правил, таких как EXP30-C «Не зависьте от порядка выполнения между точкам и последовательности".

Интересно рассмотреть, как это делается автоматически, и, следовательно, рассмотреть усилия, которые потребуются для проверки соблюдения посредством ручного рассмотрения кода.

Точка последовательности - это точка в выполнении программы, при которой все побочные эффекты рассчитываются до перехода к следующему шагу. Они часто упоминаются в ссылках на C и C ++, потому что они являются основной концепцией для определения правильности и, если это допустимо, возможных результатов выражений.

В качестве практического примера рассмотрим две функции f() и g() в C или C ++.
Сценарий 1: выражение f () + g ()
Оператор плюс не связан с точкой последовательности, и поэтому в выражении возможно, что сначала будут выполнены либо f(), либо g().

Сценарий 2: выражение f(), g()

Оператор запятой связан с точкой последовательности, поэтому в коде f(), g() определяется порядок расчета: сначала вызывается f(), а затем g()[4].

Основная задача правила заключается в том, чтобы гарантировать, что результат выражения, такого как в сценарии 1, не зависит от того, что происходит до его выполнения. Чтобы быть уверенным в этом, необходимо рассмотреть различные потенциальные порядки расчета между точками последовательности.

Покажем, что вручную потребуется много времени: проверяющему потребуется очень высокий уровень знаний и концентрации, и даже это не исключает риска человеческой ошибки. Автоматизация процесса является гораздо более прагматичным подходом, поскольку статический анализ LDRA способен распознавать эти ситуации, анализируя не только то, что используются операторы и переменные, но также и контекст структуры выражения и как переменные в этом выражении могут быть затронуты перед использованием.

Фаза анализа перекрестных ссылок

Как только "изображение" потока данных будет завершено, можно использовать эту информацию для перекрестных ссылок на символы, используемые во всей системе. В этом анализе представлена информация о том, где, как и как часто используются переменные, а также обеспечивает основу для оценки потенциальных несоответствий. Например, различия между тем, как определяется и объявляется символ, могут привести к неопределенному поведению, и являются еще одним примером проверки, относящейся к рекомендации CERT C MSC15-C «Не полагайтесь на неопределенное поведение».

[4]http://publications.gbdirect.co.uk/c_book/chapter[8]/sequence_points.html

Перекрестные ссылки также допускают расширение более поверхностных проверок, выполненных на более ранних этапах, таких как распространение анализа возможных нарушений использования массива на экземпляры, охватывающие более чем одну процедуру, относящуюся к правилу CERT C ARR30-C «Не формировать и не использовать указателей или индексы массивов за пределами массивов ". Этот многоуровневый подход распространен во всем статическом анализе LDRA и является значимым, потому что это более тяжелые нарушения, охватывающие несколько функций, которые, вероятно, будут наиболее трудными для обнаружения - и, к сожалению, чаще всего, скорее всего, будут присутствовать.

Сравнение инструментов статического анализа

На рынке существует множество различных инструментов статического анализа, и легко составить список требований. В этом контексте полезно рассмотреть сводку информации, связанной с инструментами автоматической диагностики, доступными на веб-сайте CERT (5).

Анализ показывает в общей сложности 306 рекомендаций CERT C. На рисунке 3 и на рисунке 4 представлены данные, собранные на основе того, что, когда есть указание на то, что конкретный инструмент способен обнаруживать нарушение в какой бы то ни было мере, соответствующую рекомендацию можно считать реализованной.

Tool	Known	Unknown	Total
Co. A	7	0	7
Co. B	96	0	96
Co. C	79	43	122
Co. D	50	0	50
Co. E	11	0	11
Co. F	63	0	63
Co. G	0	3	3
Co. H	36	12	48
Co. I	20	1	21
Co. J	135	0	135
LDRA tool suite	200	1	201
Co. L	10	0	10
Co. M	2	24	

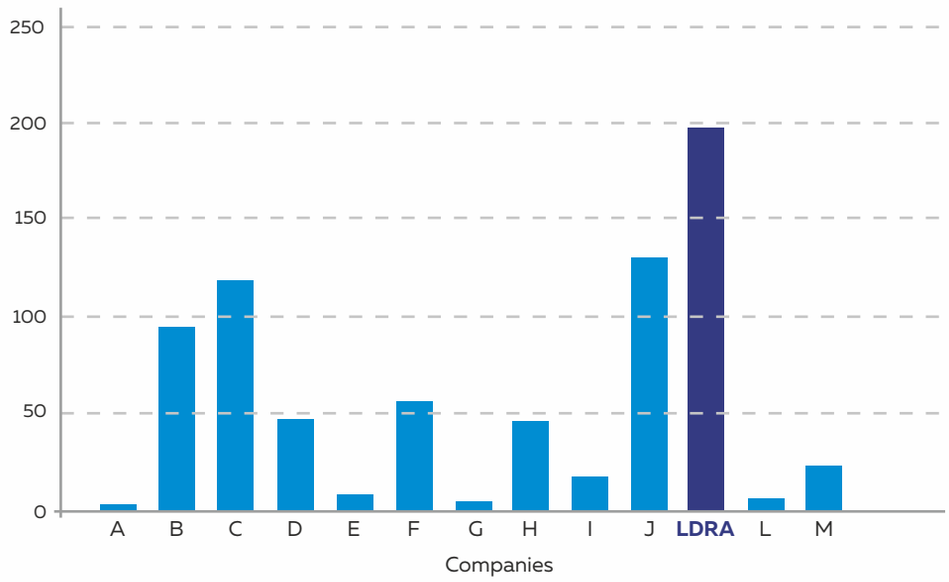
Рисунок 3: Сводка, показывающая количество рекомендаций, реализованных каждым инструментом (из общего числа, 306 в стандарте).

На рисунке 3 категории «Known» и «Unknown» относятся к описаниям веб-сайта CERT как руководств CERT C, так и инструментов статического анализа.

Столбец «Known» показывает количество правил, для которых есть либо подтверждение покрытия, описание покрытия, либо и то, и другое.

Столбец «Unknown» показывает количество правил, для которых нет ни подтверждения покрытия, ни описания его.

По оценкам, по крайней мере 42 руководящих принципа CERT C не проверяются статическим анализом, потому что они не определены точно или потому, что для реализации может потребоваться знание намерений пользователя. Одним из примеров последнего случая является рекомендация CERT C ARR00-C «Понять, как работают массивы». Ни один инструмент не может гарантировать, что программист полностью понимает каждый аспект функциональности массива, применимый к его программе, но инструмент статического анализа может определять конкретные методы кодирования, которые предполагают иное.



Tool	No. of guidelines
A	7
B	96
C	122
D	50
E	11
F	63
G	3
H	48
I	21
J	135
LDRA tool suite	**201**
L	10
M	26

Рисунок 4: Сравнение поставщиков инструмента для соответствия CERT C

Вывод

Можно внедрить использование CERT C Secure Coding Standard, а также проверить соответствие ему с помощью ручных рассмотрений кода. Однако такая задача является постоянной и подверженной ошибкам, и для большинства руководств CERT C существуют инструменты статического анализа, которые предлагают гораздо более эффективный и эффективный подход.

Сравнивая эти инструменты, важно помнить, что, несмотря на то, что количество поддерживаемых рекомендаций CERT C может обеспечить очевидную сравнительную статистику, но часто поддерживаются простые руководства, которые являются общими для всех инструментов. К сожалению, проверки на нарушения, которые трудно реализовать поставщикам инструментов, как правило, являются теми же, которые сложно анализировать с помощью проверки!

Также полезно сравнить полноту обнаружения для каждого из правил. Многие из руководств охватывают несколько различных возможных нарушений. Все ли они принимаются во внимание? Насколько полно они проверяются?

+7 (495) 009-65-85
info@exponenta.ru
ldra.exponenta.ru