МЕТОД ОЦЕНКИ ЭФФЕКТИВНОСТИ ТЕСТИРОВАНИЯ ВСТРОЕННОГО ПО ТВЕРДОТЕЛЬНЫХ НАКОПИТЕЛЕЙ

Мусин С. Б., Иванюк А. А.
ИООО Softeq Development
Кафедра вычислительных методов и программирования, Белорусский государственный университет информатики и радиоэлектороники
Минск, Республика Беларусь
E-mail: sergei.musin@softeq.com, ivaniuk@bsuir.by

В докладе рассматриваются проблемы разработки «прошивок» накопителей на базе флэш памяти. Показана актуальность проблемы тестирования встроенного ПО. Предлагается метод получения компактных характеристик стеков вызовов функций программы на базе адаптивного сигнатурного анализа.

Введение

Современные твердотельные накопители (SSD – solid state drives) обладают целым рядом значительных преимуществ по сравнению с накопителями на жёстких магнитных дисках (hard disk drives – HDD). Эти характеристики – надёжность, производительность, энергопотребление, вес, рабочая температура и уровень шума – в первую очередь являются результатом отсутствия мехнических частей в связи с использованием флэш памяти для хранения информации.

Флэш память – это энергонезависимая память, в которой для хранения информации используются транзисторы с плавающим затвором, причем компромисс между скоростью стирания записанной информации и уровнем износа запоминающих элементов достигается путём стирания блоками. Наибольшая плотность упаковки элементов на кристалле достигается при использовании архитектуры NAND, где к каждому транзистору подводятся не индивидуальные, а общие продольный и поперечный контакты, таким образом, запись и считывание информации производится страницами. Кроме того, для увеличения объема хранимых данных часть транзисторов программируется таким образом, чтобы хранить не один (SLC – single level cell), а два (MLC - multiple level cell) или три (TLC – triple level cell) бита информации. Баланс надёжных, быстрых для записи SLC ячеек и ёмких MLC ячеек оказывает существенное влияние на все характеристики устройства, в особенности на его стоимость. Указанные особенности приводят к тому, что скорость и надёжность работы накопителя существенно вариируются в зависимости от способов его использования приложениями. Поэтому накопитель укомплектовывается встроенным контроллером, который исполняет сложные, обычно защищенные патентами алгоритмы, реализованные встроенным ПО («прошивкой») накопителя. Как правило, «прошивка» обеспечивает эффективное размещение данных в зависимости от записываемого объема данных, равномерное распределения износа ячеек, атомарность операций записи и стирания данных, а также обработку ошибок.

Согласно [1] разработка новой прошивки твердотельного накопителя занимает минимум 2 года. Причем существенное внимание при разработке уделяется тестированию, так как качество «прошивки» является наиболее важным фактором обеспечения конкурентноспособности производителя на рынке.

I. Проблема оценка эффективности тестирования встроенного ПО

Одной из самых сложных для тестирования задач, реализуемых «прошивкой», является проверка сохранности данных пользователя в случае потери питания [1, 2]. Существует практически бесконечное количество внутренних состояний устройства хранения и в каждом состоянии может произойти потеря питания. Задача «прошивки» определить это состояние при следующей инициализации устройства. Таким образом, для контроля эффективности тестирования охват тестами должен определяется не только исходя из факта выполнения контролируемого кода, но и с учетом состояния накопителя, т.е. с учетом выполнения функций программы в определенном контексте. Хорошим критерием такой оценки является контроль стеков вызовов функций программы [3]. Описано множество методов [3–7] осуществления данной процедуры, однако, все они ориентированы на использование больших объемов памяти для хранения результатов, что является неприемлиемым для ПО встроенных систем. Для компактного представления информации о вызовах функций в программе обычно используется дерево вызываемых контекстов (CCT - Calling-Context Tree). Тем не менее, представление в виде дерева даже для короткого времени запуска программ среднего размера может требовать значительных объемов памяти. Альтернативный подход исходит из того факта, что глубина стека вызовов и текущая исполняемая функция идентифицируют вызываемый путь в большинстве случаев [6]. В [7] было предложено использовать хэш-функцию, кото-

рая с определенной вероятностью соответствует каждому из контекстов вызовов программы. Периодический опрос значения функции во время тестового прогона позволяет управлять объемом сохраняемых данных, причем затраты на вычисление хэш-значения (32-битное число) невелики. Основным недостатком данного подхода является низкая диагностическая способность (можно только определить факт различия контекстов, но не детали различий).

II. Предлагаемый метод

В настоящей работе предлагается использовать метод адаптивного сигнатурного анализа (АСА) для вычисления компактной характеристики контекста вызовов. В отличие от других подходов, диагностическая способность данного метода будет обеспечиваться математическим аппаратом теории кодирования.

Используя методику предложенную в [8], будем рассматривать множество всех адресов функций программы, как столбцы проверочной матрицы выколотого кода Хэмминга (все адреса различны, нет нулевого адреса). Тогда вычисление компактной оценки (сигнатуры) стека вызовов программы эквивалентно расчету синдрома кода Хэмминга для двоичного вектора, в котором единицам соответствуют функции представленных в стеке. Иными словами, рассчитывается сумма по модулю два адресов вызванных функций. Однако, так как это операция коммутативна, порядок вызовов функций не будет учитываться. Поэтому будем использовать q-значный код Хэмминга, где q - максимальная глубина вложенных вызовов функций. Как правило, рекурсивные вызовы не используются в ПО встроенных систем, можно ожидать, что значение q невелико и может быть рассчитано при линковке программы.

Таким образом, сигнатура представляет собой пару значений:

- Сумма по модулю 2 значений глубины вложенности вызванных функций;
- Сумма по модулю два произведений адреса вызванных функции на глубинувызова, причем произведение рассчитывается по модулю неприводимого полинома степени n, где $2^n = q$.

Изначально эти значения равны нулю. В процессе работы программы сигнатура поддерживается в актуальном состоянии («адаптируется»), для чего предыдущие значения сигнатуры суммируются по модулю два с новым значением.

Как было показано в [8], достоверность метода АСА на базе q-значного кода Хэмминга соответствует минимальному расстоянию кода ($d = 3$), однократные ошибки в адресах и значениях обнаруживаются. В нашем случае, диагностируются различия в вызванных функциях. Возможно повышение достоверности метода путем построения алгоритма расчета сигнатуры на базе более мощного, чем код Хэмминга, кода (код Рида–Маллера, БЧХ), если установить связь между столбцами проверочной матрицы этого кода и адресами вызываемых функций. Причем последовательно сохраненные значения сигнатуры также можно интерпретировать как итеративный код («код произведения»).

При реализации предлагаемого метода в случае *gcc*-совместимого компилятора можно использовать возможность «instrument-functions», которая позволяет вызывать пользовательский код адаптации сигнатуры при входе в/выходе из функций программы. Что касается процесса вычисления сигнатуры, то наиболее трудоемкой операцией является умножение. Поэтому целесообразно рассмотрение оптимизаций, например [9].

Одним из направлений дальнейших исследований является одновременный контроль стека вызовов и внутреннего состояния накопителя. Метод АСА позволяет производить обе функции с заданной степенью достоверности. Кроме того, необходимо исследовать возможность хранения группы сигнатур, распределение адресов вызываемых функций и возможности для изменения его диапазона в целях повышения достоверности.

III. Список литературы

1. SSD Firmware Development [Electronic resource] / A. Tomlin. – Flash Memory Summit Proceedings, 2012. – Mode of access: `http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2012/20120823_S304A_Tomlin.pdf`. – Date of access: 1.09.2013.
2. Solving the Power Cycling Challenge in SSDs [Electronic resource] / A. Tomlin. – Flash Memory Summit Proceedings, 2013. – Mode of access: `http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2013/20130813_B11_Tomlin.pdf`. – Date of access: 1.09.2013.
3. McMaster, S. Call Stack Coverage for Test Suite Reduction / S. McMaster, A. Memon // IEEE International Conference on Software Maintenance (ICSM) . – 2005. – P. 539-548.
4. Zhang, X. Matching execution histories of Program Versions / Z. Zhang, R. Gupta // SIGSOFT Softw. Eng. Notes. – 2005. – Vol. 30, № 5. – P. 197–206.
5. Mytkowicz, T. Inferred Call Path Profiling / T. Mytkowicz D. Coughlin, A. Diwan// SIGPLAN Not., – 2009. – Vol. 44, № 10. – P. 175–190.
6. D'Elia, D. C. Mining Hot Calling Contexts in Small Space /D. C. 'Elia, C. Demetrescu, I. Finocchi // SIGPLAN Not., – 2011. – Vol. 46, № 6. – P. 516–527.
7. Bond, M. D. Probabilistic calling context / M. D. Bond, K. S. McKinley // SIGPLAN Not., – 2007. – Vol. 42, № 10. – P. 97–112.
8. Ivaniuk, A. A. Detecting multiple errors in RAM by Self-Adjusting output data compression / A. A. Ivaniuk, S. B. Musin, V.N. Yarmolik // Russian Microelectronics, – 2007. – Vol. 36, № 4. – P. 271–277.
9. Musin, S. B. Self-Adjusting Output Data Compression for RAM with Word Error Detection and Correction / S. B. Musin, A. A. Ivaniuk, V.N. Yarmolik // MIXDES '07. 14th Int. Conf., – 2007. – P. 535–538.
10. Silverman, J. H. Fast Multiplication in Finite Fields $GF(2^n)$ / J. H. Silverman // Cryptographic Hardware and Embedded Systems Lect. Not., – 1999. – P. 122–134.