Использование аппаратных средств профилирования для обеспечения информационной безопасности критически важных систем

В. А. Галатенко[1], К. А. Костюхин[2]

[1]Федеральное государственное учреждение «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук», Москва, РФ, galat@niisi.ras.ru;

[2]Федеральное государственное учреждение «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук», Москва, РФ, kost@niisi.ras.ru

Аннотация. Работа посвящена исследованию возможностей применения аппаратных счетчиков производительности (специальных регистров центрального процессора) для выявления потенциальных угроз безопасности критически важных систем и комплексов. Авторами был доработан открытый прикладной программный интерфейс измерения производительности, с помощью которого осуществляется управление аппаратными счетчиками.

Ключевые слова: счетчики производительности, информационная безопасность, атаки по сторонним каналам, PAPI

1. Введение

Многие современные процессоры поддерживают методы анализа программного кода за счет использования аппаратных счетчиков (специальных регистров, записывающих определенные типы аппаратных событий). К примерам аппаратных событий относятся общее количество циклов процессора, общее количество выполненных инструкций, количество выполненных операций с плавающей запятой, количество промахов при доступе к кэш-памяти и т. д. Изначально аппаратные счетчики использовались специально для построения профилей выполнения и последующей оптимизации, но они могут выполнять и другую важную функцию — помогать разработчикам и системным архитекторам оперативно выявлять так называемые атаки по сторонним каналам (side-channels attacks [1]). В рамках этой работы была исследована возможность использования аппаратных счетчиков для обнаружения таких атак и адаптирован программный интерфейс измерения производительности (Performance Application Programming Interface, PAPI [2]) для аппаратной платформы, работающей под управлением отечественного операционной системы.

2. Архитектура PAPI

Целью проекта PAPI является разработка, стандартизация и внедрение портативного и эффективного интерфейса прикладного программирования для доступа к аппаратным средствам профилирования. Сегодня PAPI стал стандартом де-факто для разработчиков программного обеспечения, имеющего доступ к аппаратным счетчикам производительности.

На рисунке 1 представлена архитектура PAPI.

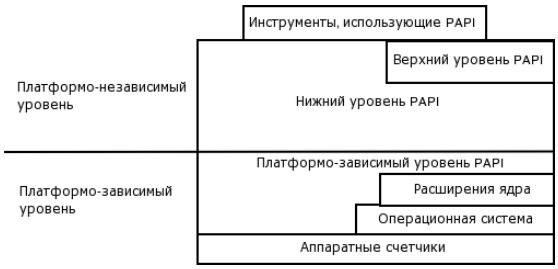

Рис. 1. Архитектура PAPI

Существует два основных уровня PAPI: платформо-независимый и платформо-зависимый, которые скрывают от пользователя детали реализации доступа к аппаратным счетчикам конкретного процессора. Для этого функции PAPI, специфичные для платформы, могут использовать расширения ядра, функции целевой операционной системы или язык ассемблера.

3. Интерфейсы PAPI

PAPI предоставляет пользователям интерфейсы верхнего и нижнего уровня, в которых различается сложность настройки и использования. В настоящее время существуют реализации интерфейса на языках высокого уровня, таких как C и Fortran.

3.1. Интерфейс высокого уровня

Интерфейс высокого уровня состоит всего из 7 функций, обеспечивающих основные операции над аппаратными счетчиками: запуск, остановка, чтение со сбросом и без сброса. В этом случае пользователь может использовать только события, предопределенные стандартом PAPI. Функции интерфейса верхнего уровня используют интерфейс PAPI низкого уровня, чтобы избавить пользователей от явных вызовов, таких как функции инициализации библиотеки PAPI.

```
int PAPI_num_counters (void)
```

Инициализирует PAPI (если требуется). Возвращает число аппаратных счетчиков.

```
int PAPI_start_counters
  (int *events, int len)
```

Инициализирует PAPI (если требуется). Связывает множество событий с аппаратными счетчиками. Запускает счетчики.

```
int PAPI_stop_counters
  (long long *vals, int  alen)
```

Останавливает счетчики и сохраняет их значения в массиве vals.

```
int PAPI_accum_counters
  (long long *vals, int  alen)
```

Прибавляет значения счетчиков к значениям в массиве vals и обнуляет счетчики.

```
int PAPI_read_counters
  (long long *vals, int  alen)
```

Считывает значения счетчиков в массив vals и обнуляет счетчики.

```
int PAPI_flips
  (float *real_time,
   float *proc_time,
   long long *flpins,
   float *mflpins)
int PAPI_flops
  (float *real_time,
   float *proc_time,
   long long *flpins,
   float *mflpins)
int PAPI_ipc
  (float *real_time,
   float *proc_time,
   long long *ins,
   float *ipc)
```

Упрощенные вызовы для измерения числа команд и операций с плавающей точкой, а также частоты выполнения команд процессора. Кроме того, эти функции возвращают реальное время работы процессора, а также виртуальное время, то есть время выполнения пользовательского процесса.

Ниже приведен пример, иллюстрирующий использование функций высокоуровневого интерфейса PAPI. В нем происходит подсчет общего числа команд и тактов процессора при выполнении функции do_work.

```
#include <papi.h>

#define NUM_EVENTS 2

long long values[NUM_EVENTS];
unsigned int Events[NUM_EVENTS] =
{PAPI_TOT_INS, PAPI_TOT_CYC};

/* Стартовать счетчики */
PAPI_start_counters ((int*)Events,
NUM_EVENTS);

/* Интересующая нас функция */
do_work ();

/* Остановить счетчики и сохранить
их значения в массиве values */
ret = PAPI_stop_counters (values,
NUM_EVENTS);
```

3.2. Интерфейс низкого уровня

Низкоуровневый интерфейс обладает по сравнению с интерфейсом высокого уровня расширенной функциональностью и большей эффективностью. В его состав входит более 50 различных функций, которые можно разделить на следующие группы:

- инициализация библиотеки PAPI;
- функции измерения времени;
- функции получения информации;
- служебные функции;
- функции управления множествами событий;
- функции управления аппаратными счетчиками.

Далее показано использование низкоуровневого интерфейса PAPI для подсчета общего числа тактов процессора, а также числа команд сопроцессора плавающей арифметики во время вызова функции do_work.

```
#include <papi.h>

#define NUM_EVENTS 2

int Events[NUM_EVENTS] =
{PAPI_FP_INS,PAPI_TOT_CYC};
int EventSet;
long long values[NUM_EVENTS];

/* Инициализация PAPI */
ret = PAPI_library_init
          (PAPI_VER_CURRENT);

/* Создать множество событий */
ret = PAPI_create_eventset
          (&EventSet);

/* Добавить новые события */
```

```
ret = PAPI_add_events
          (&EventSet,
           Events,
           NUM_EVENTS);

/* Стартовать счетчики */
ret = PAPI_start (EventSet);

do_work(); /* Искомая функция */

/* Остановить счетчики и сохранить
результат в массиве values */
ret = PAPI_stop (EventSet,values);
```

В проекте PAPI предложено стандартизованное, переносимое решение для профилирования кода посредством управления аппаратными счетчиками событий. На сегодняшний день существуют реализации PAPI в виде библиотек для многих современных платформ. Следует отметить также хорошую документированность проекта и простоту использования предлагаемых интерфейсов.

Для используемой целевой системы авторами была доработана и портирована версия papi-c 3.9.0. Такой выбор объясняется слабой зависимостью этой версии от системных вызовов современных ОС семейства Windows или Linux.

4. Потенциальные угрозы, которые можно выявлять с помощью аппаратных счетчиков

4.1. Атаки повторного использования кода

Атаки повторного использования кода (Code Reuse Attacks, CRA [3]), ставящие под угрозу целостность потока управления программы, направлены на изменение нормального потока управления для выполнения вредоносных действий. Среди примеров можно привести возвратно-ориентированное программирование, (Return Oriented Programming, ROP), используя методы которого, злоумышленник получает контроль над стеком вызовов, чтобы заменить адрес возврата из функции. Другим примером является переходо-ориентированное программирование (Jump Oriented Programming, JOP), в котором злоумышленник использует команды перехода для объединения фрагментов вредоносного кода.

Собранная с помощью аппаратных счетчиков информация, такая как, например, события промаха в кэш-памяти или неправильные предсказания ветвления, является, на наш взгляд, хорошим эвристическим индикатором атак этого типа.

4.2. Внедрение кода

Атаки на внедрение кода (Code Injection [4]) реализуют вставки в атакуемое приложение вредоносного кода. Многие из атак этого типа выполняются с помощью переполнения буфера и могут быть выполнены различными способами. В некоторых случаях, для изменения поведения программы могут быть введены ложные данные, такие как ложные показания датчиков в системах управления технологическим процессом. Целями таких атак могут быть захват контроля, саботаж или повреждение атакуемой системы таким образом, чтобы помешать выполнению ее миссии.

Для противодействия этим атакам можно использовать эталонные профили выполнения программы, построенные на аппаратных событиях во время ее первых «эталонных» запусков. В дальнейшем аппаратные счетчики можно использовать для выявления аномального поведения программы, т.е. отклонения текущего профиля выполнения от эталонного. В зависимости от внедренного кода количество различных аппаратных событий, а также их соотношений (как будет показано ниже) резко изменяется за короткое время, что может служит индикатором такого рода атак.

4.3. Атаки по сторонним каналам

Атаки этого типа обычно направлены на кражу информации из целевой системы. Это могут быть пароли, ключи или другие секретные данные.

Для извлечения нужной информации в настоящее время злоумышленники все чаще прибегают к атакам с использованием кэш-памяти: Flush+Reload, Evict+Time, Prime+Probe, Evict+Reload [1].

Все вышеперечисленные атаки порождают определенные аппаратные события, такие, как, например, промахи в кэш-памяти, что является хорошим индикатором для их раннего обнаружения.

4.4. Атаки типа «отказ в обслуживании»

Некоторые атаки типа «отказ в обслуживании» (Denial of Service, DoS) также могут быть обнаружены с помощью аппаратных счетчиков. Как и в п. 4.2 здесь следует использовать эталонные профили выполнения программы, поскольку предполагается, что число, последовательность возникновения и определенные соотношения событий при нормальной работе приложения сильно отличаются от работы во время DoS атаки, которая обычно характеризуется чрезвычайно высокой аппаратной активностью,

в частности должно серьезно возрасти число таких событий, как промахи в TLB и запись в кэш-память первого уровня [5].

5. Применение аппаратных счетчиков в обеспечении информационной безопасности

В качестве экспериментальной платформы был выбран процессор Intel Core I3-6100, под управлением ОС Fedora Core 22. Для имитации атаки использовался инструмент Mastik [6]. Поскольку атаки по сторонним каналам с использованием кэш-памяти предполагают увеличение числа промахов по кэш-памяти 3-го уровня (L3), то вполне логично использовать это событие (L3_MISS) в качестве индикатора потенциальной угрозы. Однако одного его недостаточно. Сама логика приложения может предполагать большое число событий L3_MISS, например, при работе с большим числом данных, не хранящихся локально. Поэтому было предложено еще одно событие (L1_REPL), показывающее, как часто замещаются строки в кэш-памяти 1-го уровня. Теперь если взять их отношение L3_MISS / L1_REPL, то полученный индикатор будет означать, что приложение значительно использует память, но при этом часто очищает кэш. Что может свидетельствовать о проводимой атаке.

Последующие эксперименты показали, что за время измерения атакуемые приложения имели в несколько раз более высокое значение предложенного индикатора (в среднем, в 5 раз), по сравнению с его же значением в обычном режиме работы.

Также были построены профили типичного выполнения вычислительных задач. В качестве критерия было предложено использовать отношение числа выполненных команд сопроцессора плавающей арифметики к числу всех выполненных команд (профиль строился для каждого критического потока управления). Замеры проводились в определенных заранее контрольных точках. Эксперименты показали, что разброс в значениях критерия при разных запусках не превысил 3%. Такой подход позволяет устранить проблему недетерминизма аппаратных счетчиков [7]. В профиль были включены и события по выполнению перехода и выполнению инструкции ветвления. Учитывая недерминизм аппаратных счетчиков, сравнение профилей выполнения задачи в эксплуатационном режиме с построенными в ходе настройки эталонными профилями проводилось по контрольным событиям перехода и ветвления (были выделены контрольные последовательности событий, нарушение которых является признаком потенциального сбоя или атаки). Стресс-тестирование показало, что предложенный подход позволил успешно выявлять некорректные последовательности событий.

6. Заключение

Был проведен анализ потенциальных угроз, реализован прикладной программный интерфейс доступа к аппаратным счетчикам производительности, проведены исследования, подтверждающие изначальное предположение о том, что профиль выполнения атакуемой системы, построенный на определенных аппаратных событиях, меняется, что позволяет построить эффективную систему мониторинга и защиты.

В качестве направления дальнейших исследований видится расширение списка индикаторов, позволяющих выявлять проникновение в систему программ-злоумышленников, путем анализа изменения количества аппаратных событий в их различных комбинациях, во время имитации атак по сторонним каналам.

«Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме «Исследование и реализация программной платформы для перспективных многоядерных процессоров» (FNEF-2022-002).»

Using Hardware Performance Counters to Ensure Information Security of Critical Systems

Vladimir Galatenko, Konstantin Kostiukhin

Abstract. The article discusses the possibility of using hardware performance counters, commonly used in the creating of system execution profiles, to identify potential security threats to critically important systems and complexes. The authors have ported an open Performance Application Programming Interface (PAPI), which is used to manage hardware counters.

Keywords: performance counters, information security, side-channels attacks, PAPI

Литература

1. F. Liu, Y. Yarom, Q. Ge, G. Heiser, R.B. Lee. Last-Level Cache Side-Channel Attacks are Practical, Security Privacy. In Proceedings of the 2015 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 17–21 May 2015.

2. PAPI User's Guide, http://icl.cs.utk.edu/papi/

3. Vishnyakov A.V., Nurmukhametov A.R., Kurmangaleev S.F., Gaisaryan S.S. Method for analysis of code-reuse attacks. Proceedings of the Institute for System Programming of the RAS (Proceedings of ISP RAS). 2018;30(5):31-54. (In Russ.)

4. Oliver Moradov. Code Injection in Brief: Types, Examples, and Mitigation, 2022, https://bright-sec.com/blog/code-injection/

5. Pablo Pessoa do Nascimento, Paulo Pereira, Jr Marco Mialaret, Isac Ferreira, Paulo Maciel. A methodology for selecting hardware performance counters for supporting non-intrusive diagnostic of flood DDoS attacks on web servers, Computers & Security, Volume 110, 2021, https://www.sciencedirect.com/science/article/pii/S0167404821002583

6. Mastik: A Micro-Architectural Side-Channel Toolkit, https://github.com/0xADE1A1DE/Mastik

7. S. Das, J. Werner, M. Antonakakis, M. Polychronakis, F. Monrose. SoK: The Challenges, Pitfalls, and Perils of Using Hardware Performance Counters for Security. 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2019, pp. 20-38.