Защищенные вычисления на примере архитектуры “Эльбрус”

Соколов Роман, 12.03.2005

В эссе представлен краткий обзор особенностей архитектуры Эльбрус, реализующих принципы защищенных вычислений - описываются основные проблемы, присущие традиционным архитектурам, и основные идеи, разработанные для их решения. Подробно об аппаратной реализации этих идей можно прочитать в работах [1, 2, 4]. Также в [3] даны ссылки на обзоры современных подходов к решению этой задачи таких компаний, как Intel и AMD.

Введение

Команда Эльбруса имеет большой опыт разработки надежных высокопроизводительных вычислительных систем. Ее последняя разработка - микропроцессор E2k - вобрал в себя как новейшие принципы организации и проведения вычислений (такие как явный параллелизм на уровне команд, двоичная компиляция), так и весь накопленный опыт, в частности, в разработке архитектур, обеспечивающих защищенность вычислений. Под защищенностью вычислений здесь будет пониматься программная защищенность[1] - когда работа программы, содержащей ошибки либо реализующей злонамеренные действия, не может повлиять на корректную работу остальной части системы.

Стоит отметить, что заслуга Эльбруса состоит не столько в изобретении принципиально новых идей, сколько в строгом, последовательном воплощении в жизнь известных и ранее парадигм работы компьютера - современная модель вычислений, заложенная в языках высокого уровня, сама по себе теоретически позволяет создавать защищенные системы. Важным фактором, гарантирующим полное решение проблемы, является поддержка на основе простых решений этих идей

1. в аппаратуре и
2. в операционной системе.

Почему опыт Эльбруса стал особенно актуален сейчас? Дело в том, что раньше, когда шла борьба за каждый мегагерц, самой приоритетной задачей было достижение максимальной эффективности вычислительной системы. Поэтому о возложении на аппаратуру каких-либо других функций, кроме вычислительной, речи не шло. Одновременно не стояла столь остро проблема защищенности вычислительных систем. Также немаловажную роль здесь сыграли неудачные попытки создания систем, реализующих принципы защищенного программирования, даже таких лидеров микропроцессорной отрасли, как компании Intel[2].

Однако в современных условиях и современном информационном окружении идеи,

[1] Хотя надежности аппаратуры при создании вычислительных комплексов Эльбрус уделяется не меньшее внимание.

[2] Подробнее об истории создания таких систем можно прочитать в работе [1].

которые закладывались при разработке отечественных вычислительных комплексов Эльбрус на самом фундаментальном уровне изначально, становятся особенно актуальными - небольшое снижение производительности более чем оправдано приобретением вычислительной системой такого свойства, как защищенность.

Что надо защищать в вычислительной системе? Основные компоненты вычислительной системы - это устройство обработки данных и память, в которой эти данные хранятся. Далее мы будем считать, что всё аппаратное окружение системы (вычислительные модули, линии связи) работает корректно.

Таким образом, проблемой является защита памяти данных вычислений от ошибок в самой программе, инициировавшей эти вычисления, и от влияния (главным образом, злонамеренного) других программ. Также отдельное внимание нужно обратить на ошибки в системных программах и традиционную организацию файловых систем, которые с успехом используются вирусами.

Память и программирование

В качестве первого тезиса здесь уместно привести цитату из работы [1]:

> *"Смысл прост - типы данных (для защищенности - только указатели[3]) должны обрабатываться правильно."*

По большому счету, в современном программировании есть три основные уязвимости:

- 1) Неинициализированные данные, когда память, выделенная для нового объекта, которому не присваивается свое значение, может содержать данные, которые использовались другой задачей. Это может быть причиной и некорректной работы программы, и, вообще говоря, нарушения защиты.

- 2) Выход за границы массива. Результат такой же.

- 3) Отсутствие контроля при работе с указателями. Например, присвоение ссылочному типу данных целочисленного значения. В итоге можно просто выйти за пределы фактически адресуемой области памяти.

Введем некоторые понятия и на примере работы абстрактной вычислительной системы постараемся понять смысл этого тезиса. Будем разбивать все вычисления на вычислительные модули (ВМ), вычисления в каждом из которых защищены от влияния других ВМ. По смыслу это понятие эквивалентно вычислениям в текущей процедуре. Обрабатывающее устройство (ОУ) выполняет все операции и хранит некоторый объем данных - аргументов операций. ВМ и обрабатываемые данные размещаются в информационном пространстве (ИП). ОУ работает под управлением ВМ, ВМ может обращаться к ИП, считывать из него информацию и изменять его. При создании в ИП узла для хранения данных в ОУ появляется ссылка на него, через которую данный ОУ (и никто более) имеет к нему доступ. Новый пустой узел не должен содержать старых ссылок, так как это было бы прямым нарушением защищенности. Попытка использовать пустой узел должна вызывать диагностическое прерывание. Обращение к "зависшим" ссылкам на уничтоженный узел также должно вызывать диагностическое прерывание.

[3]Проверка остальных типов позволяет диагностировать ошибки в программе.

Ссылка содержит всю информацию, необходимую для доступа к данным, которые она описывает (в том числе и права), что означает, что если ВМ может обращаться к данным с помощью ссылки, то он имеет право обращаться (читать, изменять) к самим данным.

> *"Система должна обеспечивать контроль над тем, чтобы в операциях, предусматривающих использование в качестве аргументов ссылки, не были использованы данные других типов (целые, вещественные и т. д.), а в операциях с аргументами других типов ссылка не могла быть модифицирована."*

Обеспечение строгого контроля типов в языке высокого уровня подразумевает, что тип данных (в том числе информация о том, является ли переменная ссылкой и какой именно) связывается с переменной во время её объявления на всё время её существования, что даёт возможность компилятору производить необходимый контроль типов во время трансляции программы. Однако выполнение этого принципа понижает эффективность использования языка высокого, поэтому, например, в реализациях таких массовых языков, как C и C++, контроль типов нарушается, работа со ссылками никак не контролируется (например, разрешается присваивать целые в переменные типа указатель), а в Java принято совсем кардинальное решение - явные указатели на произвольные типы данных исключены из языка вообще (что делает язык неуниверсальным и неэффективным уже с точки зрения программирования), оставлены ссылки только на объекты. Подход Эльбруса - это поддержка контроля типов в аппаратуре.

Контекстом вычислений называется множество данных, доступных данному ВМ (по ссылкам, в том числе и по косвенным). Контекстной ссылкой (КС) называется ссылка, через которую можно получить доступ ко всему контексту данного ВМ. Она должна храниться в ОУ во время выполнения вычислений ВМ. Когда управление ОУ переходит к другому ВМ, происходит переключение контекста и, соответственно, КС в ОУ, поэтому к этому процессу необходимо предъявлять определенные требования для обеспечения защищенности контекстов переключаемых модулей. Здесь вводится понятие метки ВМ (МВМ): она хранится, например, в модуле А, указывает на команду в контексте В и используется только в качестве аргумента операции переключения контекста и кода программы. МВМ формируется следующим образом: контекст В сам готовит информацию (ссылку на контекст В и код, доступные в нем) для переключения на самого себя и передает ее в контекст модуля А, который в нужный момент использует ее для одновременного переключения на контекст и код. Это приводит к тому, что любой контекст защищён от неправильной или злонамеренной работы программ, работающих в других контекстах и в то же время два контекста могут активно взаимодействовать, обмениваясь параметрами и значениями через механизм МВМ. Такая защита в корне отличается от механизма виртуальных пространств, когда такое взаимодействие невозможно. Таким образом, введено еще одно ключевое понятие рассматриваемой архитектуры - контекстная защита.

Несколько слов о реализации. Аппаратно Эльбрус (E2K) поддерживает базовые типы (все адресные типы - указатели и т.д. - и пустой тип "NULL"), типовый контроль и формирование контекста с помощью ссылок. Кроме того, в E2K поддержаны программно определяемые типы данных, то есть, по

существу, объектно-ориентированное программирование.

Проблема традиционных машин заключается в том, что в качестве ссылки используется машинное слово без указания типа (что это ссылка), содержащее только адрес начала области адресуемых данных без указания ее размера и прав доступа к ней. То есть, невозможно контролировать выполнение операций со ссылкой, а также можно легко выйти за пределы адресуемой области данных. Сама ссылка может быть произвольно (случайно или злонамеренно) изменена.

Для описания типа информации, хранящейся в машинном слове, в архитектуре Эльбруса вводятся дополнительные (к имеющимся информационным) разряды в памяти (ТЭГ), число которых может варьироваться. В частности, для микропроцессора E2K - по 2 бита на каждое 32-х разрядное слово. Сам типовый контроль вводится в аппаратных операциях обработки данных. Указатели помечаются специальным разрядом для правильной его идентификации аппаратурой и состоят из ссылки на контекст и кода. С помощью указателей реализуется контекст процедуры. Указатель на массив содержит также информацию о размере массива. Указатель на объект содержит описание границ public- и private-полей объекта. При всем при этом не требуется какая-то специальна память - используются дополнительные комбинации ECC-битов (биты контроля и коррекции аппаратных ошибок), дополнительные разряды нужно вводить только в кристаллах процессора и кешпамяти (дополнительно 7-12 % аппаратуры).

Проблема ошибок в системных программах, которыми могут воспользоваться вирусы, ослабляется технологией, описанной выше. Даже если злонамеренная программа каким-то образом будет запущена, она будет работать только в своем контексте со своими данными. Однако, хотя описанные механизмы контроля ссылок запрещают такой программе осуществлять доступ к данным в других контекстах, необходимо еще запретить ей доступ к данным, хранящимся в файлах. Об этом пойдет речь ниже.

Память и операционная система

Здесь предметом обсуждения является вторая проблема - организация файловой системы (ФС). Ее архитектура в значительной степени определяет способность вредоносной программы реализовать свое предназначение.

В традиционнных ФС в качестве "указателя" на файл используется символьная строка, задающая путь к файлу из корня ФС.

Пример: допустим, скачанной извне и запущенной программе мы передаем файл параметров, внутри которого могут находиться имена других файлов. При этом загруженной программе придется передать корень ФС. В итоге запущенная программа имеет полный доступ ко всем файлам ФС ее владельца (так как запущена от его имени[4]), хотя обычно для работы может быть необходимо значительно меньше файлов (или не нужно вообще) - это чрезвычайно благоприятная ситуация для агрессивной программы. Либо просто опасная ситуация, если программа содержит ошибку.

В идеале же система защиты ФС должна предоставлять доступ данной программе к минимально необходимому объему информации, при этом оставаясь предельно простой в реализации и использовании.

> *"...защита от АП (агрессивная программа) в Эльбрусе – это не борьба с конкретными приёмами*

[4]К сожалению, зачастую так и происходит.

авторов АП, но скорее создание системы, обеспечивающей защиту от любых АП."

В ней присутствуют два типа указателей - указатель файла и указатель программы. Второй состоит из ссылки на файл кода программы и ссылки на словарь - файловый контекст этого кода. Файлы имеют заголовки, которые собраны в отдельный файл заголовков (ФЗ). То же самое и со словарями. Файлы заголовков и словарей находятся в контексте ОС и непосредственно пользователю недоступны. В итоге, во время исполнения программе доступен только ее ограниченный контекст - как в виртуальной памяти, так и в системе файлов! Это главное отличие системы Эльбруса от существующих ОС: каждой процедуре соответствует индивидуальный контекст файлов, в то время как в традиционных системах у всех исполняемых на данной машине файлов один файловый контекст - это корень ФС. В Эльбрусе корень системы в общем случае не доступен пользователю. Еще один небольшой пример: пользователь запускает на компьютере 1 программу, загруженную из компьютера 2. Эта программа будет иметь доступ только к явно переданным ей пользователем параметрам и к своему контексту в компьютере 2, если он существует (здесь предполагается, что реализованы межкомпьютерные указатели).

В отличии от поддержки контроля типов данных поддержка новой архитектуры файловой системы не требует поддержки аппаратуры. В плане эффективности такая система не уступает существующим системам, в которых поиск файла осуществляется по имени через системы словарей. В рассматриваемой системе файл, как правило, доступен прямо по физической ссылке, а это значительно эффективнее.

Резюме

Таким образом, ключевой идеей в защищенной технологии Эльбруса является контроль типов в аппаратуре, в языке и в операционной системе.

Реализация этой технологии продемонстрировала важнейшие заложенные в нее свойства. В такой системе работающая программа имеет доступ строго только к необходимым для ее исполнения данным, которые определяются разработчиком. Эти ограничения надежны благодаря наличию небольшого объема простых инструментов, отвечающих за корректную работу с указателями и корректную смену контекста. Создание описанной ОС, которая сама будет разработана и отлажена на данной вычислительной системе, значительно увеличивает вероятность сокращения ошибок во время работы такой системы и программ внутри нее.

Несомненно, одним из главных плюсов для конечного пользователя (в первую очередь, программиста) такой архитектуры является то, что ему самому ничего не нужно делать для обеспечения защищенности. Возможно, придется отказаться от крайне небольшого числа приемов программирования, что, однако, ведет только к более строгому стилю программирования и фактически не идет в ущерб эффективности.

Более того, такая вычислительная система:

- упрощает и ускоряет отладку программных систем;
- упрощает разработку системных программ, таких как ОС, так как нет необходимости заботиться об обеспечении защиты между модулями системы;
- позволяет обнаруживать ошибки в ПО, отлаженном на традиционных архитек-

турах (даже в эталонных тестах Spec92, которые считаются хорошо отлаженными программами, было обнаружено более 30 ошибок).

То есть, фактически эта технология превращается в строгий инструмент, на котором должна быть отлажена любая программа.

Справедливости ради нужно заметить, что ввод этой технологии нарушает совместимость с традиционными архитектурами. Однако, как было сказано выше, затраты на ее внедрение будут куда меньшими, чем потери, которые несет сегодня компьютерная индустрия от вредоносных программ. Вообще говоря, решение самой проблемы вирусов невозможно без введения той или иной несовместимости.

Список литературы

[1] Борис БАБАЯН. Защищенные информационные системы // http://www.elbrus.ru/mcst/SECURE_INFORMATION_SYSTEM_V5_2r.pdf (pdf 1.12 МБ)

[2] Babayan B.A. Main Principles of E2K Architecture // Free Software Magazine, Vol. 1, Issue 02, Feb 2002.

[3] Цикл статей "Технологии компьютерной безопасности":
Часть 1. Intel LaGrande Technology — архитектура компьютерной безопасности // http://www.terralab.ru/system/32571/
Часть 2. AMD Execution Protection — маленькая революция // http://www.terralab.ru/system/32706/
Часть 3. "Эльбрус" — защита без латания дыр // http://www.terralab.ru/system/32754/
Часть 4. AMD Alchemy Au1550 — алхимия сетевой защиты // http://www.terralab.ru/system/32758/

[4] Волконский В.Ю., Тихонов В.Г., Эльцин Е.А., Матвеев П.Г. Реализация объектно-ориентированных языков программирования, гарантирующая межмодульную защиту - Высокопроизводительные вычислительные системы и микропроцессоры, сборник научных трудов, выпуск 4, М., ИМВС РАН, 2003, с. 18-37.