

Московский Государственный Университет

Факультет Вычислительной Математики и Кибернетики

Лаборатория Вычислительных Комплексов

Курсовая Работа

Разметка программ контрольными точками для контроля
поведения приложений в подсистеме безопасности ядра
Linux.

Федор Сахаров

группа 422

Научный руководитель:

Денис Гамаюнов

Москва, 2010

Аннотация

В работе рассматривается возможность расширения функциональности механизма контроля поведения программ, используемого в SELinux, при помощи повышения гранулярности контроля поведения приложений в указанной системе за счет отслеживания внутреннего состояния программы из ядра. Предлагается делать это при помощи разметки исполнимого кода приложения контрольными точками на уровне исходных текстов.

Содержание

| | | |
|-----|---|----|
| 1 | Введение. | 3 |
| 2 | Постановка задачи | 3 |
| 2.1 | Расшифровка темы | 3 |
| 2.2 | Актуальность | 4 |
| 2.3 | Цель работы | 4 |
| 3 | Обзор существующих систем безопасности уровня ядра ОС Linux и ОС Trusted BSD | 4 |
| 3.1 | SELinux | 5 |
| 3.2 | AppArmor | 8 |
| 3.3 | GRSecurity | 9 |
| 3.4 | Trusted BSD | 10 |
| 3.5 | Результаты рассмотрения существующих систем безопасности . . . | 12 |
| 4 | Решение задачи | 12 |
| 4.1 | Реализация системы контроля поведения наблюдаемого приложения | 12 |
| 4.2 | Выбор уязвимого сетевого приложения | 13 |
| 5 | Заключение | 14 |

1 Введение.

Стандартные системы безопасности ОС, основы которых были заложены несколько десятилетий назад, давно не являются удовлетворительными. Система безопасности Unix предоставляет одинаковые права всем пользователям в определенной группе, все процессы, запущенные от имени конкретного пользователя, обладают его привилегиями. Любая уязвимость становится потенциальной причиной компрометации учетной записи пользователя.

В 1985 году был введен стандарт «Критерии оценки доверенных компьютерных систем» более известный под названием «Оранжевая книга». Данный стандарт получил международное признание и оказал сильное влияние на последующие разработки в области информационной безопасности. Появилось семейство так называемых "trusted" операционных систем — TrustedBSD, Trusted Solaris, Trusted UNICOS 8.0, HP-UX 10.26, PitBull for AIX 5L, XTS-400. На сегодняшний день результатами разработки более современных и продвинутых систем безопасности, работающих поверх стандартных, стали такие продукты, как SELinux (NSA, Red Hat) [1], AppArmor (Immunix, Novell) [2], GRSecurity[3], Seatbelt(Apple). Кроме этого, разработчики некоторых систем пытаются расширить стандартные системы безопасности, улучшая их и добавляя новые способы защиты.

2 Постановка задачи

2.1 Расшифровка темы

Разработать и реализовать набор инструментов для разметки приложений (исходного кода и соответствующих бинарных программ) контрольными точками (software breakpoints).

Детализация постановки:

- Разработать и реализовать набор инструментов для простановки контрольных точек в программе, а также динамической установки таких точек в работающей программе.
- Выбрать пример сетевого приложения, в котором:
 - а) есть удалённо эксплуатируемая уязвимость,
 - б) данная уязвимость позволяет нарушить конфиденциальность, целостность

или доступность системы в случае компрометации даже при наличии политики SELinux (контрпример к SELinux).

Показать, как можно расставить контрольные точки в данном приложении, чтобы ущерб от атаки был минимальным.

- Разработать и реализовать механизм защиты контрольных точек от модификации в процессе выполнения.

2.2 Актуальность

Существующие механизмы защиты ядра и контроля поведения приложений в ОС Linux (SELinux, AppArmor) имеют ряд недостатков, в частности, используемые в них методы выявления аномального поведения приложений не учитывают внутреннее состояние защищаемого приложения. В ряде случаев это накладывает сильные ограничения на допустимое поведение, что ограничивает применимость этих механизмов. Учет внутреннего состояния контролируемого приложения позволит избежать жестких обобщенных ограничений на его поведение.

2.3 Цель работы

Расширение функциональности систем защиты уровня ядра Linux (SELinux) за счет повышения гранулярности отслеживания поведения приложений и разделения их внутренних состояний. Привести актуальный пример, позволяющий показать работоспособность разработанного механизма.

3 Обзор существующих систем безопасности уровня ядра ОС Linux и ОС Trusted BSD

Сравним системы безопасности уровня ядра ОС Linux (SELinux, AppArmor, GRSecurity) и Trusted BSD. Для сравнительного анализа были выбраны следующие критерии:

Реализованные модели безопасности.

Существуют различные модели безопасности такие, как Дискреционная(DAC), Мандатная(MAC), Принудительное присвоение типов на основании определенной политики(TE), Списки контроля доступа(ACL). Данный критерий отражает реализованные в рассматриваемой системе модели безопасности.

Наличие возможности изменять матрицу доступа во время исполнения.

Во многих из упомянутых выше моделях безопасности контроль событий в системе осуществляется на основании матрицы доступа. Матрица доступа является отображением декартова произведения множеств объектов и субъектов системы на множество, элементами которого являются наборы прав. Критерий отражает, есть ли возможность изменять матрицу доступа во время исполнения.

Возможность динамической смены контекстов приложения.

Для учета внутреннего состояния приложения в процессе контроля за его поведением может быть использована динамическая смена контекста безопасности приложения. Критерий описывает, существует ли в рассматриваемой системе возможность менять права приложения в процессе исполнения.

Классы вредоносных действий, предотвращаемых системой безопасности.

Разные системы безопасности предотвращают различные классы вредоносных действий. Некоторые системы идут по пути предотвращения заранее известных действий злоумышленника, другие позволяют минимизировать нанесенный ущерб от успешной атаки. Из-за этого на практике часто приходится комбинировать различные системы безопасности.

3.1 SELinux

SELinux является системой безопасности уровня ядра Linux, основанной на подсистеме LSM. LSM позволяет создавать модули безопасности. Данные модули должны реализовывать определенную логику принятия решений о разрешении или запрещении различных взаимодействий между объектами и субъектами системы.

Под объектами системы здесь понимаются файлы, объекты межпрограммного взаимодействия, объекты сетевого взаимодействия и прочие. Субъекты представляют собой пользовательские процессы, демоны, ядро и т.д..

SELinux обеспечивает возможность комплексной защиты системы, ограничивая поведение приложений и пользователей в рамках политик безопасности. В первую очередь, SELinux направлена на борьбу с успешными атаками, в частности, с атаками нулевого дня, когда уязвимость уже известна злоумышленнику, но лекарства еще не было выпущено. В таких случаях уязвимость локализуется на уровне политики. Компания Tresys ведет подсчет конкретных случаев угроз безопасности, которые, в частности, могли быть предотвращены SELinux. В их числе: перепол-

нение буфера в Samba (may 2007), Apache DoS (jun 2007), Mambo exploit (jul 2007), hplip Security flaw (oct 2007).

Конфигурация политик является сложной задачей из-за необходимости описывать профили для каждого приложения вручную на специальном языке описания политик. Добавление новых профилей может повлечь за собой необходимость в модификации уже имеющихся профилей, что может привести к появлению ошибок и росту накладных расходов на администрирование системы.

Реализованные модели безопасности.

Принудительное присвоение типов (TE).

Основной идеей принудительного присвоения типов является явная разметка всех объектов в системе специальными структурами данных (метками безопасности), хранящими в себе информацию об атрибутах объекта, используемую при принятии решений внутри логики политики. Для процессов и объектов используется один и тот же тип атрибутов. Поэтому достаточно одной матрицы для описания взаимодействий между разными типами, при этом объекты одного типа могут рассматриваться по-разному, если их ассоциированные классы безопасности различны. Пользователи не привязаны к типам безопасности напрямую, вместо этого используется RBAC.

Ролевой контроль доступа (RBAC)

Данный метод используется для определения множества ролей, которые могут быть назначены пользователям. SELinux расширяет модель RBAC до жесткой привязки пользовательских ролей к определенным доменам безопасности, роли могут быть организованы в виде иерархии приоритетов. Такая привязка ролей к доменам позволяет принимать большинство решений на основе конфигурации TE. Контекст безопасности, кроме всего прочего, включает в себя атрибут роли.

Многоуровневая система безопасности (MLS)

SELinux предоставляет MLS для случаев, когда есть необходимость в традиционной многоуровневой системе безопасности. У объектов и субъектов могут быть различные уровни и категории. Как правило, используется лишь один уровень.

Наличие возможности изменять матрицу доступа во время исполнения.

SELinux Предоставляет возможность перезагружать политику во время работы системы.

Возможность динамической смены контекстов приложения

SELinux предоставляет разработчикам приложений инструментарий, позволяющий создавать более безопасные приложения. Этого можно достичь путем изменения текущих привилегий приложения во время его исполнения. Последнее реализуется путем изменения домена приложения. Приложение должно запросить у ядра ОС смену своего текущего домена на указанный. При этом возможность такой смены доменов должна быть явно описана в политике безопасности. Далее данный метод будет рассмотрен более подробно.

Классы вредоносных действий, предотвращаемых системой безопасности.

В отношении системы безопасности SELinux было бы неправильно говорить о предотвращении угроз. Кроме этого, система не оперирует классами угроз. Основной идеей SELinux является минимизация ущерба от успешных атак на приложение. Для этого накладываются жесткие рамки на поведение приложений.

Принципы работы

Главными элементами системы безопасности являются субъект, объект и действия. В классы объектов входят классы файлов (`blk_file`, `chr_file`, `dir`, `fd`,...), классы межпрограммного взаимодействия (`ipc`, `msg`, `msgq`, `sem`, `shm`), классы сетевого взаимодействия (`key_socket`, `netif`, `node`, `packet_socket`, `tcp_socket`), классы объектов (`passwd`), системные классы (`capability`, `process`, `Security`, `System`). Под субъектами понимаются процессы, демоны, ядро и т.д.. Действия, которые субъекты SELinux могут производить над объектами различны для различных классов объектов. Для классов файлов это, например, будут создание, исполнение, ссылки, чтение, запись, удаление.

SELinux ассоциирует атрибуты безопасности с субъектами и объектами и основывает свои решения на этих атрибутах. Атрибутами являются: идентификатор пользователя, роль и тип. Идентификатор пользователя — пользовательская учетная запись, ассоциированная с субъектом или объектом. У каждого пользователя может быть несколько ролей, но в какой-то конкретный момент времени ему может быть предписана только одна из них. Пользователь может менять роли командой `newrole`. Типы (для процессов — Домены) делят субъекты и объекты на родственные группы. Это — главный атрибут безопасности, используемый SELinux для принятия решений.

Типы позволяют помещать процессы в "песочницы" и предотвращать повышение привилегий. К примеру, роль суперпользователя — `sysadm_r`, его тип — `sysadm_t`. Политика безопасности SELinux загружается системой из бинарного файла политики, который, как правило, находится в `/etc/selinux`. Бинарная по-

литика собирается при помощи `make`, исходные коды, как правило, находятся в `/etc/selinux/${POLNAME}/src/policy`.

Инструменты работы с SELinux могут быть разделены на три категории: специальные утилиты для настройки и использования SELinux, модифицированные версии стандартных команд и программ Linux, некоторые добавочные инструменты, к примеру, для настройки и анализа политик. Среди основных команд можно выделить следующие: `chcon` – помечает файл или группу файлов указанным контекстом безопасности, `checkpolicy` – позволяет выполнять множество действий, связанных с политиками, в том числе, компиляцию политики и ее загрузку в ядро; `getenforce` — позволяет узнать в каком режиме работает SELinux, `newrole` – позволяет пользователю перемещаться между ролями; `run_init` — позволяет запускать, останавливать или контролировать сервис; `setenforce` позволяет менять режим работы системы; `setfiles` присваивает метки указанной директории и ее поддиректориям. Некоторые из измененных программ: `cron`, `login`, `logrotate`, `pam`, `ssh`. Некоторые инструменты: `Apol` – инструмент для анализа файла `policy.conf`; `SeAudit` – инструмент для анализа логов, имеющий графический интерфейс; `SeCmds`; `SePCuT` — инструмент для просмотра и редактирования файлов политик; `SeUser` — модификация пользовательских учетных записей.

3.2 AppArmor

AppArmor является системой безопасности уровня ядра ОС Linux, разрабатываемой компанией Novell. AppArmor использует LSM аналогично SELinux.

Тем не менее, данная система безопасности не использует явную разметку всех объектов в системе. Вместо этого она контролирует поведение приложений, опираясь на профили поведения приложений, описанные на некотором интерпретируемом языке. В данных файлах хранится информация, основанная на путях к объектам в файловой системе, о том, к каким объектам и с какими правами имеет доступ приложение.

В отличие от SELinux'а, в котором настройки глобальны для всей системы, профили AppArmor разрабатываются индивидуально для каждого приложения. Таким образом, гораздо меньше вероятность необходимости изменения существующих профилей при генерации новых профилей. Кроме этого, AppArmor предоставляет инструменты автоматической генерации профилей на основе поведения приложения и возможность производить контроль в двух режимах: режиме обучения и режиме принуждения.

Реализованные модели безопасности.

В AppArmor реализовано принудительное присвоение типов.

Наличие возможности изменять матрицу доступа во время исполнения

Это возможно сделать путем редактирования конфигурационных файлов политики.

Классы вредоносных действий, предотвращаемых системой безопасности

Аналогично СБ SELinux, AppArmor позволяет минимизировать ущерб от успешных атак на систему. Контроль за всеми событиями в системе производится на основании определенной администратором политики безопасности.

Возможность динамической смены контекстов приложения

Система предоставляет возможность смены текущих привилегий для веб-сервера Apache (Change Hat). Тем не менее, из-за фактического прекращения разработки AppArmor данная система так и не стала доступной для использования с произвольным приложением.

3.3 GRSecurity

GRSecurity является набором патчей ядра Linux. Основными отличительными чертами данной системы безопасности являются возможность функционировать без настройки, защита от всех видов уязвимостей, связанных с модификацией адресного пространства процесса, возможность определять списки доступа и способность функционировать на различных аппаратных архитектурах.

Обнаружение атак на приложения осуществляется системой аудита. Механизм предотвращения атак реализован в PaX. Механизм ограничения действий приложений реализован в списках контроля доступа (Access Control Lists).

ACL представляет собой вариант матрицы контроля доступа, где с каждым объектом ассоциируется множество пар. Каждая из этих пар содержит субъект и набор правил.

PaX является набором патчей ядра, которые позволяют предотвращать атаки, связанные с модификацией адресного пространства приложений. Существует три класса угроз, предотвращением которых занимается PaX. Это внедрение и исполнение кода с повышенными привилегиями, исполнение кода самого процесса путем изменения нормального течения исполнения процесса, нормальное ис-

полнение программы, но над данными, для которых предусмотрены повышенные привилегии.

Несмотря на то, что система может функционировать без дополнительного администрирования политики безопасности, при необходимости, можно вносить изменения в политику. Во многом язык описания политик схож с языком в AppArmor.

Реализованные модели безопасности

В GRSecurity реализована модель принудительного контроля доступа на основании списков контроля доступа (ACL). Кроме этого, реализованы методы рандомизации ключевых локальных и сетевых информационных данных, ограничения на /proc, контроль сетевых сокетов, добавочные функции аудита.

Наличие возможности изменять матрицу доступа во время исполнения
Отсутствует.

Возможность динамической смены контекстов приложения
Отсутствует.

Классы вредоносных действий, предотвращаемых системой безопасности.

Позволяет заблокировать вредоносные действия, связанные с модификацией содержимого памяти внедрением вредоносного кода и последующим его исполнением.

К классам предотвращаемых вредоносных действий относятся

- Внедрение кода в приложение.
- Изменение нормального течения исполнения процесса.
- Исполнение программы с повышенными привилегиями.

3.4 Trusted BSD

TrustedBSD является проектом разработки расширения существующей системы безопасности FreeBSD, который включает в себя расширенные атрибуты UFS2, списки контроля доступа, OpenPAM, аудит событий безопасности с OpenBSM, мандатное управление доступом и TrustedBSD MAC Framework. Расширенные атрибуты UFS2 позволяют ядру и пользовательским процессам пометать файлы именованными метками. В этих метках хранятся данные, необходимые системе

безопасности. ACL и метки MAC в их числе. Списки контроля доступа являются расширениями дискреционного контроля доступа. Аудит системных событий позволяет вести избирательный аудит важных системных событий для последующего анализа, обнаружения вторжений, и мониторинга. Начиная с версии 5.0 в ядре FreeBSD появилась поддержка MAC Framework, прошедшая испытания в TrustedBSD. Данная подсистема позволяет создавать политики, определяющие принудительное присвоение доменов и типов (DTE), многоуровневую систему безопасности (MLS). MLS предоставляет интерфейсы управления этой подсистемой, примитивы для синхронизации, механизм регистрации политик, примитивы для разметки объектов системы, разные политики, реализованные в виде модулей политики MAC и набор системных вызовов для приложений. При регистрации политики, происходит регистрация специальной структуры (`struct mac_policy_ops`), содержащей функции MAC framework, реализуемые политикой.

Реализованные модели безопасности

В системе реализованы списки контроля доступа (ACL), мандатный контроль доступа (MAC), аудит событий безопасности.

Наличие возможности менять матрицу доступа во время исполнения

Такая возможность отсутствует.

Возможность динамической смены контекстов приложения

Фреймворк MAC позволяет реализовать в модуле безопасности возможность изменения приложением собственных прав. При этом, аналогично SELinux, приложение должно быть модифицировано соответствующим образом для использования данной возможности.

Классы угроз

Аналогично SELinux и AppArmor, TrustedBSD позволяет накладывать жесткие ограничения на поведение приложений. Эти ограничения описываются в политике безопасности системы. Основной целью ограничения поведения приложений является минимизация ущерба от атак "нулевого дня".

3.5 Результаты рассмотрения существующих систем безопасности

| Система Безопасности | Модели | Возможность менять матрицу доступа в процессе выполнения | Динамическая смена контекстов | Классы вредоносных действий, предотвращаемых системой |
|----------------------|-----------------------|--|--------------------------------------|--|
| SELinux | TE, MAC, RBAC | Существует | Существует | Минимизация ущерба от успешных атак |
| AppArmor | TE | Существует | Существует только для сервера Apache | Минимизация ущерба от успешных атак |
| GRSecurity | ACL | Отсутствует | Отсутствует | Внедрение кода в приложение и его исполнение, изменение нормального течения исполнения приложения, исполнение с повышенными привилегиями |
| Trusted BSD | MAC, ACL, RBAC, Audit | Отсутствует | Существует | Минимизация ущерба от успешных атак |

Существующие системы безопасности уровня ядра ОС Linux предоставляют широкие возможности контроля за поведением приложений. В частности, SELinux и Trusted BSD предоставляют возможность создавать приложения, интегрированные с этими системами безопасности. Тем не менее, рассмотренные механизмы уровня ОС не позволяют динамически менять права доступа приложения в зависимости от его состояния. Данная работа направлена на увеличение дискретности контроля поведения приложений за счет использования информации об их внутреннем состоянии.

4 Решение задачи

4.1 Реализация системы контроля поведения наблюдаемого приложения

Результатом работы прошлого года стали изменения в ядре Linux, позволяющие динамически переключать контексты приложения. Изменения были внесены в ядро версии 2.6.26. Также было предложено использовать подсистемы utrace и uprobes для простановки контрольных точек в наблюдаемом приложении.

В этом году был реализован модуль ядра, позволяющий динамическую смену контекста безопасности приложения при прохождении исполнения через кон-

трольные точки. В связи с переносом всего решения на ядро версии 2.6.32 возникла необходимость полностью переделать работу проделанную в прошлом году. Реализацию указанного модуля можно разделить на две составляющие.

Первой из них является использование систем `uprobes` и `utrace` для расстановки контрольных точек в наблюдаемых процессах. Были реализованы списки наблюдаемых процессов, списки контрольных точек в этих процессах, функции обработчики событий происходящих в наблюдаемых процессах.

Второй частью является полностью переделанная из-за переноса на ядро новой версии система поддержки динамических изменений контекстов SELinux. Эта система используется в реализованном модуле для обработки событий перехода процесса в новое состояние, которые предусматривают смену политики, применяемой к приложению.

Кроме этого было решено отказаться от модификации синтаксиса политик SELinux с целью сделать изменения вносимые в исходную систему минимальными. Вместо этого были реализованы интерфейсы получения из пользовательского пространства всей необходимой информации о наблюдаемых процессах и контрольных точках в них.

4.2 Выбор уязвимого сетевого приложения

Для доказательства актуальности разработанного механизма контроля и его корректной работы, необходимо продемонстрировать пример уязвимого приложения, которое является уязвимым несмотря на наличие политики SELinux, а реализованное средство позволяло бы минимизировать эффект от успешной атаки на данное приложение.

В качестве примера такого приложения был выбран сетевой демон OpenSSH. OpenSSH является типичным примером приложения, работающего с системными правами и с практически полным набором разрешающих прав SELinux. В том числе, в случае успешной атаки на буфер, политика SELinux, применяемая к данному приложению не запрещает злоумышленнику получить удаленный шелл, сделав вызов `system()` в уязвимом процессе. Это логично, так как демону `ssh` необходимы права на вызов функций `fork()` и `exec()`.

Тем не менее, пример уязвимости в этом приложении должен быть достаточно простой для эксплуатации.

За последнее время в указанном приложении не было обнаружено таких уязвимостей, поэтому было решено внести такую уязвимость в код приложения вруч-

ную.

В ходе более пристального исследования архитектуры OpenSSH оказалось, что разработчики реализовали разбиение приложения на состояния, обладающие различными правами. Технически это осуществляется при помощи порождения новых процессов с меньшими привилегиями, которые обрабатывают пользовательский ввод, операции с сетью и криптоалгоритмы. Эти новые процессы меняют свою корневую директорию на `/var/empty` делая невозможным запуск любых процессов из кода этого дочернего процесса, в том числе, в случае успешной атаки на обнаруженные в нем уязвимости.

Это существенно осложняет реализацию искусственного внесения уязвимостей в OpenSSH. Кроме того, описанная выше архитектура ставит под сомнение сам выбор OpenSSH в качестве требуемого примера. Кроме этого, стоит отметить, что множество других защитных механизмов, работающих в Linux делают искусственное внесение уязвимости в приложение достаточно сложным процессом. В числе таких механизмов можно отметить различные системы защиты стека от исполнения и подмены адреса возврата из функции, рандомизацию адресного пространства, права чтение, запись и исполнение различных секций памяти процесса.

Из-за указанных технических сложностей на данный момент не удалось привести требуемый пример уязвимого приложения.

4.3 Защита контрольных точек от модификации в процессе выполнения

При использовании контрольных точек сразу возникает вопрос в возможности их модификации и фальсификации злоумышленником. Для успешности контроля переходов приложения в новые состояния мы должны быть уверены в подлинности контрольных точек.

В реализации работы контрольной точкой является инструкция прерывания, записываемая в сегмент кода процесса. Очевидно, что сегмент кода не является модифицируемым, что позволяет считать контрольные точки надежными.

5 Заключение

В процессе решения поставленной задачи с использованием результатов работы прошлого года была разработана система контроля поведения приложения, раз-

личающая его внутренние состояния и корректирующая относительно изменений внутренних состояний ограничения, накладываемые системой SELinux на поведение приложения. Данная система была реализована в качестве модуля ядра. При этом, изменения в самом ядре Linux являются минимальными, а утилиты пользовательского пространства, входящие в проект SELinux остаются неизменными. Это позволяет легко поддерживать решение в процессе выхода новых версий ядра Linux.

Кроме этого, реализованное решение достаточно слабо зависит от SELinux, используя лишь функции изменения контекста приложения в процессе исполнения. Это позволит в случае необходимости легко перенести реализованную систему на другие системы контроля поведения приложений, существующие в Linux такие, как AppArmor и Tomoyo.

Исполнимый код наблюдаемого приложения размечается контрольными точками, которые используются для разграничения пространства его внутренних состояний. Расставленные в коде контрольные точки не могут быть изменены или удалены злоумышленником, как не могут быть добавлены и новые контрольные точки. Это позволяет считать информацию о смене состояний надежной.

На данный момент из-за высокой технической сложности не удалось привести требуемый пример уязвимого приложения или создать такой пример искусственно. Такой пример крайне желателен для обоснования актуальности проделанной работы и дальнейшие усилия будут сконцентрированы именно на этом.

Список литературы

- [1] Официальная документация SELinux [HTML]
(<http://www.nsa.gov/research/selinux/docs.shtml>)
- [2] Документация по проекту AppArmor [HTML]
(http://en.opensuse.org/AppArmor_Geeks)
- [3] Сайт проекта GRSecurity [HTML]
(<http://pax.grsecurity.net/>)
- [4] Jonathan Corbet, Greg Kroah-Hartman, Allesandro Rubini Linux Device Drivers, O'Reilly, 2005. 640 с.

- [5] Daniel P. Bovet, Marco Cesati, Understanding the Linux Kernel, O'Reilly, 2002, 568 с.
- [6] Страница utrace [HTML]
(<http://people.redhat.com/roland/utrace/>)