DOI 10.17223/20710410/2/16

УДК 681.322

РАСШИРЕНИЕ ФУНКЦИОНАЛЬНОСТИ СИСТЕМЫ БЕЗОПАСНОСТИ ЯДРА LINUX НА ОСНОВЕ ПОДМЕНЫ СИСТЕМНЫХ ВЫЗОВОВ

М.А. Качанов, Д.Н. Колегов

Томский государственный университет

E-mail: kden@sibmail.com

Обсуждаются некоторые известные средства безопасности для ядра Linux, основанные на перехвате и подмене системных вызовов, и предлагается модель системы реализации политик безопасности для ОС GNU/Linux на основе подмены системных вызовов, представляющая собой модуль ядра.

Ключевые слова: *перехват системных вызовов*, *политики безопасности*, *модуль ядра.*

В настоящее время существует значительное количество средств расширения систем безопасности и повышения уровня защищенности компьютерных систем (КС) на основе изменения функциональности ядер операционных систем (ОС) класса UNIX и Linux в частности. Большая их часть (SELinux, LSM, GrSecurity, Openwall, AppArmor и др.) устраняет уязвимости путем изменения исходного кода ядра ОС. Другие средства (Systrace, Janus, Ostia) используют перехват и подмену системных вызовов (СВ), но их реализация часто также затрагивает изменение кода ядра, библиотек или приложений и требует их перекомпиляцию.

СВ предоставляют интерфейс доступа к пространству ядра ОС из пространства пользователя для выполнения привилегированных операций. Основная идея заключается в перехвате СВ и выполнении функций, реализующих заданную политику безопасности (ПБ), вместо стандартных.

Так, на основе перехвата СВ возможно:

- реализовывать «песочницы» – среды, в которых действия процессов ограничены в соответствии с заданной ПБ [1, 2, 3];
- реализовывать различные формальные ПБ, например политику мандатного управления доступом Low-Watermark [4];
- обнаруживать и предотвращать атаки на уровне узла, осуществлять аудит в КС [3].

При этом имеется возможность не изменять исходный код ядра, а добавить требуемую функциональность применением загружаемых модулей ядра Linux. Такой подход используется, например, в реализации модели Low-watermark средства LOMAC. Недостатками данного подхода является сложность реализации, потенциальное увеличение количества уязвимостей в ядре и ограниченная переносимость. Целью данной работы является разработка и реализация загружаемого модуля ядра Linux, реализующего заданную ПБ на основе подмены СВ.

1. Обзор некоторых известных средств реализации ПБ

Исследования, связанные с подменой СВ, как одного из механизмов создания «песочниц», ведутся приблизительно с 1994 г. Классические системы такого рода, как правило, имеют *гибридную фильтрующую архитектуру* – на уровне ядра реализуются механизмы перехвата СВ и реализации ПБ, а на уровне пользователя – механизмы управления доступом [5, 6].

Основными проблемами систем с подобной архитектурой являются:

- неверное представление о состоянии ОС – для принятия решения в соответствии с ПБ необходимо учитывать возможные преобразования объектов;
- использование псевдонимов ресурсов;
- косвенные пути к ресурсам (UNIX domain sockets, core dumps, передача дескрипторов);
- состояние гонок;
- побочные эффекты запрещения системных вызовов.

Для устранения недостатков систем с фильтрующей архитектурой разработаны системы с *делегирующей архитектурой* (Ostia), где на уровне ядра запрещаются СВ, не соответствующие ПБ, а на пользовательском уровне агенты от имени ограниченных процессов осуществляют доступ к ресурсам [6].

Janus [1, 2, 3] – одно из первых средств реализации ограниченных сред выполнения недоверенного программного обеспечения (ПО) с помощью перехвата СВ. Первая версия Janus использовала механизмы виртуальной файловой системы /proc и системного вызова ptrace. Позднее было показано, что данные методы плохо подходят для изолирования процессов с точки зрения безопасности. Одним из недостатков реализа-

ции оригинальной системы Janus являлось отсутствие разрешения изменения корневого каталога для процесса [3].

В настоящее время Janus (J2) состоит из mod_janus – модуля ядра Linux, реализующего перехват СВ, и интерпретатора политик janus, выполняемого в пространстве пользователя. Механизмы обнаружения и предотвращения атак, автоматической генерации ПБ и аудита не реализованы.

Работа осуществляется по следующей схеме:

1) изолированный процесс выполняет СВ, который перехватывается;

2) модуль mod_janus уведомляет janus о наличии системного вызова, а процесс, выполнивший СВ, переводит в состояние сна;

3) janus запрашивает у mod_janus данные о процессе и принимает решение в соответствии с ПБ;

4) в зависимости от ПБ, системный вызов выполняется или возвращается сообщение об ошибке.

Systrace [3] является одним из самых известных средств создания ограниченных сред. Данное средство дополнительно позволяет обнаруживать и предотвращать атаки, осуществлять аудит и изменять (повышать) привилегии, что позволяет не использовать механизм setuid/setgid. На уровне ядра реализован быстрый механизм проверки системных вызовов на соответствие политике безопасности. При выполнении изолированным приложением СВ ядро выполняет его проверку без инспекции аргументов, при этом взаимодействия с препроцессором политик, расположенным в пространстве пользователя, не происходит. Обычно на этом уровне разрешаются такие системные вызовы, как read или write. Если ядро не может принять решение по системному вызову, то оно обращается к «демону» политик через устройство /dev/systrace, при этом системный вызов и его аргументы транслируются в слова системо-независимого языка.

Для файлов производится определение абсолютного имени и разрешение символических ссылок. При этом транслятор осуществляет нормализацию – оригинальные аргументы замещаются транслированными. Процесс на время принятия решения блокируется. Найдя политику для соответствующего вызова, «демон» политик возвращает в ядро соответствующий ответ, и процесс переходит в состояние готовности. Если политика не найдена, то решение запрашивается у пользователя через соответствующий интерфейс или СВ запрещается с возможностью возврата кода ошибок. Для схожих системных вызовов используется механизм псевдонимов для создания виртуальных СВ.

При аварийном завершении Systrace ядро завершает все приложения, находящиеся в состоянии мониторинга. Вновь созданные процессы наследуют политику процессов-родителей. Состояние гонок предотвращается описанным выше замещением аргументов СВ в ядре на аргументы, проверенные Systrace. Ядро выполняет только СВ, которые прошли проверку. Таким образом, можно, например, путем замещения аргументов, соответствующих именам файлов, организовать виртуальную файловую систему для приложения. Имеется возможность автоматической генерации ПБ. Для этого все выполняемые системные вызовы и их аргументы преобразуются в правила ПБ. Также реализована возможность повышения привилегий процессу для выполнения некоторых операций, например создания raw-сокета, благодаря чему приложения могут выполняться не с правами администратора системы. В настоящее время Systrace доступна для ОС NetBSD, OpenBSD и GNU/Linux.

Средство **LOMAC** [4] представляет реализацию модели мандатной ПБ Биба Low-Watermark в качестве загружаемого модуля ядра. После загрузки модуля происходит выделение сущностей высокого и низкого уровней целостности. К сущностям высокого уровня относятся службы ядра, системные файлы и приложения, библиотеки, а к низким – сервисы, к которым имеют доступ удаленные пользователи, скачанные файлы и т.д. Субъект получает доступ к объекту, если это не противоречит ПБ, реализуемым средствами ОС и LOMAC.

Ostia разработана Garfinkel [6] на основе анализа недостатков реализации Janus как делегирующий монитор ссылок, который осуществляет перенаправление СВ от изолированного приложения. Основными компонентами являются модуль ядра, библиотека эмуляции и агенты. Модуль ядра предназначен для запрещения выполнения всех СВ от изолированного ПО к защищаемым ресурсам напрямую. Когда модуль ядра перехватывает СВ, он передает управление обработчику, находящемуся в библиотеке эмуляции. Важно, что обработчик должен быть помещен в адресное пространство изолированной программы до ее выполнения и получения управления загрузчиком разделяемых библиотек – это достигается применением собственного загрузчика ELF-файлов. Затем происходит преобразование СВ в запрос к агенту. Запрос осуществляется через интерфейс сокетов. Каждый изолированный процесс имеет своего агента, который выполняет СВ в соответствии с принятой ПБ. В целях повышения производительности по умолчанию разрешенными считаются системные вызовы read, write, dup. Недостатком является необходимость исследования символических ссылок в пространстве пользователя для определения возможности доступа к файлу.

В [7] подмена СВ исследуется для возможности моделирования различных ПБ и реализации криптографических методов защиты информации в ОС. Развитием такого подхода является реализация ПБ в КС с помощью аспектно-ориентированного программирования (АОП). В [8] представляется инструмент, состоящий из языка АОП AspectTalk, виртуальной машины и транслятора с AspectTalk на язык этой машины, предназначенный для автоматизации процессов проектирования ПБ и их реализации в КС.

2. Описание схемы реализации ПБ

Расширение функциональности системы безопасности представляет собой реализацию политик безопасности, обеспечивающих контроль над выполнением системных вызовов. Каждая политика безопасности представляет собой однонаправленный связный список элементов отношения $S \times O \times A \times P$, где S, O, A и P – множества соответственно субъектов, объектов, субъектно-объектных взаимодействий и возможных действий. В качестве объектов могут выступать процессы и файлы; субъектами всегда являются процессы. Субъектно-объектными взаимодействиями могут быть отправка процессом сигнала другому процессу, чтение и запись в файл, порождение процесса и т.д. Возможными действиями являются разрешение, обход, запрет с возвратом кода ошибки, изменение самого взаимодействия (например, изменения содержимого буфера записи и посылаемого сигнала, комбинированные с протоколированием субъектно-объектного взаимодействия). В элементах политики безопасности субъекты идентифицируются по следующим параметрам: уникальный идентификатор, идентификатор пользователя, эффективный идентификатор пользователя, идентификатор группы, имя исполняемой команды, режим использования указанных параметров. Идентификация объектов происходит либо по вышеизложенному методу для процессов, либо по полным путям файлов в файловой системе для файлов. Субъектно-объектные взаимодействия определяются экземплярами политик безопасности – для каждого взаимодействия он свой. Действия задаются уникальным идентификатором, допустимым для данного субъектно-объектного взаимодействия, и кодом возможной ошибки. Под допустимостью понимается возможность применения действия к заданным субъекту, объекту и взаимодействию. Например, действие «удалить из буфера записи строку “string”» не может быть применено, если субъектом является процесс, объектом – файл, а взаимодействием – «открытие файла». Другими словами, допустимость действия определяется типами аргументов системного вызова и непосредственно самим системным вызовом (его номером).

Стоит уточнить, что каждому субъектно-объектному взаимодействию поставлено в соответствие некоторое подмножество множества всех системных вызовов (рассматриваются только те системные вызовы, разрешение структур ядра по аргументам которых, в принципе, позволяет определить некоторый файл или процесс). Данные подмножества не обязаны не пересекаться, но различным подмножествам не могут соответствовать одинаковые взаимодействия. При исполнении системного вызова идентификация текущего субъектно-объектного взаимодействия происходит путем поиска соответствий для множеств, которым принадлежит выполняемый в данный момент системный вызов. Если такое соответствие не единственно, то производятся анализ параметров, переданных системному вызову, и устранение такой неоднозначности. Например, пусть субъектно-объектному взаимодействию «создание файла» поставлено в соответствие множество системных вызовов {open, creat, mkdir}, а взаимодействию «открытие файла» соответствует множество {open}, и пусть выполняющийся системный вызов – open, и он не принадлежит никаким другим подмножествам, поставленным в соответствие другим субъектно-объектным взаимодействиям. Тогда указанный поиск соответствий даст в качестве результата два взаимодействия: «открытие файла» и «создание файла». В этом случае будет произведен анализ параметра flags (32-битный булев вектор) вызова open. Если побитовая конъюнкция вектора flags с наперед заданным вектором O_CREAT (24-битным), дополненным 8 нулевыми битами со стороны старших разрядов, даст ненулевой вектор, то результатом разрешения неоднозначности взаимодействия будет «создание файла», иначе – «открытие файла». Соответствие взаимодействий подмножествам системных вызовов задается так, что неоднозначность всегда можно устранить.

Дадим определения. *Активный субъект* – процесс, от имени которого выполняется системный вызов. *Активный объект* – объект, на который направлено действие системного вызова. *Соответствие* субъекта и объекта активным субъекту и объекту – совпадение характеристик первых с соответствующими характеристиками вторых с учетом режимов использования, указанных в элементе некоторой политики.

Контроль субъектно-объектных взаимодействий на основе политик безопасности осуществляется за счет перехвата системных вызовов следующим образом (см. рис. 1): в политике текущего взаимодействия осуществляется поиск элемента по субъекту и объекту, соответствующим текущим активным субъекту и объекту. Если такой элемент обнаружен, то выполняется действие, указанное в элементе политики; просмотр списка заканчивается, если только действие не является обходом, иначе выполняется действие по умолчанию. Таким образом, каждая политика является упорядоченной последовательностью фильтров для заданного взаимодействия.

На основе вышеизложенной схемы разработано средство, представляющее собой загружаемый модуль ядра Linux 2.6.x. Достоинствами такого подхода являются более высокая производительность по сравнению со средствами, использующими уровень пользователя, а также отсутствие необходимости изменения исходного кода ядра и его перекомпиляции и возможность динамической загрузки и выгрузки модуля в процессе работы операционной системы.

Политики безопасности представлены элементами структур ядра, имеющими поля, соответствующие характеристикам политик безопасности. Выбрано около 20 основных системных вызовов, удовлетворяющих оговоренным выше условиям.

Для каждого системного вызова написана функция-перехватчик с теми же аргументами, в действия которой входит:

1) определение текущего взаимодействия;

2) разрешение по переданным аргументам элементов структур ядра и абсолютных имен файлов с целью определения активных субъекта и объекта (разрешались дескрипторы, уникальные идентификаторы и относительные пути);

3) анализ соответствующего системному вызову экземпляра политики безопасности и выполнение вышеопределенного действия (рис. 1);

4) возврат кода ошибки, если это предусмотрено в элементе политики, либо возврат результата работы оригинального системного вызова.

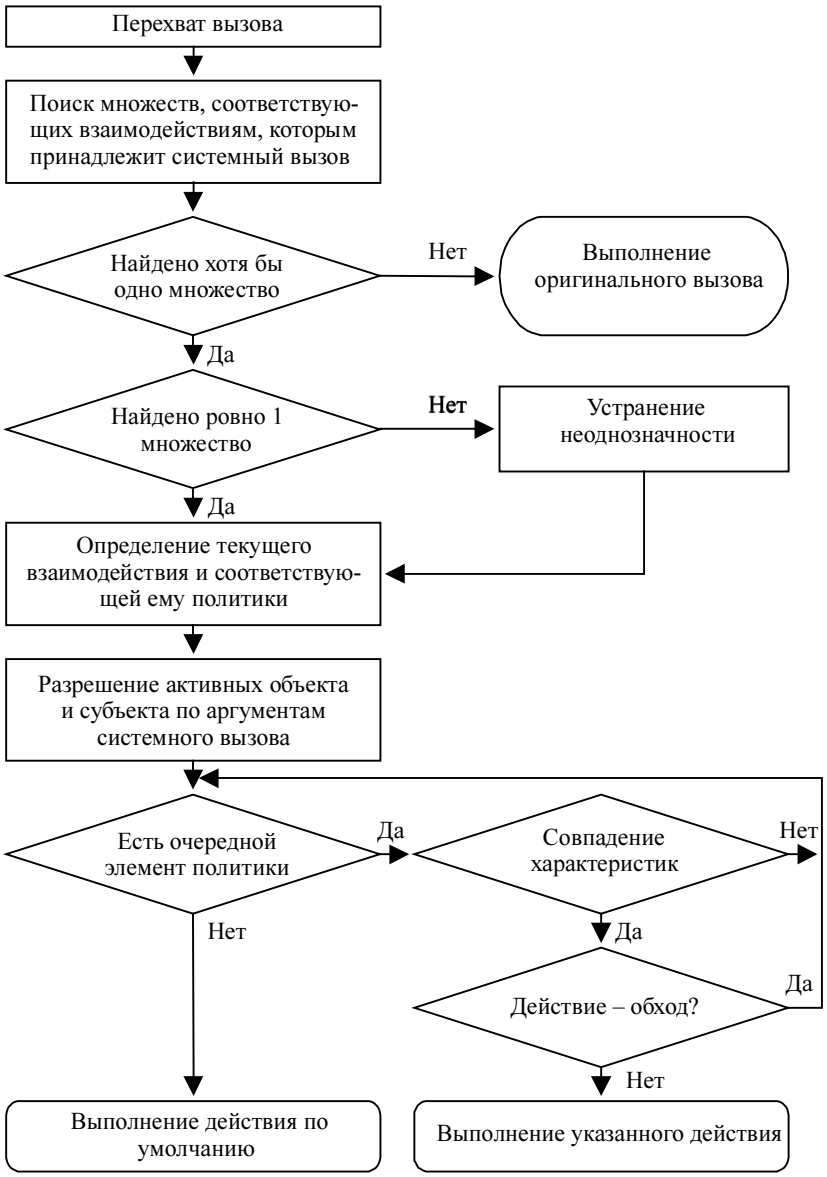

Рис. 1

3. Загружаемый модуль ядра, реализующий ПБ

При инициализации модуля вычисляется адрес таблицы системных вызовов, а адреса точек входа в ядро заменяются адресами соответствующих им функций-перехватчиков, при этом адреса оригинальных функций сохраняются в памяти. Таким образом, при всяком выполнении процессом системного вызова вызывается функция-перехватчик. Также в процессе инициализации модуля выполняются создание экземпляров политик и подготовка файла протоколирования, имя которого можно задать при загрузке модуля.

Для задания ПБ реализован интерпретатор, разбирающий строки ввода пользователя, написанные на определенном языке. Функциональными возможностями интерпретатора являются: добавление и удаление

элементов политик безопасности с указанием объектов, субъектов, взаимодействия и действия модуля, вывод политик безопасности и изменение порядка элементов в списке, т.е. управление фильтрацией. Причем вывод осуществляется на терминал, соответствующий текущему процессу, определяемому макросом current.

Взаимодействие с модулем ядра происходит по следующей схеме: интерпретатор разбирает строку ввода пользователя и в случае, если она корректна, формирует массив аргументов системного вызова execve, в котором в определенном формате указываются действия, выполняемые модулем ядра для формирования списка ПБ, и выполняет системный вызов. Модуль перехватывает системный вызов execve, разбирает строку аргументов и изменяет ПБ.

При выгрузке модуля происходит заполнение измененных элементов таблицы системных вызовов оригинальными значениями, выполняется освобождение памяти, выделенной под политики, и закрытие файла-протокола. Схема работы средства представлена на рис. 2.

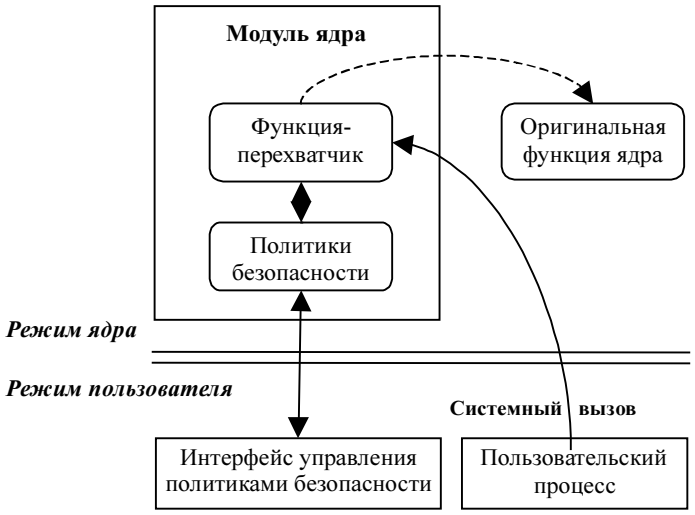

Рис. 2

В данной реализации с помощью модуля возможно осуществить сокрытие процессов и файлов, контроль порождения и клонирования процессов, контроль доступа к файлам, глубокий аудит выполняемых в компьютерной системе действий.

Заключение

В данной работе были описаны некоторые существующие средства расширения функциональности системы безопасности ядра Linux на основе подмены системных вызовов. Предложена собственная модель такой системы, использующая политики безопасности. Разработано средство, реализующее данную модель, в виде загружаемого модуля ядра с возможностью задания политик безопасности на определенном языке. Перспективными направлениями развития являются введение абстрактного типа ПБ на уровне реализации, исследование возможности реализации формальных моделей безопасности. Планируется также исследование возможностей применения данного средства и создание гибкого интерфейса работы со средством.

ЛИТЕРАТУРА

1. *Goldberg I*., *Wagner D*., *Thomas R*., *Eric A*. Brewer. A Secure Environment for Untrusted Helper Applications // Proceedings of the 6th Usenix Security Symposium, July 1996.
2. *Wagner D.A*. Janus: an approach for confinement of untrusted applications. Technical Report CSD-99-1056. 12. 1999. http://citeseer.ist.psu.edu/wagner99janus.html.
3. *Provos N*. Improving Host Security with System Call Policies, November 2002. http://citeseer.ist.psu.edu/provos02improving.html.
4. *Fraser T*. LOMAC: MAC You Can Live With. – USENIX, 2001. – http://portal.acm.org/citation.cfm?id=71575.
5. *Garfinkel T*. Traps and Pitfalls: Practical Problems in System Call Interposition Based Security Tools. www.stanford.edu/~talg/papers/traps/traps-ndss03.pdf.
6. *Garfinkel T*., *Pfaff B.*, *Rosenblum M*. Ostia: A Delegating Architecture for Secure System Call Interposition. http://citeseer.ist.psu.edu/garfinkel03ostia.html.
7. *Куликов М.Л.*, *Ромашкин Е.В.*, *Стефанцов Д.А.* Разработка средств моделирования политик безопасности операционных систем // Вестник ТГУ. Приложение. 2007. № 23. С. 189 – 193.
8. *Стефанцов Д.А.* Реализация политик безопасности в компьютерных системах с помощью аспектно-ориентированного программирования // Прикладная дискретная математика. 2008. № 1. С. 94 – 100.