УДК 004.056.53

А.В. Анисимов, Е.Ю. Иванников

Подсистема криптографического обеспечения конфиденциальности данных в защищенной ОС на базе ОС *GNU/Linux* с расширением *RSBAC*

Рассмотрена задача обеспечения конфиденциальности данных, обрабатываемых в защищенной операционной системе, при их случайной либо намеренной утечке. Предложен проект подсистемы комплекса средств защиты, обеспечивающей автоматизированную избирательную криптографическую защиту информации. Описаны изменения в ядре базовой операционной системы, необходимые для реализации подсистемы.

A problem of ensuring the data confidentiality in a trusted OS in case of the accidental or intentional leakage is considered. A design for a subsystem of a trusted computing base is suggested which provides the automated selective cryptographic information protection. The modifications of the OS kernel needed to implement the subsystem are described.

Розглянуто задачу забезпечення конфіденційності даних, що оброблюються у захищеній операційній системі, у випадку їхнього випадкового або навмисного витоку. Запропоновано проект підсистеми комплексу засобів захисту, що забезпечує автоматизований вибірковий криптографічний захист інформації. Описано зміни в ядрі базової операційної системи, необхідні для реалізації підсистеми.

Введение. Обеспечение конфиденциальности обрабатываемой информации (наряду с обеспечением целостности) – одна из важных задач, возлагаемых на защищенную операционную систему (ЗОС). Она с успехом решается с помощью средств административного и/или доверительного размежевания доступа [1], основывающихся на известных моделях безопасности, например, на моделях Белла–ЛаПадула [2] и Харрисона–Руззо–Ульмана [там же]. В то же время механизмы размежевания доступа не способны противостоять таким угрозам безопасности, как кража конфиденциальных данных (отдельно или вместе с оборудованием) или случайная утечка вследствие неосторожных действий персонала (например, потеря носителей). При этом вероятность случайных утечек оказывается неожиданно большой. Так, согласно данным аналитической лаборатории *InfoWatch* [3], из всех известных случаев утечки в 2008 г. на долю случайных приходится 46%, еще 42% – на долю намеренных (оставшиеся 12% составляют утечки неустановленного характера).

Естественное решение этой проблемы лежит в применении криптографических механизмов, призванных обеспечить конфиденциальность и целостность данных в ситуации, когда они выходят из сферы контроля комплекса средств защиты (КСЗ). Криптографические механизмы могут применяться либо ко всему носителю (разделу) целиком, либо к отдельным файлам. Преимущество первого подхода заключается в полном охвате криптографической защитой всей информации, хранимой на носителе. К недостаткам следует отнести отсутствие избирательности, что приводит к избыточному шифрованию некритической информации и, как следствие, к падению производительности компьютерной системы (КС). Наконец, все данные на одном носителе (разделе) окажутся зашифрованными с помощью одного ключа (возможно, мастер-ключа), утечка которого приведет к их компрометации. Альтернативный подход заключается в шифровании отдельных файлов, содержащих критическую информацию. Обеспечивая высокую избирательность защиты, этот подход выдвигает очень жесткие требования к персоналу, перекладывая на конечных пользователей задачу выбора информации, нуждающейся в защите, и главное, задачу по осуществлению криптографических операций (запуск программ для расшифрования данных перед их обработкой, а затем для зашифрования результатов).

Следует также учесть, что обработка информации – процесс, сопровождающийся созданием временных файлов и, возможно, новых фай-

лов, содержащих итоговые результаты обработки, что еще более усложняет задачу контроля. Последнее обстоятельство имеет очень большое значение в указанной ранее значительной вероятности именно случайных утечек.

Для авторов статьи очевидно, что конфиденциальность критических данных не может быть гарантирована в такой КС, где ее обеспечение зависит от конечных пользователей.

Таким образом, существует актуальная задача по разработке подсистемы криптографической защиты критической информации, лишенной отмеченных недостатков, а именно, обеспечивающей автоматизированную избирательную защиту.

Эта статья – часть научно-исследовательской работы по созданию ЗОС, которая проводилась под руководством проф. Анисимова А.В. на факультете кибернетики КНУ имени Тараса Шевченко. Профиль безопасности разрабатываемой ЗОС включает в себя услугу административной конфиденциальности «КА-2» [1] на основе использования мандатной модели Белла–ЛаПадула.

В основе предлагаемого решения лежат следующие требования: подсистема криптографической защиты должна функционировать в составе КСЗ; при этом подсистема должна быть частью административной политики безопасности. Необходимо добиться того, чтобы любой обозначенный меткой секретности (уровнем доступа) файл хранился на носителе в зашифрованном виде, причем шифрование должно осуществляться без участия пользователя. Поэтому реализация политики мандатной модели безопасности будет расширена с тем, чтобы присвоение уровней доступа существующим файлам, а также создание новых файлов субъектами, имеющими соответствующие уровни допуска, сопровождалось шифрованием данных.

Обзор защищенной среды

Разрабатываемая ЗОС базируется на дистрибутиве *Debian* ОС *GNU*/*Linux* с расширением безопасности *RSBAC*. Расширение *RSBAC* представляет собой надстройку над ядром ОС *Linux* и набор утилит администрирования. *RSBAC* функционирует на уровне ядра ОС и добавляет собственные проверки к системным вызовам. Система *RSBAC* разработана на основе обобщенной модели размежевания доступа *GFAC* (*Generalized Framework for Access Control*), предложенной в [4].

Подробнее с системой *RSBAC* можно ознакомиться в работе ее создателя [5], а также в статьях одного из авторов [6, 7]. Дадим краткий обзор.

В архитектуре *RSBAC* средства для принудительного размежевания доступа, средства для принятия решения о допустимости или недопустимости доступа, а также данные про атрибуты безопасности сторон – участников доступа реализуются как отдельные компоненты. Структурно механизм защиты состоит из трех частей:

- блок контроля системных вызовов (*Access Enforcement Facility – AEF*);
- блок принятия решений (*Access Decision Facility – ADF*);
- хранилище информации про атрибуты безопасности объектов и субъектов КС (*Access Control Information – ACI*).

Системные вызовы дополняются кодом блока контроля *AEF*, преобразующего их в один из стандартных (определенных в системе *RSBAC*) запросов к блоку принятия решений *ADF*. Последний проводит опрос модулей политик безопасности, принимая решение о допустимости или недопустимости вызова. Модули политик безопасности получают информацию для принятия решений из хранилища атрибутов безопасности *ACI*.

Требования к подсистеме криптографической защиты

Основное выдвигаемое требование – следующее: любой защищенный объект с уровнем доступа, превышающим пороговое значение, должен быть представлен на носителе (разделе) в зашифрованном виде. Мы не требуем выполнения обратной импликации, поскольку оставляем возможность для конечного пользователя осуществлять шифрование данных, неклассифицированных в модели мандатного управления, по своему усмотрению (подлежащих, например, доверительному контролю).

Требования к криптографической подсистеме естественны и состоят из требований к стойкости шифрования и удобству эксплуатации. Таким образом, подсистема криптографической защиты должна:

1. Использовать современные стойкие алгоритмы, а также обеспечивать возможность конфигурирования параметров.

2. Хранить ключи шифрования в зашифрованном виде, возможно, на отчуждаемом носителе.

3. Быть прозрачной для конечного пользователя и не требовать от него каких-либо специальных знаний и/или действий.

4. Обеспечивать автоматизированную избирательную криптографическую защиту.

5. Исключить возможность утечек расшифрованных данных (например, в виде каких-либо временных файлов или страниц памяти, перемещенных в область подкачки).

В качестве основы для реализации требований (1–3) выбрана стековая шифрующая файловая система (ФС) *eCryptFS*, описанная в следующем подразделе. Требование (4) будет обеспечено благодаря внедрению шифрующей ФС в состав КСЗ, чему посвящается отдельный подраздел статьи. Выполнение требования (5) – часть функциональности, обеспечиваемой услугой безопасности по контролю объектов (КО-1).

Описание стековой шифрующей файловой системы *eCryptFS*

Для ОС *GNU*/*Linux* толчком к развитию шифрующих файловых систем послужила работа [8], в которой введена технология стековых ФС. Суть технологии составляет то, что некоторая ФС монтируется над другой смонтированной ФС. Первая ФС (верхнего уровня) отвечает за необходимые преобразования данных (например, за шифрование) и вызывает соответствующие процедуры ФС нижнего уровня, отвечающей за непосредственное осуществление операций чтения, записи и т.д. На рис. 1 представлена схема стековой технологии на примере ФС *eCryptFS*.

Стековые ФС – один из наиболее перспективных путей развития ФС для ОС типа *Linux*. Их несомненным преимуществом является то, что они могут использовать в качестве ФС нижнего уровня одну из хорошо известных ФС, например *ext*3. Длительный период развития ФС *ext*3 и ее большая распространенность дает гарантии отсутствия программных ошибок. Вся же дополнительная функциональность обеспечивается верхним слоем, что облегчает как тестирование, так и сопровождение ПО. Приведенные аргументы имеют особое значение, поскольку ЗОС предназначена для обработки информации, имеющей высокую ценность.

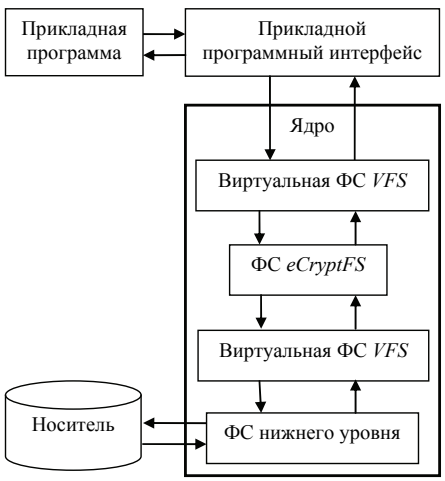

Рис. 1. Стековая файловая система *eCryptFS*

Одна из наиболее известных стековых шифрующих ФС – ФС *eCryptFS* [9], корни которой уходят в прототип, предложенный в работе [8]. Будучи смонтированной в определенном каталоге, ФС *eCryptFS* обеспечивает прозрачное шифрование и расшифрование файлов. При этом, для осуществления всех криптографических операций *eCryptFS* использует современные алгоритмы шифрования, реализованные в виде модулей ядра ОС.

Последнее обстоятельство позволяет использовать для шифрования файлов иной алгоритм (например, отвечающий национальным стандартам), для чего достаточно реализовать такой алгоритм в виде нового модуля ядра, не модифицируя ФС.

Отметим некоторые (основные) преимущества ФС *eCryptFS*.

- ФС реализована на уровне ядра ОС, что уменьшает накладные расходы, связанные с необходимостью переключения между контекстом ядра и контекстом пользователя.

• Шифрование и расшифрование файлов осуществляется прозрачно для пользователя. Фактически, пользователь не замечает разницы при доступе к обычным и зашифрованным файлам.

• Криптографические метаданные являются частью зашифрованного файла, хранясь в его заголовке. Это обеспечивает простоту при обмене файлов как в пределах одного узла, так и между узлами, поскольку отпадает необходимость в переносе вместе с файлом дополнительной информации. Зашифрованные данные могут быть восстановлены в другой среде лишь с помощью верного ключа.

• Гибкость в выборе криптографических алгоритмов и их параметров.

• Ключи шифрования (мастер-ключи) могут вводиться пользователем, могут храниться на носителе, в том числе, отчуждаемом.

Несколько слов об особенностях криптографического механизма, используемого в ФС *eCryptFS*. Для каждого файла создается уникальный ключ, используемый симметрическим алгоритмом для шифрования данных в файле. Ключ шифрования файла шифруется с помощью мастер-ключа пользователя. Пользовательский ключ может формироваться на основе идентификационной фразы либо быть открытым ключом ключевой пары.

Шифрование и расшифрование данных осуществляется посредством прикладного программного интерфейса ядра. Операции выполняются над областями, размеры которых составляют размер страницы памяти в архитектуре целевой КС. Каждая область шифруется отдельно, при этом используется вектор инициализации, который получают из корневого вектора инициализации, связываемого с каждым файлом. Данные в пределах одной области шифруются симметрическим алгоритмом блочного шифрования, в режиме CBC (сцепления блоков по шифротексту).

Модуль административного контроля системы размежевания доступа *RSBAC*

В состав системы *RSBAC* входит модуль *MAC* (*Mandatory Access Control*), реализующий мандатную модель размежевания доступа на основе расширения модели Белла–ЛаПадула. Необходимость расширения модели продиктована потребностью контролировать большое число возможных операций доступа к объектам (всего более 50 запросов в терминах *RSBAC*), а также сложностью использования модели в современной КС. Классическая модель Белла–ЛаПадула рассматривает лишь операции чтения *read* и записи *write*. В модуле *MAC* на основе направления потока информации между объектом и субъектом, вводится сопоставление каждому запросу одной из трех операций: чтение (*read*), запись (*write*) и чтение–запись (*read–write*). В таблице указано такое соответствие для некоторых контролируемых запросов.

Т а б л и ц а. Соответствие некоторых запросов *RSBAC* операциям модели Белла–ЛаПадула

Запрос *RSBAC*	Операция доступа
APPEND_OPEN	*read*
READ_OPEN	*read*
EXECUTE	*write*
WRITE_OPEN, CREATE	*write*
MOUNT	*read–write*
READ_WRITE_OPEN	*read–write*

Управление атрибутами безопасности мандатной модели, в том числе уровнями допуска/доступа, может осуществлять лишь уполномоченный пользователь (администратор безопасности). Другое направление расширения классической модели состоит в том, что в результате выполнения операций доступа уровень допуска субъектов и уровень доступа объектов может изменяться (без участия администратора безопасности) в заданных пределах. Остановимся на этой возможности несколько подробнее, поскольку это имеет значение для дальнейшего изложения.

С каждым субъектом (процессом) модуль мандатного управления связывает такие атрибуты безопасности:

- $l_{\min}$ – минимальный уровень допуска;
- l_0 – начальный уровень допуска;
- $l_{\max}$ – максимальный уровень допуска;
- l_{cur} – поточный уровень допуска;
- $l_{\min\_\text{write}}$ – минимальный среди уровней доступа всех тех объектов, доступ к которым осуществлялся субъектом в режиме *write* или *read–write* (с момента инициации субъекта);

• $l_{\max\_read}$ – максимальный среди уровней доступа всех тех объектов, доступ к которым осуществлялся субъектом в режиме *read* или *read–write* (с момента инициации субъекта);

• *MAC_auto* – атрибут субъектов, позволяющий им корректировать свой уровень допуска; этот атрибут также может быть установлен у объектов, позволяя субъектам изменять уровень доступа таких объектов в результате выполнения операций;

• некоторые другие, не имеющие непосредственного отношения к данной статье.

Уровень доступа объекта будем обозначать через l_{target}.

На основе анализа исходных кодов системы *RSBAC* авторами были восстановлены алгоритмы проверки операций доступа. Эти алгоритмы отличаются некоторой громоздкостью, поэтому ниже приведены их фрагменты, отвечающие за изменение уровней допуска/доступа.

Доступ типа *read*

1. **если** $l_{\text{cur}} < l_{\text{target}} \le l_{\max}$, **то**
2. **если** у субъекта установлен атрибут MAC_auto, **то**
3. **если** $l_{\text{target}} \le l_{\min\_write}$, **то** поднять уровень допуска субъекта до уровня доступа объекта: $l_{\text{cur}} \leftarrow l_{\text{target}}$.
4. **конец если**
5. **конец если**

Доступ типа *write*

1. **если** $l_{cur} < l_{target} \le l_{max}$, **то**
2. **если** у субъекта установлен атрибут *MAC_auto*, **то** поднять уровень допуска субъекта до уровня доступа объекта: $l_{\text{cur}} \leftarrow l_{\text{target}}$.
3. **конец если**
4. **иначе если** $l_{\min} \le l_{\text{target}} < l_{\text{cur}}$, **то**
5. **если** у субъекта установлен атрибут MAC_auto, **то**
6. **если** $l_{\text{target}} < l_{\max\_read}$, **то**
7. **если** субъект не может нарушать запрет на запись вниз, **то**
8. **если** для объекта может быть нарушен запрет на запись вниз и у объекта установлен атрибут MAC_auto, **то** поднять уровень доступа объекта до уровня допуска субъекта: $l_{\text{target}} \leftarrow l_{\text{cur}}$.
9. **конец если**
10. **иначе если** $l_{\text{target}} \ge l_{\max\_read}$, **то** поднять уровень допуска субъекта до уровня доступа объекта: $l_{\text{cur}} \leftarrow l_{\text{target}}$.
11. **иначе если** у субъекта не установлен атрибут MAC_auto, **то**
12. **если** субъект не может нарушать запрет на запись вниз, **то**
13. **если** для объекта может быть нарушен запрет на запись вниз и у объекта установлен атрибут MAC_auto, **то** поднять уровень доступа объекта до уровня допуска субъекта: $l_{\text{target}} \leftarrow l_{\text{cur}}$.
14. **конец если**
15. **конец если**
16. **конец если**
17. **конец если**

Для доступа типа *read–write* логика работы по корректировке уровней допуска/доступа соответствует логике работы при доступе типа *write*.

Таким образом, имеет место такой факт. Уровень доступа объекта файловой системы может быть задан или изменен лишь в следующих случаях:

• создание нового файлового объекта процессом, имеющим некоторый назначенный уровень допуска (т.е., отличное от нуля значение);

• явная установка администратором безопасности нового уровня доступа для файлового объекта; при этом уровень доступа может быть как повышен, так и понижен;

• изменение уровня доступа файлового объекта, происходящее в результате выполнения запросов имеющих тип «запись» и «чтение–запись» согласно приведенным алгоритмам; при этом уровень доступа может лишь повыситься.

Проект реализации автоматизированной избирательной криптографической защиты

Обозначим некоторый уровень доступа файлов как пороговый уровень (для криптографической защиты). Смысл определения заключа-

ется в том, что любой файл с уровнем доступа, равным либо превышающим пороговое значение, должен быть зашифрован.

Для включения ФС *eCryptFS* в состав КСЗ необходима ее модификация, вызванная тем, что *eCryptFS* не поддерживает монтирования в директорию, содержащую незашифрованные файлы. Изменения исходных кодов позволят *eCryptFS* осуществлять операции над незашифрованными файлами. Кроме того, *eCryptFS* зашифровывает все файлы при их создании. Модификация ФС обеспечит избирательное шифрование, контролируемое политикой административной безопасности.

Также метод ФС, предназначенный для возврата справочной информации о ФС (метод *ecryptfs_statfs*(), являющийся членом структуры *super_operations*), должен быть изменен для возвращения вызывающей функции нового маркера ФС (так называемое «магическое» число). Это связано с тем, что КСЗ должен уметь идентифицировать ФС *eCryptFS* (т.е. иметь информацию о том, находится ли некоторый файл на ней). В настоящее время метод *ecryptfs_statfs*() возвращает значение маркера ФС нижнего уровня.

В структуры данных ФС *eCryptFS* будет введен новый внутренний атрибут, связываемый с файлами и указывающий на то, что файл необходимо зашифровать.

Операция шифрования файла должна быть реализована с учетом того, что файл может быть одновременно открыт несколькими процессами. Предлагаем проводить шифрование файла во время его освобождения. Освобождение файла происходит при сбрасывании последней открытой ссылки на него и обслуживается методом *release*(), являющимся членом структуры *file_operations*. Метод автоматически вызывается виртуальной файловой системой *VFS*.

Реализация метода *release*() в ФС *eCryptFS* представлена функцией *ecryptfs_release*(). Эта функция будет модифицирована для проверки значения нового атрибута. Если значение указывает на то, что файл должен быть зашифрован, а файл не является таковым, то:

- для каждой страницы данных файла будут вызваны внутренние функции *eCryptFS*, отвечающие за шифрование данных;
- при шифровании страницы будут помечены как «грязные», т.е., требующие записи на диск;
- в случае успешного шифрования, будут вызваны внутренние функции *eCryptFS*, формирующие криптографический заголовок зашифрованного файла.

Отдельного рассмотрения заслуживают следующие два вопроса, вызванные особенностями совместного использования системы размежевания доступа *RSBAC* и (любой) стековой ФС. Система *RSBAC* хранит атрибуты безопасности объектов отдельно от них самих, во внутреннем хранилище информации *ACI*. Как известно, идентификация объектов файловой системы в ядре производится с помощью пары вида (идентификатор устройства, номер индексного дескриптора). Эта комбинация уникальна в пределах всей виртуальной ФС *VFS* и используется системой *RSBAC* в качестве ключа для сопоставления файловым объектам их атрибутов безопасности. Проведенный анализ исходных кодов ФС *eCryptFS* позволяет утверждать, что создаваемый *eCryptFS* индексный дескриптор для некоторого файлового объекта имеет тот же идентификатор устройства и тот же номер индексного дескриптора, что и индексный дескриптор, создаваемый ФС нижнего уровня для этого файлового объекта. Таким образом, все атрибуты безопасности файловых объектов, установленные до монтирования ФС *eCryptFS*, будут видны и после монтирования *eCryptFS*. Верно и обратное утверждение: любое назначение атрибутов безопасности файловых объектов не будет утеряно при размонтировании *eCryptFS*.

Далее, использование технологии стековых ФС приводит к тому, что для осуществления некоторой операции с файловым объектом методы *VFS* будут вызваны дважды. Поскольку расширение *RSBAC* дополняет методы *VFS* вызовами процедур для проверки допустимости запросов, это приводит к двойной обработке одного запроса. Общая схема уже приводилась на рис. 1, однако не будет лишним проиллюстрировать ее еще одним примером.

Рассмотрим операцию записи данных в файл, находящийся на смонтированной ФС *eCryptFS*.

Запись данных осуществляется системным вызовом *write*(). Системный вызов передает управление функции *vfs_write*(), дополненной проверкой допустимости операции. В случае успешной проверки функция *vfs_write*() вызывает метод *write*() файловой системы. Для *eCryptFS* этот метод реализован функцией *ecryptfs_write*(). Последняя функция, при необходимости, осуществляет шифрование страницы с данными и вызывает функцию *vfs_write*(), но уже для нижнего файла. В результате будет выполнена еще одна (избыточная) проверка допустимости вызова. Аналогично избыточной проверке будут подвергнуты вызовы *read*() и другие.

Если для некоторых системных вызовов такая ситуация может считаться допустимой в виду их не столь частого использования (например, удаление, переименование файлов), то для вызовов *read*() и *write*() это неминуемо приведет к определенному падению производительности КС.

Избежать падения производительности можно следующим образом. Заменим вызовы функций *vfs_read*() и *vfs_write*(), осуществляемые из функций *ecryptfs_read*() и *ecryptfs*_write(), новыми функциями *bypass_rsbac_read*() и *bypass_rsbac_write*() соответственно. Новые функции являются клонами указанных функций виртуальной ФС, не включающими однако проверки допустимости вызовов.

Перейдем теперь к модификации системы размежевания доступа *RSBAC*. Выше были указаны случаи, при которых может происходить изменение уровня доступа объекта. КСЗ, распознав одно из этих событий, должен инициировать процесс шифрования файла (разумеется, при условии, что файл представлен в незашифрованном виде). Также для третьего случая следует предусмотреть осуществление операции расшифрования зашифрованного файла при понижении уровня доступа.

Распознавание случаев будет проводиться во внутренних функциях модуля *MAC*, отвечающих за проверку доступов типа *write* (функция *auto_write_attr*()) и *read–write* (функция *auto_read_write_attr*()). Логика работы этих функций будет дополнена проверкой следующего вида:

- **если** в результате выполнения операции уровень доступа объекта будет повышен до уровня порогового значения или будет превышать его, **то**;
- **если** файл находится на файловой системе *eCryptFS*, **то** установить у файла новый атрибут;
- **иначе** отказать в доступе;
- **конец если**.

Кроме того, изменения необходимо внести в функцию – диспетчер модуля *MAC*. Эта функция осуществляет сопоставление запросов и типов доступа (*read*, *write*, *read–write*), вызывая соответствующие функции для их проверки. Одним из запросов является запрос *CREATE*, инициируемый в том числе перед созданием нового файла. Обработка запроса *CREATE* будет дополнена новой проверкой, гарантирующей то, что новый файл с уровнем доступа, равным или превышающим пороговый уровень, будет создан, только если он находится на ФС *eCryptFS*. У файла будет установлен новый атрибут, который вызовет его шифрование при освобождении. В противном случае, в доступе на создание нового файла будет отказано.

Управление уровнем доступа объектов со стороны администратора безопасности осуществляется с помощью специальных утилит системы *RSBAC*: *attr_set_fd*, *attr_set_file_dir*, а также меню пользователя *rsbac_fd_menu*. Утилиты формируют запрос на изменение значения указанного атрибута некоторого объекта, который обслуживается системным вызовом *sys_rsbac*(). Модификация этого системного вызова позволит проверять выполнение условия о необходимости осуществления криптографических операций с файлом (шифрование или расшифрование). В случае если файл подлежит шифрованию, но не находится на ФС *eCryptFS*, в повышении уровня доступа будет отказано.

Как часть политики по ведению журнала регистрации событий, шифрование и расшифрование файлов будет сопровождаться соответствующими записями. В случае отказа в доступе при невозможности зашифровать файл в журнал будет выдано сообщение о причине отказа.

Выше уже отмечалось, что от реализации подсистемы криптографической защиты мы тре-

буем исключения утечек расшифрованных данных в виде временных файлов или страниц памяти, перенесенных в область подкачки. Отметим следующее: в своем функционировании ФС *eCryptFS* не создает никаких временных файлов. Зашифрованные страницы считываются в память ядра и расшифровываются в памяти ядра.

Частью функциональности, которую обеспечивает услуга «КО-1», входящая в профиль ЗОС, является шифрование области подкачки. Шифрование осуществляется с помощью одноразового сеансового ключа, генерируемого автоматически при загрузке КС. Таким образом, гарантируется, что в случае перемещения страниц памяти в область подкачки, данные будут сохранены на диске в зашифрованном виде.

Перспективы дальнейшей работы

Одно из перспективных направлений развития подсистемы криптографической защиты – обеспечение шифрования имен файлов. Разработчики ФС *eCryptFS* отмечают, что препятствием для реализации такой возможности является потенциальная коллизия имен (проблема пространства имен).

Рассмотрим проблему на следующем примере. Пусть пользователь Алиса создает в некоторой директории файл с именем α, и пусть имя этого файла зашифровано с помощью ключа k_1, известного лишь Алисе и Бобу. Файл будет записан на диск под зашифрованным именем $a_1 = e(\alpha, k_1)$. Далее, пусть Кэрол создает в той же директории файл с именем α, зашифрованным с помощью ключа k_2, известного лишь Кэрол и Бобу. Файл записан на диск под именем $a_2 = e(\alpha, k_2)$.. Поскольку Кэрол не обладает ключом k_1, ФС не в состоянии определить коллизию имен. Теперь при доступе Боба в директорию расшифрование имен приведет к появлению в одной директории разных файлов с одинаковыми именами $\alpha = d(a_1, k_1)$. и $\alpha = d(a_2, k_2)$, что является недопустимым.

В качестве решения проблемы пространства имен разработчиками ФС предлагается использование отдельного ключа для шифрования имен файлов. Этот ключ – общий для шифрования имен всех файлов в данной точке монтирования.

Заключение. Предложенный проект реализации подсистемы криптографической защиты данных имеет ряд несомненных преимуществ. Если сравнивать его с решениями, осуществляющими избирательное шифрование данных, инициируемое конечными пользователями, то здесь гарантируется полное соответствие метки мандатного контроля и способа обработки критической информации. Подчеркнем, что доля случайных утечек сравнима с долей намеренных утечек и составляет 42%. Исключая возможность ошибки персонала при осуществлении криптографических операций, мы обеспечиваем существенно более надежную защиту критической информации.

В сравнении с решениями, осуществляющими полное шифрование данных, предлагаемая реализация оказывает существенно меньшее влияние на производительность КС, избегая избыточного шифрования.

Подготовка этой статьи сопровождалась реализацией проекта в рамках работы по созданию ЗОС. В техническом смысле реализация не вызвала особых сложностей. Все необходимые изменения были хорошо определены и локализированы. Наибольшей переработке подвергся код ФС *eCryptFS*. Однако благодаря тому, что эта ФС использует стековую технологию, изменения не затрагивают базовых операций с файловыми объектами и поэтому могут быть сравнительно легко протестированы.

1. *ТЗІ* 2.5-004-99. Критерії оцінки захищеності інформації в комп'ютерних системах від несанкціонованого доступу. – К.: ДСТСЗІ СБ України, 1999. – 59 с.
2. *Зегжда Д.П.*, *Ивашко А.М.* Основы безопасности информационных систем. – М.: Горячая линия – Телеком, 2000. – 452 с.
3. *http*://www.infowatch.ru/threats_and_risks/threats_model/
4. *LaPadula L.J.* Essay 9: Rule-Set Modeling of a Trusted Computer System // Information Security: An Integrated Collection of Essays / Ed. by M.D. Abrams, S. Jajodia, H.J. Podell. – Los Alamitos, Cal., USA: IEEE Computer Society Press. – http://www.acsac.org/secshelf/book001/09.pdf
5. *Ott A.* Regelsatz-basierte Zugriffskontrolle nach dem «Generalized Framework for Access Control» Ansatz am Beispiel Linux. – Diplomarbeit. Universität Hamburg, 10. Nov. 1997. – http://www.rsbac.org/doc/media/dipl-ps.zip
6. *Іванников Є.Ю.* Адміністративне керування в ОС Linux на основі системи RSBAC. Матеріали всеукр. наук.-техн. конф. студентів, аспірантів, молодих вчених з міжнар. участю «Інформаційно-керуючі системи і комплекси» – ІКСК'2008, Миколаїв, НУК. – 2008. – С. 16–21.
7. *Іванников Є.Ю.* Довірче керування в ОС GNU/Linux на основі системи розмежування доступу RSBAC // Матеріали 4-ї міжнар. наук.-техн. конф. «Комп'ютерні науки та інформаційні технології» – CSIT'2009. – Львів: Вежа і Ко, 2009. – С. 286–290.
8. *Zadok L.*, *Badulescu L.*, *Shender A.* Cryptfs: A stackable vnode level encryption file system. Technical Report CUCS-021-98, Comp. Sci. Dep., Columbia University, 1998. – http://filesystems.org/docs/cryptfs/ index.html
9. *Halcrow M.* eCryptfs: An Enterprise-class Encrypted Filesystem for Linux // Proc. of the Ottawa Linux Symp. 2005. – **1**. – Ottawa, 2005. – P. 201–226. – http://www.linuxsymposium. org/2005/ linuxsymposium_ procv1.pdf

Поступила 18.03.2010
Тел. для справок: (044) 259-0427 (Киев)
E-mail: *ava@unicyb.kiev.ua*, *ivannikoff@meta.ua*