Воронин Д.А. 012гр.

Архитектура Прозрачной Криптографической Файловой Системы (Transparent Cryptographic Filesystem abbr. TCFS)

TCFS[1] – это прозрачная(т.е. тот факт, что файловая система не является локальной, должен быть скрыт от приложений и пользователей) криптографическая файловая система (напоминает NFS). Она предоставляет пользователям защищенный доступ к файлам, расположенным на удаленном сервере. Она борется с получением несанкционированного доступа и подделыванием данных как на стороне сервера, так и при передаче через сеть, при помощи кодирования и обзора сообщений(message digest). Приложения получают данные от TCFS файловой системы при помощи системных вызовов операционной системы, что дает пользователям полную прозрачность.

TCFS для операционных систем Linux, NetBSD и OpenBSD доступна на сайте http://www.tcfs.it

Введение

В наши дни организация сети делает возможным разделять ресурсы. Файловые системы были исторически одними из первых сервисов, которые были распределены в сети(например файловая система NFS фирмы Sun). Сейчас такие сервисы даже более необходимы. Фактически, широкое распространение мобильного оборудования(наладонники и лэптопы) требует доступ к публичным файловым репозитариям из любой точки мира с различной организацией сети. Распределенные приложения и сервисы посредством сети предоставляют очевидные преимущества, но также создают некоторые проблемы, связанные с безопасностью: не авторизованные пользователи могут получить доступ к серверам с критическими данными.

В распределенных файловых системах имеется два типа участников: серверы, которые имеют прямой доступ к локальной файловой системе, и клиенты, которые хотят получить доступ к файлам, содержащимся в этих файловых системах. Серверы и клиенты взаимодействуют через сеть. Давайте в качестве примера рассмотрим работу файловой системы NFS. Грубо говоря,сервер получает запрос на блок данных от клиента и посылает этот блок данных через сеть. Подслушать этот разговор сервера с клиентом, и, тем самым, прочитать пересылаемые данные, является простой задачей. Более того, доступ к данным со стороны сервера строится на принципе уникальных идентификационных номеров(uid) клиентов. Поэтому, ничто не может предотвратить получения “правильной информации” нежелательным клиентом, и, как следствие, получения доступа ко всей файловой системе.

Пользователи, которые хотят защитить свои файлы, должны принимать меры по предотвращению раскрытия критических данных. Эта проблема может быть адресована к различным уровням: уровню пользователя, уровню приложения или к уровню операционной системы. TCFS решает эту проблему на системном уровне.

Криптографические файловые системы

Криптографическая файловая система Matt'a Blaze'a (CFS)[2], возможно, самая широко используемая, и она наиболее близка по архитектуре к TCFS. CFS шифрует данные перед передачей их недоверяемым компонентам, и расшифровывает их перед передачей доверяемым компонентам. Пользователи CFS создают директории при помощи ключей, и

любой файл, созданный в защищенной директории, автоматически шифруется.

Криптографическая файловая система CryptFS[3], реализованная как виртуальный inode уровень, может быть использована как на локальной так и на удаленной файловой системе. Она предоставляет механизм шифрования Blowfish.

Self-certifying File System(SFS)[4], решая вопрос управления ключами в криптографических файловых системах, предлагает разделение управления ключами от безопасности файловой системы. Серверы имеют открытый ключ, и клиенты используют этот открытый ключ для аутентификации сервера и установления защищенного частного канала связи. Для того, чтобы клиетны могли аутентифицировать сервер, SFS представляет механизм “самоудостоверения имени пути”(self-certifying pathname). Данный механизм содержит хеш открытого ключа сервера, поэтому клиенты могут удостовериться, что действительно говорят с легитимным сервером.

Архитектура TCFS

TCFS имеет очень простую архитектуру. Данные хранятся в закодированном виде в файловой системе сервера. Каждый раз, когда работающее на стороне клиента приложение нуждается прочесть данные, ядро операционной системы клиента делает запрос соответствующего блока данных на сервере. Сервер шлет закодированный блок данных клиенту. Клиент декодирует данные перед передачей их приложению. Операция записи данных происходит таким же методом: приложение передает данные клиенту, который кодирует их и передает серверу; сервер после получения данных от клиента через сеть сохраняет из в файловой системе.

Эта архитектура имеет несколько преимуществ:

1. Минимальная модель доверия(trust model). Архитектура TCFS не подразумевает, что сервер и/или сеть находятся в так называемой “зоне доверия”. Фактически, сервер видит только закодированные данные, и данные передаются через сеть также в закодированном виде. Как мы увидим далее, клиент может определить любое несанкционированное изменение данных.
2. Минимальные требования к администрированию. TCFS не требует никаких дополнительных вмешательств со стороны системного администратора сервера. Все операции поддержки файловой системы не подразумевают никаких знаний от TCFS. Системный администратор может игнорировать тот факт, что его локальная файловая система является в действительности TCFS.
3. Слабое влияние на приложения клиента. TCFS была спроектирована таким образом, чтобы уменьшить влияние на приложения. Приложения клиента получают доступ к файлам из файловой системы TCFS при помощи обычных системных вызовов, таким образом они не нуждаются в изменениях и перекомпиляции для работы с TCFS. Приложения клиента не имеют дела с управлением ключами.
4. Низкие требования со стороны пользователя. Кроме вопроса, связанного с управлением ключами, TCFS имеет низкие требования со стороны конечного пользователя. Для пользователей и приложений TCFS предоставляет такой же уровень прозрачности, как и NFS. Более того, она предоставляет пользователям контроль за политикой кодирования/декодирования, т.е. возможность управлять тем, какие файлы каким методом кодировать, а какие не кодировать вовсе.

Аутентификация серверов

TCFS подразумевает очень маленькую модель доверия: пользователю нужно только доверять клиентской машине, использующейся для доступа к файловой системе TCFS.

Соответственно, пользователю нет нужды доверять серверу, на котором физически расположена файловая система. Сервер имеет доступ только к закодированным данным, которые не пригодны для использования. Очевидно, что сервер может модифицировать данные, и пользователь никакими средствами не может это предотвратить. Тем не менее, т.к. TCFS имеет механизм аутентификации данных, если сервер изменил данные, пользователь немедленно узнает, что данные были изменены. Клиенту нет никакой необходимости аутентифицировать сервер. Допустим, что пиратский хост имитирует легитимный TCFS сервер. Подчеркиваем, что даже в этом случае серкретность пользователя не будет нарушена. Если клиент запишет данные, то пиратский хост получит только зашифрованные данные. С другой стороны, если клиент попытается произвести операцию чтения, данные, которые он получит от сервера, не будут аутентифицированы, и, поэтому, немедленно отвергнутся клиентом.

Управление ключами

Рассмотрим вопрос об управлении ключами отдельно от самой криптографической файловой системой. В двух реализациях TCFS для Linux и BSD-like ядер, TCFS предоставляет простой интерфейс для передачи ключа ядру(при помощи системного вызова ad-hoc ioctl, или при помощи модифицированной операции монтирования файловой системы). TCFS имеет три схемы управления ключами, которые называются Raw, Basic и Kerberized, которые мы вкратце рассмотрим далее. TCFS может предоставить управление ключами на разных уровнях: на уровне процесса, когда каждый процесс имеет свой собственный ключ для доступа к файловой системе TCFS; на уровне пользователя, когда каждый пользователь имеет свой собственный ключ, и все процессы с одним и тем же идентификатором пользователя(uid) используют один и тот же ключ.

Схема управления ключам и "*Raw*" (RKM)

Используя RKM API(Raw Key Management Application Interface – интерфейс приложения в схеме управления ключами Raw), приложение может передать ключ ядру операционной системы. Впоследствии, ядро будет использовать предоставленный ключ для шифрования и дешифрования. Никакой проверки ключа ядро не осуществляет, т.е. приложение должно быть уверено, что передало ядру правильный ключ. Схема RKM не используется конечным пользователем, но служит основой для построения более сложных схем управления ключами.

Схема управления ключами "*Basic*"(BKM)

Эта схема позволяет пользователям сгенерировать ключи и хранить их в базе данных в закодированной форме, используя логин и пароль в качестве аутентификации. Таким образом, пользователям TCFS нужно помнить не главный ключ(master), и только их логин и пароль. Для использования схемы BKM необходимо наличие системного администратора, который осуществляет процесс регистрации(обычно сведения о зарегистрированных пользователях хранятся в файле /etc/tcfspwdb). Использование схемы BKM предполагает следующие этапы:

1. Системный администратор регистрирует пользователя в базе данных ключей(при помощи команды tcfsadduser).
2. Пользователь создает главный ключ(используя команду tcfsgenkey) Команда tcfsgenkey генерирует случайный ключ, шифрует его при помощи пароля пользователя и сохраняет базе данных.
3. Когда пользователю понадобится доступ к его зашифрованным файлам, он должен

извлечь свой главный ключ из базы данных(предоставив свой пароль), и передать его на уровень TCFS. Для этой операции требуется команда tcfsputkey.

4. Пользователь заканчивает свою сессию при помощи команды tsfcsrmkey, которая удаляет ключ из ядра.

Схема управления ключам и "*Kerberized*"(KKM)

Kerberos – это распределенный сервис аутентификации, разработанный в M.I.T [5]. Kerberos аутентификация - это доверенная аутентификация третьей стороны, которая предоставляет аутентификацию всем участникам в распределенной среде. Kerberos делает возможным для клиента и сервера аутентифицировать друг друга и установить частный (private) канал связи. Схема KKM выступает полной альтернативой к схеме BKM. Введем новое понятие: серверный ключ TCFS(TCFSKS), который содержится в базе данных главных ключей. Клиент аутентифицируется в системе Kerberos самостоятельно и получает сессионный ключ и мандат(ticket), отсылает TCFSKS запрос(например: дай ключ пользователя или сохрани новый ключ) через сеть. Административные опции, такие как добавление/удаление пользователей и групп, могут быть реализованы таким же образом. Взаимодействие между клиентом и TCFSKS содержит следующие шаги:

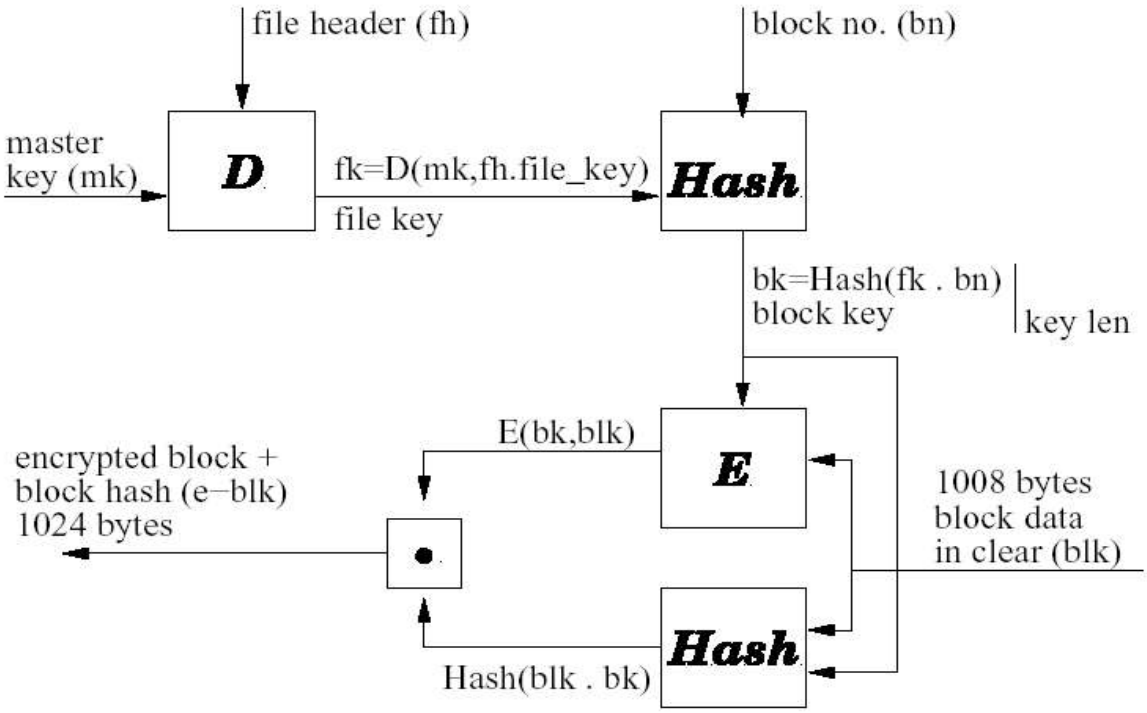
1. В конце Kerberos аутентификации, клиент получает сессионный ключ.
2. Клиент посылает запрос и свой мандат TCFSKS.
3. Сервер расшифровывает сообщение и отсылает обратно ответ и мандат.
4. Клиент получает ответ и сбрасывает мандат.

Механизм шифрования

TCFS не предоставляет фиксированной схемы шифрования, для каждого файла может быть определена своя схема шифрования. Механизм шифрования должен удовлетворять некоторому определенному интерфейсу и, в Linux-реализации, должен быть загружаемым модулем ядра(loaded kernel module). Модули для всех основных механизмов предоставляются вместе с реализацией TCFS. Модульное шифрование позволяет пользователю реализовать и использовать свой собственный модуль шифрования, что позволяет увеличить безопасность.

Далее будем обозначать символами E(. , .) и D(. , .) алгоритмы шифрования и расшифрования, связанные с используемой схемой шифрования(см. рис.). Размер зашифрованного блока может быть не равен размеру входного блока. Каждый файл имеет заголовок(header), который содержит некоторую информацию о самом файле(версию TCFS, идентификатор шифра).

Каждый пользователь A имеет главный ключ K_A . Для каждого файла f случайным образом выбирается файловый ключ K_f. Файловый ключ шифруется при помощи главного ключа пользователя и помещается в поле файлового ключа в заголовке. Каждый блок TCFS файла состоит из двух частей: из данных и из аутентификационного тэга. Каждый блок зашифрованного TCFS файла шифруется алгоритмом E в режиме сцепления блоков CBC (Cipher Block Chaining mode) при помощи блокового ключа. Блок незашифрованного TCFS файла сохраняется в первоначальном виде. Блоковый ключ вычисляется как хеш-функция от числа, получаемого конкатенацией файлового ключа и номера блока. Аутентификационный тэг аутентифицируемого TCFS файла вычисляется как хеш-функция от числа, получаемого конкатенацией блока данных и блокового ключа. Неаутентифицируемый TCFS файл имеет аутентификационный тэг, состоящий из нуля байт.



Такой метод шифрования и хеширования блоков демонстрирует следующие характеристики безопасности:

1. Если используется “сильная” схема шифрования, то зашифрованные файлы нельзя прочесть без знания файлового ключа или главного ключа пользователя.
2. Т.к. каждый файл шифруется при помощи своего уникального файлового ключа, то невозможно определить когда два зашифрованных файла соответствуют одному и тому же тексту.
3. Более того, т.к. каждый блок шифруется при помощи уникального блокового ключа, невозможно определить когда два блока одного файла соответствуют одному и тому же тексту.
4. Аутентификационный тэг предотвращает изменение данных на сервере. Очевидно, что т.к. сервер имеет физический доступ к файловой системе, ничто не может помешать серверу модифицировать данные. Функция аутентификационного тэга состоит в том, чтобы пользователь мог проверить, были ли несанкционированные изменения данных. Данный аутентификационный механизм гарантирует следующее:
   (a) Изменение блока без вычисления заново аутентификационного тэга легко устанавливается TCFS клиентом. Вычисление заново аутентификационного тэга требует знания блокового ключа, который зависит от файлового ключа, который, в свою очередь, зашифрован при помощи главного ключа пользователя.
   (b) Т.к. каждый блок аутентификационного тэга зависит от смещения блока, без знания блокового ключа невозможно вставить дополнительный блок данных или поменять местами два блока в одном файле. Более того, т.к. аутентификационный тэг зависит также и от файлового ключа, невозможно импортировать блок из другого файла.

Производительность.

Для оценки производительности был использован стандарт(benchmark) Andrew [6]. Этот стандарт оценки производительности состоит из пяти частей:

1. Создание директорий(directory creation).
2. Копирование файлов(file copy).
3. Рекурсивное чтение директорий(recursive directory stats).
4. Рекурсивное чтение файлов(recursive files scan).
5. Компиляция(compilation).

Мы представим 4 варианта теста. Первый вариант – показатели производительности NFS. Второй вариант – показатели TCFS на файлах, которые не шифровались. Последние 2 – результаты использования NULL модуля шифрования и модуля 3DES. Характеристики тестовой машины: Pentium II 233MHz, 64Mb mem. Будут представлены 2 серии измерений. В первой серии файловая система источника находилась на локальной машине, а TCFS располагалась на удаленной машине. В второй серии наоборот.

Экспорт данных в удаленную файловую систем

Phase	NFS	TCFS NONE	TCFS NULL	TCFS 3DES
Creating directories	0.109	0.178	0.648	0.698
Copying files	1.385	2.777	9.047	15.924
Recursive directory stats	2.215	4.798	5.558	6.537
Recursive file scan	3.074	7.489	10.047	16.129
Compilation	36.802	57.791	1m3.874	1m27.929

Импорт данных из удаленной файловой системы

Phase	NFS	TCFS NONE	TCFS NULL	TCFS 3DES
Creating directories	0.052	0.065	0.081	0.093
Copying files	0.282	1.545	2.548	5.462
Recursive directory stats	1.355	1.449	2.273	2.388
Recursive file scan	2.261	2.464	4.038	4.267
Compilation	34.634	36.653	35.448	48.364

Ссылки:

[1] G. Cattaneo, L. Catuogno, A. Del Sorbo, P. Persiano, "*The Design and Implementation of a Transparency Cryptographic Filesystem for UNIX*", Technical report, Dip. Informatica e Appl., Universita di Salerno, July 1997.
http://www.tcfs.it/docs/freenix01.pdf

[2] M.Blaze, "*A Cryptographic File System for UNIX*", First ACM Conference on Communication and Computing Security, pp. 158-165, Fairfax VA 1993. http://www.crypto.com/papers/cfs.pdf

[3] E. Zadok, I. Badulescu, A. Shender, "*Cryptfs: A stackable vnode Level encryprion file system*", 1998.
http://www.cs.columbia.edu/~ezk/research/cryptfs/cryptfs.pdf

[4] D. Mazieres. N. Kaminsky, M.F. Kaashoek, E. Witchel "*Separating key management from file system security*", Proceedings of 17th ACM Symposium on Operating System Principle (SOPS' 99), Kiawah, Island, South Carolina, December 1999. http://www.pdos.lcs.mit.edu/papers/sfs:sosp99.pdf

[5] J.G. Steiner, C.Neuman, J.I. Schiller, "*Kerberos, an authentication service for Open Network Systems*", USENIX Association Conferences Proceedings, pp.191-202, February 1988.
http://www.cybersafe.ltd.uk/docs_other/Kerberos%20%96%20An%20Authentication%20Service%20for%20Open%20Network%20Systems%20(Project%20Athena).pdf