

Kryptographische Dateisysteme

Unix CFS und CryptFS und Windows2000 NTFS-Verschlüsselung

Reinhard Rösch

Reinhard.Roesch@stud.informatik.uni-erlangen.de

17.06.2002

Abstract

Wichtige Daten sollen nicht nur während der Übertragung vor unbefugtem Zugriff gesichert werden, sondern auch wenn sie auf Datenträgern gespeichert sind. Um über die vom Betriebssystem bereitgestellten Mechanismen hinaus Sicherheit zu gewährleisten, werden Dateisysteme eingesetzt, die alle Daten verschlüsselt auf Medien speichern. Vorgestellt werden im folgenden das Crypto File System (CFS), CryptFS, sowie das Encrypted File System (EFS), Teil von NTFS.

1 Einführung

Moderne Betriebssysteme bieten standardmäßig Zugriffsschutz für vertrauliche Dateien, allerdings kann der Benutzer eines Computers sich der Vertraulichkeit seiner Daten nur solange sicher sein, wie alle an der Bearbeitung beteiligten Komponenten unter seiner Kontrolle stehen. Insbesondere wenn ein Angreifer physischen Zugang zum Datenmedium erreicht, kann er die Daten auf einem seiner Rechner auslesen, und seine Administratorrechte auf diesem Rechner erlauben ihm ohne weiteren Aufwand volle Zugriffsrechte auf alle Daten. Aber auch wenn die Daten zur Speicherung den Rechner verlassen, etwa bei der Nutzung von Servern im Netzwerk oder bei Backups auf entfernte Systeme kann der Benutzer oft nur mit Mühe feststellen, ob seine Daten Unbefugten nicht zugänglich sind.

Verschlüsselung ist eine mögliche Lösung dieses Problems, da alle erreichbaren Daten nur bei Kenntnis des Schlüssels die darin enthaltene Information offenbart. Allerdings ist Benutzung von Dateiverschlüsselung, wie sie zum Beispiel PGP/GPG zur Verfügung stellt sehr aufwendig und fehlerträchtig, da häufiges manuelles Eingreifen erforderlich ist, oder das Verschlüsseln oder Löschen des Klartexts schlicht vergessen wird.

Die kryptographischen Vorgänge können automatisiert werden, indem eine Schicht des Dateisystems sie übernimmt. Dann ist sichergestellt dass alle so geschriebenen Daten abgesichert sind. Die erste solche Erweiterung eines Dateisystems war das 1993 von Matt Blaze (AT&T Bell Laboratories) vorgestellte Cryptographic File System (CFS)[1]. Andere Verfahren zur Dateiverschlüsselung gehen mit ähnlichen Zielsetzungen und Methoden vor, weshalb hier hauptsächlich auf CFS eingegangen wird, und ansonsten nur Unterschiede herausgestellt werden.

2 Das Cryptographic File System

2.1 Zielsetzung

Ziele der Entwicklung von CFS war es angreifbare Teile des Dateisystems abzusichern, ohne dabei unangenehm in den normalen Arbeitsablauf einzugreifen. Die Eingabe der kryptographischen Schlüssel soll deshalb nur einmal pro Benutzer erfolgen, ebenso soll der Schlüssel nach der Benutzung wieder einfach entfernt werden können.

Der Zugriff auf die Datei soll transparent für den Benutzer erfolgen, das heißt, alle Applikationen sollen ohne Änderung weiterhin funktionieren wie gewohnt und auch ohne grosse Einbussen an Performance, auch die Semantik von nebenläufigen Dateizugriffen soll erhalten bleiben. Die Forderung der Transparenz erstreckt sich dabei auch auf die verschlüsselten Dateien, sie sollen sich nicht prinzipiell anders verhalten als unverschlüsselte, so soll zum Beispiel Anlegen von Sicherungskopien möglich sein. Ohne Kenntnis des Schlüssels sollen aber weder Inhalte, noch Informationen wie Dateiname lesbar sein.

Mehrere Dateien sollen mit demselben Schlüssel bearbeitet werden können, als kleinste Einheit der Schlüsselverwaltung wurde daher das Verzeichnis gewählt. Auch sollte es genügen, wenn Nutzer nur ihrem eigenen Rechner vertrauen, und nicht der Sicherheit entfernter Server oder Netzverbindungen.

2.2 Benutzerinterface

Wie bereits erwähnt ist die kleinste Einheit, der ein Schlüssel zugewiesen wird ein Verzeichnis. Entsprechend beziehen sich alle Tools auf Verzeichnisse.

Der `cmkdir` Befehl erstellt ein neues Verzeichnis in dem später alle Dateien in verschlüsselter Form gespeichert werden, und fragt nach dem Passwort, das beim Zugriff auf dieses Verzeichnis benutzt wird, und dessen Hashwert zur Verifikation in einer versteckten Datei dort gespeichert wird.

Um kryptographisch abgesicherte Dateien im Klartext zu erreichen benutzt man `cattach`. `cattach` wird der Name des verschlüsseltem Verzeichnisses und der ihm zuzuweisende Klartextname als Parameter mitgeteilt, und fordert zur Eingabe eines Schlüssels auf, der mit dem bei der Erstellung mit `cmkdir` angegebenen übereinstimmen muss. Danach ist der Inhalt aller Dateien des verschlüsselten Verzeichnisses unter (üblicherweise) `/crypt/<Klartextname>` ohne weiteren Aufwand zugreifbar.

Daneben gibt es noch die Tools `cname` und `ccat` die Zugriff auf Daten und Dateinamen gestatten, ohne dass CFS installiert sein muss, und `cdetach` zum Entfernen eines Schlüssels aus dem System. CFS unterstützt auch die Nutzung von Smart Cards zur Speicherung der Passphrasen, wofür weitere Tools zur Verfügung stehen.

2.3 Implementierung

Das Original CFS benutzt DES im *electronic code book* (ECB) Format, in dem jeder 8 Byte Block unabhängig von vorhergehenden Blocks verschlüsselt wird, und für alle Blocks wird derselbe Schlüssel verwendet, der aus der angegebenen Passphrase erzeugt wird. Dies erlaubt schnellen Zugriff mit identischer Zugriffszeit unabhängig von der Position des Blocks, im Gegensatz zur Verwendung feedbackbasierten Methoden da hier beim Lesen eines Blocks alle vorher liegenden Blöcke ebenfalls entschlüsselt werden müssten, und beim Schreiben würden sich alle Daten nach einem geänderten Block ebenfalls ändern, die Berechnungen im Fall grosser Dateien wären also zu aufwendig.

Allerdings wären bei alleiniger Benutzung dieses Modus kryptographische Analysen möglich,

die ausnutzen dass - eventuell sehr grosse - Datenmengen mit immer wieder demselben Schlüssel verschlüsselt werden, und möglicherweise sogar known-plaintext Angriffe auf bekannte Header von vermutlich benutzten Dateiformaten. Aus diesem Grund wird bei der Ausführung von `cattach` mit Hilfe des DES *output feedback* (OFB) Modus aus einem anderen Teil des Passworts ein grosses (halbes Megabyte) Bitfeld erzeugt. Beim Schreiben werden die Daten mit der korrespondierenden Position des Bitfeldes XOR-verknüpft, und das Ergebnis mit dem ECB Verfahren verschlüsselt, beim Lesen entsprechend in umgekehrter Reihenfolge.

Der eigentliche CFS-Dienst, `cfstd`, ist als Network File System (NFS) Server realisiert, mit um Austausch von Schlüsselinformationen erweiterten remote procedure calls, die von `cattach`, `cdetach` und `cmkdir` benutzt werden. Ein `cattach` führt nach einer Verifikation des Schlüssels dazu, dass die genannten Klartextverzeichnisse mit dem DES Schlüssel, der berechneten Bitmaske und dem verschlüsselten Verzeichnis assoziiert werden und die genannten Directoryeinträge in `/crypt` erstellt werden.

Alle nachfolgenden System Calls auf dieses Verzeichnisse und seine Inhalte werden kompatibel zu einem normalen NFS Server abgewickelt, so dass alle Unixversionen, die NFS benutzen können ohne Änderung der Kernelstrukturen auch mit CFS umgehen können. `cfstd` läuft dabei, wie NFS Server auch, als User Level Prozess, es lässt sich aus Sicherheitsgründen aber nur von privilegierten Ports von *localhost* aus zugreifen. Bei jedem Zugriff erfolgt jedoch eine Prüfung ob sie von einem Prozess mit der gleichen UID stammt, der das Verzeichnis *attached* hat. `cfstd` greift auf die zugrundeliegenden Dateien mit normalen System Calls zu, nur werden eben alle Daten automatisch vor dem Schreiben verschlüsselt, und nach dem Lesen entschlüsselt.

Durch das Aufbauen auf dem NFS Protokoll wurde sowohl eine schnelle Entwicklungszeit gewährleistet da die Behandlung des Dateizugriffs vom Dateisystem des Hostsystems übernommen wird, als auch flexibler Einsatz auf vielen verschiedenen Zielsystemen ermöglicht, da prinzipiell nur Kompatibilität mit NFS erforderlich ist. Der Weg über mehrere Kontextwechsel, und natürlich die Verschlüsselung selbst, senkt jedoch die Leistungsfähigkeit, insbesondere bei Benutzung kleiner Dateien, aber in der Praxis sind Werte von 70% der Performance ohne Verschlüsselung erreichbar.

2.4 Sicherheitsaspekte

CFS stellt nur die Vertraulichkeit der geschriebenen Daten auf kryptographischem Weg sicher, für alles andere, etwa zur Absicherung gegen Verlust der Daten, oder der Sicherheit der Schlüssel während der Eingabe und solange `cfstd` ihn im Speicher behält wird auf den Zugriffsschutz des Betriebssystems und die Sorgfalt des Benutzers vertraut. Da die Schlüssel das System des Users jedoch nicht verlassen muss nur diesem einem System vertraut werden. CFS kann auch nicht sicherstellen, dass wichtige Daten nicht auf dem Umwege über virtuellen Speicher auf ungeschützte Datenträger gelangen.

Die Stärke der Verschlüsselung im ECB+OFB Modus liegt wohl zwischen der von 3DES und einfacher DES Verschlüsselung im ECB Modus (dem schwächsten der DES Modi), allerdings kann eventuell getrennt nach den einzelnen Teilschlüsseln der beiden Methoden gesucht werden, wenn mehrere ähnliche Dateien (z.B. verschiedene Versionen eines Quellcodes), oder bekannte Texte (z.B. bekannte Header von vermutlich benutzten Dateitypen) in einem Verzeichnis, und damit mit demselben Schlüssel, verschlüsselt wurden. Zusammen mit der begrenzten DES Schlüssellänge von 56bit ist die Originalversion von 93 nicht mehr zeitgemäß.

2.5 Erweiterungen von CFS

Basierend auf dem Konzept von CFS wurden verschiedene Verbesserungen eingebracht. So enthält das Transparent Cryptographic File System (TCFS)[2,3] der Universität Salerno ein mächtigeres Schlüsselverwaltungskonzept, das auf Wunsch auch auf Dateiebene arbeitet, Sharing von Dateien erlaubt und grössere Automatisierung des *attach*-Prozesses, durch ein modifiziertes *login*. Oft sind Loginpasswörter jedoch schwächer, zumindest jedoch kürzer wie die Passwörter von CFS. Neue Versionen verschieben die Verschlüsselung in den (Linux-/OpenBSD-/NetBSD-) Betriebssystemkern, und unterstützen auch mehrere Kryptoalgorithmen (IDEA, Blowfish).

Das an der University of Berkeley entwickelte Extended Cryptographic File System (ECFS)[4] verschlüsselt ebenfalls Dateien und nicht Verzeichnisse und es erlaubt digitale Signatur von Dateien um sicherzustellen, dass Dateien nicht verändert wurden.

3 CryptFS

CryptFS[5], entwickelt an der Columbia University unterscheidet sich von den bisher vorgestellten Verfahren hauptsächlich dadurch, dass es stark auf Performance ausgerichtet ist. Aus diesem Grund wurde es von vornherein für Integration in den Kernel entwickelt, es wurde optimiert für schnelle Operation mit memory mapped Dateien und benutzt wird der Blowfish Algorithmus. CryptFS arbeitet üblicherweise auf Blöcken in der Grösse einer virtuellen Speicherseite der verwendeten Architektur, z.B. 8KB bei UltraSPARC, da dies die Grösse ist, die bei memory mapped Operationen verwendet wird. Die verschlüsselten Daten werden physikalisch auf den Datenträger durch ein loopback Dateisystem geschrieben. Durch diese Maßnahmen wird die Geschwindigkeit auf etwa 80% des unverschlüsselten Dateisystems gebracht, aber vor allem wird dieser Wert konsistent erreicht als bei CFS.

CryptFS wurde entwickelt auf Solaris 2.5.1, aber da bereits früh darauf geachtet wurde von den darunterliegenden Dateizugriffen zu abstrahieren, wurde es erfolgreich portiert auf Linux und FreeBSD. Durch die Benutzung von Kernspeicher ist die Sicherheit noch etwas höher als bei CFS, auch ist es hier dem Betriebssystem möglich, die Daten zumindest vor Überschieben ohne Passwort zu sichern, indem das Loopbacksystem ansonsten schreibgeschützt ist.

4 NTFS Encrypted File System

Mit Windows2000 hat Microsoft das Encrypted File System (EFS)[6] als Teil von NTFS 5.0 herausgebracht. Ähnlich wie die bereits vorgestellten Verschlüsselungen benutzt EFS für die Ver- und Entschlüsselung aus Performancegründen einen symmetrischen Schlüssel, den sogenannten *File Encryption Key* FEK, der bei der ersten Verschlüsselung automatisch erzeugt wird und in einem speziellen NTFS-Datenstream gespeichert. Bei jedem Verschlüsselungsvorgang wird der Inhalt der Datei zunächst mit dem FEK und dem DESX Algorithmus verschlüsselt, danach wird der FEK durch ein asymmetrisches Verfahren mit dem Public Key des Benutzers, der von der in Windows integrierten CryptoAPI automatisch bei der Erstellung des Accounts erzeugt wurde, verschlüsselt. Beim Lesen einer abgesicherten Datei wird umgekehrt zuerst der FEK gelesen und entschlüsselt mit dem Private Key des Nutzers, und danach mit Hilfe des FEK der Inhalt der Datei gelesen.

Der Vorteil dieser Aufteilung liegt darin, dass jede Datei individuell verschlüsselt wird, und damit auch einzeln für jede Datei Benutzern die Möglichkeit des Lesezugriffs gegeben wird. Es muss einfach nur der FEK mit dem Public Key eines neu hinzugekommenen Nutzers

verschlüsselt werden und an die Datei angehängt werden, und schon kann dieser User ebenfalls ungehindert mit dieser Datei arbeiten. Tatsächlich ist in der Grundeinstellung jede verschlüsselte Datei neben dem Eigentümer zumindest für dem sogenannten *Recovery Agent* lesbar, für den Fall dass das Passwort, oder zumindest der Private Key des Eigentümers nicht mehr verfügbar ist. Der Private Key des Nutzers wird automatisch beim Login an der Konsole mit Hilfe des Passworts freigeschaltet, was auch erlaubt das von einem unbeaufsichtigtem Terminal alle Dateien des dort eingelogten Nutzers lesbar sind. Für hoch vertrauliche Daten empfiehlt Microsoft die Speicherung des Private Keys auf Smart Cards.

Literatur

- [1] Matt Blaze, A Cryptographic file system for Unix, Aus Proceedings of the First ACM Conference on Computing and Communications Security, 1993, <http://www.crypto.com/papers/cfs.pdf>
- [2] G. Cattaneo und G. Persiano, Design and Implementation of a Transparent Cryptographic File System for Unix, Università di Salerno, 1996, <http://www.tcfs.it/docs/tcfs.ps>
- [3] Ermelindo Mauriello, TCFS: Transparent Cryptographic File System, Linux Journal Ausgabe 40, 1997, <http://www.tcfs.it/docs/tcfs-lj.pdf>
- [4] David Bindel, Monica Chew und Chris Wells, Extended Cryptographic File System, Berkeley University, 1999, <http://www.cs.berkeley.edu/~cswells/papers/ecfs.ps>
- [5] Erek Zadok, Ion Badulesen und Alex Shender. CryptFS: A stackable vnode level encryption file system. Technical Report CUCS-0121-98, Columbia University, 1998, <http://www.cs.columbia.edu/~ezk/research/cryptfs/cryptfs.pdf>
- [6] Unbekannter Autor, Encrypting File System for Windows 2000, Microsoft Web Site, 1999, <http://www.microsoft.com/windows2000/docs/encrypt.doc>
Zusätzlich verwendete Quellen:
- [7] Bruce Schneier, Applied Cryptography, Second Edition, John Wiley & Sons, 1996
- [8] Aviel Rubin, Hackerabwehr und Datensicherheit, Addison-Wesley, München, 2002
- [9] Matt Aderson, Encrypting File System, The Evolution of NTFS: NTFS 5.0, Ars Technica, 2000, <http://www.arstechnica.com/paedia/n/ntfs/ntfs5-3.html>
- [10] Alexander Yuriev, Cryptographic File System under Linux HOW-TO, LINUX SECURITY FAQ, 1996, <http://www.ibiblio.org/pub/Linux/docs/faqs/security/Cryptographic-File-System>