Seminarvortrag *Secure NFS*

Michael Stilkerich *michael.stilkerich@informatik.stud.uni-erlangen.de*

am 12. Mai 2003

Einleitung

Das Network File System ist ein sehr eleganter Weg, gemeinsam genutzte Dateisysteme auf Servern transparent in das Dateisystem der Clients einzubinden. NFS hat jedoch einen entscheidenden Mangel: die Sicherheit. In dieser Ausarbeitung wird eine sicherere Variante des NFS, das Secure NFS und die zugrundeliegenden Verfahren vorgestellt.

1 Historie

Mitte der 80er entwickelte Sun Microsystems eine Reihe von Netzwerkprotokollen — den Remote Procedure Call (RPC) und darauf basierend das Network Information System[1] (NIS) und das Network File System (NFS) — die ein Netzwerk aus Workstations wie ein alleinstehendes Computersystem agieren ließen. Diese Protokolle waren maßgebliches Kriterium für den Erfolg von Sun als Computer-Hersteller. Sie ermöglichten den Benutzern in einer Organisation die Freiheit und Leistung eines alleinstehenden Systems und die gleichzeitige Nutzung der Vorteile eines zentral verwalteten Systemes und wurden bald zum Standard.

Ein wichtiges Kriterium in Netzwerken hatte Sun bei der Entwicklung jedoch kaum berücksichtigt: die Sicherheit – RPC und NFS hatten praktisch keine Sicherheitsmechanismen. Trotz dieses schweren Mangels ist die Protokollsammlung auch heute noch weit verbreitet.

2 NFS und NIS

NFS erlaubt das einhängen von Dateisystemen auf einem Server in das lokale (virtuelle) Dateisystem, als wären die Partitionen lokal verbunden. Der Benutzer kann mit allen Datei-Operationen des Betriebssystems transparent auf die Dateien dieses Systems zugreifen.

[1] vorher auch bekannt als Yellow Pages (YP)

System	Technik	Bemerkungen
AUTH_NONE	Keine	Keine Authentifizierung, anonymer Zugriff
AUTH_UNIX	Client sendet UID und GIDs des Users	Nicht sicher. Der Server traut den übermittelten Daten.
AUTH_KERB	Authentifizierung auf Kerberos basierend	Sehr sicher, jedoch muss ein Kerberos Server aufgesetzt werden. Nur wenige Implementierungen.
AUTH_DES	Basiert auf DES und öffentlicher Schlüssel Kryptographie	Relativ sicher, aber fast nur von Sun implementiert. Für Secure RPC verwendetes System.

Tabelle 1: Authentifizierungsmechanismen für RPC

NIS ist ein verteiltes Datenbanksystem, das es vielen Computern eines Netzes ermöglicht Passwort- und Gruppendateien, Hosttabellen und andere Dateien gemeinsam zu benutzen. NIS Clients können diese sog. NIS maps so benutzen als wären sie lokal gespeichert.

Sowohl NFS als auch NIS basieren auf RPC.

3 Der Remote Procedure Call (RPC)

RPC ermöglicht es einem Programm auf einem Rechner eine Funktion, die auf einem anderen Rechner abläuft, transparent wie eine lokale Funktion aufzurufen. RPC erlaubt die einfache Erstellung von Client-Server Anwendungen.

3.1 eXternal Data Representation (XDR)

Zum Transport von Daten zwischen Rechnern eventuell verschiedener Architekturen (z.B. mit unterschiedlicher Byte Order) verwendet RPC das Protokoll XDR zur einheitlichen Darstellung von Daten unabhängig von der Architektur des Systems.

3.2 Authentifizierungsmechanismen von RPC

Clients, die einen RPC Server kontaktieren, müssen sich auf irgendeine Weise authentifizieren, damit der Server entscheiden kann, auf welche Daten und Dienste sie Zugriff erhalten sollen. Für RPC existieren die in Tabelle 1 aufgelisteten Authentifizierungsmechanismen.

4 Secure NFS

Ende der 80er entwickelte Sun eine sichere(re) Version des NFS Protokolls, das Secure NFS. Dieses ist nichts anderes als die Kombination des NFS Protokolls mit

Secure RPC.

Secure RPC benutzt den AUTH_DES Mechanismus, welcher auf DES und öffentlicher Schlüssel Kryptographie basiert. Diese beiden Systeme sollen zunächst in einem kleinen Kryptographie Exkurs erläutert werden.

4.1 Data Encryption Standard (DES)

Bei DES wird ein 56bit Schlüssel zum Kodieren von 64bit Blöcken Klartext in 64bit Blöcke von Chiffretext. Problem des Verfahrens ist, dass Sender und Empfänger den gleichen Schlüssel kennen müssen, welcher auf einem wiederum sicheren Weg transportiert werden muss. Dieses Problem löst das Verfahren des exponentiellen Schlüsseltausches.

4.2 exponentieller Schlüsseltausch (nach Diffie-Hellman)

Beide Seiten wählen jeweils einen geheimen Schlüssel und erzeugen einen öffentlichen Schlüssel. Dieser ist letztendlich nichts anderes als ein wohlbekannter Wert in Klartext, der mit dem geheimen Schlüssel verschlüsselt wurde. Dieser öffentliche Schlüssel wird bekanntgegeben[2] und steht nun für Authentifizierungsdienste zur Verfügung. Auch die privaten Schlüssel werden, verschlüsselt mit dem Passwort des Benutzers, in einer NIS Map gespeichert und beim Login vom NIS Server geholt.

Die Chiffrierung erfolgt mittels eines sehr komplizierten Algorithmus, der sich Exponential- und anderer arithmetischer Operatoren mit kommutativen Eigenschaften bedient. Die Schlüssel besitzen eine Länge von 50-100 bit und der Algorithmus ist hinreichend komplex, um zu gewährleisten dass der öffentliche Schlüssel in einem sicherheitsrelevanten Zeitraum nicht dechiffriert werden kann.

Beim exponentiellen Schlüsseltausch[3] erzeugen nun beide Seiten voneinander unabhängig *denselben* Konversationsschlüssel, indem jeweils der eigene geheime Schlüssel zur Verschlüsselung des öffentlichen Schlüssels der Gegenseite benutzt wird. Aufgrund der Kommutativität ist die Reihenfolge der Verschlüsselung egal und es wird jedesmal derselbe Schlüssel erzeugt.

$$P(A) = \alpha^{K(A)}$$

$$P(B) = \alpha^{K(B)}$$

$$C(A,B) = \alpha^{K(A)K(B)} = P(A)^{K(B)} = P(B)^{K(A)}$$

wobei P(A) und P(B) die öffentlichen Schlüssel von A und B sind, K(A) und K(B) entsprechend die privaten Schlüssel und C(A,B) der Konversationsschlüssel.

[2] beispielsweise über NIS in der MAP *publickey.byname*

[3] der Name kommt von den exponentiellen Verfahren des Algorithmus

4.3 Die Funktionsweise von Secure RPC

Die Validierung einer RPC Anfrage findet auf dem Server statt und beruht auf Informationen die in der Anfrage enthalten sind:

- Client und Server einigen sich auf einen Konversationsschlüssel, der mit den folgenden Schritten erzeugt wird:

  1. Der Client wählt zufällig eine grosse Zufallszahl zum Konversationsschlüssel. Dieser bleibt für die Zeit des Logins gültig.
  2. Dieser Schlüssel muss dem Server mitgeteilt werden, dazu wird der exponentielle Schlüsseltausch angewendet.
  3. Der Konversationsschlüssel wird mit einem Sitzungsschlüssel chiffriert, den der Client aus Kombination seines privaten Schlüssels mit dem öffentlichen Schlüssels der Servers erzeugt.
  4. Der chiffrierte Konversationschlüssel wird an den Server gesendet. Dies ist der einzige Zeitpunkt zu dem ein Schlüssel über das Netz wandert.
  5. Der Server erhält den Sitzungsschlüssel durch Kombination seines privaten Schlüssels mit dem öffentlichen Schlüssel des Clients und kann den Konversationsschlüssel dechiffrieren.

- Bei jeder RPC Anfrage verschlüsselt der Client die aktuelle Zeit mit dem Konversationsschlüssel und sendet diese mit.

- Der Server entschlüsselt den Zeitstempel und prüft ihn. Fällt er in ein festgelegtes Zeitfenster wird die Anfrage akzeptiert.

Der Server weiß, dass der Benutzer am Client jener ist für den er sich ausgibt denn

- Das empfangene Paket wurde mit dem Konversationsschlüssel chiffriert.

- Dieser wurde dem Server verschlüsselt mit einem Sitzungsschlüssel mitgeteilt

- Der einzige Weg auf dem der Client den Sitzungsschlüssel wissen konnte, war ihn durch Kombination des öffentlichen Schlüssels des Server mit dem privaten Schlüssel des Benutzers zu erzeugen.

- Um den privaten Schlüssel zu wissen, musste der Client den Schlüssel vom NIS Server holen und entschlüsseln.

- Zum entschlüsseln des privaten Schlüssels musste der User das Passwort kennen.

4.4 Anwendung von Secure NFS in der Praxis

Für die Aktivierung von Secure NFS ist lediglich das Schlüsselwort *secure* in die Optionsliste des jeweiligen Eintrages in der */etc/exports* Datei des Servers einzufügen. Der Eintrag

```
/mnt/ext 10.10.10.0/255.255.255.0(rw,secure)
```

würde beispielsweise das Dateisystem unter /mnt/ext auf dem Server via Secure NFS an das Netz 10.10.10.0/24 exportieren.

Auf den Clients muss beim mounten des Dateisystems die Option *secure* angegeben werden. Wäre obiges Dateisystem auf dem Server *xan* exportiert, wäre z.B. ein gültiges mount-Kommando

```
mount xan:/mnt/ext /mnt/netshares/ext -o secure,rw
```

Natürlich kann dieses Dateisystem auch automatisch während des Bootvorganges durch einen entsprechenden Eintrag in der */etc/fstab*[4] eingehängt werden, dieser könnte für das Beispiel wie folgt aussehen:

```
xan:/mnt/ext /mnt/netshares/ext nfs auto,secure,rw 0 0
```

Um den Betrieb von Secure NFS für den Benutzer transparent zu halten, läuft auf jedem Client der Dämon *keyserv*. Dieser holt sich beim Login eines Benutzers automatisch den privaten Schlüssel des Benutzers und dechiffriert diesen mit dem Benutzer Passwort. Der Schlüssel wird dann zwischengespeichert und für alle Secure RPC Aufrufe verwendet. Zur Sicherheit sollte jeder Benutzer vor dem Logout das Kommando *keylogout* ausführen um den Schlüssel explizit aus dem Cache zu entfernen.

Falls der Benutzer auf dem Rechner kein Kennwort für den Login eingegeben hat, beispielsweise beim Remote Login via ssh unter Verwendung eines anderen Authentifizierungsmechanismus, so kann der Dämon *keyserv* den privaten Schlüssel nicht dechiffrieren. Damit der Benutzer Secure RPC dennoch verwenden kann, muss er das Kommando *keylogin* ausführen, welches das Benutzerpasswort erfragt und dem *keyserv* Prozess übergibt.

Literatur

[1] Stern, Hal: *Verwaltung von UNIX-Netzwerken mit* NFS und NIS. O'Reilly Verlag 1998

[2] Garfinkel, Simson & Spafford, Gene: *Practical UNIX & Internet Security.* O'Reilly Verlag 1996

[4] oder für größere Netzwerke via NIS