Authentifizierung und Autorisierung unter Linux/Solaris mit PAM

Ausarbeitung zum Vortrag im Seminar „Konzepte von Betriebssystemkomponenten“

Clemens Lang
sicslang@stud.informatik.uni-erlangen.de

ABSTRACT

Die *Pluggable Authentication Modules* sind seit vielen Jahren der de-facto Standard für Authentifizierung, Konto-, Sitzungs- und Passwortverwaltung unter Linux, Solaris und Mac OS X. Gegenüber der 1996 spezifizierten ursprünglichen Version wurden durch verschiedene Parteien wie z. B. den Entwicklern der Linux-Implementierung oder Sun Microsystems zahlreiche Änderungen vorgenommen. Dieses Paper soll einen Überblick über den aktuellen Stand von PAM geben.

1. EINLEITUNG

Der Zugriff von Benutzern auf Rechnersysteme wird durch spezielle Dienste, sogenannte *system-entry services* ermöglicht. Solche Dienste sind u. a. SSH-Server, `/bin/login`, FTP-Server, graphische Display-Manager von Desktopumgebungen, wie z. B. Gnome (`gdm`) oder KDE (`kdm`), oder diverse Bildschirmschoner.

Alle diese Dienste haben eine Gemeinsamkeit: Die Notwendigkeit, Benutzer vor dem Erlauben des Zugriffes zu authentifizieren. Vor der Entwicklung der *Pluggable Authentication Modules (PAM)* griffen solche Dienste direkt auf die Benutzerdatenbank(en) des Systems zu. Welche Datenbanken auf welche Weise gelesen wurden, bestimmten dabei die Dienste selbst. Wollte man als Systemadministrator eine andere Datenbank zum Speichern von Benutzern verwenden (etwa um von mehreren Rechnern auf die gleichen Benutzerdatenbanken zu verweisen) musste man jeden einzelnen Dienst entsprechend konfigurieren bzw. gegebenenfalls die Zugriffsmethoden selbst implementieren. Korrekturen, die Fehler beim Zugriff auf diese Datenbanken behoben, mussten ebenfalls in allen entsprechenden Diensten implementiert werden. [5]

Eine Lösung dieses Problems stellen die *Pluggable Authentication Modules* dar: Sie führen eine Abstraktionsebene zwischen den authentifizierenden Diensten und den modularisierten Authentifizierungsmethoden ein. Sie erlauben Systemadministratoren eine detaillierte Konfiguration, welche Authentifizierungsmethoden (in Form von Modulen) für Authentifizierungsanfragen welcher Dienste eingesetzt werden sollen, ohne die Programme selbst ändern zu müssen. Diese Abstrahierung ermöglicht des Weiteren eine schnellere Adaption neuer Authentifizierungsmethoden, da an den Diensten beim Wechsel von Authentifizierungsmethoden keine Änderungen vorgenommen werden müssen.

2. ZIELE VON PAM

Schon von Beginn der Entwicklung von PAM an existierte eine durchdachte Liste von Zielen des Frameworks. Bewusst wurden einige Punkte nicht aufgenommen: In der Spezifikation wird explizit darauf hingewiesen, dass keine Annahmen über den Transportweg der Kommunikation mit dem Benutzer getroffen werden [5]. Entsprechende Verschlüsselungsmaßnahmen sind demnach Aufgabe des Dienstes und/oder anderer Bibliotheken.

Die wichtigsten Punkte aus der Liste der Ziele sind [5]:

- Feingranulare Konfigurierbarkeit durch den Administrator: Es soll möglich sein, verschiedene Dienste getrennt von einander zu konfigurieren. Pro Dienst sollten mehrere Authentifizierungsmethoden durch beliebige logische Operatoren verbunden werden können.
- PAM sollte unabhängig von der Darstellungsform des Diensts sein: X-basierte graphische Eingabemethoden sollten genauso unterstützt werden, wie Terminal-basierte Logins (z. B. über SSH- oder Telnet-Verbindungen)
- Authentifizierungsmethoden sollten geändert oder ausgetauscht werden können, ohne die benutzenden Dienste verändern zu müssen.
- Neben Authentifizierung sollen Konto-, Sitzungs- und Passwortverwaltung modularisiert werden.

3. UMSETZUNG VON PAM

Unter Beachtung dieser Ziele wurde PAM zunächst von Sun Microsystems als interne Schnittstelle für Solaris 2.3 entwickelt. Diese Schnittstelle wurde im Vorfeld der *3rd ACM Conference on Computer and Communications Security* im März 1996 von Vipin Samar spezifiziert und veröffentlicht [5].

Diese Spezifikation wurde anschließend in diversen Betriebssystemen implementiert und erweitert. So ist Linux-PAM z. B. die üblicherweise in GNU/Linux-basierten Distributionen verwendete Implementierung. Einige Erweiterungen die-

ser, die auch Linux-PAM genannt wird, wurden 2001 spezifiziert [3]. Neben Linux und Solaris gibt es aktuell Implementierungen für Free-/Open-/NetBSD und Mac OS X. Es existiert weiterhin eine Implementierung für Windows NT [2], die sich jedoch nicht durchgesetzt hat und mit der Umstellung des Authentifizierungs-Frameworks ab Windows Vista als veraltet gelten kann.

Dieser Abschnitt beschreibt die Schnittstellen von PAM. Dazu gehört ein *Application Programming Interface (API)* zur Verwendung durch Dienste und das sogenannte *Service Provider Interface (SPI)* das die Kommunikationsschnittstelle zwischen PAM und den PAM-Modulen beschreibt. Wir betrachten ebenfalls die vier Teilbereiche von PAM Authentifizierung, Konto-, Sitzungs- und Passwortverwaltung, die zugehörigen Funktionen der API und das Stapeln von Modulen, dass einen Großteil der Flexibilität von PAM ausmacht.

3.1 Aufbau

Abbildung 1 zeigt den schematischen Aufbau von PAM. Die verschiedenen Dienste (*system-entry services*) greifen auf das *Application Programming Interface (API)*, das durch PAM bereitgestellt wird, zu, welches seinerseits die Konfigurationsdatei (normalerweise `/etc/pam.conf` und das Verzeichnis `/etc/pam.d/`) liest und anhand der Konfiguration entscheidet, welche Module zur Authentifizierung verwendet werden sollen. Anschließend werden diese Module geladen und ihnen über das *Service Provider Interface* die Authentifizierungsanforderung des Diensts mitgeteilt. Sie kommunizieren mit dem Nutzer und prüfen die Authentifizierungsanforderung mit den jeweils zutreffenden Authentifizierungsmechanismen.

Die PAM-API und -SPI sind weiterhin in die verschiedenen Teilbereiche ihrer Anwendung unterteilbar. Diese sind:

Authentifizierung Ist der Benutzer, wer er vorgibt zu sein?

Kontoverwaltung Ist das Konto abgelaufen oder gesperrt?

Sitzungverwaltung Ist der Benutzer bereits eingeloggt? Wie lange ist der Benutzer eingeloggt?

Passwortverwaltung Ändern der Passworts

Diese vier Teile sind jeweils unabhängig voneinander konfigurierbar. Authentifizierende Dienste müssen nicht jeden dieser Teile nutzen – möchte ein Dienst einen Benutzer lediglich authentifizieren, jedoch nicht seine Autorisierung zur Benutzung des Systems prüfen, kann z. B. auf die Kontoverwaltung verzichtet werden, indem der entsprechende Aufruf der API (vgl. Abschnitt 3.2) im Programm übergangen wird.

3.2 Die PAM-API

Applikationen, die PAM benutzen möchten, steht das *Application Programming Interface (API)* zur Verfügung. Dieses wird durch die Header-Datei `security/pam_appl.h` beschrieben. Einige der Funktionen der API können den in Abschnitt 3.1 beschriebenen Teilbereichen von PAM zugeordnet werden. Im einzelnen sind das [5]:

Authentifizierung `pam_authenticate()` authentifiziert den Benutzer. Nach erfolgreicher Authentifizierung sollte `pam_setcred()` aufgerufen werden, um Modulen zu erlauben, sogenannte „credentials“ (etwa: Eigenschaften, Berechtigungsnachweise) wie z. B. Kerberos-Tickets oder spezielle Gruppenberechtigungen zu setzen.

Kontoverwaltung `pam_acct_mgmt()` wird normalerweise nach der Authentifizierung aufgerufen und prüft, ob das Konto des Benutzers abgelaufen oder gesperrt ist. Auch Zeitbeschränkungen für das Login (z. B. eine Kindersicherung) können durch diese Funktion realisiert werden.

Sitzungsverwaltung `pam_open_session()` teilt PAM mit, dass eine neue Sitzung für einen Benutzer geöffnet wurde. Eine geöffnete Sitzung sollte mit einem Aufruf von `pam_close_session()` wieder geschlossen werden.

Passwortverwaltung `pam_chauthtok()` stellt die nötige Funktionalität bereit, um Passwörter zu ändern. Dienste können diese Funktion aufrufen, wenn ein Rückgabewert einer Funktion darauf hinweist, dass das Passwort eines Benutzers abgelaufen ist (Auftreten des Fehlercodes `PAM_AUTHTOK_EXPIRED`) oder wenn der Benutzer sein Passwort ändern möchte (z. B. mittels des dafür geschriebenen Programms `/usr/bin/passwd`).

Alle Funktionen der PAM-API[1] erwarten einen Zeiger auf ein `pam_handle_t`, das durch einen Aufruf von `pam_start()` angelegt und initialisiert wird. Dieses Handle speichert den internen Zustand der Transaktion von PAM und ermöglicht mehrere parallele Authentifizierungsprozesse durch PAM. Für nutzende Dienste ist die Struktur `pam_handle_t` als leer deklariert, um Änderungen an den internen Datenstrukturen von PAM zu verhindern. Dennoch können einzelne Eigenschaften der Struktur durch `pam_set_item` und `pam_get_item` verändert beziehungsweise gelesen werden [4].

Die Kommunikation zwischen PAM-Modulen und dem Benutzer findet über die sogenannte *conversation function* [4] statt. Diese Funktion muss vom nutzenden Service bereitgestellt und beim Aufruf von `pam_start()` als Funktionszeiger übergeben werden. Die Funktion bekommt ein Array von `pam_message`[2]s und soll auf jede dieser Nachrichten mit einer `pam_response`[3] antworten. Dabei gibt es vier verschiedene Typen von `pam_message`s:

`PAM_PROMPT_ECHO_OFF` Weist die *conversation function* an, einen Text vom Benutzer einzulesen und maskiert den Text während der Eingabe.

`PAM_PROMPT_ECHO_ON` Weist die Funktion an, einen Text vom Benutzer einzulesen, ohne ihn zu maskieren.

`PAM_ERROR_MSG` Weist die Funktion an, eine Fehlermeldung auszugeben.

[1] außer `pam_start()`, das benötigt wird, um das Handle zu erzeugen

[2] Eine Struktur aus einer Zahl, die den Typ angibt und einer Zeichenkette, die für den Benutzer lesbar ausgegeben werden soll

[3] Ebenfalls eine Struktur aus einer Zahl für den Typ und einer optionalen Zeichenkette, die vom Benutzer als Antwort eingegeben wurde

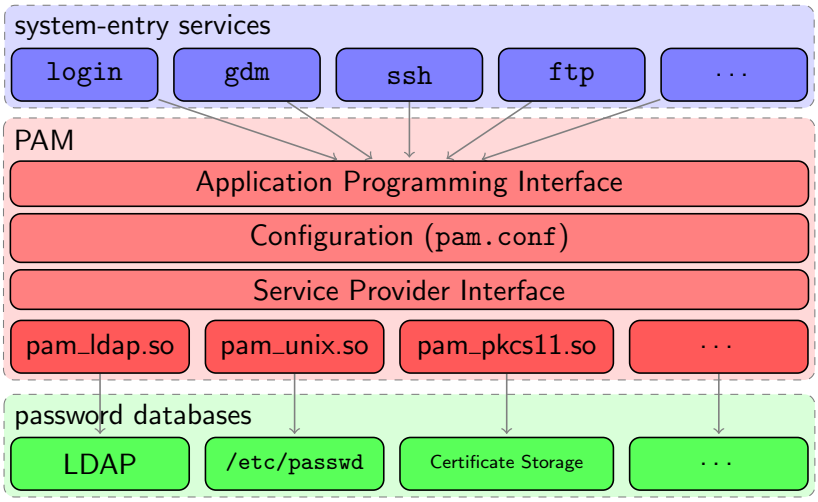


Figure 1: Schematischer Aufbau von PAM

`PAM_TEXT_INFO` Weist die *conversation function* an, einen textuellen Hinweis auszugeben.

Dieser Ansatz wurde gewählt, um die Nutzung von PAM unabhängig vom Medium zu machen. Je nach Darstellungsform der Applikation kann und muss die Implementierung dieser Funktion variieren – graphische Logins müssen offensichtlich andere Anweisungen ausführen, um diese Aktionen durchzuführen, als dies bei einer Anmeldung auf einem Terminal oder über eine Netzwerkverbindung nötig wäre.

Diese Annahme schränkt PAM jedoch in der Flexibilität der Authentifizierungsmethoden deutlich ein. PAM-Module, die spezielle Hardware wie etwa USB-Tokens oder Peripherie zur Erkennung biometrischer Merkmale benötigen, existieren zwar, funktionieren aber nur lokal, d. h. wenn der Besitzer sich physikalisch vor dem Rechner, gegen den authentifiziert werden soll, befindet. Diese Module greifen in der Regel direkt auf die Hardware zu. Versucht sich der Benutzer allerdings über eine Netzwerkverbindung zu authentifizieren, kann auf die Hardware, selbst wenn sie am Rechner des Benutzers verfügbar ist, nicht zugegriffen werden.

3.3 Clientseitige Unterstützung für PAM

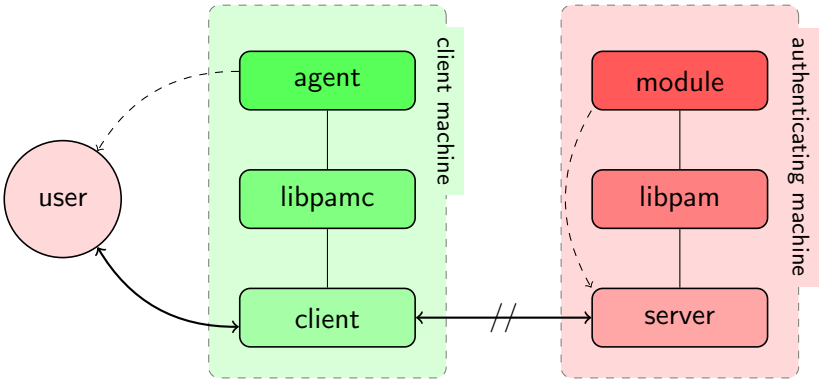
Um diese Beschränkung in der Spezifikation zu umgehen, wurde für Linux-PAM mit `libpamc` eine Bibliothek zur clientseitigen Unterstützng von PAM entwickelt und das Protokoll um binäre Nachrichten erweitert, die am Client durch eben diese Bibliothek an sogenannte *Agents* weitergegeben werden [3]. Diese arbeiten dann mit entsprechenden PAM-Modulen auf dem Server zusammen und können so zum Beispiel auf am Client vorhandene Hardware zugreifen. Abbildung 2 visualisiert diese Erweiterung. Zwischen Server und Client besteht ein Kommunikationskanal, über den Nachrichten ausgetauscht werden. Dabei ist wichtig festzustellen, dass PAM keine Annahmen über die Art der Übertragung trifft und auch nicht sicherstellt, dass die Übertragung sicher ist. Die auf dem Server über `libpamc` angebundenen Module steuern die Authentifizierung bzw. Autorisierung. Damit kann ein Modul neben der *conversation function* über sogenannte *binary prompts*, die an einen entsprechenden *Agent* am Client weitergegeben werden, mit dem Benutzer kommunizieren. Aktuell existiert allerdings kein PAM-Modul, das die Möglichkeit der clientseitigen Unterstützung verwendet.

3.4 Konfiguration

Alle gängingen PAM-Implementierungen stellen den Systemadministratoren zur Konfiguration eine tabellenartig strukturierte Datei als Schnittstelle zur Verfügung (vgl. Beispiel in Tabelle 1). Die erste Spalte nennt den Namen des Diensts (oder „OTHER“ als Standard-Konfiguration für Dienste, für die keine anderen Module konfiguriert sind). Die zweite Spalte bestimmt, für welchen Teilbereich (vergleiche Abschnitt 3.1) die Zeile gilt. Mögliche Werte sind „account“, „auth“, „password“ und „session“. Spalte drei („Flags“) wird in Abschnitt 3.4, Absatz „Stapeln von Modulen“ ausführlich behandelt. Die vierte Spalte „module path“ gibt den Pfad der Shared-Object-Datei, die das zu nutzende Modul bereitstellt, relativ zu einem (systemspezifischen) Include-Verzeichnis an. In der fünften und letzten Spalte können schließlich modulspezifische Konfigurationsoptionen gesetzt werden.

Stapeln von Modulen. Um die Forderung nach mehreren Authentifizierungsmethoden für einen Dienst umzusetzen, erlaubt es PAM verschiedene Module zu stapeln. Abbildung 3 zeigt eine beispielhafte Konfiguration einer solchen auch *Stacking* genannten Anordnung von Modulen. Für die drei Teilbereiche Sitzungsverwaltung, Authentifizierung und Kontoverwaltung sind im Beispiel unterschiedliche Module konfiguriert worden. In der Abbildung nicht sichtbar sind die logischen Operatoren zwischen den drei gestapelten Modulen im Teilbereich Authentifizierung `pam_kerberos_auth.so`, `pam_unix_auth.so` und `pam_rsa_auth.so`. Die Konfigurationsdatei (vgl. Tabelle 1) erlaubt das Setzen dieser Operatoren in der Spalte *Flags*. Mögliche Werte sind [4, 1]:

- **required**:

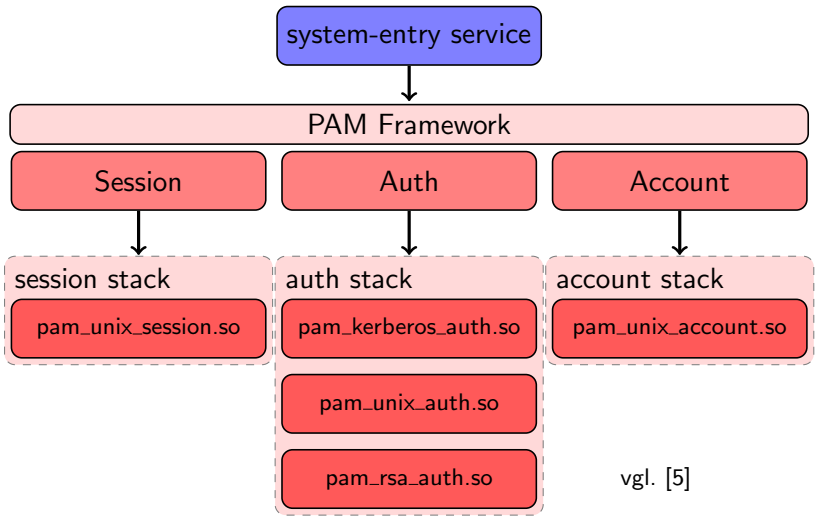


[3]

Figure 2: Erweiterung `libpamc` in Linux-PAM

Service	Type	Flags	Module Path	Options
login	auth	required	pam_kerb_auth.so	debug
login	auth	required	pam_unix_auth.so	use_mapped_pass
login	auth	optional	pam_rsa_auth.so	try_first_pass

Table 1: Beispiel-Konfiguration von PAM [5]



vgl. [5]

Figure 3: Module Stacking in PAM

Der Erfolg dieses Moduls ist für ein Gelingen des Prozesses notwendig. Bei Misserfolg werden alle weiteren Module dennoch abgearbeitet, um vor dem Benutzer (bzw. potentiellen Angreifer) zu verschleiern, welches Modul fehlgeschlagen ist.

- **requisite**:
  Wie **required**, jedoch ohne im Fehlerfall den Rest der Module weiter abzuarbeiten.

- **sufficient**:
  Der Erfolg dieses Moduls genügt für ein Gelingen des kompletten Authentifizierungsprozesses (es sei denn, ein Modul, das mit **required** markiert war, ist bereits vorher fehlgeschlagen). Ein Misserfolg wird behandelt wie der eines mit **optional** konfigurierten Moduls.

- **optional**:
  Erfolg oder Misserfolg dieses Moduls sind nur dann relevant, wenn es das einzige Modul im aktuellen Stapel ist.

- **include**:
  Bindet eine andere Konfigurationsdatei gleichberechtigt zu den bereits spezifizierten Modulen in diesem Stapel ein.

- **substack** (nur Linux):
  Bindet eine andere Konfigurationsdatei als Unter-Stapel zu den bereits spezifizierten Modulen in diesem Stapel ein. Damit lassen sich Klammerausdrücke in logischen Ausdrücken modellieren.

- **binding** (nur Solaris):
  Wie **sufficient**, jedoch mit dem Unterschied, dass ein Misserfolg behandelt wird wie der eines **required**-Moduls.

Neben diesen Werten für die *Flags*-Spalte ist es in Linux-PAM ebenfalls möglich, mit eckigen Klammern geklammerte Ausdrücke anzugeben, die für jeden der verschiedenen Rückgabewerte[4] mit verschiedenen Aktionen reagieren. Solche Aktionen können in diesem Fall sein [4]:

- **ignore**: Der Rückgabewert wird ignoriert, die Abarbeitung des Stapels wird fortgesetzt.

- **bad**: Der Stack wird weiter abgearbeitet, der Prozess schlägt fehl.

- **die**: Die Abarbeitung des Stacks wird abgebrochen, der Prozess schlägt fehl.

- **ok**: Der Stack wird weiter abgearbeitet.

- **done**: Die Abarbeitung des Stacks wird beendet, der Prozess kehrt erfolgreich zurück.

- **reset**: Setzt den Zustand (und damit eventuell aufgetretene *bad*s) zurück und fährt mit dem nächsten Modul fort.

[4]Für eine Liste dieser Rückgabewerte siehe [4]

Betrachten wir einen fiktiven Aufruf von `pam_authenticate()` im Beispiel aus Tabelle 1. Damit der Authentifizierungsprozess gelingt, müssen die Module `pam_kerb_auth.so` und `pam_unix_auth.so` eine erfolgreiche Authentifizierung signalisieren. Die Authentifizierung durch das Modul `pam_rsa_auth.so` wird versucht, kann aber fehlschlagen, ohne Auswirkungen auf den Authentifizierungsprozess zu haben.

4. ZUSAMMENFASSUNG

Die *Pluggable Authentication Modules* sind der am weitesten verbreitete Standard für Authentifizierungs- und Autorisierungsanforderungen unter Linux, Solaris, BSD und Mac OS X. Trotz einiger Versuche, die Spezifikation zu erweitern ist die ursprüngliche Fassung von 1996 immer noch der kleinste gemeinsame Nenner, der am häufigsten Anwendung findet. Betrachtet man die langjährige Stabilität der Schnittstelle, erhält man den Eindruck, dass PAM wohl am Ende seines Entwicklungsprozesses angekommen ist. Für die Einsatzzwecke, für die clientseitige Unterstützung von Linux-PAM benötigt wird, wurden häufig Alternativen gefunden. Die von SSH bekannte key-basierte Authentifizierung wäre z. B. durch ein PAM-Modul und einen passenden *Agent* auf dem Client durchführbar – die Entwickler des Protokolls haben sich aber für die (portablere) Variante entschieden, die Authentifizierung durch eine Public/Private-Key-Infrastruktur direkt ins Protokoll und damit in Server und Client zu integrieren [6].

Dennoch ist PAM keineswegs ein totes Projekt. Es ist eine Vielzahl von PAM-Modulen verfügbar und immer wieder kommen neue, wie beispielsweise `pam-face-authentication`, ein PAM-Modul, das mit Gesichtserkennung durch eine Webcam arbeitet, hinzu.

5. REFERENCES

[1] *man pam.conf(4)*, sunos 5.10 edition.

[2] N. Itoi and P. Honeyman. Pluggable authentication modules for windows nt. In *Proceedings of the 2nd USENIX Windows NT Symposium*, pages 1–12, Seattle, Washington, USA, Aug 1998. USENIX.

[3] A. G. Morgan. *Pluggable Authentication Modules (PAM)*, Dec 2001. Draft for the Open-PAM working group PAM standard.

[4] A. G. Morgan and T. Kukuk. *The Linux-PAM Guides*, 1.1.1 edition, Dec 2009. Manuals for Linux-PAM.

[5] V. Samar. Unified login with pluggable authentication modules (pam). In *CCS '96: Proceedings of the 3rd ACM conference on Computer and communications security*, pages 1–10, New York, NY, USA, 1996. ACM.

[6] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Authentication Protocol. RFC 4252 (Proposed Standard), Jan. 2006.