



Abwehr

Sichere Authentifikationssysteme

Sonnleitner Erik



Schwierigkeitsgrad



Auch in Zeiten von biometrischen Identifikationssystemen und Public-Key Kryptographie, sorgen Passwörter immer noch in nahezu allen IT-Bereichen für die Authentifikation, nicht zuletzt aufgrund der Einfachheit und Bequemlichkeit der Benutzer sowie Administratoren. In diesem Artikel erfahren Sie, wie Sie Login-Systeme härten und Authentifikationsmechanismen gegen unbefugten Zugriff absichern.

Passwörter werden in Zeiten der globalen Vernetzung nahezu überall verwendet, obgleich sie eine einerseits schwache und andererseits äußerst leicht angreifbare Basis für Diebstahl und Angriff darstellen.

Im Gegensatz zu biometrischen Authentifikationstechniken wie Iris-Scans oder Ohrvermessungen, können Passwörter gezielt generiert und oftmals auch gleich auf Validität geprüft werden. Brute-Force-Werkzeuge und Wort-Listen sei Dank, ist die Ausführung eines solchen Angriffs ein Kinderspiel und erfordert somit keine tiefgehenden Kenntnisse von den verwendeten Betriebssystemen und Netzwerkprotokollen.

Auf einer etwas abstrakteren Ebene unterteilt man generelle Systematiken zur Authentifizierung von Objekten in drei mögliche Teilbereiche: Wissen, Besitz und Eigenschaft, sortiert nach ihrer Sicherheit. Während sich Biometrie gänzlich im Gebiet der Eigenschaften von Personen bewegt, welche obendrein als relativ sicher angesehen werden, sind Passwörter lediglich eine Form von Wissen, und somit ein schwaches Verfahren zur Authentifikation.

Diese Tatsache bringt zwei große Probleme mit: Einerseits ist der Hang zur Bequemlichkeit

oft größer als das Bedürfnis sich möglichst sichere und unartikulierbare Passwortphrasen zu merken, und andererseits können sie leicht ausgespäht werden.

In diesem Artikel erfahren Sie...

- Wie man Einmal-Passwörter anhand von s/Key für Services konfiguriert und generiert;
- Wie Sicherheitsrelevante PAM Module funktionieren und angewendet werden;
- Wie Benutzer schlüsselbasierende Authentifikation nutzen können;
- Wie Sie Dienste verstecken und durch Port-Knocking aktivieren;
- Wie Sie USB-Speicher zur Authentifikation nutzen können.

Was sie vorher wissen/können sollten...

- Umgang mit Unixsystemen und deren Services;
- Grundlagen asymmetrischer Kryptographie;
- Umgang mit iptables.

Glücklicherweise sind die Zeiten von rlogin, rsh und telnet vorüber, doch auch das Verschlüsseln von übertragenen Passwörtern ist nicht der Weisheit letzter Schluss, lediglich das direkte Sniffing im Netzwerk und gegebenenfalls Man-in-the-middle Attacken werden erschwert. Keylogger sind nicht nur in Hardware sowie Software erhältlich sondern auch leicht zu verstecken und, mit etwas Geschick, auch schwer zu entdecken. Darüber hinaus werden auch heutzutage einerseits bei zahlreichen Protokollen die nicht-kryptographischen Varianten noch immer vorgezogen wie es beispielsweise bei FTP oder POP3 der Fall ist, andererseits sind oftmals Protokolle

im Einsatz welche bis dato kein kryptographisch abgesichertes Pendant bieten (man denke hier an DNS oder ICMP).

One-time-passwords

Einmal-Passwörter bilden eine großartige Alternative zu herkömmlichen Logins. Da ein Passwort nur einmal über das Netzwerk übertragen wird, ist bei Diebstahl nicht die Gefahr gegeben, dass sich der Angreifer aus der gewonnenen Information große Vorteile verschaffen könnte, zumal der abgefangene String seine Gültigkeit unmittelbar nach dem Login wieder verliert. Zusätzlich ist man nicht von einer (möglicherweise sowieso nicht gegebenen)

Verschlüsselung der verwendeten Protokolle auf dem Application Layer abhängig.

Die Verteilungen der Passwörter kann grundsätzlich auf zwei Arten erfolgen:

- (1) Der Administrator erstellt die Passwortlisten für alle betroffenen Benutzer im voraus und verteilt diese im Anschluss. Diese Variante repositioniert die Verwendung von Passwörtern von der Kategorie Wissen zum eigentlichen Besitz;
- (2) Der Benutzer verfügt selbst über einen Client über den die jeweiligen Einmal-Passwörter generiert werden können. Hierfür muss ebenfalls ein vordefiniertes Passwort eingegeben werden, welches jedoch nicht im Netzwerk übertragen wird. Diese Variante ist komfortabler, und speziell für nicht-verschlüsselte Protokolle zu empfehlen.

Listing 1. PAM Konfiguration

```

1 auth      required      pam_env.so
2 auth      required      pam_env.so envfile=/etc/default/locale

3 #@include common-auth
4 auth      sufficient    pam_opie.so
5 auth      sufficient    pam_unix.so
6 auth      required      pam_deny.so

7 account   required      pam_nologin.so
8 @include  common-account

9 @include  common-session
10 session  optional      pam_motd.so
11 session  optional      pam_mail.so standard noenv
12 session  required     pam_limits.so

11 @include common-password
    
```

OPIE via PAM

s/Key stellt ein One-Time-Password-System für unix-ähnliche Betriebssysteme zur Verfügung und ist bei diversen BSD-Derivaten bereits Standard. Für Linux gibt es die freie Implementierung OPIE (One-time Passwords In Everything), welche dankenswerter Weise auch gleich die nötigen PAM-Schnittstellen mitliefert. Die meisten großen Distributoren stellen Pakete für OPIE als Server sowie Client und die nötigen Bibliotheken zur Verfügung, welches es zu installieren gilt.

Obgleich das passwd/shadow-Konzept von Linux eine einfache Modifizierung der Benutzerkonten ermöglicht, so wäre es sehr problematisch wenn jedes Programm welches nach Authentifizierung verlangt direkt auf diese Dateien zugreifen müsste. Darüber hinaus wäre es sehr komplex, solche Applikationen über zentralisierte Login-Systeme wie Kerberos oder LDAP anzuwenden. Hierfür wurde PAM entwickelt, und stellt eine Schnittstelle zwischen den zur Verfügung gestellten Authentifikationsmecha-

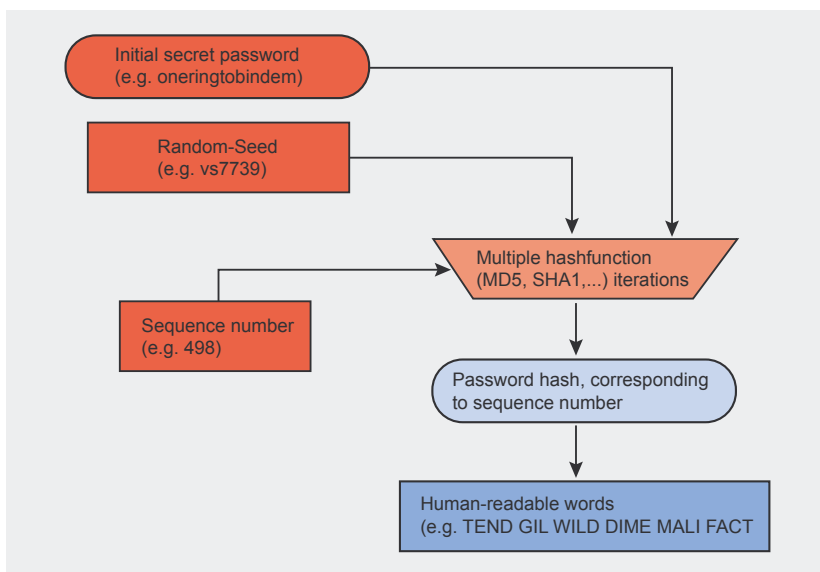


Abbildung 1. Funktionsweise von s/Key

nismen und den einzelnen Applikationen dar. Die Funktionsweise von PAM wird im Kasten PAM detaillierter betrachtet.

Absichern der Dienste

Im Verzeichnis `/etc/pam.d/` liegen sämtliche PAM-Konfigurationen der am System verfügbaren PAM-fähigen Services. Um s/Key, respektive OPIE am Beispiel von SSH zu aktivieren, sehen wir uns die Datei `/etc/pam.d/ssh` genauer an. Die (auskommentierte!) Zeile 3 in Listing 1 teilt PAM mit, die Datei `/etc/pam.d/`

`common-auth` auszulesen und die dort definierten Module zum normalen Login auf einem Unix-System zu laden und auszuführen, also lokale Benutzerkennungen zu überprüfen.

Wir wollen im folgenden, dass sich Benutzer abseits von der normalen SSH-Loginprozedur auch mittels One-time-Passwörter einloggen können. Dies ist nützlich wenn sie s/Key nur bei bestimmten Benutzerkonten anwenden bzw. anwenden müssen.

Das Schlüsselwort `sufficient`, welchem nun sowohl `pam_opie` als

auch `pam_unix` zugewiesen ist, teilt PAM mit, dass lediglich eine der beiden Authentifikationsmethoden positive Rückmeldung geben muss, um den Login zu gewähren. Ersetzen wir `sufficient` mit `required`, so müssten beide Mechanismen, also die reguläre Passwordeingabe zusammen mit dem generierten Einmal-Passwort korrekt eingegeben werden.

Um SSH anzuweisen das neue PAM Modul auch zu nutzen, muss in der Konfigurationsdatei `/etc/ssh/sshd_config` noch die der Variablenwert `ChallengeResponseAuthentication` noch auf `yes` gesetzt werden, da es sich hier um ein klassisches Challenge/Response-Verfahren handelt.

OTPs aktivieren

Nun müssen natürlich noch die Passwörter am Server aktiviert werden, was mit dem Werkzeug `opiepasswd` geschieht, siehe Listing 2. Zu unterscheiden ist lediglich für welche Benutzer das s/Key-System aktiviert werden soll. Möchte man

Listing 2. OPIE Konfiguration

```
[root@delta-xi:~]# useradd ssh_otp
[root@delta-xi:~]# opiepasswd -c -f ssh_otp
Adding ssh_otp:
Only use this method from the console; NEVER from remote. If you are using
telnet, xterm, or a dial-in, type ^C now or exit with no password.
Then run opiepasswd without the -c parameter.
Using MD5 to compute responses.
Enter new secret pass phrase:
Again new secret pass phrase:

ID ssh_otp OTP key is 499 de4875
BARN BOND SIRE BRAY RULE DOUR
```

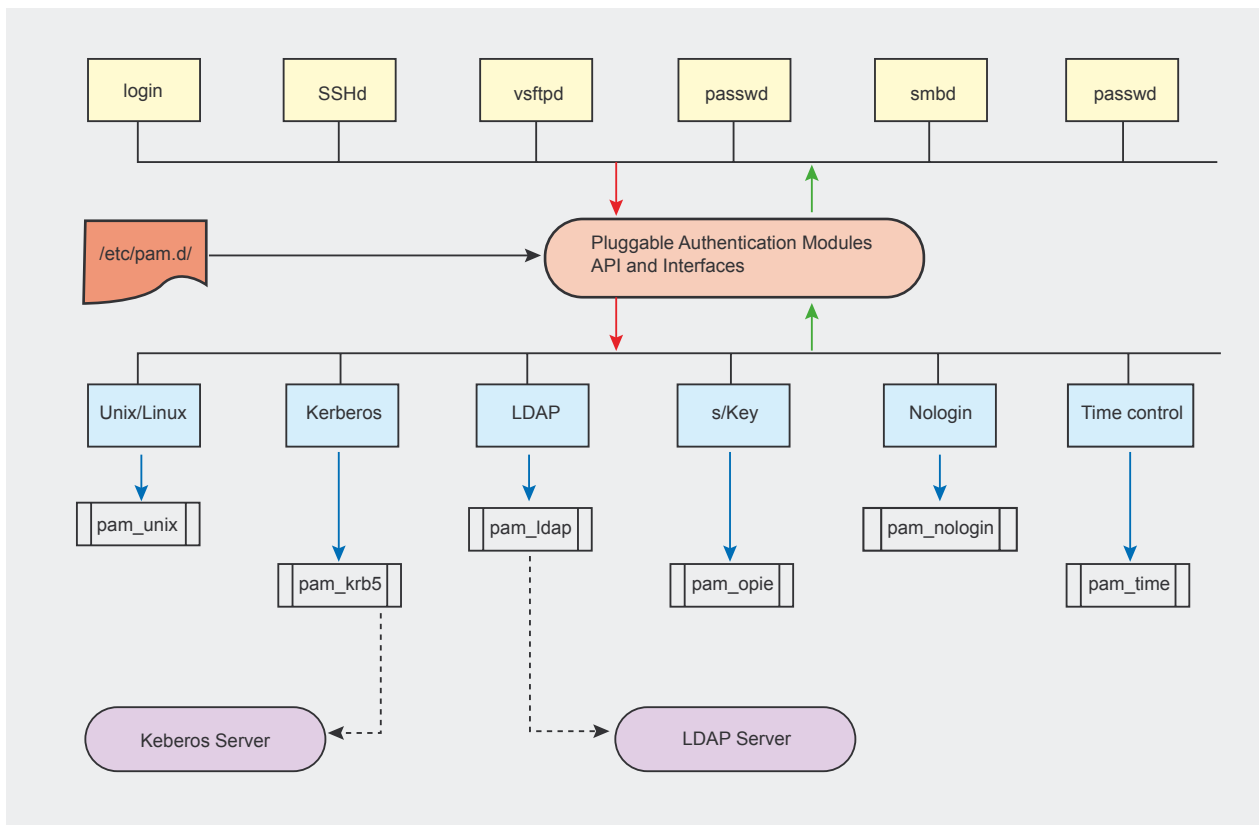


Abbildung 2. Funktionsweise von PAM

dies für alle Benutzer am System, so reicht es aus den spezifischen Benutzernamen als letzten Parameter von `opiepasswd` wegzulassen.

Die am Ende von Listing 2 ausgegebenen Werte geben nur der Vollständigkeit halber das erste Passwort, sowie die Nummer (499) und den zufällig genierten Seed-Wert (de4875) an, welche aber nicht zwingend benötigt werden.

Nun müssen die Passwörter nur mehr erzeugt werden, wofür `opiekey` zuständig ist. Möchte der Administrator Passwortlisten verteilen, so empfiehlt sich gleich mehrere Passwörter für jeden Benutzer zu generieren und auf Papier auszuhändigen. Das hat den Vorteil, dass der Benutzer gar nicht erst in Besitz eines heiklen Pass-

wortes ist, sondern lediglich eine Liste mit begrenztem Umfang bei sich trägt, welche bei Verlust und dementsprechender rascher Reaktion auch gleich wieder mit neuen Werten errechnet werden kann, so dass die alten Passwörter ihre Gültigkeit verlieren:

```
opiekey -n 100 499 de4875
```

Nach der Eingabe des korrekten Passworts (welches natürlich dasselbe sein muss wie beim `opiepasswd` Aufruf) werden 100 Passwörter erzeugt, beginnend von 499 bis 399.

Die zweite Alternative ist den Benutzern das Passwort zur Erzeugung von eigenen Passwörtern zu nennen. Dieses kann an einem beliebigen s/Key kompatiblen Client

(z.B. Windows-Clients, Palmtops oder auch Java-Applikationen auf Mobiltelefonen) errechnet werden. Denn versucht sich der Benutzer am System via SSH anzumelden, werden ihm Seed und benötigte Passwörter mitgeteilt:

```
tomhet@vortex:~$ ssh ssh_otp@delta-
                               xi.net
otp-md5 498 de4875 ext, Response:
```

Daher reicht der Aufruf von `opiekey 498 de4875` um das benötigte Passwort zu generieren.

Um den Client die Generierung der Passwörter zu erleichtern, wäre etwa ein Browser-Plugin denkbar, dass diese Aufgabe für einen erledigt.

Absicherung von FTP Diensten

Die meisten unter Linux verbreiteten FTP Services nutzen die PAM API oder bieten die Möglichkeit sie zu nutzen. `vsftpd` ist ein für Sicherheit bekannter FTP Service, der jedoch standardmäßig kein sFTP anbietet. Über die Konfigurationsdatei `/etc/pam.d/vsftpd` kann auch hier `pam_opie.so` zur Verwendung von s/Key verwendet werden. Es ist ebenfalls ratsam, die normale Unix-Login-Prozedur von PAM zu deaktivieren.

Sollten Sie ihren FTP Service nicht permanent in Gebrauch haben, bietet sich an auch das PAM Modul `pam_nologin.so` zu verwenden:

```
auth required pam_nologin.so
```

Dieses Modul kontrolliert beim Loginversuch eines Benutzers, ob die Datei `/etc/nologin` existiert. Ist dies der Fall, wird unabhängig von der gewählten Authentifikationsvariante der Zugriff auf den Dienst verweigert und der Inhalt der Datei ausgegeben, sofern es sich beim Benutzer nicht um den Administrator handelt.

Des Weiteren kann die Sicherheit wesentlich erhöht werden, indem die Möglichkeit eines remote Logins zeitlich limitiert wird. Das Modul

PAM

Die Pluggable Authentication Modules, oder kurz PAM, wurden ursprünglich von Sun Microsystems entwickelt und verleihen dem Administrator die Möglichkeit, sämtliche Authentifikationsmethoden zentral zu verwalten sowie frei zu konfigurieren, welche Services durch welche Methoden, wie streng und unter welchen Voraussetzungen Benutzer authentifizieren dürfen (siehe Abbildung 2).

Die meisten Systeme bieten unter `/etc/pam.d/` für jeden PAM-fähigen Dienst eine Datei in welcher sequentiell Einträge der Form:

```
type control module arguments
```

zu finden sind. `type` kann dabei vier Werte annehmen:

- `account` für nicht-authentifikationsbezogene Restriktionen, wie beispielsweise zeitabhängige Login-Sperren;
- `auth` für die eigentliche Authentifizierung;
- `password` zur Änderung von Benutzerpasswörtern, sowie;
- `session` für spezielle Operationen welche vor, bzw. nach dem erfolgten Login ausgeführt werden sollen, wie z.B. das Einrichten von Systemumgebungen oder aktives Logging.

`control` legt fest, was bei korrekter oder inkorrekt Abarbeitung des Moduls zu geschehen hat, und besitzt folgende Ausprägungen:

- `required` besagt, dass die Authentifizierung im Fall inkorrekt Modularbeitung in jedem Fall negativ verläuft, die restlichen angeführten Module jedoch trotzdem ausgeführt werden;
- `requisite` realisiert im Wesentlichen dieselbe Funktionalität wie `required`, nur dass PAM der aufrufenden Applikation sofort einen negativen Rückgabewert liefert;
- `sufficient` lässt negative Authentifikationen zu, sofern mindestens ein Modul positiv abgearbeitet wurde;
- `optional` lädt zusätzliche Module, bei denen die Korrektheit der Ausführung nicht von Relevanz ist.

Am Ende jeder Zeile befindet sich das eigentliche Modul, relativ zu `/lib/security`. Sämtliche dort installierten Module können angeführt werden.

`pam_time.so` liest die auszuführenden zeitlichen Beschränkungen über `/etc/security/time.conf` ein und kehrt negativ zurück, sofern sich die aktuelle Zeit nicht innerhalb des erlaubten Fensters befindet.

Passwortlose Authentifikation über asymmetrische Kryptographie

OpenSSH bietet in der Version 2 eine sehr gute Basis für sichere verschlüsselte Sitzungen. Obgleich die Gefahr von Sniffing-Attacken sinkt, sollte man sich trotzdem nicht in vollkommener Sicherheit wiegen, denn zahlreiche alternative Angriffsmethoden bleiben weiterhin bestehen, da das Passwort (obwohl nicht im Klartext) über das Netzwerk übertragen wird. Damit bleibt das Protokoll immer noch anfällig für simple Brute-Force Attacks, oder durch ausgefeiltere Angriffstaktiken wie etwa SSH Timing Attacks.

Wer gänzlich auf passwortbasierte Authentifikation verzichten will, kann stattdessen auf asymmetrische Kryptographie setzen. Dabei hat jeder Benutzer ein unikates *Private-/Public-Key* Schlüsselpaar, von denen der private Schlüssel sicher am Clientsystem vorliegen muss.

Der Server benötigt lediglich den öffentlichen Schlüssel aller Benutzer, und zumal diese keiner Geheimhaltung unterliegen, können sie auch auf unsicherem Wege über das Netzwerk transportiert werden.

Zuerst muss für jeden Benutzer ein Schlüsselpaar erstellt werden, via:

```
ssh-keygen -t <type>
```

wobei `<type>` den zu verwendenden Public-Key Algorithmus beschreibt, welcher bei SSHv2 wahlweise `rsa` oder `dsa` sein kann. SSH verlangt nun die Eingabe einer Passphrase, über welche der private Schlüssel auf dem lokalen Dateisystem symmetrisch verschlüsselt wird, um die Auswirkungen von eventuellem Diebstahl zu minimieren.

Listing 3 verdeutlicht den Prozess zur Erstellung des Schlüsselpaares. SSH akzeptiert hier auch leere Passwörter, wodurch der resultierende private Schlüssel jedoch Plain Text auf dem Datenträger verbleibt, was generell nicht zu empfehlen ist.

Der öffentliche Schlüssel wird i.d.R. unter `~/.ssh/id_dsa.pub` (respektive `id_rsa.pub`), der private unter `~/.ssh/id_dsa` gespeichert. Erstgenannter muss nun noch an

die Datei `~/.ssh/authorized_keys` am Server angehängt werden (siehe Listing 3).

Um den Public-Key Loginmechanismus von OpenSSH zu aktivieren muss noch die Option `PubkeyAuthentication` in `/etc/ssh/sshd_config` auf `yes` gesetzt werden. Soll dies die einzige Möglichkeit bleiben sich Zugang zum System zu gewähren kann die Weitergabe der Logindaten zu PAM zusätzlich

Listing 3. Public-Key Authentifizierung

```
tomhet@vortex:~$ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (~/.ssh/id_dsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ~/.ssh/id_dsa.
Your public key has been saved in ~/.ssh/id_dsa.pub.
The key fingerprint is:
7b:6d:92:28:bc:c0:02:0a:5f:2e:9d:95:b4:d4:c2:6e tomhet@vortex
tomhet@vortex:~$ scp id_dsa.pub root@server:~/.ssh/.tmppk
  && ssh root@server "cat ~/.ssh/.tmppk >> ~/.ssh/authorized_keys && rm ~/.ssh/.tmppk"
```

Listing 4. knockd.conf

```
[options]
logfile = /var/log/knockd.log
[unblockSSH]
sequence = 3748,2994,2338
seq_timeout = 5
tcpflags = syn
start_command = /sbin/iptables -I INPUT 1 -s %IP% -p tcp --dport 22 -j ACCEPT
cmd_timeout = 10
stop_command = /sbin/iptables -D INPUT 1 -s %IP% -p tcp --dport 22 -j ACCEPT
```

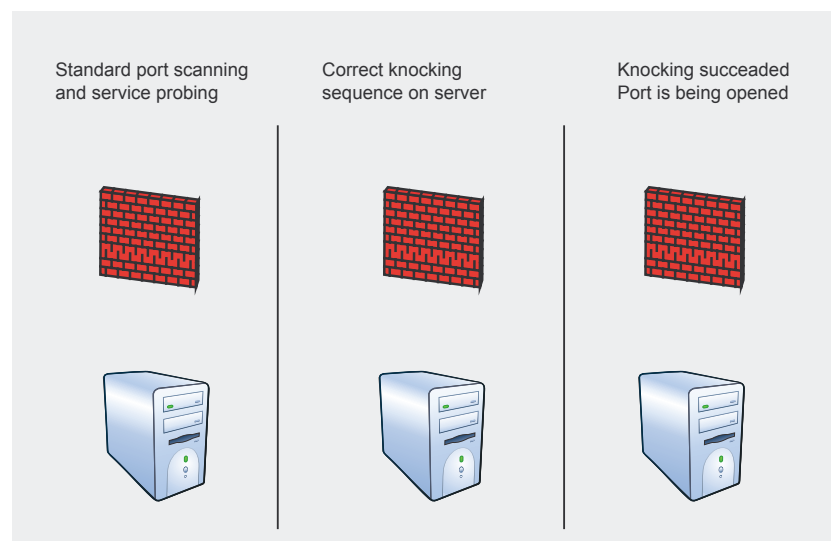


Abbildung 3. Port-Knocking

unterbunden werden, indem `UsePAM` auf `no` gesetzt wird. Beim Versuch eines remote Logins zum SSH Service wird nun lediglich lokal das Passwort zur Entschlüsselung des sicher abgelegten privaten Schlüssels abgefragt, sofern diesem ein Passwort zugeteilt wurde. Dabei wird lediglich eine mit Hilfe des privaten Schlüssels erzeugte Signatur übertragen, welche sicherstellt dass es sich um den vorgegebenen Benutzer

handelt und aus dem der verwendete Schlüssel nicht errechenbar ist.

Authentifikation für Paranoide: Port-Knocking

Selten aber doch ist es wünschenswert einen Service gänzlich verschwinden zu lassen um Angreifern bereits beim Fingerprinting und Scannen ihres Ziels den Boden unter den Füßen wegzuziehen.

Das Prinzip des Port-Knocking ist genauso simpel wie effizient: Die Firewall des Servers sperrt grundsätzlich den Dienst welcher geheimgehalten werden soll nach außen. Das bedeutet, dass eingehende Port-Scanner sowie direkte Verbindungsversuche zu dem betroffenen Port vollkommen ins Leere gehen, da die Firewall jede Anfrage blockt.

`knockd` ist die Implementierung eines Port-Knocking-Daemons, welcher selbst keinen Service anbietet sondern ausschließlich auf Verbindungeanfragen von geschlossenen Ports am Server horcht. Stellt dieser eine vordefinierte Folge von abgetasteten Ports, in der richtigen Reihenfolge und innerhalb eines (meist sehr kleinen) Zeitfensters fest, so werden bestimmte Aktionen durchgeführt welche der Administrator frei festlegen kann. Im Normalfall also ein Firewall-Eintrag erstellt, welcher die IP von der aus die korrekte Knocking-Kombination stammt auf den gewünschten Port zugreifen lässt – und dies wiederum ebenfalls nur für ein Zeitfenster von wenigen Sekunden. Ist die Verbindung zum Service einmal hergestellt, lässt eine stateful Firewall die bestehende Verbindung weiterhin durch.

Listing 4 zeigt eine typische Konfiguration für `/etc/knockd.conf`. Der `sequence` Parameter gibt dabei die Ports an, welche in genau der vorgegebenen Reihenfolge *geknockt* werden müssen um den Trigger `[unblockSSH]` zu aktivieren. Werden an diese Ports innerhalb 5 Sekunden, wie in `seq_timeout` definiert jeweils TCP-Pakete mit gesetztem SYN-Flag (Option `tcpflags`) gesendet, so wird das Kommando `start_command` am Remotesystem ausgeführt.

Nach Ablauf des in `cmd_timeout` definierten Zeitfensters wird das Kommando hinter `stop_command` ausgeführt. Prinzipiell finden hier alle möglichen Shellkommandos Anwendung, zur Absicherung unseres SSH Services bringen wir `iptables` einfach dazu Port 22 zu öffnen bzw. zu sperren, alles in Abhängigkeit

Listing 5. Modul `pam_usbauth.c`

```
/* pam_usbauth module, (c) 2007 Erik Sonnleitner. Licensed under GPL */
#include <stdio.h>
#include <security/pam_modules.h>
#include <security/_pam_macros.h>

#define PAM_SM_AUTH
#define USBDEV "/dev/sdb1"
#define MAX_HASH_LEN 256
#define LOCAL_VERIFY "/etc/usbauth.d/"

PAM_EXTERN int pam_sm_authenticate(pam_handle_t *pamh, int flags, int argc,
                                   const char **argv) {
    int retval;
    const char *user = NULL;
    char* hash_usb = malloc(sizeof(char) * MAX_HASH_LEN);
    char* hash_local = malloc(sizeof(char) * MAX_HASH_LEN);

    retval = pam_get_user(pamh, &user, NULL);

    char* pwoff = malloc(sizeof(char) * (strlen(LOCAL_VERIFY) + strlen(user)
                                         + 2));
    if(!pwoff || !hash_usb || !hash_local)
        return PAM_SYSTEM_ERR;

    snprintf(pwoff, strlen(LOCAL_VERIFY)+strlen(user)+1, "%s%s", LOCAL_VERIFY,
             user);

    FILE* localvrfy = fopen(pwoff, "r");
    FILE* usbdrive = fopen(USBDEV, "r");

    if(!localvrfy || !usbdrive)
        return PAM_SYSTEM_ERR;

    hash_usb = fgets(hash_usb, MAX_HASH_LEN, usbdrive);
    hash_local = fgets(hash_local, MAX_HASH_LEN, localvrfy);
    fclose(usbdrive);
    fclose(localvrfy);

    if(!strcmp(hash_usb, hash_local))
        return PAM_SUCCESS;
    else
        return PAM_AUTH_ERR;
}

PAM_EXTERN int pam_sm_setcred(pam_handle_t *pamh, int flags, int argc,
                              const char **argv) {
    return PAM_SUCCESS;
}
```

von der IP von der der Port-Knock stammt (welche via `%IP%` im Kommando angegeben werden kann und von `knockd` substituiert wird).

Um an den Ports vom Client aus beim Server anzuklopfen, können im Grunde alle TCP-Paketgeneratoren verwendet werden, sofern die richtigen TCP Flags gesetzt werden. Überaus komfortabel ist der implizit bei `knockd` mitgelieferte Client `knock` welcher nur mehr über den Zielhost und die Portsequenz informiert werden muss:

```
$ knock server 3748 2994 2338
  && ssh user@server
```

Dabei können UDP und TCP Ports auch in gemeinsam genutzt werden, sofern in `/etc/knockd.conf` entsprechend konfiguriert:

```
$ knock server 3748:tcp 2994:udp
  2338:udp && ssh user@server
```

Selbstverständlich bleibt das eigentliche Port-Knocking nicht vor Sniffen verborgen, ist bei geeigneter Konfiguration und mäßigem Netzwerkverkehr jedoch auch für Kenner ihres Faches nicht leicht aufzuspüren. Zu beachten ist, dass das Knocking-Prinzip ausschließlich eine zusätzliche Absicherung bietet und die herkömmliche Authentifikation (wie auch immer diese aussehen mag) nicht ersetzen sollte.

Authentifikation über USB-Sticks (Key-Drives)

Passwortlose Authentifikation lässt sich jedoch auch über Hardware regeln, wie etwa durch Crypto-Smartcards. Diese haben jedoch zwei Nachteile: Einerseits verwenden zahlreiche Hersteller solcher Karten proprietäre Hard- wie auch Software bei welchen der Kostenfaktor nicht zu unterschätzen ist, andererseits wird bei jedem Login ein Kartenlesegerät benötigt, was insbesondere beim Gebrauch von Laptops meist nicht überall möglich ist.

Aus diesem Grund möchte ich hier eine einfache Methode vorstellen, sich lokal über einen handelsüblichen USB-Speicher via

PAM authentifizieren zu lassen. Die Größe des USB-Sticks ist nicht entscheidend, legen Sie auf dem Datenträger lediglich eine kleine zusätzliche Partition an. Im folgenden Beispiel befindet sich das USB-Medium auf `/dev/sdb1`.

Das Prinzip ist einfach: Der USB-Stick beinhaltet den Hashwert des Passwortes. Selbiger ist in einer geschützten Datei auf der lokalen Festplatte enthalten, ähnlich zu `/etc/shadow`. Sind beide Werte identisch, erfolgt der Login, andernfalls schlägt die Authentifikation fehl.

Wir erstellen das Verzeichnis `/etc/usbauth.d/` und legen für jeden Benutzer welcher sich via USB-Stick authentifizieren darf eine Datei an,

mit dem Namen des Benutzers als Dateiname und dem Hashwert seines Passwortes als Inhalt:

```
mkdir /etc/usbauth.d/ &&
echo "password" | sha512sum |
awk '{print $1}' > /etc/usbauth.d/user
&& echo -e "\n"
>> /etc/usbauth.d/user
```

Listing 5 zeigt den Code für das PAM-Modul, welches mit

```
gcc -shared -o pam_usbauth.so
pam_usbauth.c
```

übersetzt, und die Datei `pam_usbauth.so` anschließend nach `/usr/lib/security` kopiert wird.

s/Key und OPIE

s/Key verwendet zur Generierung von Einmal-Passwörtern drei Werte (von denen lediglich einer der Geheimhaltung unterliegt):

- Die eigentliche Passwordeingabe, sowie;
- eine randomisiert erzeugte Seed-Sequenz um die Streuung der möglichen Ergebnisstrings größtmöglich zu halten, und;
- die Nummer des zu erstellenden Passwortes (i.d.R. bei 500 beginnend).

Der eingegebene Passwortstring (welcher bei der OPIE-Implementierung nicht kürzer als 10 Zeichen sein darf) durchläuft zuerst eine kryptographische Hashfunktion (MD5 oder SHA1), woraus das erste Passwort erzeugt wird. Das iterative Wiederaufrufen der Hashfunktion erzeugt nun die Basis für die Folgepasswörter bis für jedes Passwort ein Hash errechnet wurde.

Diese Hashes dienen als Grundlage für die Passwörter welche von OPIE verlangt werden, nämlich 6 kurze englische Wörter (welche beim Login übrigens nicht case sensitive sind), siehe Abbildung 1.

Abbildungs-Beschreibungen

Abbildung 1. Funktionsweise von s/Key: zeigt die Erzeugung von Passwörtern über s/Key. Das Passwort sowie die generierte Seed-Sequenz und die fortlaufende Login-Nummer werden zusammen einer Einweg-Hashfunktion unterzogen, woraus die Basis für das erste Passwort resultiert. Alle weiteren Passwörter ergeben sich durch das iterative Rehashing der erzeugten Prüfsumme, aus denen die finale 6-Wort-Sequenz errechnet wird.

Abbildung 2. Funktionsweise von PAM: PAM dient als Abstraktionsschicht zwischen Authentifikationsmethoden und den Services, welche nach Authentifikation verlangen. Dabei steuert der Service die Authentifikation rein über die PAM API, während die Art und Funktionsweise der PAM Module transparent bleibt und über die PAM SPI gesteuert wird. So können verschiedenste Services über eine einheitliches Interface alle vorhandenen Strategien zur Authentifikation bei Bedarf nutzen, und ggf. an weitere Server leiten.

Abbildung 3. Port-Knocking: Ein normaler Portscan (links) wird von der Firewall vollständig geblockt, der aktive aber hinter der Firewall nicht erreichbare Service wird nicht entdeckt. Tritt jedoch die korrekte Portknocking-Sequenz (mitte) auf, wird dies von `knockd` erkannt und die Firewall lässt den entsperren den Dienst für die Source-IP von der das Knocking stammt für ein gewisses Zeitfenster.

Die Partition des USB-Sticks wird im RAW-Format, also ohne darunterliegendem Dateisystem angesprochen. Der Hashwert des Passwortes muss nun auf den USB-Speicher übertragen werden:

```
dd if=/etc/usbauth.d/user of=/dev/sdb1
```

Zu guter letzt weisen wir PAM noch darauf hin, unser Modul `pam_usbauth.so` zur Authentikation zu verwenden, indem die Zeile `@include common-auth` auskommentiert, und stattdessen folgende zwei Zeilen eingefügt werden:

```
auth sufficient pam_usbauth.so
auth required pam_deny.so
```

Auch wenn diese Variante nur für lokale Logins Sinn macht, bieten sich zahlreiche PAM Schnittstellen an, wie beispielsweise `/etc/pam.d/su`, `/etc/pam.d/sudo`, `/etc/pam.d/login`, `/etc/pam.d/xlock`, `/etc/pam.d/kscreensaver` und `/etc/pam.d/gdm`.

Somit werden alle lokalen Loginversuche über den USB-Stick umgeleitet. Selbstverständlich ist dies auch auf alle anderen lokalen PAM-fähigen Programme anwendbar. Sollten Ihnen dennoch Ihren Key-Drive abhanden kommen, so

Im Internet...

- <http://www.kernel.org/pub/linux/libs/pam/>;
- <http://www.zeroflux.org/cgi-bin/cvstrac.cgi/knock/wiki>;
- <http://www.openssh.com/>.

Über den Autor...

Der Autor studiert Techn. Informatik in der Masterstudienrichtung Netzwerke und Computersicherheit an der JKU Linz und leitet nebenberuflich Seminare zu den Themen Linux Systemadministration und Netzwerktechnik. Seine besonderen Interessen gelten der Kryptographie und der Netzwerksicherheit. Seine Webseite: www.delta-xi.net, Kontakt: esonn@gmx.net

hat der Finder lediglich einen Hashwert in der Hand – welchen sie natürlich jederzeit ändern können, ohne Ihr eigentliches Passwort verraten zu haben.

Es sei an dieser Stelle jedoch gesagt, dass es sich bei dem hier vorgestelltem Code lediglich um eine Proof-of-Concept Implementierung handelt, und durch Mechanismen wie die Verwendung eines verschlüsselten USB-Dateisystems, der Verschlüsselung des Hashes bzw. Passwortes an sich oder der Verwendung von Public-Key-Verfahren noch wesentlich sicherer gestaltet werden könnte.

Die eigentliche Implementierung von USBAuth beinhaltet zahlreiche sicherheitsrelevante Zusatzmechanismen – die Quellen des vollständigen Moduls in aktueller Version können via SVN vom Delta Xi Server geladen werden (unter <http://usbauth.sf.net> finden sie auch passende Source Tarballs sowie Debian Pakete und Dokumentation):

```
svn co svn://delta-xi.net/
svn/pam_usbauth
```

Fazit

Dass Authentifikation immer ein gewisses Risiko beinhaltet eine Identität falsch zu verifizieren sollte jedem seriösen Administrator klar sein. Durch den Einsatz von Verschlüsselung, s/Key, Public Keys oder Knocking-Verfahren sowie lokale USB Key-Drives kann das Risiko jedoch abermals minimiert werden.

Der Benutzer an sich bleibt immer noch eine der größten Schwachstellen in Netzwerken, jedoch kann die drohende Gefahr, beispielsweise bei Verlust und/oder Diebstahl von Einmal-Passwörtern in Grenzen gehalten werden, zumal der Anwender sofort merken sollte falls ein Logintan seiner Liste *übersprungen* wurde. Und spätestens wenn auch am Clientsystem der Login aufgrund eines gestohlenen Key-Drives verwehrt bleibt, werden Sie es als Administrator als erstes erfahren (und so sollte es auch sein). ●