c't magazin für computer technik

Linux mit dem Gesicht entsperren

| 26.04.2019 07:15 Uhr Jürgen Schmidt

Linux beherrscht komfortable Anmeldefunktionen wie eine Gesichtserkennung. So richten Sie die Funktion richtig und vor allem sicher ein.

Genervt vom ewigen Eintippen meines supersicheren und deshalb natürlich superlangen und superschwer einzugebenden Passworts schielte ich etwas neidisch auf die Kollegen, die sich mit Grinsen im Gesicht an ihrem Windows-Hello anmelden. Das muss doch auch bei Linux gehen. Und in der Tat: Auch wenn es noch nicht bei den Systemeinrichtungs-Tools typischer Linuxe angekommen ist – die Grundlagen für eine komfortable und dennoch sichere Anmeldung am System sind bereits vorhanden. Man muss sie nur richtig verdrahten.

Also machte ich mich auf eine Reise in die Innereien meines Linux-Systems – einem **Ubuntu Desktop 18.04 LTS (Download) [1]**. Es kostete einiges an Zeit und Nerven. Aber am Ende hatte ich, was ich mir vorgestellt hatte: einen Linux-PC, der nach dem Stand der Technik abgesichert und trotzdem komfortabel zu benutzen ist. Wenn Sie Lust haben, begleiten Sie mich auf diese Reise durch die Tiefen der Gesichtserkennung, Pluggable Authentication Modules (PAM) und des Gnome-Desktops. Diese Einladung gilt übrigens auch, wenn Sie eine andere Linux-Distribution wie Fedora benutzen. Die Konzepte lassen sich nämlich einfach übertragen.

Ich werde in diesem und den weiteren Artikeln dieser Serie nicht nur eine konkrete, praktische Anleitung liefern, wie ich mein System konfiguriert habe. Darüber hinaus erkläre ich die Sicherheitsanforderungen, die mich zu genau diesem Kompromiss zwischen Sicherheit und Komfort geführt haben. Und schließlich erkläre ich ausreichend viel von den involvierten Konzepten und Mechanismen, dass Sie damit Ihren eigenen Kompromiss realisieren können, der vielleicht aus guten Gründen von meinem abweicht.

LINUX KOMFORTABEL ENTSPERREN

- **Linux mit dem Gesicht entsperren [2]**
- **PIN, Gesichtserkennung, zweiter Faktor: Linux-Authentifizierung [3]**
- **Linux mit komfortabler Bildschirmsperre [4]**

Dabei wird auch einiges an Hintergrundwissen zur Funktionsweise unter anderem von PAM, Hardware-Tokens und dem Gnome Desktop abfallen, ohne diese Themen dabei jedoch erschöpfend abzuhandeln. Insbesondere werde ich große Bögen um einige für dieses Projekt nicht relevante Spezialitäten machen. Im Zweifelsfall gibt es jedoch Pointer zu den entsprechenden Hintergrundinformationen.

Mehr Komfort vor dem PC

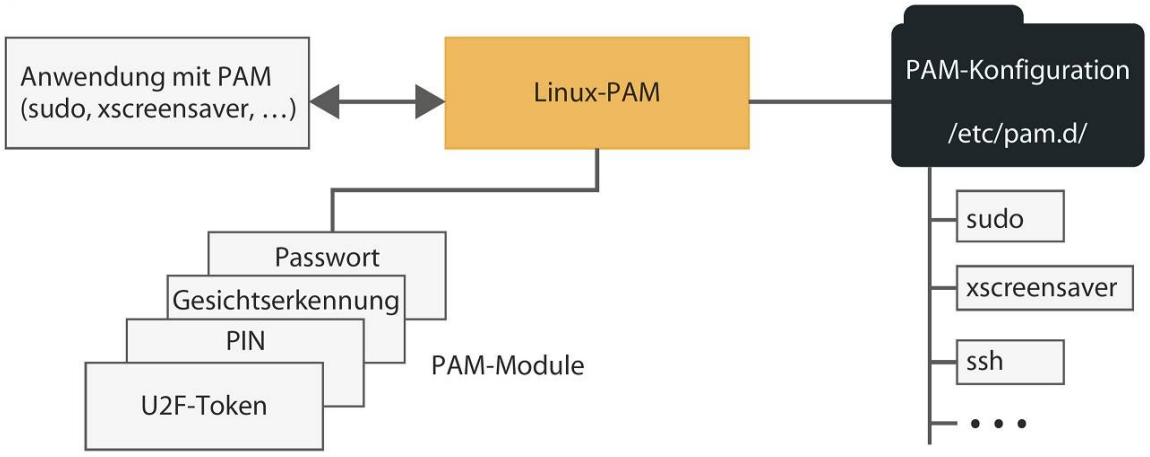
Genug der Vorrede – jetzt wird es konkret: Mein Ziel war es, dass der PC in meiner Abwesenheit – also etwa wenn ich mal zwischendurch heißes Wasser für meine Mate holen gehe – zügig gesperrt wird. Das lässt sich im Prinzip ganz einfach einstellen. Doch damit das erträglich ist, muss sich diese Bildschirmsperre möglichst einfach aufheben lassen. Schließlich muss ich dies gefühlt zwanzig- bis dreißig Mal am Tag erledigen und da will ich nicht jedes mal ein zwölfstelliges Passwort mit Sonderzeichen, Ziffern, Groß- und Kleinbuchstaben eintippen müssen.

Naheliegend war etwas mit Biometrie – etwa Gesichtserkennung. Doch ein System, das die Kollegen für etwas Schabernack **schon mit einem ausgedruckten Foto austricksen können [5]**, erschien mir nicht ausreichend sicher. Also eher eine Zweifaktor-Authentifizierung etwa mit Gesicht und einer einfach und schnell einzutippenden PIN. Das bekam ich zwar hin, aber meine am Ende umgesetzte Lösung sah dann doch noch etwas anders aus.

Was ich explizit nicht ändern wollte, war die eigentliche Anmeldung am System – etwa morgens zum Arbeitsbeginn oder nach einem Neustart. Da sollte weiterhin das zwölfstellige Passwort nötig sein. Schließlich hängt daran unter anderem die Sicherheit der Datenträgerverschlüsselung. Das könnte man zwar durchaus auch noch durch einen zweiten Faktor ergänzen – doch das ist dann ein Folgeprojekt. Hier geht es erst mal um etwas mehr Komfort, ohne dabei allzu viel Sicherheit einzubüßen.

PAM: Pluggable Authentication

Wenn man an den Anmeldemechanismen eines Linux-Systems Änderungen vornehmen will, stößt man eher früher als später auf PAM. Die Pluggable Authentication Modules sind eine raffinierte Architektur, mit deren Hilfe man im Prinzip ganz einfach etwa eine Passwortabfrage durch einen Fingerabdruckcheck ersetzen kann – und zwar ohne dass man an einer Anwendung etwas ändern muss.



Die PAM-Architektur: Wenn ein Programm für die Authentifizierung PAM benutzt kann man mit wenigen Handgriffen eine

Passwort-Abfrage durch Gesichtserkennung ersetzen. Dazu muss man nur die Konfiguration /etc/pam.d/ anpassen.

Die PAM-Konfiguration erfolgt im Verzeichnis /etc/pam.d/. Dort legen Anwendungen wie `sudo` und `ssh` jeweils in einer eigenen Konfig-Datei fest, wie der Anwender seine Berechtigung zur Nutzung der jeweiligen Dienste nachzuweisen hat. Den Grundstein legen Dateien, deren Namen mit `common` beginnt. Bei `common-password` geht es um das Ändern des Passworts, `common-session` ist die Sitzungsverwaltung und `common-auth` legt den Grundstein für eine normale Authentifizierung. Die meisten Dienstprogramme binden die common-Dateien ein

```
@include common-auth
```

und ergänzen diese dann bei Bedarf durch eigene Vorgaben für `auth`, `session` und `password`. Änderungen an den common-Dateien wirken sich also potenziell auf alle Programme des Systems aus, was man beim Basteln tunlichst vermeiden sollte. Stattdessen sollte man immer die Konfigurationsdateien einzelner Dienste anpassen.

Der für dieses Projekt relevante Teil ist die Authentifizierung. Die erfolgt standardmäßig gemäß `common-auth` durch die Eingabe eines Passworts. Doch wie man sieht, ist die Datei etwas komplizierter als man erwartet:

```
auth     [success=1 default=ignore]     pam_unix.so nullok_secure
auth     requisite                      pam_deny.so
auth     required                       pam_permit.so
```

Die erste Spalte beschreibt den Zweck, in diesem Fall also die Authentifizierung. Die zweite die Relevanz für das jeweilige Ansinnen – also etwa "required". Die dritte Spalte legt das eingesetzte PAM-Modul fest und alles was danach kommt, sind dessen Parameter. Die Modulnamen enden dabei immer auf `.so`, weil es sich um Bibliotheken handeln muss, die jedes Programm einbinden kann.

[6]

Flexibles Regelwerk

Diese PAM-Regeln werden von oben nach unten der Reihe nach abgearbeitet, außer eine der Regeln ändert dies explizit. So bedeutet das `success=1`, dass im Erfolgsfall die nächste Regel übersprungen werden soll.

`pam_unix.so` ist der reguläre Unix-Passwortcheck, die Option `nullok_secure` bedeutet, dass hier auch leere Passwörter akzeptiert werden dürfen, wenn das aktuelle Terminal als sicher gelistet ist (siehe `/etc/securetty`).

Gibt der Anwender also bei der durch `pam_unix` initiierten Passwortabfrage das richtige Passwort ein, springt PAM direkt zu `pam_permit.so` in der dritten Zeile, was den Zugang gestattet. In allen anderen Fällen kommt die zweite Regel zur Anwendung und `pam_deny` verweigert den Zugang final. Das sieht unnötig kompliziert aus, ermöglicht aber sehr flexible Regelwerke wie das später vorgestellte.

Die Schlüsselwörter `required` und `requisite` signalisieren, dass die beiden Module für den Erfolg des Vorgangs erfolgreich abgeschlossen werden müssen – und zwar zwingend. Bei `required` arbeitet PAM auch bei einem Fehlschlag trotzdem noch die folgenden Regeln ab; nach einem fehlgeschlagenen `requisite` bricht PAM hingegen sofort ab. Module mit `optional` beeinflussen den Ausgang der Authentifizierung nicht. Sie sind dazu da, beispielsweise mit dem für den Login eingegebenen Passwort auch gleich ein verschlüsseltes Laufwerk einzubinden.

Erste Schritte

Für erste eigene PAM-Experimente sollte man nicht gleich mit dem Screensaver oder gar dem Login anfangen. Denn mit etwas Pech sperren Sie sich dabei aus dem System aus. Besser ist es, neue Regelwerke mit `sudo` zu testen. Folgendes hat sich bei meinen Basteleien bewährt: Zunächst öffnet man ein neues Terminalfenster und startet mit `sudo -i` eine Root-Shell. Die sollten Sie nicht schließen, bis Sie fertig sind – sie stellt sicher, dass Sie Änderungen rückgängig machen können.

Dann legen Sie von jeder Datei, die Sie ändern wollen, zunächst ein Backup an:

```
cd /etc/pam.d
cp sudo sudo-org
```

Jetzt können Sie versuchsweise mal die Passwortabfrage abschalten. Das geht ganz einfach: In der Datei `sudo` die Zeile mit `common-auth` durch ein vorangestelltes # auskommentieren und dahinter eine äußerst großzügige Regel einfügen:

```
# @include common-auth
auth required pam_permit.so
```

Es lohnt sich übrigens, dies mit vi zu machen, weil dessen Syntax-Highlighting signalisiert, ob man eine gültige Zeile produziert hat. Einen Verschreiber wie `reqired` quittiert vi sofort mit dem Entfernen der Farben.

PROBLEME MIT PAM?

Beim Basteln an der Authentifizierung kann es durchaus zu Problemen kommen. Ubuntu protokolliert alle Authentifizierungsvorgänge in `/var/log/auth.log`; auf Systemen mit Systemd kann man das mit `journalctl -f` live verfolgen und sieht dann Einträge wie:

```
pam_userdb(sudo:auth): user 'ju' granted access
```

Hier sieht man auch fehlgeschlagene Anfragen. Die Log-Einträge verraten das verantwortliche PAM-Modul (`pam_userdb`), den aufrufenden Prozess (`sudo`), die Art der Anfrage (`auth`) und manchmal sogar einen Grund, warum der PAM-Check nicht funktioniert hat ("incorrect password"). Viele PAM-Module lassen sich über eine debug-Option auch zu ausführlicheren Meldungen überreden (siehe etwa `man pam_userdb`).

Beim Entwickeln komplexerer PAM-Regularien hilft das Tool `pamtester`, das man via `apt` installieren kann. Dazu legt man in /etc/pam.d/ eine Testdatei `test1` mit PAM-Anweisungen an. Die arbeitet dann beispielsweise

```
pamtester -v test1 ju authenticate
```

ab. Das startet eine Authentifizierung für den Benutzer "ju" gemäß der in `test1` spezifizierten Regeln und gibt dazu Statusinformationen aus. Das abschließende Ergebnis ist dann entweder "Authentication failure" oder "successfully authenticated".

Ob die Änderung Erfolg hatte, kann man dann in einem anderen Terminalfenster mit normalen Anwenderrechten testen. Der Befehl

```
sudo -k id
```

fragt normalerweise nach einem Passwort. Das `-k` entfernt dazu vorhandene Auth-Tokens, die dafür verantwortlich sind, dass man nach einem erfolgreichen `sudo` ein paar Minuten ohne weitere Nachfrage als Root werkeln darf. Mit der neu erstellten PAM-Konfiguration entfällt die Passwortabfrage jedoch ganz. `sudo` ohne Passwort ist zwar superbequem, aber auch ein riesiges Sicherheitsloch. Da das nicht so bleiben darf, stellen Sie anschließend sofort im Root-Fenster den Originalzustand wieder her:

```
cp sudo-org sudo
```

Gratulation – damit sind Sie bereits gut gerüstet für anspruchsvollere Aufgaben wie die anvisierte Zweifaktor-Authentifizierung. Deren Umsetzung ist eigentlich ganz einfach. Denn schon ein

```
auth required pam_faktor1.so
auth required pam_faktor2.so
```

fordert zwingend zwei erfolgreiche Authentifizierungs-Schritte. Eine Reihe von PAM-Modulen, die die Rolle von `faktor1/2` übernehmen können, **zeigt ein anderer Artikel [7]** . Das könnte dann etwa die Eingabe einer kurzen PIN gepaart mit einer erfolgreichen Gesichtserkennung sein.

Gesichtserkennung und PIN einrichten

Das Ziel waren jedoch noch differenziertere Anmeldemöglichkeiten. Zum Start des Systems und zur ersten Anmeldung sollte man zwingend das lange und sichere Passwort eingeben müssen. Für das Entsperren nach einer kurzen Kaffeepause oder auch ein `sudo` im laufenden Betrieb genügt dann eine komfortable Authentifizierung – also etwa eine erfolgreiche Gesichtserkennung gepaart mit der Eingabe einer kurzen PIN oder die Aktivierung des Hardware-Tokens. Doch das Passwort soll nach wie vor als Fallback in allen Szenarien funktionieren – also etwa, wenn man den Büroschlüssel mit dem daranhängenden Token zu Hause vergessen hat.

Oder wenn man sich aus der Ferne via SSH anmeldet und weder Gesichtserkennung noch lokal am System zu aktivierende U2F-Tokens funktionieren.

Das folgende Regelwerk setzt diese Vorgaben ganz konkret um:

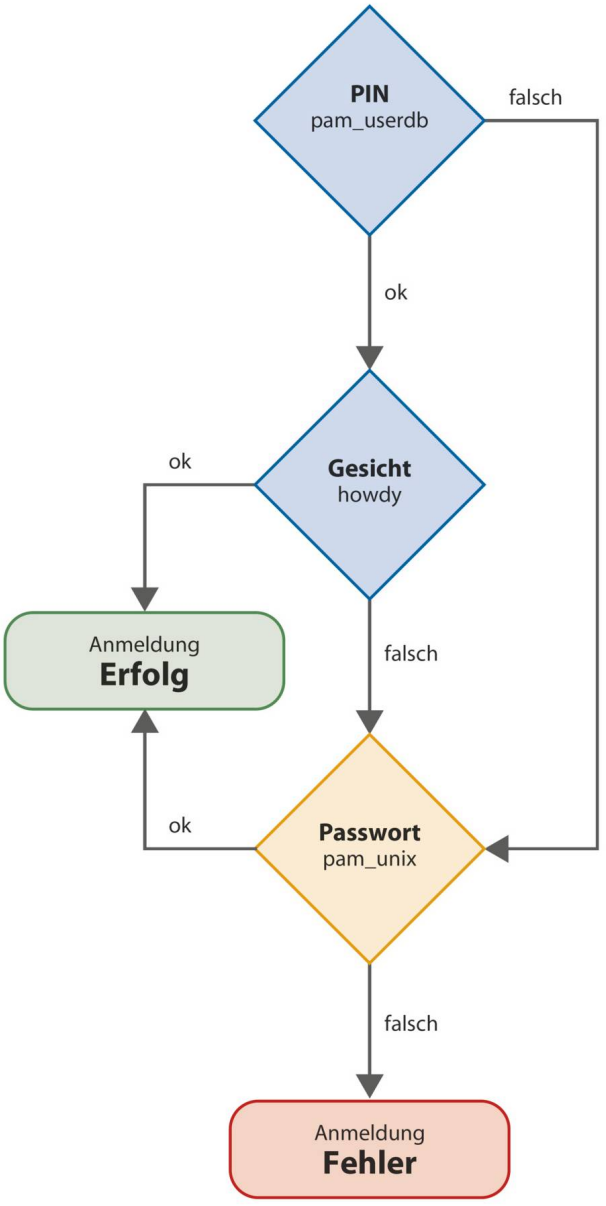
```
# erster Faktor: PIN
auth     [success=ignore default=1]     pam_userdb.so crypt=crypt db=/etc/pinlist
# zweiter Faktor: Gesichtserkennung
auth     [success=2 default=ignore]     pam_python.so /lib/security/howdy/pam.py
# Fallback -- funktioniert immer: Passwort
auth     [success=1 default=ignore]     pam_unix.so try_first_pass
# FAIL
auth     requisite                      pam_deny.so
# SUCCESSS
auth     required                       pam_permit.so
```

Sie können das einfach glauben **und im nächsten Abschnitt weiterlesen [8]** oder es mit der folgenden Erklärung Schritt für Schritt nachvollziehen.

Mit diesen PAM-Regeln bedeuten Angaben wie `=2`: "überspringe die nächsten 2 PAM-Module". Die erste fragt ein Passwort ab und `pam_userdb` vergleicht es mit den in `pinlist.db` hinterlegten Authentifizierungsdaten.

Hat das Erfolg, bedeutet das noch keine erfolgreiche Anmeldung. Das auf den ersten Blick etwas seltsame `success=ignore` sorgt dafür, dass der Anwender noch den zweiten Faktor vorweisen muss: `pam_python` respektive `pam.py` aus dem PAM-Projekt Howdy checken das Gesicht. Erst wenn das passt, meldet PAM Erfolg, denn `success=2` springt zu `pam_permit`. **Die Einrichtung dieser Module beschreibt der Folgeartikel [9]** .

Ist der erste Passwort-Test durch `pam_userdb` nicht von Erfolg gekrönt, überspringt PAM die Gesichtserkennung (`default=1`) und fährt gleich mit dem Unix-Passwort-Check fort. Dort landet man auch, wenn Howdy das Gesicht nicht mag. `pam_unix.so` testet wegen der Option `try_first_pass` zunächst das zu Beginn bereits eingegangene Passwort gegen das Login-Passwort. Ansonsten erfragt es ein neues. Bei Erfolg gewährt es den Zugang; ein Fehlschlag des Passwort-Checks bedeutet hingegen eine finale Ablehnung durch `pam_deny` .



PAM-Komfort mit Fallback.

[10]Konzept anpassen

Selbstverständlich kann man dieses Konzept auch modifizieren. Wer ein Notebook sein Eigen nennt, dessen Fingerabdruck-Scanner unter **Linux [11]** funktioniert, kann den mit `pam_fprintd` einsetzen. Ich habe etwa derzeit statt der PIN-Abfrage via `pam_userdb.so` ein U2F-Token im Einsatz. Dazu habe ich einfach `pam_userdb.so …` durch `pam_u2f.so` ersetzt.

Da sich das U2F-Verfahren anders als Passwörter oder Gesichtserkennung nach derzeitigem Kenntnisstand nicht mit realistischen Aufwand austricksen lässt – sprich: ohne Token kein Zugang – kann man es auch guten Gewissens als alleinige Authentifizierung etwa für `sudo` einrichten. Dazu kommentiert man die Zeile für den ersten Faktor aus und richtet in der zweiten Zeile `pam_u2f` ein. Wer dann das Token klaut, muss ja immer noch einen Kommandozeilen-Prompt ergattern, um `sudo`-Kommandos abzusetzen. Und dazu muss er an Login oder Bildschirmsperre vorbei.

Als Ausgangspunkt für eigene Experimente haben wir diese und einige weitere, **kommentierte Beispiel-Konfigurationen auf GitHub zusammengetragen [12]**, die Sie als Ersatz für `common-auth` in der PAM-Konfig-Datei `/etc/pam.d/sudo` einbinden können. Das Ganze funktioniert aber natürlich nicht nur mit `sudo`, sondern auch mit beliebigen Programmen, die eine Authentifizierung via PAM abwickeln. Das PolicyKit moderner Linux-Systeme beispielsweise setzt ebenfalls auf PAM auf und gehorcht dabei den in `/etc/pam.d/polkit-1` vorgegebenen Regeln. Testen kann man das einfach via `pkexec id`.

Damit bleibt eigentlich nur noch, der Bildschirmsperre den komfortablen Zugang mit Passwort-Fallback beizubringen. Doch bis das so funktioniert, wie man es haben will, sind einige weitere kleine Ausflüge unter anderem in die Gnome-Desktop-Konfiguration, **Ubuntus [13]**Hotkey-Verwaltung und das Linux-Geräte-Management via `udev` erforderlich.

Wie die Bildschirmsperre konfiguriert wird, **zeigt ein anderer Artikel [14]** . (**ju [15]**)

URL dieses Artikels:

`https://www.heise.de/-4404861`

Links in diesem Artikel:

[1] https://www.heise.de/download/product/ubuntu-22040/download

[2] https://www.heise.de/ratgeber/Linux-mit-dem-Gesicht-entsperren-4404861.html

[3] https://www.heise.de/ratgeber/PIN-Gesichtserkennung-zweiter-Faktor-Komfortable-Linux-Authentifizierung-4404929.html

[4] https://www.heise.de/ratgeber/Linux-komfortabel-und-sicher-entsperren-4418407.html

[5] https://www.heise.de/meldung/Face-Unlock-42-von-110-Handys-lassen-sich-mit-Portrait-Fotos-austricksen-4269897.html

[6] https://www.heise.de/ct/

[7] https://www.heise.de/ratgeber/PIN-Gesichtserkennung-zweiter-Faktor-Komfortable-Linux-Authentifizierung-4404929.html

[8]

[9] https://www.heise.de/ratgeber/PIN-Gesichtserkennung-zweiter-Faktor-Komfortable-Linux-Authentifizierung-4404929.html

[10]

[11] https://www.heise.de/thema/Linux-und-Open-Source

[12] https://github.com/ju916/pam-demos

[13] https://www.heise.de/thema/Ubuntu

[14] https://www.heise.de/ratgeber/Linux-komfortabel-und-sicher-entsperren-4418407.html

[15] mailto:ju@ct.de