c't magazin für computer technik

PIN, Gesichtserkennung, zweiter Faktor: komfortable Linux-Authentifizierung

| 26.04.2019 09:21 Uhr Jürgen Schmidt

Noch regelt das Passwort den Zugang zum Linux-Desktop. Doch mit passenden PAM kann man seine Berechtigung mit anderen Möglichkeiten nachweisen.

Fertige Pluggable Authentications Modules (PAM) für Linux gibt es zuhauf, die Zusatzfunktionen für die Authentifizierung bereitstellen. Bei Ubuntu zeigt der Befehl

```
apt search libpam
```

eine ganze Reihe, die direkt zur Installation bereitstehen. Weitere finden sich in diversen Entwickler-Repositories etwa auf GitHub. Man kann auch einfach mal in /lib/security/ beziehungsweise /lib/x86_64-linux-gnu/security/ stöbern. Dort liegen die jeweiligen .so-Dateien.

Zu beachten ist, dass manche PAM-Module nicht zur Authentifizierung gedacht sind, sondern diese um Zusatzfunktionen erweitern. So kann man etwa mit `pam_cracklib` neue **Passwörter [1]** gegen eine Liste beliebter aber schwacher Kennwörter abgleichen und diese dann ablehnen. `pam_ecryptfs` bindet nach erfolgreicher Authentifizierung mit dem Passwort auch gleich noch eCryptfs-verschlüsselte Home-Verzeichnisse ein. Und mit `pam_time` kann man bestimmte, authentifizierungspflichtige Vorgänge auf einen festgelegten Zeitraum beschränken und etwa einzelnen Anwendern den Login außerhalb der Arbeitszeiten untersagen (was aber keine existierenden Sitzungen beendet). Erste Details zum Einsatz liefert in aller Regel schon ein Aufruf von `man pam_XXXX`.

Das Standard-Modul für die Passwort-Eingabe ist `pam_unix`. Auf manchen Linux-Notebooks funktioniert `pam_fprintd` mit dem eingebauten Fingerabdruck-Sensor. Ich hatte mit den für diese Artikel ausprobierten externen USB-Sensoren kein Glück, sodass diese hier nicht weiter vorkommen. Man kann das Modul analog zu den folgenden einsetzen.

PIN statt Passwort

Wenn man für das Entsperren nicht das Unix-Passwort verwenden, sondern etwa einen kurzen PIN-Code einsetzen will, braucht man eine Alternative zu `pam_unix`, das fest mit den in /etc/shadow hinterlegten Passwörtern verdrahtet ist. Vorhang auf für `pam_userdb`. Das stammt aus Unix-Urzeit, ist nur schlecht dokumentiert und etwas sperrig in der Handhabung. Aber es tut, was man will, wenn man etwas Arbeit investiert. Die Grundlage für die Authentifizierung ist eine Datenbankdatei im Berkeley-DB-Format, die Kombinationen aus Benutzernamen und optional verschlüsselten Passwörtern enthält. Für ihre Erstellung muss man spezielle Tools installieren (`sudo apt install db-util`).

Meine Tests zeigten, dass zumindest die bei **Ubuntu 18.04 LTS (Download) [5]** mitgelieferte Version auch bereits mit gesalzenen SHA-512-Hashes umgehen kann, wie sie auch in der regulären Unix-Authentifizierung via /etc/shadow zum Einsatz kommen. Für deren Erzeugung muss man auch noch das Paket "whois" installieren, das – aus welchen Gründen auch immer – das Utility `mkpasswd` beheimatet.

Damit liegen die PIN-Codes nur verschlüsselt auf der Festplatte. Man muss sich jedoch darüber im Klaren sein, dass bei vierstelligen PINs mit lediglich zehntausend Variationen das stumpfe Durchprobieren nicht sonderlich lange dauert. Also sollte man die Zugriffsrechte auf die Datei möglichst restriktiv setzen.

Zunächst legt man also als Root eine PIN-Datei an:

```
USR=ju
HASH=`mkpasswd -s -m sha-512`
{ echo $USR; echo $HASH; } | db_load -T -t hash /etc/pinlist.db
chmod 0600 /etc/pinlist.db
```

Das fragt nach einem Passwort, das dabei auch angezeigt wird, und legt die PIN-Datenbank in /etc/pinlist.db an. Der abschließende `chmod`-Befehl sorgt dafür, dass nur Root die Datei lesen darf. Durch wiederholte Aufrufe dieser Kommandozeilensequenz kann man weitere Benutzer hinzufügen. **Wer das öfter erledigt, bastelt sich daraus ein kleines Skript [6]**. Da der Benutzername als Index fungiert, kann man die Einträge auch aktualisieren. Der Befehl `db_dump -p pinlist.db` zeigt den aktuellen Inhalt an. Ein PAM-Eintrag wie

```
auth required pam_userdb.so crypt=crypt db=/etc/pinlist
```

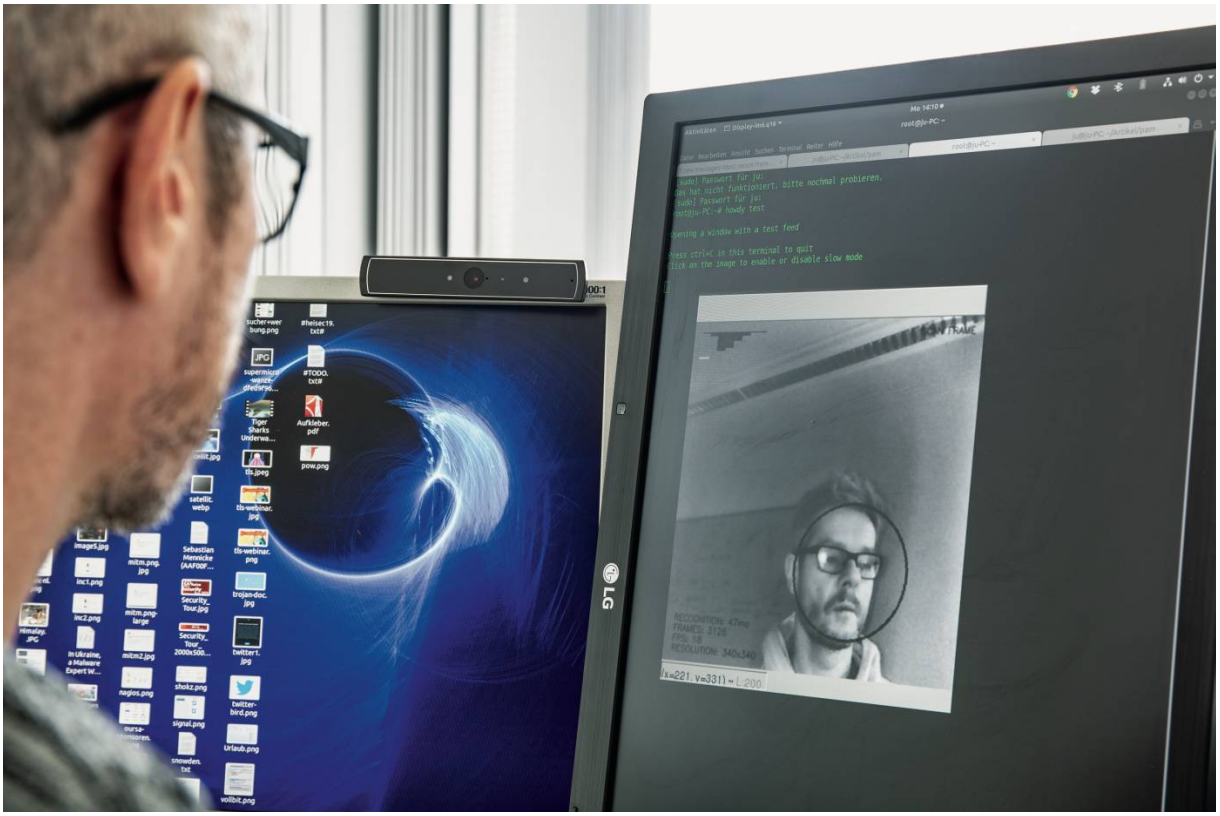
sorgt für eine Authentifizierung gegen die neue PIN-Datenbank. Etwas unkonventionell und schlecht dokumentiert ist, dass man bei der Angabe der Datenbankdatei die Dateiendung `.db` weglassen muss. Alles in allem ist `pam_userdb` ein ausgereiftes Modul, das man ohne große Bedenken einsetzen kann, wenn man eine Passwort- respektive PIN-basierte Authentifizierung realisieren will. Auf Github findet sich übrigens auch ein PAM-Modul für eine SQLite-Datenbank. Das kann jedoch nur veraltete Krypto-Verfahren wie DES und MD5, weshalb ich es nicht weiter in Betracht gezogen habe.

Alternativ kann man sich auch gegen eine **MySQL [7]**-Datenbank authentifizieren. Dazu übergibt man `pam_mysql.so` die Koordinaten der Datenbank als Parameter (`host, user, passwd, db, table, usercolumn, passwdcolumn, crypt`). Natürlich benötigt man unter dieser Adresse dann auch einen MySQL-Server, der die Login-Daten bereitstellt. Das mag in Firmennetzen sinnvoll sein; für einen einzelnen PC ist es Overkill. Deshalb soll hier ein **kurzer Verweis auf ein Howto genügen [8]**.

[9]

Gesichtserkennung mit Howdy

Biometrische Authentifizierung hat unschätzbare Vorteile: Sein Gesicht oder seinen Daumen hat man immer dabei. Sie vorzuzeigen ist im Vergleich zur Eingabe langer Passwörter sehr komfortabel. Die Nachteile sind nicht ganz so offensichtlich: Fingerabdrucksensoren, Gesichtserkennung und andere biometrische Systeme ließen sich bislang von findigen Hackern immer austricksen. Und wenn mein Passwort kompromittiert ist, wähle ich ein neues; mit Gesicht oder Fingern geht das nicht so leicht.



Mit Howdy und einer passenden Kamera kann man komfortable Gesichtserkennung für Linux nachrüsten.

Das spricht dafür, biometrische Merkmale nicht für höchste Sicherheitsansprüche zu verwenden, sondern eher als Komfortfunktion in eng begrenzten Szenarien. Also etwa zum Entsperren des PCs – ähnlich wie bei Smartphones das Auflegen des Fingers oder ein Blick in die Kamera genügt, das Gerät zu entriegeln. Nach jedem Neustart ist jedoch trotzdem ein Passwort respektive Code einzugeben.

Das Projekt Howdy ermöglicht Linux-Anwendern ein Entsperren nach dem Vorbild von Microsofts Windows Hello mit Gesichtserkennung. Dazu nutzt es die Infrarot-Bilder von Windows-Hello-kompatiblen Kameras und analysiert diese mit Hilfe der Machine-Learning-Bibliothek DLib. Die eingesetzten **Modelle wurden als Forschungsprojekte zur Gesichtserkennung auf GitHub veröffentlicht [10]**.

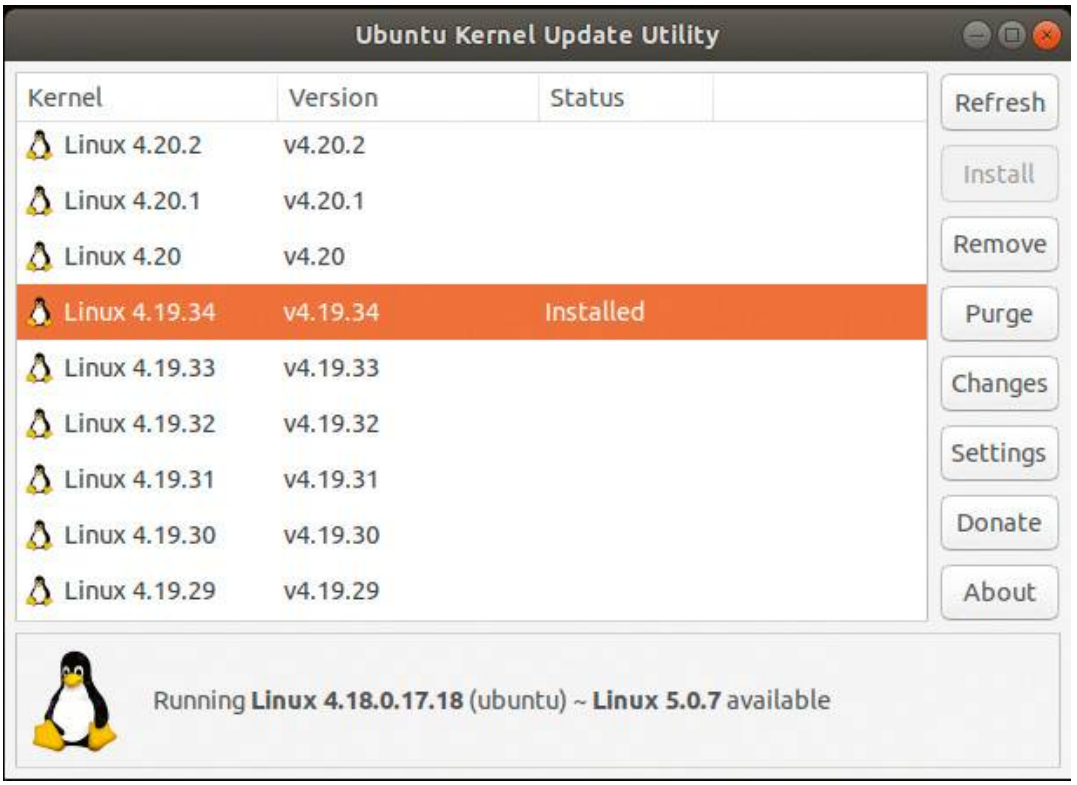
Keine Windows-Qualität

Auch wenn Howdy meine einfachen Funktionstests anstandslos absolvierte, sollte man nicht davon ausgehen, dass es an die Qualität der Windows-Gesichtserkennung heranreicht. Selbst der Howdy-Autor warnt klar und deutlich: "Dieses Paket ist in keiner Weise so sicher wie ein Passwort und wird es nie sein. … Schon eine ähnlich aussehende Person oder ein gutes Foto könnte es austricksen."

Hinzu kommt, dass Howdy sehr hemdsärmelig an Probleme herangeht und dabei durchaus Ärger verursachen kann. Es ist in Python geschrieben und nutzt als Brücke `pam_python.so`. Dessen Autoren erklären, dies sei "schon aus Performancegründen nicht die beste Option für Module, die weite Verbreitung finden". Explizite Sicherheits-Reviews gab es nach Auskunft der Howdy-Autors bisher nicht. Es ist also nicht auszuschließen, dass der Einsatz auch zu Sicherheitsproblemen etwa in Form von Privilege-Escalation-Angriffen führt – schließlich läuft der Python-Code mit Root-Rechten.

Howdy sollte mit allen Windows-Hello-kompatiblen Kameras funktionieren. Die sind in vielen Notebooks eingebaut; als externe USB-Cam gibt es sie ab circa 70 Euro. Die Kameras haben zwei Modi: Beim normalen Video-Streaming leuchtet typischerweise ein kleines blaues Lämpchen auf. Eine rote LED signalisiert hingegen, dass gerade die Infrarot-Kamera aktiv ist.

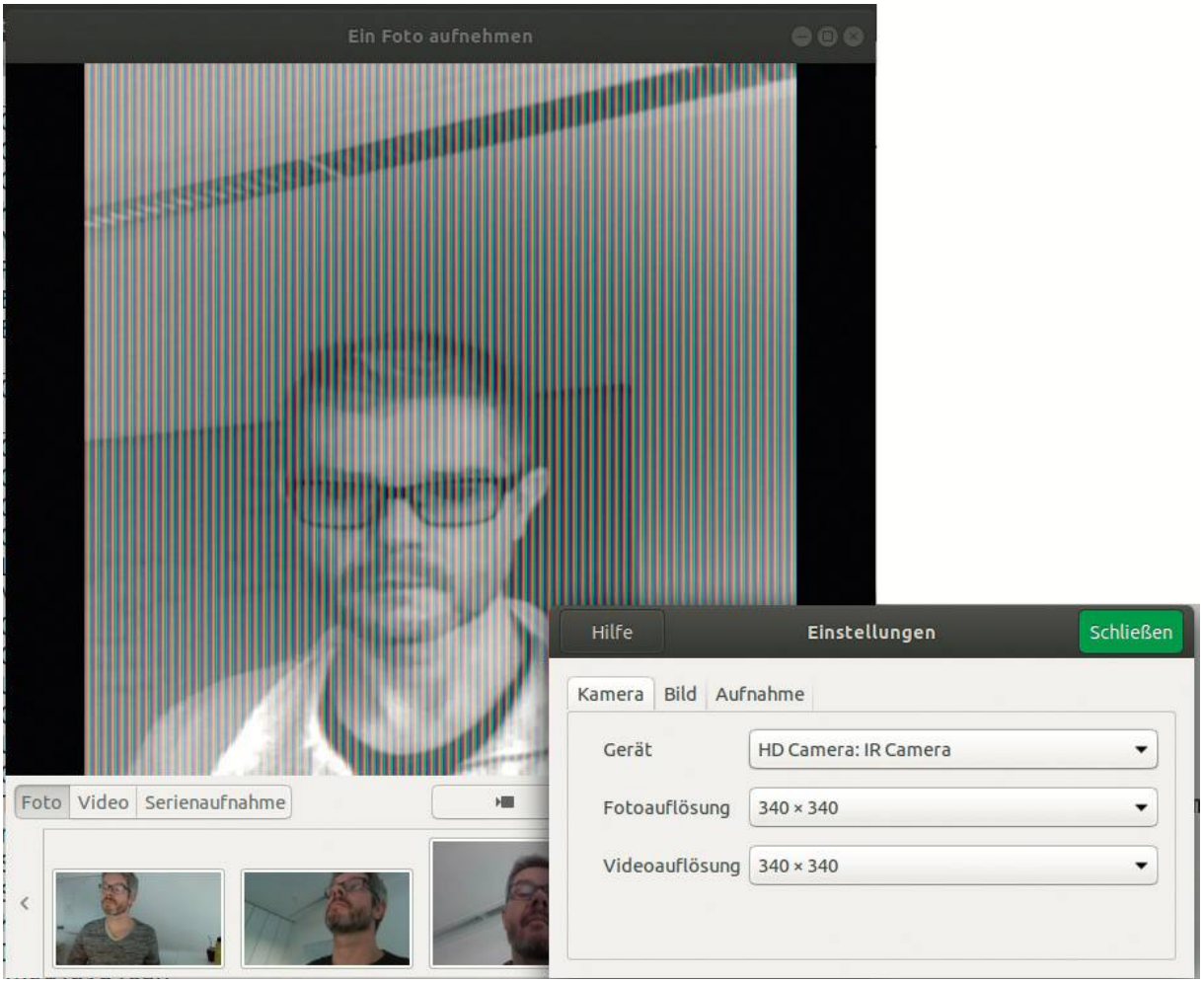
Berichten zufolge funktionieren einige Hello-kompatible Kameras unter Linux out of the box. Bei dem von mir gekauften No-Name-Produkt war das jedoch nicht der Fall. Die Kamera lieferte im System-Log Fehlermeldungen (uvcvideo: Unknown video format). Konkret funktionierte sie zwar als Webcam, aber der Infrarotbetrieb scheiterte an einem speziellen 8-Bit-Videoformat von Microsoft, das der Linux-Kernel erst mit Version 4.19 eingeführt hat. Ubuntu liefert für 18.04 standardmäßig nur Kernel bis maximal 4.18. Abhilfe schaffte die Installation des Mainline-Kernels 4.19, was mit dem Utility `ukuu` recht einfach vonstatten ging. Trotzdem ist es sicher nichts für jeden, sich vom offiziellen Ubuntu-Kernel zu verabschieden und dann bei Problemen und Updates weitgehend auf sich selbst gestellt zu sein.



Nichts für jederman: Mit dem Utility ukuu kann man unter Ubuntu auch Mainline-Kernel installieren.

Howdy installieren

Bevor etwa die einfache Webcam-App `cheese` nicht in den Einstellungen eine zweite Kamera anzeigt, die dann ein streifiges Graustufen-Bild liefert, braucht man sich nicht an der Installation von Howdy zu versuchen. Die verläuft dann jedoch **gemäß der Anleitung auf der Projektseite problemlos [11]**; Howdy findet sogar selbst das richtige Device der IR-Kamera.



Windows-Hello-kompatible Kameras nutzen ein Infrarot-Bild, das in Graustufen dargestellt wird.

Die Gesichtserkennung Howdy trägt sich als allein ausreichendes Anmeldeverfahren in `/etc/pam.d/common-auth` ein:

```
auth [...] pam_python.so /lib/security/howdy/pam.py
```

Ich rate sehr dazu, das umgehend zu ändern und durch ein vorangestelltes Kommentarzeichen '#' abzuschalten. Wie man zu einer komfortablen und sicheren **PAM-Konfiguration [12]** mit Howdy kommt, beschreibt **ein anderer Artikel [13]** . Nach der Installation kann man mit `sudo howdy test` überprüfen, ob ein Bild erscheint. Bleibt das Fenster leer, hilft es eventuell, die X11-Shared-Memory-Nutzung für das Toolkit QT abzuschalten:

```
QT_X11_NO_MITSHM=1 howdy test
```

Erscheint damit das angeforderte Testbild, empfiehlt der Howdy-Entwickler, das `QT_X11_NO_MITSHM=1` in /etc/environment systemweit zu fixieren. Wegen der möglichen Nebenwirkungen wäre jedoch eine weniger rabiate Lösung vorzuziehen. Anschließend muss man Howdy noch das eigene Gesicht beibringen:

```
sudo howdy add
```

Es hat sich bewährt, das bei verschiedenen Lichtverhältnissen zu wiederholen, um die Erkennung zu verbessern.

Als Brillenträger kann man auch noch ein Bild ohne Brille hinterlegen. `howdy list` zeigt die aktuell gelernten Gesichter und welchem Benutzer sie jeweils zugeordnet sind.

Howdy befindet sich noch in der Entwicklung; die Community ist rege und der Autor antwortet auf Fehlerberichte sehr zügig. Insgesamt ist Howdy eine coole Spielerei mit hohem Nerd-Faktor. Für Produktionssysteme würde ich es im aktuellen Zustand eher nicht empfehlen.

Zweiter Faktor

Sein Handy hat man eigentlich immer in der Nähe. Da liegt es nahe, den erfolgreichen Login von dessen Sichtbarkeit abhängig zu machen. Das Modul `pam_beacon` erlaubt es, Bluetooth-Adressen festzulegen, die zum Smartphone oder einem speziellen Beacon am Schlüsselbund gehören und beim Login in Funkreichweite sein müssen. Leider gab es bei meinen Tests nur Abstürze, die der Entwickler mit dem Hinweis kommentierte, dass er wohl mal die D-Bus-Kommunikation unter Ubuntu genauer anschauen müsse. Bisher ist das jedoch offenbar nicht geschehen. Schade.

Login per U2F

Universal Two Factor Authentifizierung (U2F) ist ein Standard für Hardware-Token zum sicheren Identitätsnachweis. Das U2F-Konzept ist eigentlich für die Anmeldung bei Web-Diensten gedacht und könnte Problemen wie Phishing und massenhaftem Passwort-Diebstahl auf Servern endlich ein Ende bereiten. Der Nachfolger FIDO2 hat das Potential, im Zusammenspiel mit WebAuthn tatsächlich in nicht allzu ferner Zukunft Passwörter überflüssig zu machen. Doch das ist Stoff für einen anderen Artikel.

U2F-Tokens gibt es ab etwa 10 Euro in verschiedenen Formen und Ausführungen.

Hier nur so viel: U2F funktioniert mit einem ausgeklügelten Challenge/Response-Verfahren und Dienst-spezifischen Public/Private-Schlüsselpaaren. Dabei zeigt das Token durch Blinken an, dass es eine Challenge erhalten hat. Doch erst wenn der Anwender einen Knopf oder Taster betätigt, antwortet es mit der Response, die den Zugang freischaltet. Die eingesetzten Schlüssel beruhen dabei auf einem Geheimnis, das physisch mit dem Token gekoppelt ist und sich nicht auslesen lässt.

U2F-kompatible Token gibt es ab circa 10 Euro; es empfiehlt sich, solche zu kaufen, die auch bereits FIDO2 können. Die Token sind äußerst robust und überleben auch eine mehrjährige Nutzung am Schlüsselbund. Standard ist ein USB-Connector. Manche Modelle können jedoch auch bereits NFC oder Bluetooth, was ich jedoch bisher nicht getestet habe.

Linux unterstützt U2F via `libpam_u2f`. Dieses PAM-Modul hat der renommierte Token-Hersteller Yubico entwickelt. Es funktionierte in meinen Tests auch mit Token anderer Marken problemlos. Nach der Installation von `libpam-u2f` via `apt` muss man lediglich dem System beibringen, welches Token zu welchem Anwender gehört. Dazu erzeugt ein Hilfsprogramm die passende Konfigurationszeile:

```
pamu2fcfg > ~/.config/Yubico/u2f_keys
```

Das Verzeichnis `Yubico` muss man dazu in aller Regel erst anlegen. `u2f_keys` kann die Einträge mehrerer Keys aufnehmen, die dann alle funktionieren. Für U2F-Kenner: Standardmäßig registriert sich PAM dabei für die Domain und App-ID `pam://$HOSTNAME`.

Ein PAM-Anweisung wie

```
auth required pam_u2f.so
```

erzwingt den U2F-Check. Für erste Tests kann man das etwa in `/etc/pam.d/sudo` einfügen, **wie bereits in einem anderen Artikel beschrieben [14]** .

Das Handling der U2F-Token ist sehr komfortabel. Verursacht etwa ein `sudo`-Aufruf eine Authentifizierungs-Anfrage, blinkt das Token dezent und signalisiert damit seine Bereitschaft, die Challenge des Systems zu beantworten. Erst wenn man es antippt, liefert es die angefragten Authentifizierungsdaten. Das Modul ist so implementiert, dass es zunächst checkt, ob überhaupt ein passendes Token für den zu autorisierenden Nutzer vorhanden ist. Wenn nicht, schlägt der PAM-Aufruf sofort fehl; es kommt also nicht erst zu lästigen Timeouts, wenn kein Token vorhanden ist.

Wenn eine Authentifizierungsanfrage vorliegt, leuchtet das Token. Erst wenn der Anwender es antippt, erfolgt die Anmeldung.

U2F ist sehr robust und sicher; man kann es guten Gewissens als alleinige Authentifizierungsmethode einsetzen. Dann darf man sich halt das Token am Schlüsselbund nicht klauen lassen. Aber den Trojaner, der das kann, muss erst jemand erfinden. Das Handicap von U2F ist, dass man eigentlich immer zwei dieser Token haben und auch bei jedem Dienst registrieren muss. Denn das Authentifizierungskonzept beruht auf einem im Token abgelegten Geheimnis, das man nicht auslesen kann; auch nicht für Backups.

Verliert man das Token, ist auch das Geheimnis weg – und der damit gekoppelte Zugang zum Dienst beziehungsweise System ebenfalls. Dann hilft nur noch das im Safe hinterlegte und vorsorglich eingerichtete Backup-Token. **Das im anderen Artikel vorgestellte Szenario umgeht dieses Problem [15]** sehr elegant: Dort funktioniert immer noch die Anmeldung mit dem Passwort als Fallback – auch ohne Token. In Kombination mit einem weiteren Faktor wie PIN oder Gesichtserkennung wird das Konzept sogar tauglich für allerhöchste Sicherheitsansprüche.

Von ganz einfach bis hochsicher

Damit stehen Passwort, PIN-Code, Gesicht und Hardware-Token als Authentifizierungs-Möglichkeit für den Linux-Desktop zur Verfügung. Durch geschickte Kombination kann man damit das komplette Spektrum von ganz einfach und komfortabel bis hochsicher und trotzdem noch erträglich umsetzen. Also erträglich im Sinne von: Immer noch viel besser, als alle paar Minuten ein zwölfstelliges Passwort einzugeben.

So ist Gesichtserkennung mit Howdy allein zwar ziemlich cool und simpel; hohe Ansprüche an die Sicherheit erfüllt sie jedoch nicht. Eine Kombination aus Howdy mit PIN-Code hingegen genügt für die meisten Ansprüche.

Kombiniert man Gesichtserkennung mit einem U2F-Token, hat man ein System, das auch höchsten Sicherheits-Ansprüchen genügt. Allerdings würde ich vom konkreten Einsatz von Howdy in Hochsicherheitsbereichen explizit abraten.

Ich persönlich bin bei meinen Experimenten zu einem Fan von U2F geworden. Das Sicherheitskonzept hat mich überzeugt und der Einsatz ist sehr komfortabel. Und nicht zuletzt sind die Token viel billiger als Kameras und lassen sich an mehreren Rechnern parallel nutzen.

Insgesamt haben für mich die U2F-Token die ursprünglich angestrebte Zweifaktor-Authentifizierung mit PIN und Gesicht ausgestochen. Bei der Gesichtserkennung kommt neben den Sicherheitsbedenken erschwerend hinzu, dass mich die ständig auf mich gerichtete Kamera irritiert. Die leuchtet zwar, wenn sie eingeschaltet wird. Doch ich erinnere mich zu gut an die Demo eines Hackers, der vorführte, wie er – bei einer anderen Kamera wohlgemerkt – die Aktivität signalisierende LED abschalten konnte.

Letztlich werde ich wohl auf eine Kombination aus PIN und U2F beziehungsweise für bestimmte Anmeldevorgänge auch U2F alleine setzen. (**ju [16]**)

URL dieses Artikels:

`https://www.heise.de/-4404929`

Links in diesem Artikel:

[1] https://www.heise.de/thema/Passw%C3%B6rter

[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/download/product/ubuntu-22040

[6] https://gist.github.com/ju916/ce58e0ce5c80030a34154ec48793cdc5

[7] https://www.heise.de/thema/MySQL

[8] https://support.asperasoft.com/hc/en-us/articles/216127348-Getting-PAM-to-authenticate-against-MySQL

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

[10] https://github.com/davisking/dlib-models/

[11] https://github.com/boltgolt/howdy

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

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

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

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

[16] mailto:ju@ct.de