

Benutzerkonfiguration - users.toml

Wichtige Informationen zur Benutzerdatenbank

BFFH selbst nutzt eine interne Datenbank in `bffh.db` für die Benutzer, jedoch wird diese erstmalig über die Datei `users.toml` gespeist.

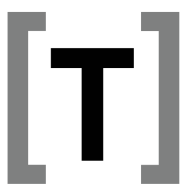
Das Speisen von Benutzern kann auch über die Python-API `pyfabapi` erfolgen (siehe [Beispielscript zum Hinzufügen und Löschen von Benutzern](#)).

Nachdem mindestens eine erste Verwaltungsrolle inkl. zugewiesenem Benutzer (Admin, Manager) zum System hinzugefügt wurde, kann anschließend auch die Benutzerverwaltung der Client-Anwendung `Borepin` genutzt werden. Über diese lassen sich Benutzer hinzufügen und entfernen, sowie Rollen und Passwörter managen.

Die in der App hinzugefügten bzw. modifizierten Benutzer **werden nicht automatisch** in die erstmalig genutzte `users.toml` geschrieben, sondern direkt in die interne Benutzerdatenbank. Das Exportieren erfolgt explizit durch `/usr/bin/bffhd --dump-users`. Für unsere Handhabe bedeutet das, dass die parallele Bearbeitung der Benutzerdatenbank per App und `users.toml` sorgfältig getätigt werden sollte, um etwaige Änderungen nicht ungewollt rückgängig zu machen.

Intro

In der TOML-Datei `users.toml` werden die Benutzer, sowie ihre jeweiligen Passwörter, Rollen und Kartenschlüssel gespeichert. Die Datei befindet sich üblicherweise in `/etc/bffh/`. Die Datei wird nicht automatisch von BFFH geladen - egal, ob sie komplett neu ist oder nur modifiziert wurde. Das Importieren der `users.toml` erfolgt durch `/usr/bin/bffhd --load-users`.



Offizielles TOML-Logo

Aufbau einer users.toml

Der grundlegende Aufbau folgt dem oben verlinkten TOML-Standard.

Benutzer

Jeder Nutzer wird dabei durch eine eigene Sektion `userid` angegeben (in einem Paar eckiger Klammern, z.B. `[AdminUser]`). Die `userid` Kennungen werden von BFFH z.B. beim Ausführen von Aktionen verwendet (Aktoren, Initiatoren) - dort werden sie in die Befehlskette hinten angefügt.

Der Benutzername kann direkt in eckige Klammern ohne Hochkommas geschrieben werden, solange keine Sonderzeichen wie Umlaute enthalten sind. Wir empfehlen aus genau diesem Grund prinzipiell jeden Nutzer ausschließlich in Hochkommas zu schreiben. So kann der Nutzer `[Ö]` von BFFH nicht erfolgreich importiert werden (da ungültige Deklaration), der Nutzer `["Ö"]` jedoch schon. Bei Nichtbeachtung folgt eine Fehlermeldung:

```
Error:  × unexpected character found: `\"{d6}` at line x column y
```

Achtung: Der Name kann unter anderem auch Leerzeichen und Sonderzeichen enthalten, sowie nahezu beliebig viele Zeichen. Folgender Username ist beispielsweise gültig:

```
["Ein seeeeeeeeeeeeeeeeeeeeeeeeeeeeeehrlanger
```

Nutzername mit Sonderzeichen_ "]. Dem Administrator wird angeraten eine Nutzernamenkonvention aufzustellen, die z.B. verschiedene Sonderzeichen vermeidet, die häufig auch bei LDAP-gestützten Anwendungen Probleme bereiten. Es wird im Zweifelsfall empfohlen sich auf Klein- und Großbuchstaben, sowie Ziffern und einfache Zeichen wie `.-_` zu beschränken.

Rollen, Passwörter, Cardkeys

Unter jeder `userid` werden Schlüssel-Wert-Paare abgelegt. Die aktuell **genutzten** Schlüsselnamen lauten:

- `roles` (Rollen) - diese Rollen werden in der Hauptkonfiguration definiert und dann der `users.toml` zugewiesen
- `passwd` (Passwort - unverschlüsselt (plaintext) oder als Argon2-Hash)

- `cardkey` (DESFire EV2 UUID) - diese wird entweder aus der Borepin App oder über das FabFire Provisioning Tool erzeugt
- `cardtoken` - eine anonyme Alternative zur `userid` und gilt als Mapping zwischen `userid` und `cardtoken`. So muss die `userid` auf einer DESFire EV2 Karte nicht gespeichert werden. Siehe auch Datei 0003.

Weitere Schlüssel und Kommentare

Beliebige weitere Schlüssel können in der `users.toml` angelegt werden und werden ebenso von BFFH geladen (key-value store) und auch wieder exportiert. Sie werden jedoch nicht vom System verwendet!

Kommentare können mit `#` vorangestellt erzeugt werden oder auf gleicher Zeile hinter dem Wert ("Inline comment"):

```
[Admin1]
# this is the admin role
roles = ["Admin"]
passwd = "secret"
cardkey = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa" #some inline comment
```

Kommentare werden spätestens beim Laden innerhalb der BFFH-Datenbank verworfen. Es empfiehlt sich im Falle des Kommentarbedarfs mit Exportstabilität einen Pseudoschlüssel anzulegen, zum Beispiel `comment`:

```
[Admin1]
roles = ["Admin"]
passwd = "secret"
cardkey = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
comment = "this is the admin role"
```

Beispielbenutzer und -rollen

Rollen werden in der bffh.dhall-Konfiguration definiert. Das Docker-compose Repository <https://gitlab.com/fabinfra/fabaccess/dockercompose> hat ein gutes Beispiel:

```
[Admin1]
roles = ["Admin"]
```

```
passwd = "secret"
cardkey = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

[Admin2]
roles = ["Admin"]
passwd = "secret"
cardkey = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

[ManagerA1]
roles = ["ManageA", "UseA", "ReadA", "DiscloseA"]
passwd = "secret"
cardkey = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

[ManagerA2]
roles = ["ManageA", "UseA", "ReadA", "DiscloseA"]
passwd = "secret"
cardkey = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

[ManagerB1]
roles = ["ManageB", "UseB", "ReadB", "DiscloseB"]
passwd = "secret"
cardkey = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

[ManagerB2]
roles = ["ManageB", "UseB", "ReadB", "DiscloseB"]
passwd = "secret"
cardkey = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

[ManagerC1]
roles = ["ManageC", "UseC", "ReadC", "DiscloseC"]
passwd = "secret"
cardkey = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
```

Manuelles Anlegen neuer Benutzer

Lasse alle neuen Nutzer ihre Passwörter verschlüsselt an den Admin übertragen, indem diese ihr Passwort als verschlüsselten Argon2 String übermitteln. Prinzipiell unterstützt FabAccess die Typen

- Argon2i
- Argon2d
- Argon2id

Die Passwörter können beliebig komplex werden. Es gibt verschiedene Parameter, die ein Passwort besonders sicher machen. Folgende Werte und ihrer empfohlenen Standardwerte können genutzt werden. Mit folgenden Settings hashed BFFH die Passwörter intern und exportiert sie auch so (`bffhd --export`):

- Salt: eine zufällige Zeichenkette , z.B. `16` bytes lang (Empfehlung)
- Parallelism Factor: `1`
- Memory Cost: `4096` kilobytes
- Iterations: `3`
- Hash Length: `32` bytes

Je sicherer das Passwort gewählt wird, desto höher ist die benötigte CPU-Leistung zum Entschlüsseln und das Einloggen in FabAccess dauert entsprechend länger. Ein Passwort mit Parallelism Factor 10, Memory Cost 100000, Iterations 20 und Hash Length 100 benötigt auf einem Raspberry Pi 3 B+ ca. 15 Sekunden zur Entschlüsselung. Siehe auch <https://www.ory.sh/choose-recommended-argon2-parameters-password-hashing> und https://de.wikipedia.org/wiki/Argon2#Empfohlene_Parameter

Argon2 Passwort auf der Kommandozeile erzeugen ...

... mit Ubuntu Linux

```
apt install argon2
PASSWORD="mypassword" && echo | argon2 $PASSWORD -i -k 4096 -p 1 -t 3 -l 32
```

... mit Windows

Eine Implementierung für Windows existiert zum Beispiel unter <https://github.com/philtr/argon2-windows>

Wir laden das Release-Zip herunter, entpacken es und führen `Argo2Opt.exe` aus:

```
set PASSWORD="mypassword"
echo %PASSWORD% | Argon2Opt.exe %PASSWORD% -i -k 4096 -p 1 -t 3 -l 32
```

Wir übermitteln die kodierte Zeichenkette in die `users.toml`, hier im Beispiel:
`$argon2i$v=19$m=4096,t=3,p=1$UEFTU1dPUkQ$VMwncjCWdW+f6x8qzshLaA`. Der Eintrag in der `users.toml` könnte so lauten:

```
[vmario]
roles = ["mitglied"]
passwd = "$argon2i$v=19$m=4096,t=3,p=1$UEFTU1dPUkQ$VMwncjCWdW+f6x8qzshLaA"
cardkey = "70AFE9E6B1D6352313C2D336ADC2777A"
```

Argon2 Passwort mit Argon2 Hash Generator Tool

Argon2 Hashes können auch online erstellt werden:

Argon2 Hash Generator

Plain Text Input

fabinfra101

Salt

TrBOLAFJHdgrBF8X

Parallelism Factor

1

Memory Cost

4096

Iterations

3

Hash Length

32

Argon2i

Argon2d

Argon2id

[How to Choose the Right Parameters for Argon2 »](#)

Output in HEX Form

COPY

ed5f68f79ee2b8e617fd4a041cca589dc99308d274e9d43d60e272667c6fdce5

Output in Encoded Form

COPY

\$argon2i\$v=19\$m=4096,t=3,p=1\$VHJCT0xBRkplZGdyQkY4WA\$7V9o957iuOYX/UoEHMpYncmTCNJ06dQ9YOJyZnxv3OU

GENERATE HASH

RESET FORM

Wir übermitteln dem Administrator von FabAccess letztlich die kodierte Zeichenkette, hier im Beispiel:

```
$argon2i$v=19$m=4096,t=3,p=1$VHJCT0xBRkplZGdyQkY4WA$7V9o957iuOYX/UoEHMpYncmTCNJ06dQ9Y
OJyZnxv3OU
```

Benutzerkonfiguration in BFFH importieren

Eine einmal angelegte `users.toml` kann und muss in die BFFH-interne Datenbank importiert werden. Details siehe

- [Cheat Sheet - Wichtigste Befehle \(Übersicht\)](#)
- [Nutzerdatenbank laden / hashen / prüfen](#)

Benutzerkonfiguration aus BFFH exportieren

Die in der BFFH-Datenbank gespeicherten Benutzer können mit dem Zusatz `--dump-users` in eine `users.toml` Datei exportiert werden (hier im Beispiel nach `/etc/bffh/config_backup/users.toml`):

```
mkdir -p /etc/bffh/config_backup/  
/usr/bin/bffhd --verbose --config /etc/bffh/bffh.dhall --dump-users /etc/bffh/config_backup/users.toml --force
```

Siehe auch [Cheat Sheet - Wichtigste Befehle \(Übersicht\)](#)

Weitere Tipps und Tricks mit TOML Dateien

Ein paar weitere Tools, die das digitale Leben ggf. vereinfachen.

TOML-Dateien sortieren (nach Sektionen, Arrays, etc.) mit toml-sort

<https://pypi.org/project/toml-sort>

```
cd /opt/fabinfra/scripts/  
python3 -m venv env  
. env/bin/activate #activate venv  
/opt/fabinfra/scripts/env/bin/pip3 install toml-sort  
chown -R bffh:bffh /opt/fabinfra/scripts/env/
```

```
toml-sort --help  
usage: toml-sort [-h] [--version] [-o OUTPUT] [-i] [-I] [-a] [--no-sort-tables] [--sort-table-keys]  
                [--sort-inline-tables] [--sort-inline-arrays] [--sort-first KEYS] [--no-header] [--no-comments]  
                [--no-header-comments] [--no-footer-comments] [--no-inline-comments] [--no-block-comments]  
                [--spaces-before-inline-comment {1,2,3,4}] [--spaces-indent-inline-array {2,4,6,8}]  
                [--trailing-comma-inline-array] [--check]
```

[F ...]

Mit folgendem Kommando lassen sich `users.toml` Dateien sortieren und gleichzeitig überschreiben.

```
toml-sort --in-place --all /etc/bffh/users.toml
```

TOML-Dateien als JSON ausgeben/verarbeiten

Analog zum bekannten Werkzeug `jq` (JSON-Parser) gibt es das Tool `yq` für das TOML-Format (als Wrapper für `jq`): <https://kislyuk.github.io/yq/#toml-support>

```
cd /opt/fabinfra/scripts/  
python3 -m venv env  
. env/bin/activate #activate venv  
/opt/fabinfra/scripts/env/bin/pip3 install yq  
chown -R bffh:bffh /opt/fabinfra/scripts/env/
```

Durch dieses Paket wird ebenfalls die Binary `tomlq` automatisch mit installiert. Bitte beachten: Das gleichnamige Paket `tomlq` gibt es separat auf [PyPI](#), ist jedoch stark veraltet und inkompatibel zu neuen Python Versionen. Es wird daher empfohlen das beinhaltende, aktuellere Paket `yq` zu installieren.

Unsere `users.toml` als JSON-Objekt ausgeben:

```
tomlq '.' /etc/bffh/users.toml
```

Mit dem nachfolgenden Befehl überschreiben wir eine `users.toml` Datei so, dass z.B. alle Kommentare entfernt werden:

```
tomlq -t -i '.' /etc/bffh/users.toml
```

Version #62

Erstellt: 23 Oktober 2024 22:53:24 von Mario Voigt (Stadtfabrikanten e.V.)

Zuletzt aktualisiert: 3 März 2025 15:01:11 von Mario Voigt (Stadtfabrikanten e.V.)