Benutzerkonfiguration - users.toml

Wichtige Informationen zur Benutzerdatenbank

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/` oder in `/opt/fabinfra/bffh-data/config/` (je nach Setupmethode). 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 `bffhd --load`.

BFFH selbst nutzt eine interne Datenbank (`bffh.db`) für die Benutzer, jedoch wird diese erstmalig über die Datei `users.toml` gespeist. 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. Das erfolgt explizit durch `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.

Wer mit der automatisierten Erstellung einer `users.toml` aus externen Nutzerquellen wie zum Beispiel CSV-Dateien oder LDAP (siehe hier) arbeitet, der sollte folgenden Workflow erarbeiten:

1. BFFH Server herunterfahren
2. Aktuelle Benutzerkonfiguration per `bffhd --dump-users users.toml` sichern
3. die soeben gesicherte `users.toml` Datei benutzen, um die aktuellsten Änderungen an der Benutzerbasis hinzuzufügen, zu löschen oder zu ändern. Hier muss selbst entschieden werden, welche Benutzerdaten aktuell sein sollen: Die durch die App geänderten Daten, oder die aus der externen Quelle? Eine von beiden muss die andere überschreiben bzw. überlagern. Entsprechend muss das verwendete Script programmiert sein: bei Vorhandensein von Nutzern, Rollen, Passwörter, Cardkeys rückfragen, ob bei Differenzen der bestehende oder der neue Datensatz verwendet werden soll oder per Config-Parameter automatisiert durchführen.
4. die neue `users.toml` Datei laden: `bffhd --load users.toml` (siehe auch Nutzerdatenbank laden / hashen / prüfen)

5. BFFH Server starten

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.

> **Achtung**: Der Name kann unter anderem auch Leerzeichen und Sonderzeichen enthalten, sowie nahezu beliebig viele Zeichen. Folgender Username ist beispielsweise gültig: `["Ein seeeeeeeeeeeeeeeeeeeeeeeeehr langer Nutzername mit Sond@rze!chen_ "]`. 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)

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 = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa" #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 = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
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 = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

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

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

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

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

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

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

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/philr/argon2-windows

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

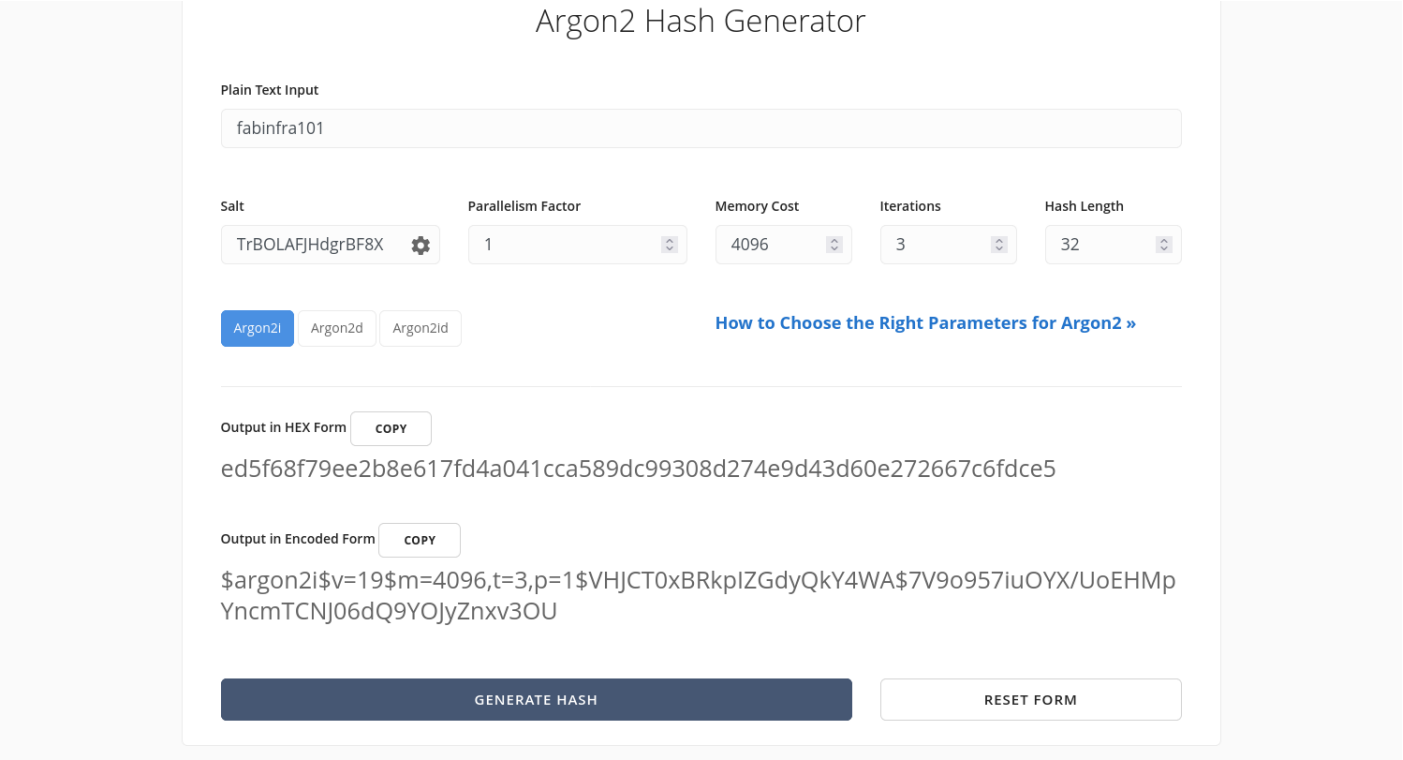
```
set PASSWORD="mypassword"
echo %PASSWORD% | Argon2Opt.exe %PASSWORD% -i -k 4096 -p 1 -t 3 -l 32
```

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:



Wir übermitteln dem Administrator von FabAccess letztlich die kodierte Zeichenkette, hier im Beispiel:
`$argon2i$v=19$m=4096,t=3,p=1$VHJCT0xBRkpIZGdyQkY4WA$7V9o957iuOYX/UoEHMpYncmTCNJ06dQ9YOJyZnxv3OU`

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:

```
BASE="/opt/fabinfra/bffh/target/release"
CFG="$DATA/config/bffh.dhall"
$BASE/bffhd --verbose --config $CFG --dump-users /tmp/users.toml --force
```

Siehe auch Cheat Sheet - Wichtigste Befehle (Übersicht)

Weitere Tipps und 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] [-l] [-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 /opt/fabinfra/bffh-data/config/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 '.' /opt/fabinfra/bffh-data/config/users.toml
```

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

```
tomlq -t -i '.' /opt/fabinfra/bffh-data/config/users.toml
```

Version #46
Erstellt: 23 Oktober 2024 22:53:24 von Mario Voigt (Stadtfabrikanten e.V.)
Zuletzt aktualisiert: 6 Januar 2025 21:39:20 von Mario Voigt (Stadtfabrikanten e.V.)