FabAccess API

Allgemeines

Die FabAccess API ist eine zentrale Komponente von FabAccess und hat einige Veränderungen hinter sich. Sie basiert auf Kern auf Cap'n Proto und ist für die Kommunikation zwischen Server (Difluoroborane) und Client (Borepin) verantwortlich.

> Die aktuelle API von FabAccess ist auf dem Versionsstand **0.3.0**. Der Umzug von FabAccess-API zu FabAccess-API-cs ist noch nicht erfolgt.

Die API wird verwendet, um eine erweiterbare und bidirektionale API bereitzustellen. Dank der Codegenerierungsfunktion von Cap'n Proto kann die API in jede Programmiersprache portiert werden. Jedes Interface wird in einer übersichtlichen Schema-Sprache dargestellt (z.B. in Python, siehe pyfabapi), was die API selbst dokumentierend macht. Die Verfügbarkeit von Interfaces im Client kann durch einfache Null-Checks überprüft werden.

Authentifizierung

Um einen übergeordneten Zugangspunkt zu bieten, ist die API in Systeme unterteilt, die erweitert und ausgetauscht werden können. Der Einstiegspunkt in die API ist das Bootstrap-Interface, das jedem Client angeboten wird. Nach der Authentifizierung werden die für den Client relevanten Interfaces bereitgestellt.

Scripting

Die API soll anderen Clients oder Skripten einen stabilen Zugang zu Ressourcen ermöglichen und die Zusammenarbeit zwischen Systemen fördern. Der Server entscheidet selbst, welche Ressourcen und Interfaces er den Clients zur Verfügung stellt, je nach Berechtigung und Konfiguration. Die Clients müssen daher in der Lage sein, damit umzugehen.

Ressourcenzugang

Durch die API und die damit verbundenen Anforderungen soll der Zugang zu Ressourcen für alle erleichtert und die Zusammenarbeit gefördert werden. Diese Grundlagen sollen dazu beitragen, ein föderiertes System für den Ressourcenverleih aufzubauen.

Details zu Cap'n Proto finden sich auf deren Homepage: https://capnproto.org und https://github.com/capnproto/capnproto

Warum gibt es keine REST API?

Der einfachste Punkt, weswegen REST eher ungeeignet ist der, dass unsere Verbindung bidirektional ist, also beide Seiten zu jedem Zeitpunkt Daten an die jeweils andere schreiben können. Das würde WebRTC prinzipiell problemlos ermöglichen (Datenaustausch z.B. per JSON), aber eine einfache TCP- oder QUIC-Verbindung ebenso und mit wesentlich weniger Setupaufwand. Sobald es um WebRTC geht, sind in einem webbasierten Client große Mengen angepasster JavaScript-Code nötig. Ist das der Fall, könnten diese TCP-/QUIC-Verbindungen über Websockets tunneln und die bestehende Cap'n Proto API ansprechen.

Cap'n Proto API ansprechen

Eine Übersicht über verschiedene Sprachen: https://capnproto.org/otherlang.html

FabAccess-API

Dies ist die ursprüngliche und aktuell verwendete API. Sie ist direkt in der Schemasprache Capn' Proto implementiert (*.capnp Dateien).

- Sprachreferenzen: https://capnproto.org/language.html
- Quellcode: https://gitlab.com/fabinfra/fabaccess/fabaccess-api
- Releases: https://gitlab.com/fabinfra/fabaccess/fabaccess-api/-/tags

Diese API sollte ursprünglich im April 2024 durch fabaccess-api-cs abgelöst werden.

Wegen 26.07.2024 // FabAccess Entwicklung kommt zum Erliegen wurde dieser Schritt jedoch bisher nicht vollzogen.

Benutzen der API

Die FabAccess-API kann mittels pyfabapi (Python) verwendet werden.

Aufbau

Im Folgenden wird der Inhalt der einzelnen Schemadateien kurz erläutert:

`connection.capnp`

Ausgangspunkt für API-Interaktionen mit einer Instanz der Schnittstelle `Bootstrap`, die beim Verbindungsaufbau an jeden Client gesendet wird.

`authenticationsystem.capnp`

SASL-basierte Authentifizierung, die bei erfolgreicher Authentifizierung eine Instanz von `struct session` aus `connection.capnp` zurück gibt und den Zugriff auf einige oder alle Subsysteme auf der Grundlage von Benutzerrollen ermöglicht.

`machinesystem.capnp`

Suchsystem für Ressourcen auf der Grundlage von URN oder konfigurierten Namen.

`permissionsystem.capnp`

Administrativer Zugang zu internen Rollen.

`usersystem.capnp`

Lookup & Admin-Zugang zur internen Darstellung der Benutzer.

`machine.capnp`

Methoden für Ressourcenobjekte, die Zustandsaktualisierungen nach dem derzeitigen Zustands-Modell und den administrativen Zugang für privilegierte Bediener ermöglichen.

`user.capnp`

Zugang zu Benutzerinformationen und Verwaltungsmethoden für privilegierte Benutzer.

`space.capnp` / `general.capnp` / `role.capnp`

In anderen Dateien verwendete Hilfstypen.

TLS Verschlüsselung

FabAccess ist wählerisch, was die verwendete Transportverschlüsselung angeht.

Kompatibel sind TLS v1.2 und TLS v1.3, jeweils begrenzte Chiffren. Details dazu finden sich auf der Server-Seite.

> Dies ist kein funktionaler Teil der Cap'n Proto API, sondern eine notwendige Information für Drittanbieter-Implementierungen des Protokolls.

Entscheidungsgründe für TLS

Kontext und Problemstellung

Die Implementierer der API sollten für jede nicht-lokale Kommunikation ein gewisses Maß an Transportverschlüsselung verwenden, denn wir leben nicht mehr in den 2000er Jahren und unsere Verschlüsselung ist tatsächlich gut, billig und sicher.

Entscheidungstreiber

- Der betreffende Softwarestack hat Sicherheitsrelevanz, selbst wenn er nur in einem LAN-Kontext verwendet wird.
- Da sich die meisten Nutzer der API über WLAN verbinden und die meisten von ihnen PSK verwenden, ist das Abhören trivial.

Geprüfte Optionen

- TLS
- DTLS
- Noise Framework Protocol

Entscheidungsergebnis

Gewählte Option: „TLS“, da TLS insgesamt am einfachsten für den verbleibenden Stack zu implementieren ist und die meisten Systemadministratoren ein gutes Verständnis der PKI von TLS haben.

Positive Konsequenzen

- Verlässliche Transportverschlüsselung ist gewährleistet
- Die PKI-Struktur von TLS kann das inhärente Problem der Vertrauensbildung in einer föderalen Umgebung leicht lösen

Negative Folgen

- Die Generierung eines vertrauenswürdigen X.509-Zertifikats ist für eine föderierte Anwendung erforderlich und verursacht entweder finanzielle Kosten oder zusätzliche Einrichtungsarbeit

- Der Verschlüsselungs-Overhead ist ein relevanter Faktor bei Geräten mit extrem niedrigem Stromverbrauch, wenn der Server für diesen Anwendungsfall schlecht konfiguriert ist (d. h. kein ChaCha20 und andere rechenschwache Algorithmen anbietet).

Vor- und Nachteile der Optionen - TLS

- Gut, weil TLS-Unterstützung auf allen Plattformen allgegenwärtig ist
- Gut, weil TLS die Aushandlung von Verschlüsselungsalgorithmen ermöglicht, so dass verschiedene Geräte die für sie am besten geeignete Verschlüsselung wählen können
- Gut, weil TLS Erweiterungen bietet, z. B. ALPN, die die Protokollversionierung erleichtern
- Schlecht, weil TLS nicht gut für SCTP geeignet ist, wohin jedoch das Protokoll in Zukunft hinwechseln sollte
- Schlecht, weil TLS von Natur aus sehr komplex ist und unter vielen Angriffsvektoren gelitten hat, am bekanntesten z. B. Heartbleed und LogJam, die bei der Konfiguration von TLS besondere Vorsicht erfordern
- Schlecht, weil die Chiffrieraushandlung von TLS (insbesondere unterhalb von Version 1.3) anfällig für Downgrade-Angriffe ist, insbesondere im Falle einer Verwendung im Stil von STARTTLS.

Vor- und Nachteile der Optionen - DTLS

Verwenden des Datagram Transport Layer Security, ein IETF-Protokoll, das TLS ähnelt, aber speziell für nachrichtenorientierte Protokolle entwickelt wurde, bei denen Nachrichtenverluste und -umwandlungen toleriert werden müssen.

- Gut, weil es die meisten Vorteile von TLS teilt, aber auch ergonomischer mit SCTP zusammenarbeitet
- Schlecht, weil DTLS deutlich weniger gut unterstützt wird als TLS
- Schlecht, weil DTLS kein Äquivalent für TLSv1.3 hat, das im Vergleich zu TLSv1.2 erhebliche Sicherheitsverbesserungen bietet

Vor- und Nachteile der Optionen - Noise Protocol Framework

Verwenden von Verschlüsselung auf der Grundlage von Noise, einem Framework mit Unterstützung für gegenseitige und optionale Authentifizierung, Identitätsverschleierung, Forward Secrecy, Zero-Round-Trip-Verschlüsselung und anderen erweiterten Funktionen.

- Gut, weil es keinen Entwurf für die Aushandlung von Chiffren hat, was Downgrade-Angriffe unmöglich macht
- Gut, weil Noise aufgrund seines geringen Gewichts und der gewählten Chiffren im Vergleich zu TLS oder DTLS nur sehr geringe Auswirkungen hat
- Gut, weil Noise sich sehr gut für ein System eignet, bei dem Verschlüsselungsschlüssel über einen Seitenkanal weitergegeben werden, z. B. durch Scannen eines QR-Codes, der auch die Adresse enthält, zu der eine Verbindung hergestellt werden soll.
- Schlecht, weil die Plattformunterstützung im Vergleich zu TLS/DTLS sehr begrenzt ist, obwohl die wichtigsten Plattformen wie Rust (bffhd), C# (Borepin), Python(1, 2) (pyfabapi) abgedeckt sind.
- Schlecht, denn Rauschen erfordert mehr Implementierungsarbeit als TLS, was die Anzahl der Codezeilen und die zu treffenden Entscheidungen betrifft.

FabAccess-API-cs

Das ist der aktuelle Rewrite der FabAccess-API in C Sharp (*.cs Dateien), jedoch **ohne bisherigen Release** und Switch auf die neue Architektur.

> Die neue API hat bereits einen recht weiten Arbeitsstand (laut Joseph Langosch) einen Arbeitsstand von Version **0.9.0**.

- Quellcode: https://gitlab.com/fabinfra/fabaccess/fabaccess-api-cs

pyfabapi (Python-Implementierung für FabAccess-API)

Das ist eine kleine Python-Bibliothek für den Zugriff auf die FabAccess-API für Verwaltung und Entwicklung.

- Quellcode: https://gitlab.com/fabinfra/fabaccess/pyfabapi

Funktionsumfang

Die Python API unterstützt alles, was im Schema (*.capnp) Dateien von FabAccess-API zu finden ist. Wir können damit also zum Beispiel Benutzer anlegen und löschen, Rollen hinzufügen und verändern, Ressourcenzustände setzen und so weiter. Ein paar

Einstiegsbeispiele sind im GitLab-Projekt dokumentiert und einsatzfähig, siehe Verzeichnis `examples/`:

- add_del_user.py - Benutzer hinzufügen oder löschen
- use_giveback_machine.py - Eine Ressource (Maschine) benutzen oder freigeben
- raw-write.py - Rohdaten an eine Ressource (Maschine) senden

Außerdem wird die API in folgenden Projekten verwendet:

- https://gitlab.com/fabinfra/fabaccess/prometheus-exporter (siehe auch Monitoring: Prometheus, Loki und Grafana)
- https://github.com/interfacerproject/zenflows-fabaccess (siehe FabCity OS (INTERFACER) Anbindung an ZENFLOWS)

Installation

```
mkdir -p /opt/fabinfra/tools/
cd /opt/fabinfra/tools/
git clone https://gitlab.com/fabinfra/fabaccess/pyfabapi.git --recursive

cd /opt/fabinfra/tools/pyfabapi/
python3 -m venv env
. env/bin/activate #activate venv
pip install -r requirements.txt

chown -R bffh:bffh /opt/fabinfra/tools/pyfabapi/
```

Upgrade

Die Installation von pyfabapi kann veralten. Über folgende Befehle können wir das Verzeichnis auf den aktuellsten Stand bringen:

```
cd /opt/fabinfra/tools/pyfabapi/
git pull
. env/bin/activate #activate venv
pip install --upgrade -r requirements.txt
chown -R bffh:bffh /opt/fabinfra/tools/pyfabapi/
```

Version #25
Erstellt: 2025-02-17 15:46:58 CET von Mario Voigt (Stadtfabrikanten e.V.)

Zuletzt aktualisiert: 2025-03-06 01:00:10 CET von Mario Voigt (Stadtfabrikanten e.V.)