Vorüberlegung zu Features: Save-Storage / Moodle Integration

Eine initiale Feature-Idee für dieses Semester war es, dem Bot eine Moodle-Integration zu implementieren, durch die der Nutzer Ereignisse in Moodle mitgeteilt und an Abgabetermine erinnert werden kann. Moodle bietet hierfür eine REST-API: https://docs.moodle.org/dev/Web_service_API_functions.

Problem 1: Login

Damit user-bezogene Daten abgerufen werden können muss der User sich in Moodle via username + passwort einloggen. Dieser Login kann nicht über die Chatbot-Funktionalitäten erfolgen, da Messenger in der Regel keine Passwort-Eingabe ermöglichen, was darin mündet, dass die Credentials im Cleartext im Chatlog landen.

Lösung: Es benötigt ein Webformular, was den Moodle-Login sicher zentralisiert. Der Bot darf im Chat nur den Link zum Login ausspielen.

Problem 2: Speicherung des Tokens

Der BeuthBot ist ein Studierenden-Projekt. Es gibt derzeit keinen Production-Server, der nicht von Studierenden eingesehen werden kann. Die hohe Fluktuation an „Administratoren“ erzeugt ein hohes Risiko für das „Leaken“ von gespeicherten Credentials.

Gefahren:

1. Böswilliger Entwickler (programmiert daten-abfluss)
2. Gutwilliger Entwickler (macht Programmierfehler)
3. Böswilliger Admin (transferiert/liest persistierte daten)
4. Gutwilliger Admin (dupliziert/transferiert daten als backups)
5. Böswilliger Nutzer (nutzt sicherheitslücken / programmierfehler)

Lösung 1: Das User-Token wird nur im RAM abgelegt

Das Token wird so nie auf die Festplatte geschrieben

1. [x] Böswilliger Entwickler (programmiert daten-abfluss)
2. [✓] Gutwilliger Entwickler (macht Programmierfehler)
3. [✓] Böswilliger Admin (transferiert/liest persistierte daten)
4. [✓] Gutwilliger Admin (dupliziert/transferiert daten als backups)
5. [✓] Böswilliger Nutzer (nutzt sicherheitslücken / programmierfehler)

Nachteil Lösung 1: Die Usability leidet stark, wenn der User sich ständig neu einloggen muss, weil der Bot die Credentials beim Neustart vergisst. Vor allem für Erinnerungs-Features ist das ein Showstopper.

Lösung 2: Das User-Token wird nur persistiert, wenn die Datenbank geschrieben wird

Die Daten werden beim Speichern mit einem One-Time-Token verschlüsselt. Beim nächsten Start des Dienstes werden die Daten entschlüsselt und die persistente Kopie gelöscht

1. [x] Böswilliger Entwickler (programmiert daten-abfluss)
2. [✓] Gutwilliger Entwickler (macht Programmierfehler)
3. [x] Böswilliger Admin (transferiert/liest persistierte daten)
4. [✓] Gutwilliger Admin (dupliziert/transferiert daten als backups)
5. [✓] Böswilliger Nutzer (nutzt sicherheitslücken / programmierfehler)

Nachteil Lösung 2: Der One-Time-Key muss auch irgendwie gespeichert werden. Da dieser auch für die Anwendung zugänglich sein muss liegt er gewissermaßen neben der verschlüsselten Datei. Ein cleveres One-Time-Verfahren kann hier zwar dafür sorgen, dass fremder Zugriff auf die Daten nicht unbemerkt bleibt - Gerade Backups können auf diese Art ganz gut abgesichert werden indem der Key dort nicht gespeichert wird. Ein echter Schutz der Daten ist aber nicht gegeben, spätestens der böswillige Admin wird einen Weg finden das Verfahren zu manipulieren

Fazit: Wir haben uns gegen eine Speicherung von User-Credentials entschieden, Solange es kein (professionell administriertes und zugriffsbeschränktes) Produktiv-Environment gibt.