Der Timadorus Authentifizierungsserver

Torben Könke

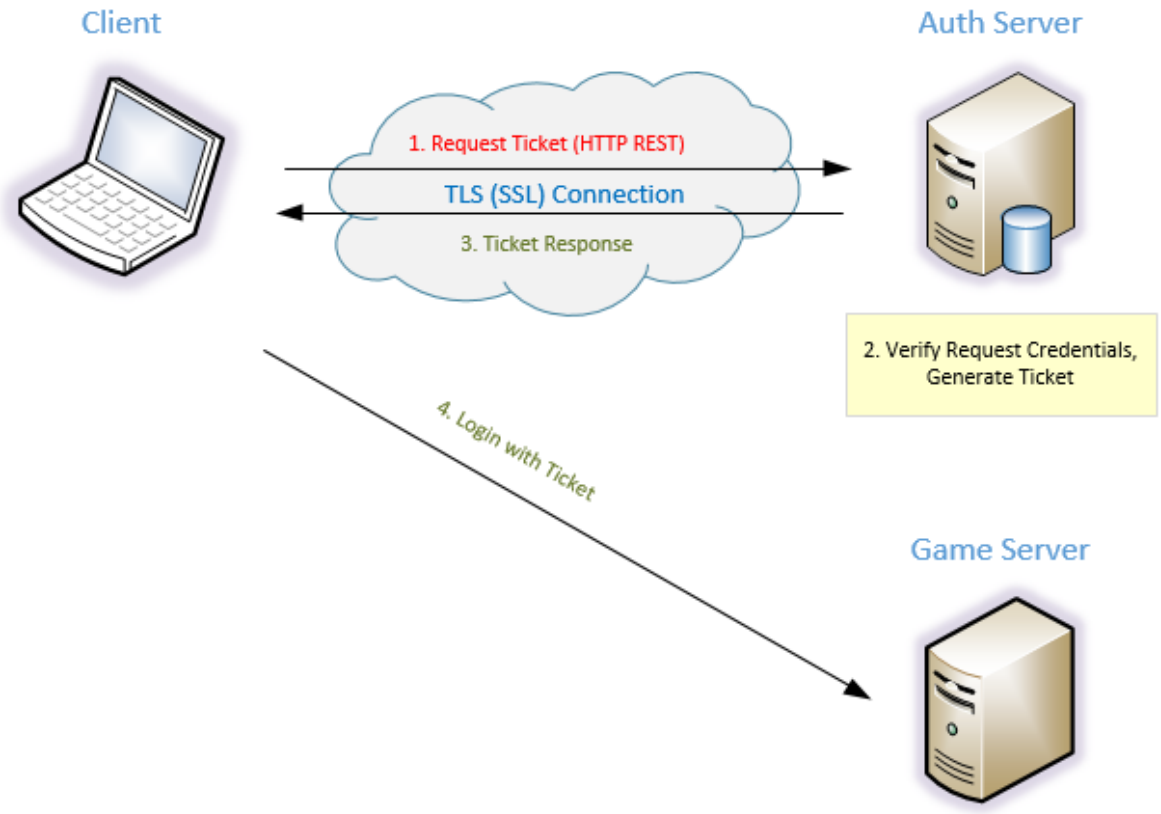
16. Juni, 2014

Inhaltsverzeichnis

1. Überblick

Der Timadorus Authentifizierungsserver ist ein eigenständiger Server, der die Authentifizierung von Benutzerkonten (*user accounts*) sowie die Auswahl eines Spielcharakters (*entity*) des jeweiligen Benutzers durchführt. Nachdem sich ein Client beim Authentifizierungsserver erfolgreich ausgewiesen hat, erhält er von diesem einen sog. „AuthToken“, der einen Identitätsnachweis darstellt und vom Client an den eigentlichen Gameserver weitergereicht wird. Der Gameserver ist wiederum in der Lage die Echtheit und Validität dieses AuthTokens zu prüfen, ohne dafür das Passwort des jeweiligen Benutzers kennen zu müssen. Auf diese Weise wird die Authentifizierung von Benutzerkonten effektiv vom Rest des Systems entkoppelt und genügt damit dem Prinzip, daß sensible Daten möglichst isoliert gehalten werden.

Nachfolgende Graphik veranschaulicht die Zusammenhänge, die in den weiteren Abschnitten näher erläutert werden:



2. RESTful Webservice

Der Authserver stellt seine API in Form eines RESTful Webservices bereit, d.h. die Kommunikation erfolgt über gewöhnliche HTTP-Requests, gerichtet an bestimmte Ressourcen, die der Authserver zur Verfügung stellt.

2.1 Was ist REST

REST (Representational state transfer) ist ein Programmierparadigma um intuitive APIs, insbesondere CRUD Operationen (Create, Read, Update, Delete) bereitzustellen. Ein gewöhnlicher HTTP-Server stellt verschiedene Ressourcen zur Verfügung, die per HTTP-Request angefordert werden können. Hierbei macht man sich zu Nutze, daß der HTTP-Standard verschiedene Verben zum Anfordern einer Ressource definiert, die sich gut mit den jeweiligen CRUD Operationen decken, nämlich:

- **HTTP GET**
  Fordert eine Ressource an
- **HTTP DELETE**
  Löscht eine Ressource
- **HTTP PUT**
  Legt eine neue Ressource an
- **HTTP POST**
  Ändert eine bestehende Ressource

2.2 REST API

Nachfolgend eine Auflistung der verschiedenen API Funktionen bzw. der Ressourcen, die der Authserver bereitstellt und an die HTTP-Anfragen gerichtet werden können.

GET /users/username

Liefert Informationen über einen Benutzeraccount zurück.

Ressource	/users/username
Methode	**GET**
Parameter	Keine
Einschränkungen	Kann nur für den eigenen Benutzeraccount angefordert werden.
Beschreibung	Liefert Statusinformationen über den Benutzer mit Benutzernamen „username“ zurück, u.a. eine Auflistung aller auswählbaren Charaktere des Benutzeraccounts.
Rückgabewert	JSON-Objekt mit folgenden Attributen: • **name** Benutzername des Benutzers • **entities**

Liste aller auswählbaren Charaktere des Benutzers

Beispielanfrage	GET /users/bob

Beispielantwort	{
	name: 'bob',
	entities: ['foo', 'bar']
	}

POST /users/username

Ändert Attribute eines bestehenden Benutzeraccounts.

Ressource	/users/username
Methode	**POST**
Parameter	Ein JSON-Objekt mit den zu ändernden Attributen. Momentan sind folgende Attribute definiert: ○ **password** Das Passwort des Benutzeraccounts
Einschränkungen	Kann nur für den eigenen Benutzeraccounts angefordert werden.
Beschreibung	Ändert Attribute des Benutzers mit Benutzernamen „username“.
Rückgabewert	Kein Rückgabewert. Erfolg ist an Hand des HTTP Statuscodes ersichtlich (Stets *200 OK* im Erfolgsfall).
Beispielanfrage	POST /users/bob HTTP Body des POST-Requests: { password: 'meinGeheimesPasswort' } Nach erfolgreichem Aufruf ist das neue Passwort des Benutzers *bob* ab sofort **meinGeheimesPasswort**.

GET /users/username/charname

Liefert Informationen über einen Charakter eines Benutzeraccounts zurück.

Ressource	/users/username/charname
Methode	**GET**
Parameter	Keine
Einschränkungen	Kann nur für den eigenen Benutzeraccount angefordert werden.
Beschreibung	Liefert Statusinformationen über den Spielcharakter mit Namen „charname“ des Benutzers mit Benutzernamen „username“ zurück.
Rückgabewert	JSON-Objekt mit folgenden Attributen: • **name** Name des Charakters • **gameServer** Der Endpunkt, zu welchem der Client sich verbinden soll.

- **authToken**
  Ein transparenter Wert, den der Client im Zuge der Loginprozedur an den angegebenen Gameserver weiterreicht, um sich auszuweisen.
- **sessionKey**
  Ein BASE64-kodierter zufälliger Schlüssel für symmetrische AES-Session-Verschlüsselung zwischen Client und Gameserver.

  Dieses Attribut ist optional und nur vorhanden, wenn der Authentifizierungsserver entsprechend konfiguriert ist.

Beispielanfrage GET /users/bob/foo

Beispielantwort

```
{
        name:       'foo',
        gameServer: '192.168.2.123:60002',
        authToken:  'f/HnTL64VRqnva+MYuWeOGS4v
                     oftYFRBKuWaTfDIGw9aQGScs83
                     4fw5v+oNkCgKPiO1APgHma/066p
                     aQ4Mlz7Ei36fxqFd/SFIBEBVLjRhl3
                     x7Wqahnd+eIerUjK',
        sessionKey: '09GTGt8D5Hmtm3/cZ8RUjQ=='
}
```

GET /users/username/charname/stats

Liefert Attributinformationen über einen Charakter eines Benutzeraccounts zurück.

Ressource /users/username/charname/stats
Methode **GET**
Parameter Keine
Einschränkungen Kann nur für Charaktere des eigenen Benutzeraccount angefordert werden.
Beschreibung Liefert Attributinformationen über den Charakter mit Namen „charname“ des Benutzers mit Benutzernamen „username“ zurück.
Rückgabewert JSON-Array mit JSON-Objekten mit jeweils folgenden Attributen:

- **name**
  Name des Attributs
- **value**
  Wert des Attributs

Beispielanfrage GET /users/bob/foo/stats

Beispielantwort

```
[
        { name: 'archery',      value: '78.4' },
        { name: 'cooking',      value: '12.3' },
```

```
        { name: 'smithing',      value: '83.9' }
    ]
```

DELETE /users/username/charname

Löscht einen Charakter eines Benutzeraccounts.

Ressource	/users/username/charname
Methode	**DELETE**
Parameter	Keine
Einschränkungen	Kann nur für Charaktere des eigenen Benutzeraccount angefordert werden.
Beschreibung	Löscht den Charakter mit Namen „charname“ des Benutzers mit Benutzernamen „username“.
Rückgabewert	Kein Rückgabewert. Erfolg ist an Hand des HTTP Statuscodes ersichtlich (Stets *200 OK* im Erfolgsfall).
Beispielanfrage	DELETE /users/bob/foo

PUT /users/username/charname

Legt einen Charakter für einen Benutzeraccount an.

Ressource	/users/username/charname
Methode	**PUT**
Parameter	Keine
Einschränkungen	Kann nur für den eigenen Benutzeraccount angefordert werden.
Beschreibung	Erstellt den Charakter mit Namen „charname“ für den Benutzer mit Benutzernamen „username“.
Rückgabewert	Kein Rückgabewert. Erfolg ist an Hand des HTTP Statuscodes ersichtlich (Stets *200 OK* im Erfolgsfall).
Beispielanfrage	PUT /users/bob/foo

POST /users/username/charname

Ändert Attribute eines bestehenden Charakters eines Benutzeraccounts.

Ressource	/users/username/charname
Methode	**POST**
Parameter	Ein JSON-Objekt mit den zu ändernden Attributen. Momentan sind folgende Attribute definiert: ○ **name** Der Name des Charakters
Einschränkungen	Kann nur für Charaktere des eigenen Benutzeraccounts angefordert werden.

Beschreibung	Ändert Attribute des Charakters mit Namen „charname“ des Benutzers mit Benutzernamen „username“.
Rückgabewert	Kein Rückgabewert. Erfolg ist an Hand des HTTP Statuscodes ersichtlich (Stets *200 OK* im Erfolgsfall).
Beispielanfrage	POST /users/bob/foo HTTP Body des POST-Requests: { name: 'bar' } Nach erfolgreichem Aufruf ist der Charakter ab sofort unter dem Namen *bar* bekannt. Die Ressource für zukünftige Anfragen lautet nun also **/users/bob/bar** statt */users/bob/foo*.

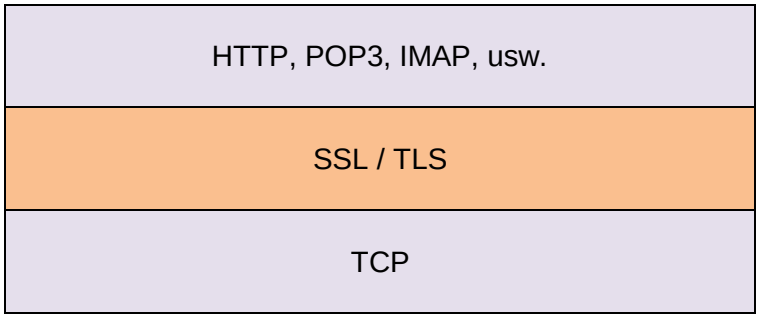
3. Übertragungssicherheit

Die Verbindung zwischen Client und Authentifizierungsserver wird per SSL/TLS verschlüsselt, so daß die Übermittlung der sensitiven Benutzerdaten (sprich, des Passworts) gesichert erfolgt.

3.1 Was ist SSL/TLS

SSL (*Secure Sockets Layer*), heutzutage offiziell unter dem Namen TLS (*Transport Layer Security*) bekannt, ist ein hybrides Verschlüsselungsprotokoll zur sicheren Datenübertragung im Internet. Die Details zu erläutern würde den Rahmen an dieser Stelle sprengen, aber vereinfacht gesagt, stellen Client und Server durch Austausch von Zertifikaten die Identität der Gegenseite sicher und tauschen im Zuge der sog. *Handshake-Phase* einen geheimen Schlüssel für symmetrische Datenverschlüsselung aus. Hiernach werden alle Nutzdaten mit diesem Schlüssel symmetrisch verschlüsselt, wobei in aller Regel AES-128 bzw. AES-256 als Verschlüsselungsverfahren zum Einsatz kommt.

Das Protokoll ist zwischen Anwendungs- und Transportschicht angesiedelt und kann demnach für beliebige Protokolle aus der Anwendungsschicht (HTTP, FTP, SMTP, IMAP, usw.) genutzt wurden, wie folgende Abbildung verdeutlicht:



3.2 Einsatz im Authentifizierungsserver

Der Authentifizierungsserver nutzt SSL, um die Datenverbindungen, über die Clients HTTP-Anfragen senden, zu verschlüsseln, sprich, es handelt sich strikt gesagt um einen HTTP**S**-Server.

Der Server verfügt über ein eigenes Zertifikat (self-signed), welches man bei Bedarf auch im entsprechenden Clientcode gegenprüfen könnte, um sicherzustellen, daß man eine Verbindung mit dem „echten“ Server aufbaut und nicht mit einem Server, welcher sich lediglich als Authentifizierungsserver ausgibt.

4. Authentifizierung

Die eigentliche Authentifizierung eines Clienten ist über HTTP Basic Access Authentication implementiert. Dies bedeutet, ein Client sendet in jeder HTTP-Anfrage seinen Benutzernamen und sein Passwort in einem gesonderten Feld des HTTP-Headers mit. Da die zugrundeliegende Verbindung bereits durch SSL verschlüsselt ist, besteht nicht die Gefahr, daß das Passwort mitgehört werden könnte. Für jedes eingehende HTTP-Request untersucht der Authentifizierungsserver das besagte Feld im HTTP-Header und extrahiert hieraus Benutzernamen und Passwort. Anschließend werden diese auf Gültigkeit geprüft und das eigentliche HTTP-Request nur bearbeitet, wenn Benutzername und Passwort gültig sind und der Benutzer authorisiert ist, die angeforderte Operation auszuführen.

4.1 HTTP Basic Access Authentication

Bei HTTP Basic Access Authentication werden Benutzername und Passwort im Header eines HTTP-Requests übertragen. Das entsprechende Headerfeld trägt den Namen *Authorization.* Benutzername und Passwort werden durch einen Doppelpunkt (‚:‘) getrennt und der so entstandene String wird BASE64-kodiert.

Lautete der Benutzername beispielsweise *foo* und das zugehörige Passwort *bar*, erhielte man nach Anwendung obiger Regeln den BASE64-kodierten String ***Zm9vOmJhcg==***, den man im *Authorization*-Feld des HTTP-Headers übertragen würde:

```
Authorization: Basic Zm9vOmJhcg==
```

Basic steht dabei für die Art und Weise, wie Benutzername und Passwort kodiert werden. Es gibt auch die Möglichkeit, nicht das Klartext-Passwort zu übertragen, sondern nur einen Wert, der sich daraus ableitet (Hash). Für Details sei auf RFC2617 verwiesen.

5. Speicherung von Passwörtern

Für die Validierung von Benutzernamen und Passwörtern muss der Authentifizierungsserver naturgemäß Informationen speichern, die solch eine Validierung ermöglichen.

5.1 Probleme

Aus naheliegenden Gründen ist es eine sehr schlechte Idee, Passwörter als Klartext in einer Datenbank zu speichern. Sollten die Daten je in falsche Hände geraten, sind die Passwörter aller Benutzer kompromittiert.

Eine erste Verbesserung ist das Speichern eines Hashes, erzeugt von einer Hashfunktion (wie beispielsweise SHA, MD und andere) mit dem Klartextpasswort als Eingabe. Auch dies ist jedoch nicht ausreichend, da identische Passwörter identische Hashes erzeugen und schwache Passwörter anfällig sind für Brute-Force bzw. Rainbow-Table Attacken. Vielmehr ist es notwendig für jedes Passwort einen zufälligen Salt-Wert zu generieren und diesen gemeinsam mit dem Hash, erzeugt aus Passwort und Salt, zu speichern.

Um Brute-Force Attacken gänzlich unmöglich zu machen, sollte zusätzlich die PBKDF2 Funktion eingesetzt werden, welche dazu dient, das Erzeugen des Passworthashes künstlich zu verlangsamen.

5.2 Implementierung

Der Authentifizierungsserver speichert für jeden Benutzer einen Hash, der durch die PBKDF2 Funktion erzeugt wird. Als Eingabe erhält die PBKDF2 Funktion das Benutzerpasswort, einen zufällig erzeugten Saltwert, sowie einen Iterationszähler, der angibt, wie oft die PBKDF2 Funktion den erzeugten Hash berechnet und so eine künstliche Verzögerung herbeiführt. Neben dem erzeugten Hashwert müssen auch der erzeugte Saltwert, sowie der Iterationszähler gespeichert werden, damit man aus diesen Werten und einer Passworteingabe den ursprünglichen Hashwert wieder berechnen kann.

Hiermit folgt der Authentifizierungsserver beim Speichern der Passwörter den generell empfohlenen *Best Practices*.

6. AuthToken

Der AuthToken wird, wie in Abschnitt 2 erwähnt, als Teil des Rückgabewerts eines HTTP-GET Requests auf eine Charakterressource zurückgeliefert und dient als Beweis, daß der jeweilige Benutzer der Eigentümer des Charakters ist. Der Client reicht den AuthToken im Zuge der Login-Prozedur an den Gameserver weiter, welcher den AuthToken auf Gültigkeit überprüfen kann. Für den Client bleibt der AuthToken dabei transparent, d.h. die Struktur und der Inhalt sind für den Client unerheblich, da er den Token lediglich weiterreicht.

6.1 Funktionsweise

Der AuthToken muß auf der einen Seite die Echtheit eines Benutzers garantieren, auf der anderen Seite darf diese Information für den Clienten nicht erkennbar oder modifizierbar sein. Andernfalls könnte ein Client sich selbst einen eigenen AuthToken für beliebige Benutzeraccounts bzw. Charaktere erzeugen und diesen an einen Gameserver senden.

Um diesen Anforderungen gerecht zu werden, wird der AuthToken von dem Authentifizierungsserver mit einem geheimen Schlüssel symmetrisch verschlüsselt und in verschlüsselter Form an den Clienten übertragen. Der geheime Schlüssel ist neben dem Authentifizierungsserver auch den Gameservern bekannt (sog. *shared-secret-key*), so daß diese in der Lage sind die verschlüsselten Informationen zu extrahieren. Da Clienten den geheimen Schlüssel nicht kennen, können sie weder existierende AuthToken einsehen oder modifizieren und auch keine eigenen AuthToken erzeugen.

6.2 Zusammensetzung des AuthTokens

Der AuthToken beinhaltet verschiedene Informationen, die es einem Gameserver ermöglichen, die Gültigkeit zu prüfen. Folgende Werte sind enthalten:

- **Benutzername**
  Der Name des Benutzers, für welchen der Token ausgestellt wurde.
- **Charaktername**
  Der Name des Spielcharakters des Benutzers, für welchen der Token ausgestellt wurde.
- **Zeitstempel**
  Ein Zeitstempel der angibt, wann der Token ausgestellt wurde. Der Zeitstempel wird als UNIX TIMESTAMP gespeichert, d.h. in Sekunden, die seit dem 01.01.1970 vergangen sind.
- **Hostname**
  Der Hostname (oder IP-Adresse) des Gameservers für welchen der Token gültig ist.
- **Sessionkey**
  Ein zufällig erzeugter Schlüssel für symmetrische Datenverschlüsselung zwischen Client und Gameserver. Dieses Feld ist optional und nur vorhanden, wenn der Authentifizierungsserver entsprechend konfiguriert ist.

6.3 Verarbeitung durch den Gameserver

Wenn ein Client eine Verbindung zu einem Gameserver aufbaut und im Zuge der Login-Prozedur seinen AuthToken an den Gameserver übermittelt, sind vom Gameserver folgende Schritte durchzuführen, um die Gültigkeit des AuthTokens zu überprüfen:

1. Den als BASE64-kodiert übertragenen AuthToken BASE64-dekodieren.
2. Die so erhaltene Bytefolge mit dem geheimen Schlüssel (*shared-secret-key*) entschlüsseln.
3. Den Benutzernamen aus dem AuthToken extrahieren und mit dem im Zuge der Login-Prozedur übermittelten Benutzernamen vergleichen.
4. Den Charakternamen aus dem AuthToken extrahieren und mit dem im Zuge der Login-Prozedur übermittelten Charakternamen vergleichen.
5. Den Hostname aus dem AuthToken extrahieren und prüfen, ob dieser sich zu einer, zu einem Netzwerkinterface des Gameservers gehörenden, Adresse auflösen lässt. Falls diese Prüfung fehlschlägt, so wurde der AuthToken für einen anderen Gameserver ausgestellt und muß abgelehnt werden.
6. Den Zeitstempel extrahieren und mit der aktuellen Zeit vergleichen. Ist die so erhaltene Differenz zu groß, sprich, der AuthToken zu alt, muß er abgelehnt werden.
7. Ggf. den Sessionkey aus dem AuthToken extrahieren (so vorhanden) und den weiteren Datenaustausch mit dem Clienten symmetrisch verschlüsseln. Ob und wie dies geschieht, muss durch das übergeordnete Netzwerkprotokoll ausgehandelt werden.

7. Quellen

- Building RESTful Web Services with JAX-RS (Oracle)
- HTTP Authentication: Basic and Digest Access Authentication (RFC 2617)
- The Transport Layer Security (TLS) Protocol Version 1.2 (RFC 5246)
- HTTP Over TLS (RFC 2818)
- Specification of PKCS #5 v2.0 (RFC 2898)
- Salted Password Hashing - Doing it Right

7.1 Weiterführendes

- Kerberos Protocol Tutorial
  Leicht verständliche Einführung in das Kerberos-Protokoll, auf dessen Grundlage das Protokoll des Timadorus Authentifizierungsservers entworfen wurde.

- Folien zur Veranstaltung IT-Sicherheit von Prof. Dr. Martin Hübner

Gibt einen guten Einblick in verschiedene hier angesprochene Themen wie AES-Verschlüsselung, Kerberos Protokoll und SSL.