Socket Sniffer

Inhalt

Überblick 1
Bedienung 1
Vorbedingung 2
Benutzung 2
Implementierung 3
Struktur 3
script.js 4
wsTool.js 4
modules 4
Anmerkungen 5

Überblick

Der Socket Sniffer ist ein Debug-Tool, welche die Informationsverarbeitung rund um die Websocket-Schnittstelle der Excon erleichtern soll. Und zwar werden mit Start des Service Managements eine Reihe von dynamischen Websocket-Ports eröffnet, über welche Informationen zu verschiedenen Services angefragt oder empfangen werden können. Der Socket Sniffer identifiziert diese dynamischen Adressen und hilft bei der weiteren Informationsfindung.

Die Anwendung basiert auf einem existierenden Tool von Gerlinde Rudolphus. Diese ermöglichte bereits die Verbindung auf den Port 7300, an welchen sich daraufhin ein Request in Form eines JavaScript-Objektes stellen ließ. Der Server antwortete ebenfalls mit einem JS-Objekt, welches als String ausgegeben wurde. Dieser ließ sich hinsichtlich weiterer Services mit jeweils zugehörigen dynamischen Port-Adressen auswerten.

Ziel dieser Tools ist es insbesondere die „Follow-Up-Verbindung“, also die zweite Stufe der Websocket-Verbindung auf einen der dynamischen Ports zu erleichtern und zum jeweils ausgewählten Service die spezifischen Requests anzubieten. Schließlich werden die teils komplexen Antworten als JS-Objekte des Servers mit diversen Informationen zur Excon in Tabellenform überführt und ermöglichen somit eine einfache Informationsauswertung.

Bedienung

Vorbedingung

Unter Firefox muss unter der URL about:config die Einstellung security.fileuri.strict_origin_policy aktuell auf false gesetzt werden.

Es gilt zu überprüfen, ob Fehler durch Browser-Einstellungen verursacht werden. Ebenso werden dynamische Modul-Imports verwendet. Eine Übersicht welche Browser-Versionen diese unterstützen, findet sich unter folgendem Link: https://caniuse.com/es6-module-dynamic-import

Für eine erfolgreiche WebSocket-Verbindung ist zunächst sicherzustellen, dass das Service Management gestartet wurde. Dies lässt sich in der Konsole durch folgenden Command überprüfen:

```
ps ax|grep svc
```

Ist dies nicht der Fall, lässt sich im Verzeichnis TDS/TDS-build-iteration/bin der Server mit folgenden Command starten:

```
./svcServiceManagement
```

Sollte es bei der Verbindung dennoch zu Komplikationen kommen, kann die Konsolenausgabe Hinweise auf das Fehlverhalten einzelner Services geben.

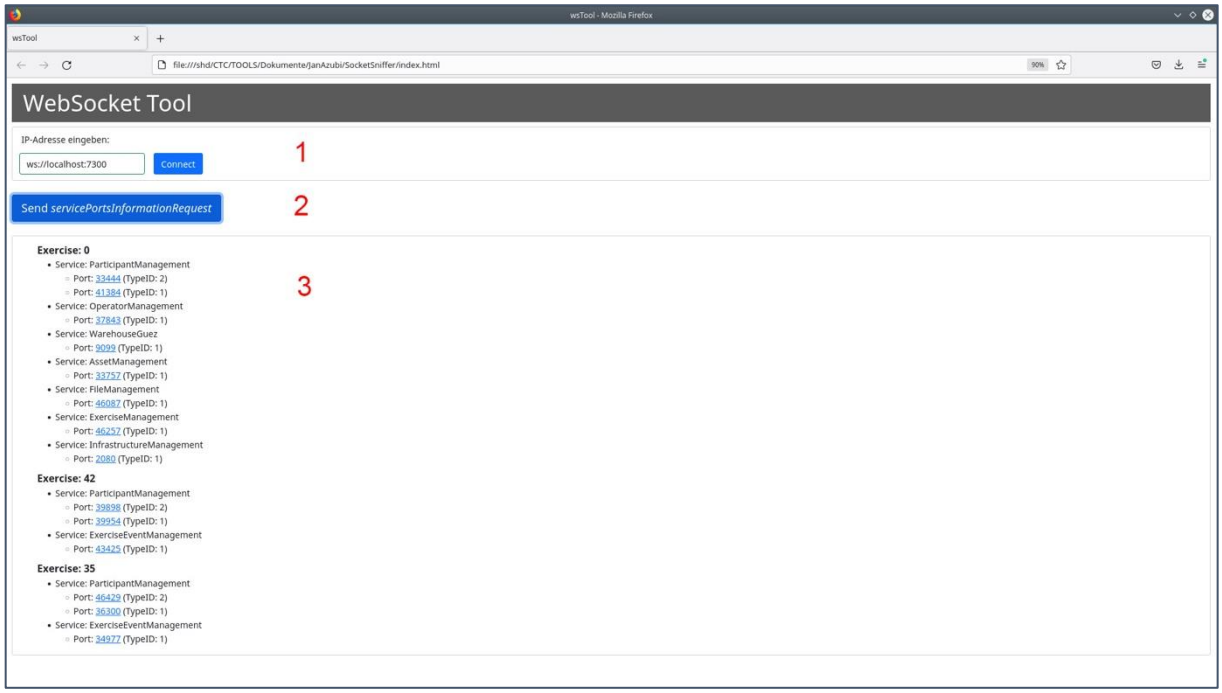
Benutzung

Das Tool gliedert sich vertikal und gibt nach erfolgreicher Durchführung eines Einzelschrittes die Nutzung des Folgeschrittes frei.

1. Mit Aufruf der index.html bietet die Startseite standardgemäß der Verbindung auf den Port ws://localhost:7300 an, welcher den Zugang zum Service Managements darstellt. Mit Connect wird die Websocket-Verbindung hergestellt, welche durch einen Farbwechsel des Adressfeldes bestätigt wird.

2. Daraufhin erweitert sich die Seite um die Möglichkeit einen sog. Service Ports Information Request zu stellen. Dies geschieht per Maus-Click.

3. Dieser Request generiert die Übersicht aller Follow-Up-Services mit zugehörigen dynamischen Port-Informationen.
   Die jeweilige Port-Adresse stellt dabei den Link für die Weiterverbindung zum Service dar. Per Klick öffnet sich eine Follow-Up-Page im neuen Tab.



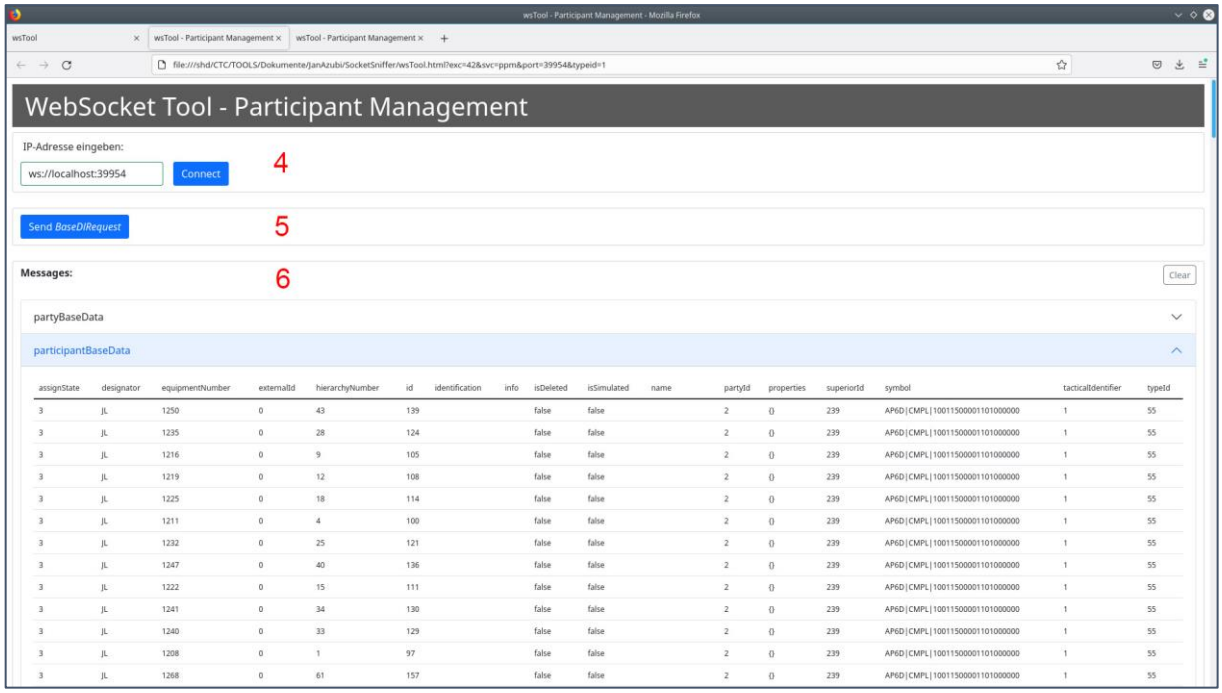
Zum Vergleich ein Screenshot des JSON-Objekts im Rohformat, aus dem die Informationen stammen:

{"data":[{"exerciseId":0,"replayPartitionId":0,"services":[{"name":"ParticipantManagement","nameAbbrevation":"ppm","ports":[{"number":33444,"typeId":2},
{"number":41384,"typeId":1}]},{"name":"OperatorManagement","nameAbbrevation":"opm","ports":[{"number":37843,"typeId":1}]},
{"name":"WarehouseGuez","nameAbbrevation":"whg","ports":[{"number":9099,"typeId":1}]},{"name":"AssetManagement","nameAbbrevation":"asm","ports":
[{"number":33757,"typeId":1}]},{"name":"FileManagement","nameAbbrevation":"flm","ports":[{"number":46087,"typeId":1}]},
{"name":"ExerciseManagement","nameAbbrevation":"exm","ports":[{"number":46257,"typeId":1}]},{"name":"InfrastructureManagement","nameAbbrevation":"ism","ports":
[{"number":2080,"typeId":1}]}]},{"exerciseId":42,"replayPartitionId":0,"services":[{"name":"ParticipantManagement","nameAbbrevation":"ppm","ports":[{"number":39898,"typeId":2},
{"number":39954,"typeId":1}]},{"name":"ExerciseEventManagement","nameAbbrevation":"eem","ports":[{"number":43425,"typeId":1}]}]},
{"exerciseId":35,"replayPartitionId":0,"services":[{"name":"ParticipantManagement","nameAbbrevation":"ppm","ports":[{"number":46429,"typeId":2},{"number":36300,"typeId":1}]},
{"name":"ExerciseEventManagement","nameAbbrevation":"eem","ports":[{"number":34977,"typeId":1}]}]}],"type":"servicePortsInformation","uuid":"UUID"}

4. Die Struktur ähnelt jener der Startseite. Der Header gibt dabei Aufschluss über den ausgewählten Service. Das Adressfeld ist bereits mit der zugehörigen Port-Adresse ausgefüllt, sodass die Verbindung nur noch durch den Connect-Button eigegangen werden muss. Die erfolgreiche Verbindung wird wie zuvor indiziert.

5. Daraufhin ergeben sich alles gültigen Optionen, um Requests zu versenden. Dies geschieht per Button-Click.
   Im Falle des Participant Managements mit Port ID 1 können Positionsdaten empfangen werden. In diesem Fall muss kein Request versendet werden. Stattdessen ist der Port für eingehende Positionsdaten geöffnet.

6. Zuletzt wird jede serverseitige Antwort auf den Request tabellarisch ausgegeben. Die Überschrift sagt aus, um welche Art von Antwort es sich handelt. Je Request werden teilweise mehrere Antworten generiert. Jede einzelne Antwort-Tabelle lässt sich daher einklappen. Im Fall der Positionsdaten wird bei der ersten Nachricht eine Tabelle angelegt, die um alle Folgepositionen erweitert wird.
   Zu jeder Antwort gibt es auch eine Konsolenausgabe, um die ursprüngliche Response einzusehen, aus der die Tabelle generiert wird.



Implementierung

Struktur

Neben html- und css-Dateien ist das Tool entsprechend der Startseite und Follow-Up-Page in zwei Javascript-Datein gegliedert. Auf der Startseite, der script.js, wird die erste Websocket-Verbindung realisiert und die dynamischen Ports ermittelt.

Mit Auswahl einer der Ports, wird der Nutzer auf wsTool.js weitergeleitet. Von dieser Datei wird je nach ausgewähltem Service dynamisch eine weitere Javascript-Datei importiert, um die Informationen des jeweiligen Services auszuwerten.

script.js

Die script.js baut eine Websocket-Verbindung auf und enthält die Funktion, um den ersten Request zu versenden (sendServicePortsInformationRequest()). Beim Erhalt einer serverseitigen Response wird die Funktion ws.ommessage() ausgelöst. Damit wird das empfangene Javascript-Objekt durchschritten und eine Liste aus den Ebenen Exercises, Services und Ports dynamisch generiert. In der Portebene werden Links generiert, die bei Betätigung auf die Folgeseite wsTool.js leiten. Für jeden dieser Links wird ein String konstruiert, der gewählte Übung, Service, Port und PortID enthält. Dieser wird bei der Weiterleitung zur URL hinzugefügt, um das korrekte Modul zur weiteren Auswertung zu importieren.

wsTool.js

Zu Beginn wird die Übergabe-URL in die Teilinformationen zerlegt, mit deren Information ein spezifisches Modul importiert wird.

Die URL /wsTool.html?exc=0&svc=opm&port=37843&typeid=1 verweist etwa auf den Service opm (Operator Management) mit Type ID 1 und führt somit zum Import des Moduls opm1.js.

Wie die script.js ruft die wsTool.js Funktionen zur Websocket-Verbindung, Request-Sendung und dynamischem Response-Handling auf. Die Implementierung dieser Funktion variiert je nach importiertem Modul. Dadurch werden je nach ausgewähltem Service nur passende Requests angeboten und die Auswertung der serverseitigen Antworten auf die Struktur der jeweiligen JavaScript-Objekte zugeschnitten.

modules

Alle Module sind gleich strukturiert, wobei sich, wie bereits beschrieben, nur die jeweilige Implementierung der Funktion unterscheidet. Folgende Funktionen werden durch wsTool.js aufgerufen und sind daher in jedem Modul vertreten:

getLongName()

Diese Funktion gibt den vollständigen Namen zur entsprechenden Abkürzung aus. Beispiel: opm = Operator Management.

createButtons()

Durch diese Funktion werden die zulässigen Requests in Form von Buttons zur Verfügung gestellt. Hinter jedem Button verbirgt sich eine spezifische Nachricht im JSON-Format, die per Websocket zum Server geschickt wird.

decryptMessage ()

Die JSON-formatierte Antwort des Servers wird hiermit in Tabellenform überführt. Das JSON-Objekt enthält wiederum einen Array aus JSON-Objekten. Über diesen wird iteriert, wobei die Keys die Kopfzeile der Tabelle bilden. Für jedes Objekt und dessen Values im Array wird eine Zeile angelegt. Besitzen spätere Objekte im Array abweichende Keys wird die Kopfzeile dynamisch erweitert und die Values werden in der richtigen Spalte eingefügt.

Für einige Services gibt es besondere Verzweigungen, wobei etwas Base64- codierte Einträge als Bilder decodiert werden.

Anmerkungen

Für das Exercise Event Managment habe ich keinen gültigen Request, der eine serverseitige Antwort generiert, gefunden. Zeitweise schien die Verbindung zum WebSocket trotz korrekter IP nicht zu funktionieren.

Es wurden folgende Requests auf die jeweiligen Services getestet:

- servicePortsInformationRequest
- baseDlRequest
- exerciseInformationRequest
- pdfListRequest
- equipmentTypesRequest
- baseDlRequest
- exercisesRequest
- participantsRequest
- participantRegisterInExconRequest