IoT4CPS
Trustworthy IoT for CPS

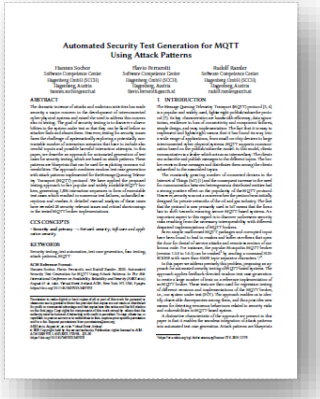
Automated Security Testing

Laboratory Demonstrator

Rudolf Ramler
Software Competence Center Hagenberg GmbH (SCCH)
rudolf.ramler@scch.at | +43 50 343 872

Background

- A demonstrator for automated security testing based on attack patterns has been developed as part of the "ICT of the Future" project IoT4CPS
- The demonstrator named *MqttRazzer* is a framework for generating random tests including security attacks for or via an MQTT broker
- For further details see
  - Sochor, H., Ferrarotti, F., Ramler, R.: An Architecture for Automated Security Test Case Generation for MQTT Systems. In International Conference on Database and Expert Systems Applications (pp. 48-62). Springer, 2020.
  - Sochor, H., Ferrarotti, F., Ramler, R.: Automated security test generation for MQTT using attack patterns. In Proceedings of the 15th International Conference on Availability, Reliability and Security (pp. 1-9). ACM, 2020.



Automated Security Test Generation for MQTT Using Attack Patterns

Hannes Sochor
Software Competence Center
Hagenberg GmbH (SCCH)
Hagenberg, Austria
hannes.sochor@scch.at

Flavio Ferrarotti
Software Competence Center
Hagenberg GmbH (SCCH)
Hagenberg, Austria
flavio.ferrarotti@scch.at

Rudolf Ramler
Software Competence Center
Hagenberg GmbH (SCCH)
Hagenberg, Austria
rudolf.ramler@scch.at

ABSTRACT

[illegible]

CCS CONCEPTS

• Security and privacy → Network security; Software and application security.

KEYWORDS

[illegible]

1 INTRODUCTION

[illegible]

An Architecture for Automated Security Test Case Generation for MQTT Systems

Hannes Sochor[✉], Flavio Ferrarotti, and Rudolf Ramler

Software Competence Center Hagenberg,
Softwarepark 21, 4232 Hagenberg, Austria
{hannes.sochor,flavio.ferrarotti,rudolf.ramler}@scch.at

Abstract. Message Queuing Telemetry Transport (MQTT) protocol is among the preferred publish/subscribe protocols used for Machine-to-Machine (M2M) communication and Internet of Things (IoT). Although the MQTT protocol itself is quite simple, the concurrent iteration of brokers and clients and its intrinsic non-determinism, coupled with the diversity of platforms and programming languages in which the protocol is implemented and run, makes the necessary task of security testing challenging. We address precisely this problem by proposing an architecture for security test generation for systems relying on the MQTT protocol. This architecture enables automated test case generation to reveal vulnerabilities and discrepancies between different implementations. As a desired consequence, when implemented, our architectural design can be used to uncover erroneous behaviours that entail latent security risks in MQTT broker and client implementations. In this paper we describe the key components of our architecture, our prototypical implementation using a random test case generator, core design decisions and the use of security attacks in testing. Moreover, we present first evaluations of the architectural design and the prototypical implementation with encouraging initial results.

Keywords: Security testing · Automated testing · IoT · MQTT

1 Introduction

The Internet of Things (IoT) connects millions of devices of different cyber-physical systems (CPSs). However, the increasing connectivity of cyber-physical objects in critical environments like industrial production, public transport, or energy distribution, also increases the vulnerability of these IoT-based, cyber-physical systems. Ensuring security in such environments introduces major challenges and high effort for quality assurance. Conventional approaches for quality assurance and testing are characterized by manual tasks and script-based automation attempts, which limit the flexibility necessary for adapting to heterogeneous, dynamic and self-organizing IoT-based systems.

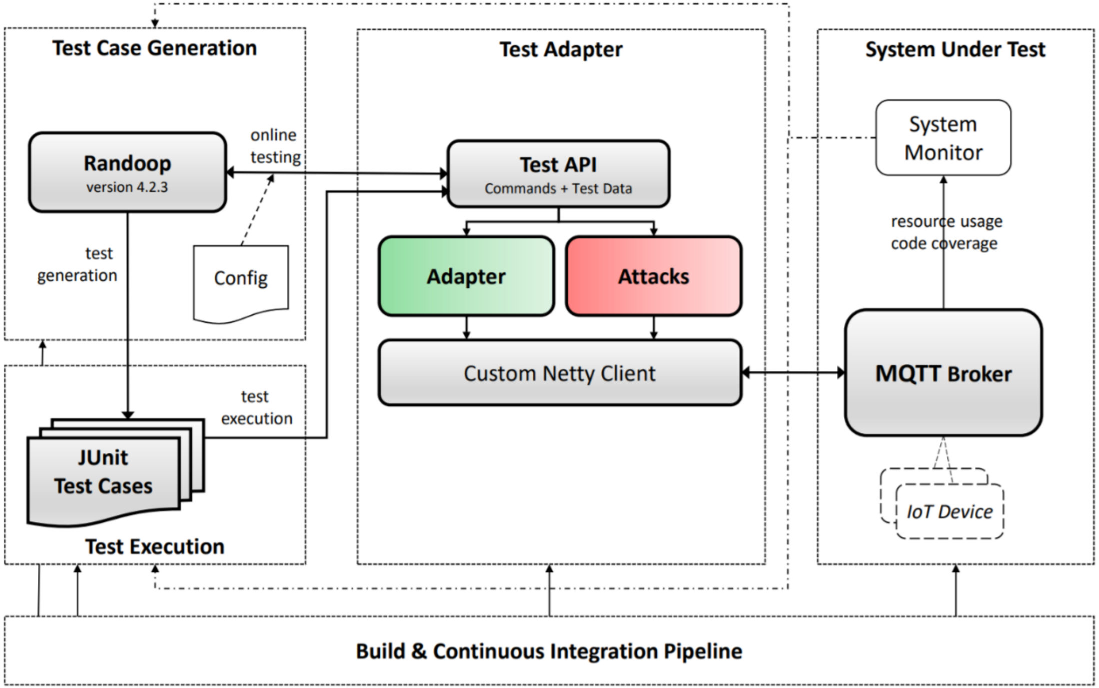
Adequate quality assurance and testing approaches are required to cope with a huge number and variety of interaction scenarios typical for IoT systems.

© Springer Nature Switzerland AG 2020
G. Kotsis et al. (Eds.): DEXA 2020 Workshops, CCIS 1285, pp. 48–62, 2020.
https://doi.org/10.1007/978-3-030-59028-4_5

Architecture Overview

Core Components

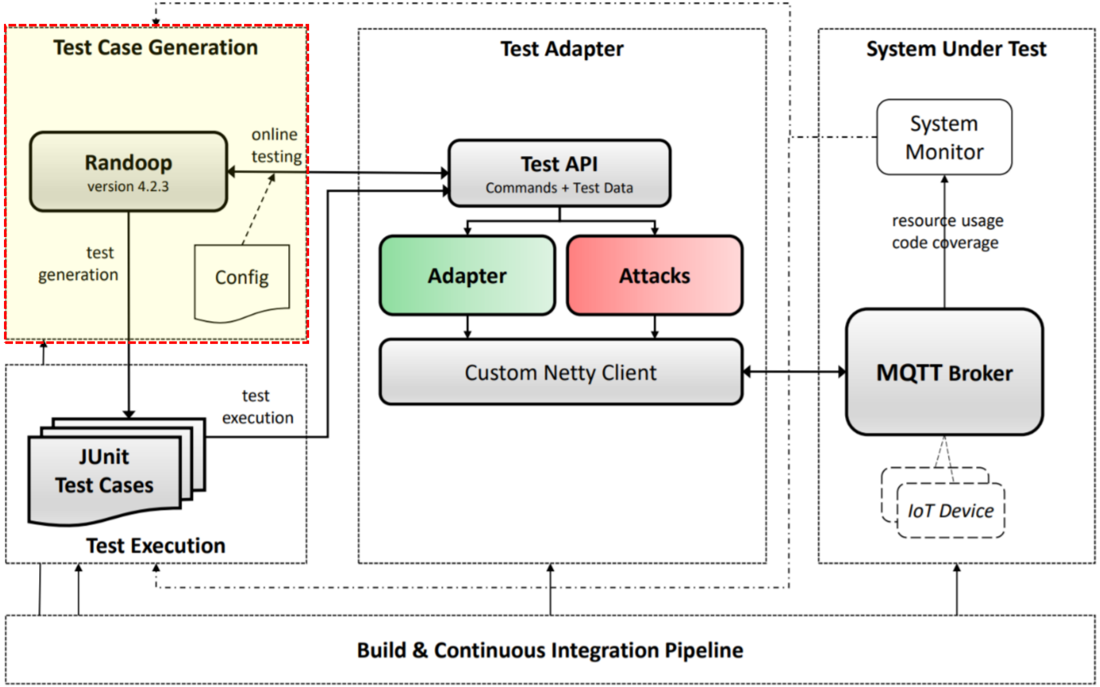
1. Test case generation
2. Test execution
3. Test adapter
4. System under test



Architecture Overview

Test Case Generation

- Open source tool *Randoop*[1] a feed-back directed random test generator
- Randoop uses the *test adapter* to access the MQTT broker
- *Config* specifies which adapter methods are used in generating test sequences
- Randoop outputs generated sequences as *JUnit test cases*

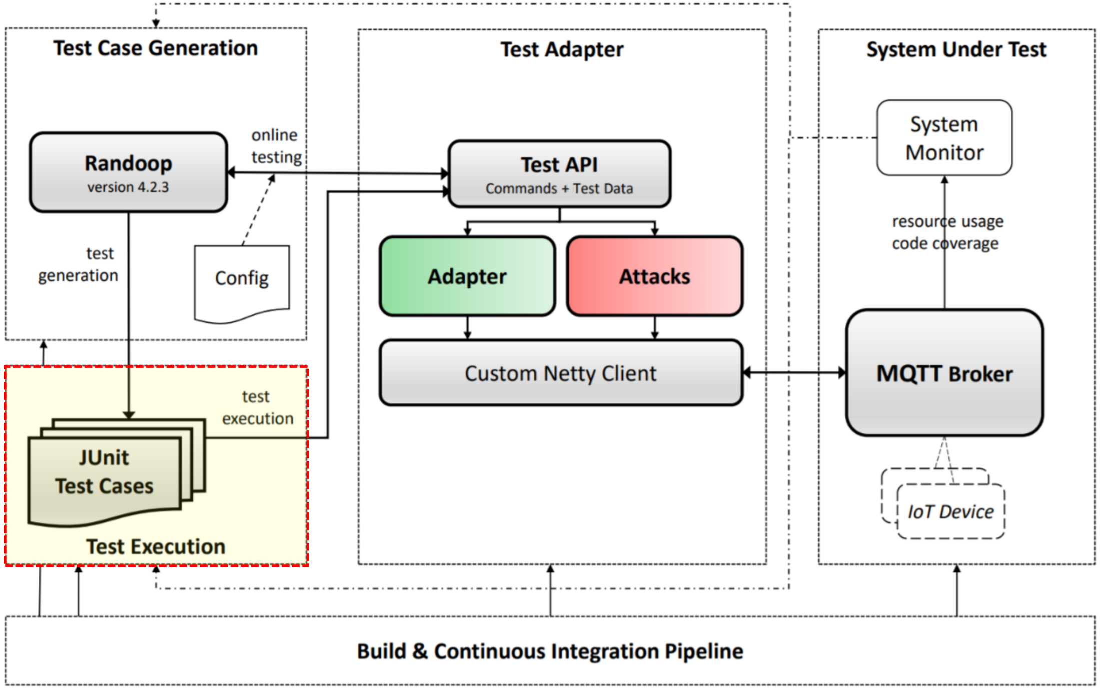


[1] https://randoop.github.io/randoop/

Architecture Overview

Test Execution

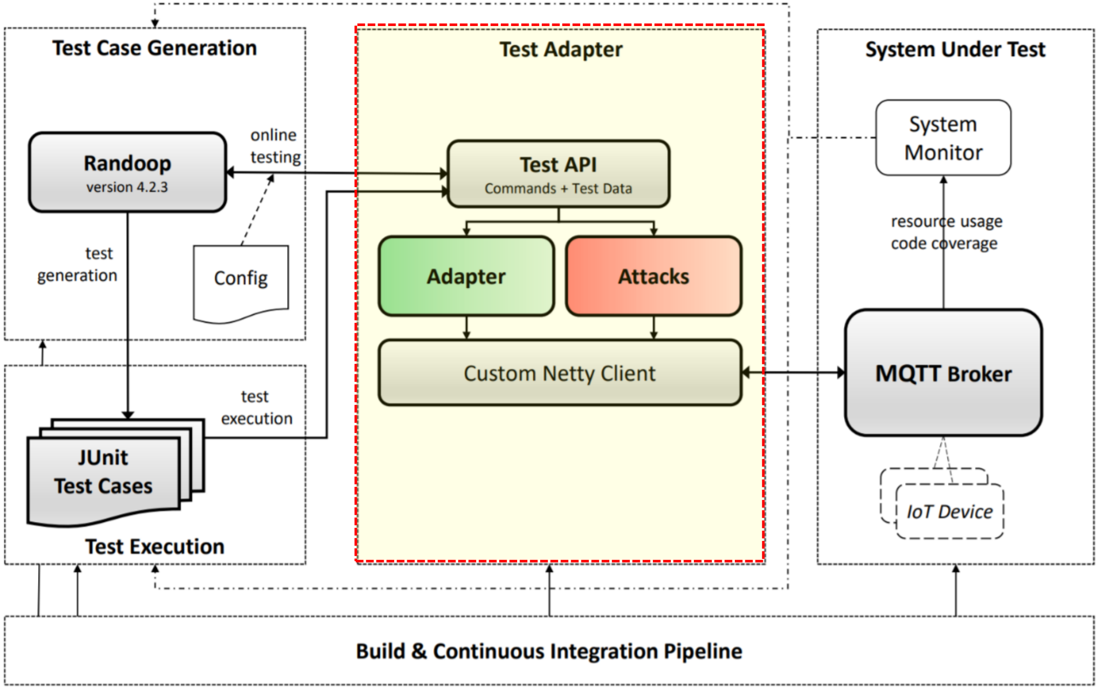
- Sequences generated by Randoop are stored as JUnit test cases
- JUnit test runner is used to execute the tests
- Tests exercise the adapter to access the MQTT broker



Architecture Overview

Test Adapter

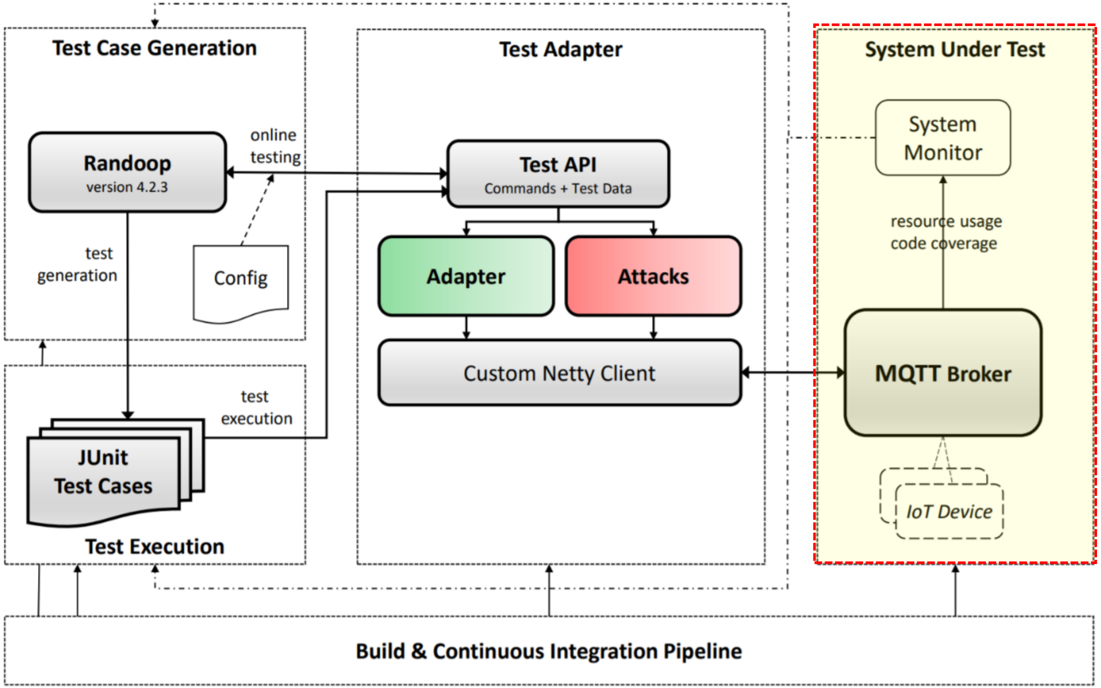
- API for interacting with an MQTT broker in testing
- The API provides
  1. Valid MQTT Commands
  2. Attacks based on invalid or malformed commands and command sequences
- A modified *Netty lib* is used to communicate with the broker; security checks have been removed to allow sending malformed/invalid data



Architecture Overview

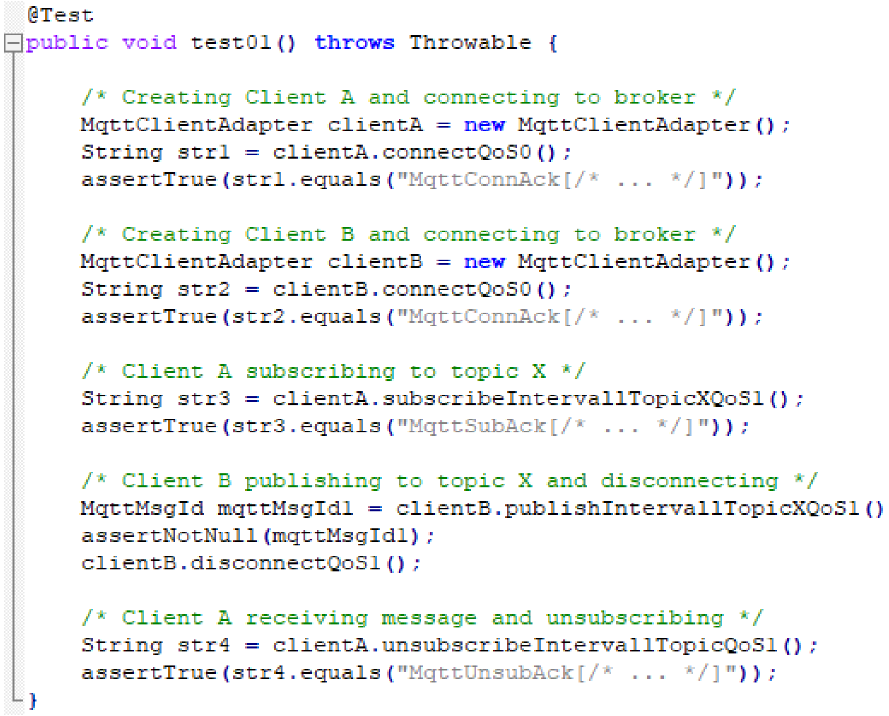
System Under Test

- The SUT is an IoT system or device accessible via MQTT and/or an MQTT broker
- In test generation and execution, the SUT is accessed via a test adapter
- System specific monitoring (e.g. MQTT broker loggin) is optionally used to directly observe the SUT‘s behavior

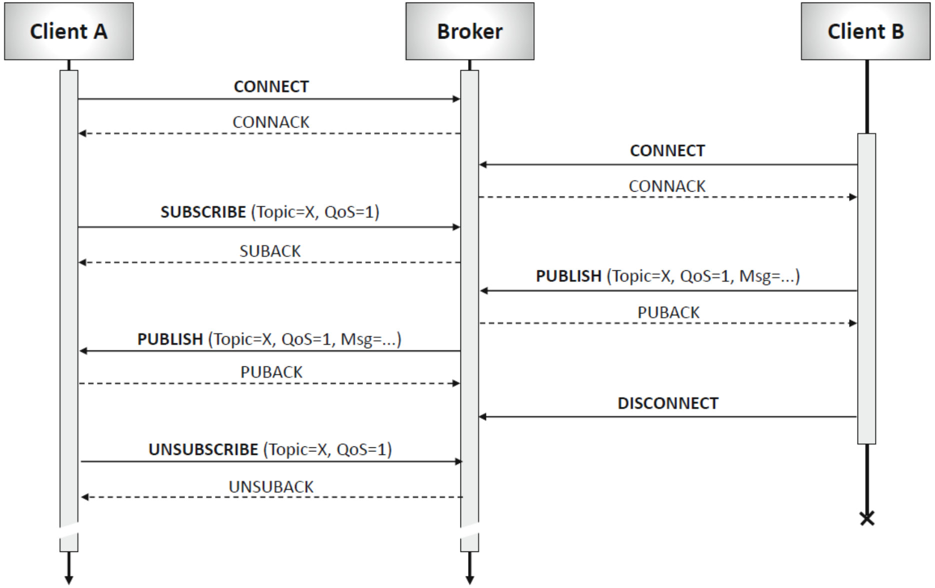


Example Generated Test Case

Covered scenario: Two clients interacting with a MQTT broker



```
@Test
public void test01() throws Throwable {

    /* Creating Client A and connecting to broker */
    MqttClientAdapter clientA = new MqttClientAdapter();
    String str1 = clientA.connectQoS0();
    assertTrue(str1.equals("MqttConnAck[/* ... */]"));

    /* Creating Client B and connecting to broker */
    MqttClientAdapter clientB = new MqttClientAdapter();
    String str2 = clientB.connectQoS0();
    assertTrue(str2.equals("MqttConnAck[/* ... */]"));

    /* Client A subscribing to topic X */
    String str3 = clientA.subscribeInterval1TopicXQoS1();
    assertTrue(str3.equals("MqttSubAck[/* ... */]"));

    /* Client B publishing to topic X and disconnecting */
    MqttMsgId mqttMsgId1 = clientB.publishInterval1TopicXQoS1()
    assertNotNull(mqttMsgId1);
    clientB.disconnectQoS1();

    /* Client A receiving message and unsubscribing */
    String str4 = clientA.unsubscribeInterval1TopicQoS1();
    assertTrue(str4.equals("MqttUnsubAck[/* ... */]"));
}
```



Client A
Broker
Client B
CONNECT
CONNACK
CONNECT
CONNACK
SUBSCRIBE (Topic=X, QoS=1)
SUBACK
PUBLISH (Topic=X, QoS=1, Msg=...)
PUBACK
PUBLISH (Topic=X, QoS=1, Msg=...)
PUBACK
DISCONNECT
UNSUBSCRIBE (Topic=X, QoS=1)
UNSUBACK

List of Demonstration Attacks

- Sample attacks have been derived from common attack patterns (e.g. CAPEC, CVE)
- Following attacks have been implemented as part of the test adapter

#	Description
0	Check if the broker accepts two clients with the same id
1	Invalid length of variable header (+1)
2	Invalid length of variable header (-1)
4	send publish message with payload size of 128MB
5	Subscribe without payload
7	subscribe with invalid wildcard in topic name
8	subscribe with escape sequences in topic name
9	publish with escape sequences in payload
10	publish with wildcards in topic name
11	connect with invalid QoS (Both QoS Bits set -> QoS=3)
13	connect with long client identifier
14	connect with invalid protocol specifier (protocol="MQQT")
15	connect with invalid protocol version (protocol ver="42")
17	connect with bad will flag combination
18	connect with usr/pwd flag set but without giving credentials
19	trigger keep alive (keepAlive=1)
20	connect with big keep alive (keepAlive=INT_MAX)
22	connect with invalid client identifier
23	subscribe with huge '/' payload
24	connect with empty client identifier
25	connect bad Username (username UTF16 encoded)

Demonstrator: Tutorial

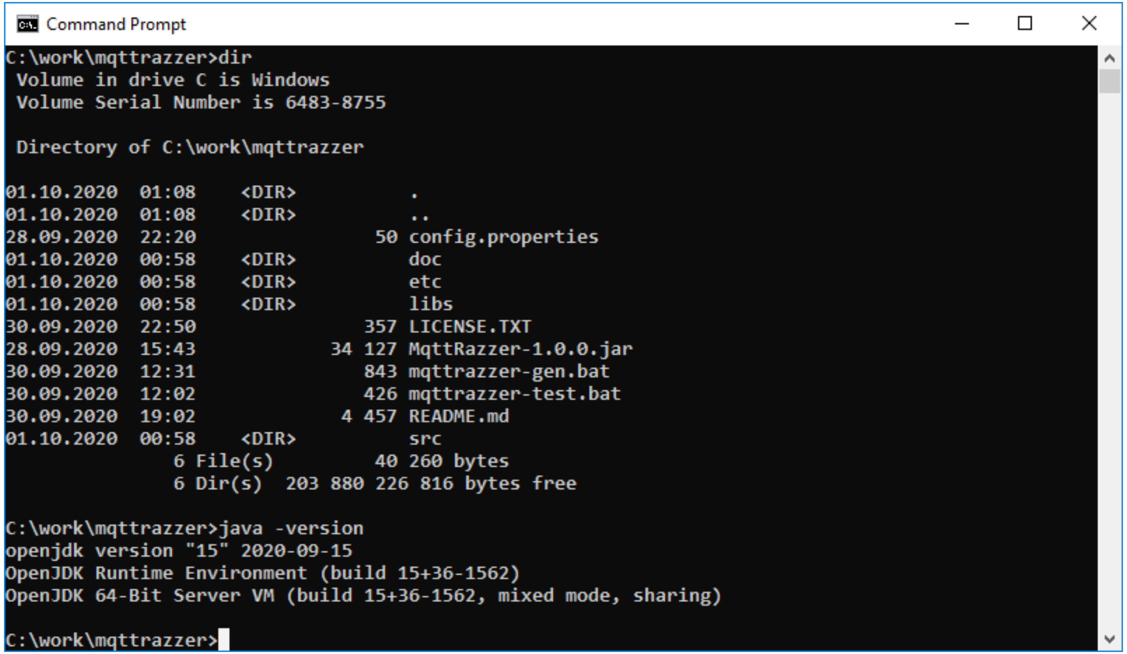
- Required setup for generating and running tests
  - JDK 1.8+
  - MQTT broker running (default is localhost:1883)

- Test Generation
  - Usage: **`mqttrazzer-gen.bat MethodList Timeout`**
  - Example: `mqttrazzer-gen.bat etc\methods_MqttSingleClientAdapter.txt 10`

- Test Execution
  - Usage: **`mqttrazzer-test.bat`**

Demonstator: Step 1 – Setup

- Mosquitto Broker running MQTT v3.1.1 broker
- Java OpenJDK 15
- Current working directory: `c:\work\mqttrazzer`



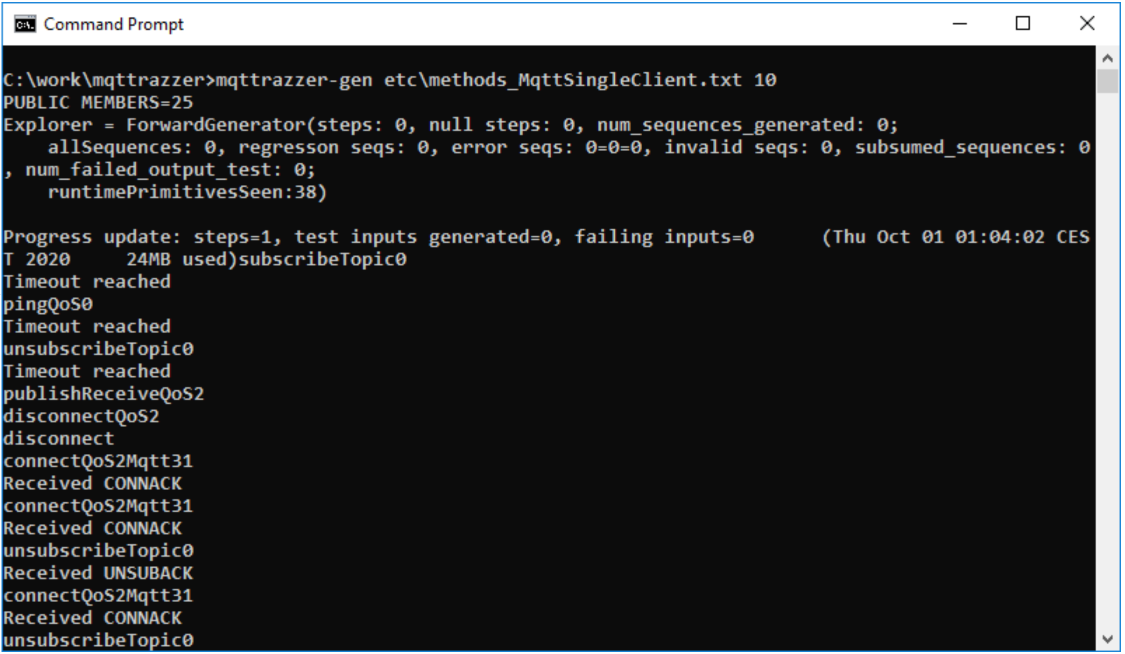
```
C:\work\mqttrazzer>dir
 Volume in drive C is Windows
 Volume Serial Number is 6483-8755

 Directory of C:\work\mqttrazzer

01.10.2020  01:08    <DIR>          .
01.10.2020  01:08    <DIR>          ..
28.09.2020  22:20                50 config.properties
01.10.2020  00:58    <DIR>          doc
01.10.2020  00:58    <DIR>          etc
01.10.2020  00:58    <DIR>          libs
30.09.2020  22:50               357 LICENSE.TXT
28.09.2020  15:43            34 127 MqttRazzer-1.0.0.jar
30.09.2020  12:31               843 mqttrazzer-gen.bat
30.09.2020  12:02               426 mqttrazzer-test.bat
30.09.2020  19:02             4 457 README.md
01.10.2020  00:58    <DIR>          src
               6 File(s)         40 260 bytes
               6 Dir(s)  203 880 226 816 bytes free

C:\work\mqttrazzer>java -version
openjdk version "15" 2020-09-15
OpenJDK Runtime Environment (build 15+36-1562)
OpenJDK 64-Bit Server VM (build 15+36-1562, mixed mode, sharing)

C:\work\mqttrazzer>
```

Demonstrator: Step 2a – Test Generation

Running ***mqttrazzer-gen.bat*** with list of adapter methods given in *methods_MqttSingleClient.txt* for a time limit of *10* seconds

Randoop test generator is started

Log output produced by test adapter from communication with MQTT broker (commands sent and response received)



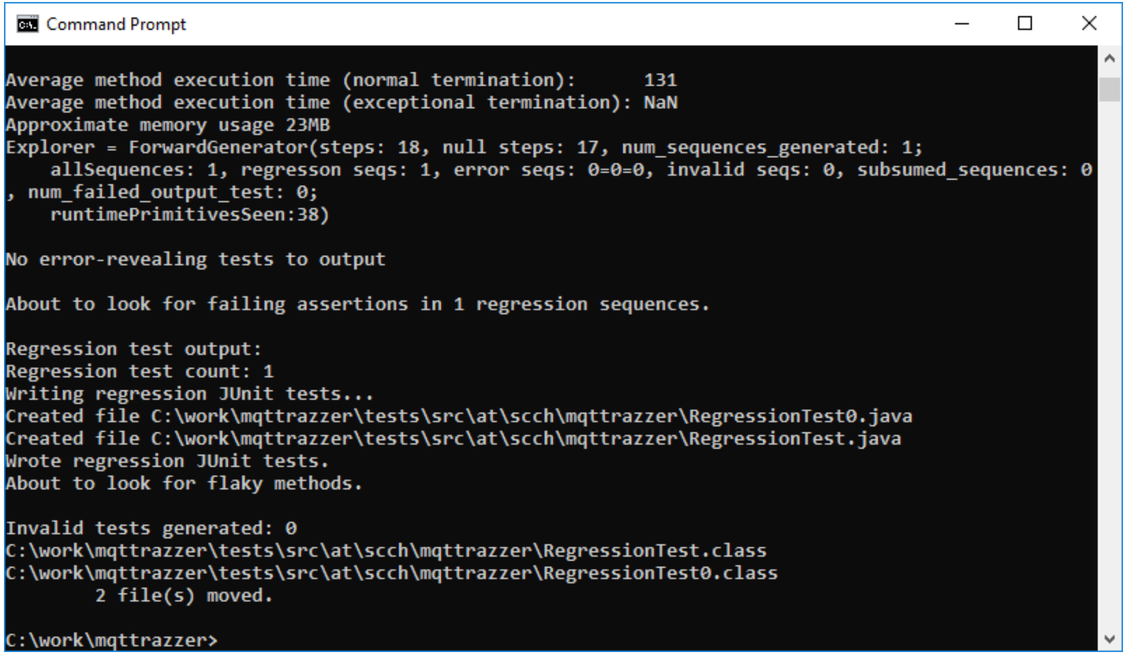
```
C:\work\mqttrazzer>mqttrazzer-gen etc\methods_MqttSingleClient.txt 10
PUBLIC MEMBERS=25
Explorer = ForwardGenerator(steps: 0, null steps: 0, num_sequences_generated: 0;
    allSequences: 0, regresson seqs: 0, error seqs: 0=0=0, invalid seqs: 0, subsumed_sequences: 0
, num_failed_output_test: 0;
    runtimePrimitivesSeen:38)

Progress update: steps=1, test inputs generated=0, failing inputs=0      (Thu Oct 01 01:04:02 CES
T 2020     24MB used)subscribeTopic0
Timeout reached
pingQoS0
Timeout reached
unsubscribeTopic0
Timeout reached
publishReceiveQoS2
disconnectQoS2
disconnect
connectQoS2Mqtt31
Received CONNACK
connectQoS2Mqtt31
Received CONNACK
unsubscribeTopic0
Received UNSUBACK
connectQoS2Mqtt31
Received CONNACK
unsubscribeTopic0
```

Demonstrator: Step 2b – Test Generation Results

Randoop test generation results; summary about explored sequences

Source files containing JUnit test cases written by Randoop

Class files after successful compilation moved to `tests\bin`

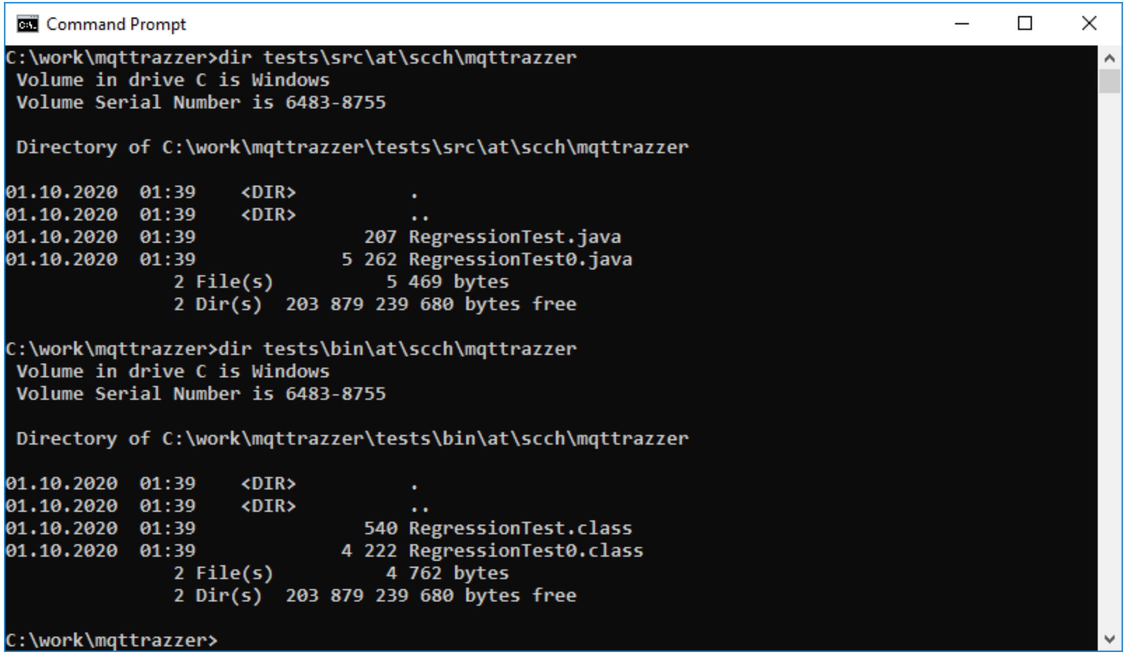


```
Average method execution time (normal termination):      131
Average method execution time (exceptional termination): NaN
Approximate memory usage 23MB
Explorer = ForwardGenerator(steps: 18, null steps: 17, num_sequences_generated: 1;
    allSequences: 1, regresson seqs: 1, error seqs: 0=0=0, invalid seqs: 0, subsumed_sequences: 0
, num_failed_output_test: 0;
    runtimePrimitivesSeen:38)

No error-revealing tests to output

About to look for failing assertions in 1 regression sequences.

Regression test output:
Regression test count: 1
Writing regression JUnit tests...
Created file C:\work\mqttrazzer\tests\src\at\scch\mqttrazzer\RegressionTest0.java
Created file C:\work\mqttrazzer\tests\src\at\scch\mqttrazzer\RegressionTest.java
Wrote regression JUnit tests.
About to look for flaky methods.

Invalid tests generated: 0
C:\work\mqttrazzer\tests\src\at\scch\mqttrazzer\RegressionTest.class
C:\work\mqttrazzer\tests\src\at\scch\mqttrazzer\RegressionTest0.class
        2 file(s) moved.

C:\work\mqttrazzer>
```

Demonstrator: Step 3a – Test Execution

Java source files containing JUnit test cases written by Randoop

Class files after successful compilation **ready for execution**



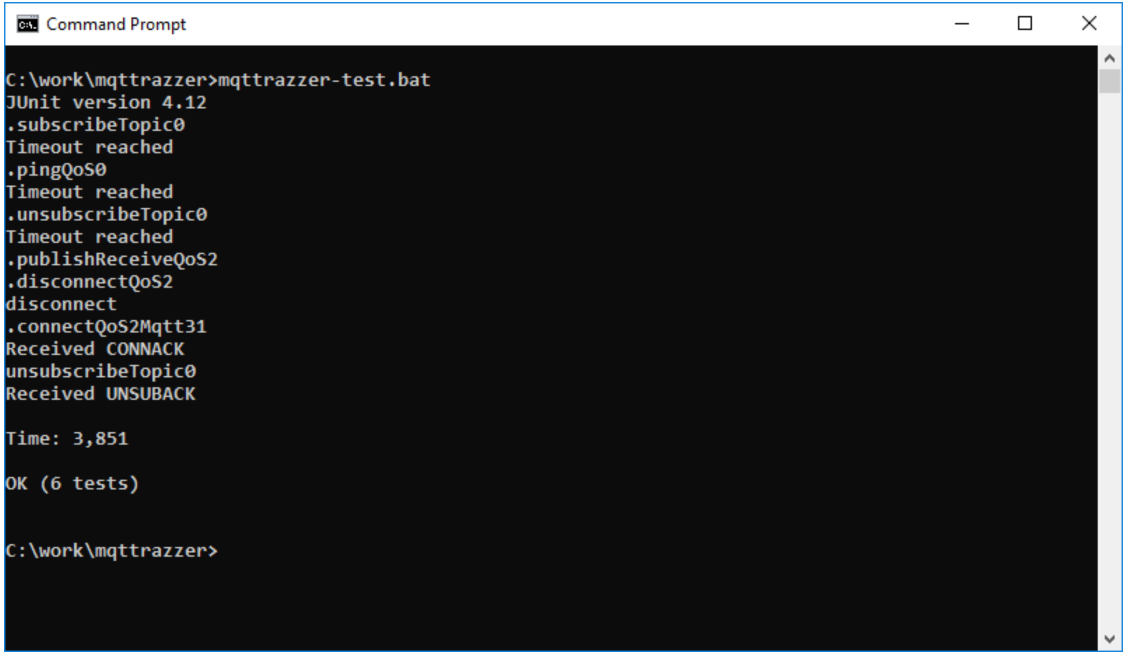
```
C:\work\mqttrazzer>dir tests\src\at\scch\mqttrazzer
 Volume in drive C is Windows
 Volume Serial Number is 6483-8755

 Directory of C:\work\mqttrazzer\tests\src\at\scch\mqttrazzer

01.10.2020  01:39    <DIR>          .
01.10.2020  01:39    <DIR>          ..
01.10.2020  01:39               207 RegressionTest.java
01.10.2020  01:39             5 262 RegressionTest0.java
               2 File(s)          5 469 bytes
               2 Dir(s)  203 879 239 680 bytes free

C:\work\mqttrazzer>dir tests\bin\at\scch\mqttrazzer
 Volume in drive C is Windows
 Volume Serial Number is 6483-8755

 Directory of C:\work\mqttrazzer\tests\bin\at\scch\mqttrazzer

01.10.2020  01:39    <DIR>          .
01.10.2020  01:39    <DIR>          ..
01.10.2020  01:39               540 RegressionTest.class
01.10.2020  01:39             4 222 RegressionTest0.class
               2 File(s)          4 762 bytes
               2 Dir(s)  203 879 239 680 bytes free

C:\work\mqttrazzer>
```

Demonstrator: Step 3b – Test Execution Results

Batch file `mqttrazzer-test.bat` executing JUnit test runner

Log output showing MQTT commands and responses from broker; log produced by adapter called from executed JUnit tests

Successful execution of generated tests (i.e. no deviations found in regression test run)



```
C:\work\mqttrazzer>mqttrazzer-test.bat
JUnit version 4.12
.subscribeTopic0
Timeout reached
.pingQoS0
Timeout reached
.unsubscribeTopic0
Timeout reached
.publishReceiveQoS2
.disconnectQoS2
disconnect
.connectQoS2Mqtt31
Received CONNACK
unsubscribeTopic0
Received UNSUBACK

Time: 3,851

OK (6 tests)


C:\work\mqttrazzer>
```

Evaluation Results

	Mosquitto	Moquette	ActiveMQ	emqx	VerneMQ
URL	https://mosquitto.org	https://github.com/andsel/moquette	https://activemq.apache.org	https://www.emqx.io	https://vernemq.com
Version	1.6.8	0.13	5.15.12	4.0.6	1.10.2
Errors	reference	>500	219	198	585
Failures	reference	>300	18	64	0

Comparison of behavior of MQTT broker with reference implementation (Mosquitto) by running regression tests generated for reference on other broker implementations

Analysis results[1]: **28 Security relevant issues discovered**

[1] Sochor, H., Ferrarotti, F., Ramler, R.: Automated security test generation for MQTT using attack patterns. In Proceedings of the 15th International Conference on Availability, Reliability and Security (pp. 1-9). ACM, 2020.

For further information please contact:

Rudolf Ramler

rudolf.ramler@scch.at

Software Competence Center Hagenberg Gmbh,
Austria, https://www.scch.at

Federal Ministry
Republic of Austria
Climate Action, Environment,
Energy, Mobility,
Innovation and Technology

Projectpartner

The IoT4CPS project is partially funded by the “ICT of the Future” Program of the FFG and the BMK.