RabbitMQ 메시지 퍼징에 관한 연구

손상진, 서민지, 권순홍, 최서윤, 이종혁*
상명대학교 소프트웨어학과
paran_son@outlook.com, alsw17317@gmail.com, soonhong@pel.smuc.ac.kr,
seoyun@pel.smuc.ac.kr, jonghyouk@pel.smuc.ac.kr

Study on RabbitMQ Message Fuzzing

Sangjin Son, Minji Seo, Soonhong Kwon, Seoyun Choi, Jong-Hyouk Lee*
Department of Software, Sangmyung University

요 약

대용량 네트워크 트래픽 처리를 위해 메시지 브로커는 널리 사용되는 기법이다. 이미 다양한 인터넷 서비스에서 널리 사용되고 있는 메시지 브로커는 공격자들의 주요한 공격 목표가 된다. 메시지 브로커의 취약점을 통해 시스템을 장악하거나 임의의 메시지를 악의적으로 변경하는 공격을 수행할 수 있다. 본 논문에서는 범용 메시지 브로커로 널리 사용되는 RabbitMQ에 메시지 퍼징 적용 방안을 소개한다. RabbitMQ의 특징과 동작 과정을 살펴보고, 메시지 퍼징 툴인 mqtt_fuzz를 활용해 퍼징에 필요한 환경 설정 및 분석 방법을 설명한다.

1. 서 론

최근 통신 네트워크의 발전에 따라 네트워크 트래픽이 증가하고 있는 추세이다. 트래픽에 따른 메시지의 사용량 증가는 시스템 장애, 서버 부하, 데이터 손실 등의 문제를 초래할 수 있다. 메시지 브로커로는 Kafka, ActiveMQ, ZeroMQ, RabbitMQ등이 있으며, 이 중 RabbitMQ는 다양한 언어와 프로토콜을 지원함으로써 다양한 서비스에서 사용되고 있다. RabbitMQ와 같은 소프트웨어의 보안 취약점을 통한 공격이 꾸준히 발생하고 있다. 이러한 문제를 해결하기 위해 초기에 취약점을 탐지하고 보안하기 위한 취약점 분석 기법에 대한 중요성이 높아지고 있다. 취약점 분석 기법으로는 코드 검토, 퍼징, 오염 분석, 기호 실행 등의 방법이 있다. 본 논문에서는 트래픽에 따른 메시지의 사용량 증가로 인해 발생되는 문제(e.g., 시스템 장애, 서버 부하, 데이터 손실 등)를 해결하기 위해 사용되는 RabbitMQ에 메시지 퍼징을 적용하는 방법에 대해 설명한다.

본 논문의 2장 관련 연구에서는 취약점 분석 기법, 메시지 프로토콜과 메시지 브로커인 RabbitMQ에 대해 설명하며, 3장에서는 MQTT 메시지 퍼징을 RabbitMQ에 적용하는 방법에 대해 설명한다. 4장에서는 본 논문의 결론을 맺는다.

2. 관련 연구

2.1 취약점 분석 기법

소프트웨어는 우리 생활의 모든 곳에 활용되고 있다. 다양한 분야에서 다량의 소프트웨어 배포가 이루어지고 있으며 소프트웨어의 품질과 복잡도 역시 날로 늘어가는 추세이다. 그러나 개발자가 예기치 못한 코드 내 결함은 소프트웨어의 잠재적 취약점이 될 수 있으며, 이러한 취약점은 시스템 가용성을 침해하거나 민감한 정보를 탈취하는 공격으로 이어질 수 있다. 따라서 소프트웨어의 취약점 탐지 및 분석은 정보보호에 있어 매우 필수적인 일이며, 이에 따라 소프트웨어 취약점 분석 기법에 관한 활발한 연구가 요구된다 [1]. 취약점 분석을 위한 기법에는 여러 가지가 존재한다. 그중 가장 대표적으로 코드 검토(Code Review), 퍼징(Fuzzing), 오염 분석(Taint Analysis), 기호 실행(Symbolic Execution)이 있다. 각 기법에 대한 설명은 다음과 같다.

코드 검토(Code Review)는 소프트웨어의 구현 단계에서 소스코드의 보안 취약점을 수동으로 검사하는 기법이다. 나머지 방법들과 다르게 정확한 결함의 위치를 파악할 수 있다는 장점이 있지만, 대규모 소프트웨어의 경우 코드를 수동으로 검토하는 것은 많은 시간을 요구한다. 또한 컴파일된 바이너리를 직접 실행하지 않으므로 런타임 에러를 발견하기에도 부적절하다 [2].

퍼징(Fuzzing)은 프로그램 실행 시 무작위 데이터를 입력하여 프로그램이 정상적으로 동작하지 않고 크래시(Crash)를 일으키는 예외를 탐지하는 무작위 테스팅 기법이다. 퍼징은 대상에 대한 기반 지식의 유무에 따라 무작위 데이터를 입력하는 덤(Dumb) 방식과 스마트(Smart) 방식으로 나뉘며, 데이터 처리 방법에 따라 생성(Generation)과 변이(Mutation)로 나뉜다. 고전적인 블랙박스 테스트 방식의 경우 소스코드에 대한 정보가 없어도 테스트할 수 있다는 장점이 있지만, 코드 커버리지(Code Coverage)가 낮다는 단점이 있다 [3]. 퍼징 관련 툴로는 Melkor, AFL Fuzzer, Peach Fuzzer 등이 있다.

오염 분석(Taint Analysis)은 프로그램 실행 시 외부로부터 입력되는 데이터를 오염된 값으로 판단하고, 데이터 흐름을 분석하여 취약한 부분을 추적하는 기법으로 다양한 방식에 적용될 수 있다. 특히 동적 오염 분석(Dynamic Taint Analysis)은 PIN Tool과 같은 도구를 이용하여 추적이 가능하다는 장점이 있다. 하지만 많은 오버헤드와 시간이 소모된다는 단점이 있다 [4]. 오염 분석 관련 툴로는 PANDA, QIRA, moflow 등이 있다.

기호 실행(Symbolic Execution)은 프로그램 실행 시 사용되는 변수 값 대신에 기호를 사용하여 입력된 데이터에 대한 경로를 분석하는 기법이다. 기호 실행은 코드 커버리지가 높다는 장점이 있지만, 고전적 기호 실행의 경우 해결할 수 없는 수식에 대해 입력을 생성할 수 없다는 단점이 있다 [5]. 기호 실행 관련 툴로는 z3, yices 등이 있다.

2.2 메시징 프로토콜

메시지 브로커를 사용하는 환경에서는 브로커가 지원하는 메시징 프로토콜을 통해 애플리케이션 간 통신이 이루어져야 한다. 가장 보편적이고 대중적인 TCP/IP 기반 메시징 프로토콜은 AMQP(Advanced Message Queuing Protocol), MQTT(Message Queue Telemetry Transport), STOMP(Simple/Streaming Text Oriented Messaging Protocol)가 있다.

AMQP(Advanced Message Queuing Protocol)는 신뢰성과 상호 운용성에 중점을 둔 바이너리 기반 메시징 프로토콜이며, 메시징과 관련된 다양한 기능을 제공한다. 다양한 기능에는 신뢰적 큐잉, 토픽 기반 메시지 발행(publish)과 구독(subscribe), 유연한 라우팅, 트랜잭션 및 보안 등이 있다. 이를 통해 세밀한 제어가 가능하며 안정성을 요구하는 곳에 적합한 프로토콜이다. AMQP는 Fanout, Topic, Headers 기반 방식으로 직접 메시지를 교환한다.

MQTT(Message Queue Telemetry Transport)는 간단하고 자원이 제한된 장치와 낮은 대역폭의 네트워크를 중점으로 하는 바이너리 기반 메시징 프로토콜로, 경량화되고 낮은 대역폭을 사용하는 IoT(Internet of Things) 환경에 적합하다. MQTT는 3가지의 QoS(Quality of Service)를 제공하며 토픽 기반 방식으로 메시지 채널을 생성하여 메시지를 발행하고 구독한다. 3가지의 QoS는 0, 1, 2 값으로 구분되는데 0은 메시지를 한 번만 전달 후 전달 여부를 확인하지 않고, 1은 메시지를 반드시 한 번 이상 전달하며 전달 여부를 확인하지 않고, 2는 한 번만 전달하며 메시지의 핸드셰이킹(Handshaking) 과정을 추적한다.

STOMP(Simple/Streaming Text Oriented Messaging Protocol)는 단순하고 상호 운용성에 중점을 둔 텍스트 기반 메시징 프로토콜이다. STOMP는 HTTP와 유사한 구조를 사용하고, 큐와 토픽을 다루지 않고 메시지를 전송하며 다양한 언어를 지원한다.

2.3 RabbitMQ 메시지 브로커

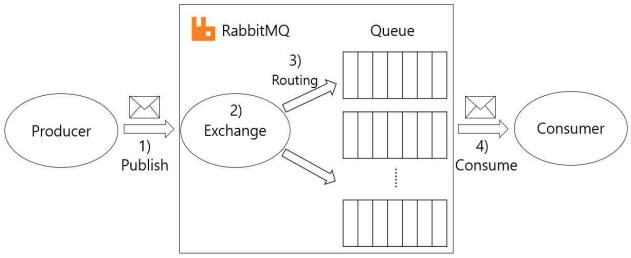
RabbitMQ는 Erlang 언어 기반의 오픈소스 메시지 브로커 소프트웨어이다. 메시지 브로커는 메시지를 송신하는 생산자와 메시지를 수신하는 소비자 사이에서 메시지를 전달하는 중간자 역할을 담당한다. RabbitMQ는 AMQP 메시징 프로토콜을 지원하기 위해 개발된 메시지 브로커이며 AMQP 이외에 MQTT, STOMP와 같은 다양한 메시징 프로토콜 및 HTTP를 지원한다.

2.3.1 RabbitMQ의 특징 [6]

RabbitMQ는 클러스터링을 통해 부하분산을 가능하게 하며 확장성을 제공한다. 클러스터링은 여러 개의 노드를 서로 연결해 마치 하나의 서버처럼 사용하는 것을 의미한다. 클러스터링 된 노드들은 서로 통신이 가능하며 업무를 분담하여 수행한다. 한 노드가 메시지 처리를 요청받으면 해당 노드가 요청을 처리하지 않고 클러스터 내의 다른 노드에게 메시지를 처리하도록 함으로써 특정 노드에 메시지가 집중되는 것을 막는다. 또한, 노드에 장애가 발생하여도 큐와 실패한 노드 내의 메시지를 다른 노드에서 복구할 수 있도록 미러링하는 High Availability 설정을 지원함으로써 메시지 손실을 최소화한다. 또한 RabbitMQ는 플러그인을 지원한다. 먼저 Management 플러그인은 명령줄 도구 “rabbitmqadmin”과 함께 RabbitMQ 노드 및 클러스터의 관리와 모니터링을 위한 HTTP 기반의 API를 제공한다. 주기적으로 시스템의 여러 측면에 대한 데이터를 수집하고 집계하여 클러스터 내 각 노드의 입출력 통계를 보여준다. 또한, 큐와 Exchange의 관리를 지원한다. 이 밖에도 라우팅, 인증, 큐 관련 플러그인과 MQTT, STOMP를 포함한 메시징 프로토콜을 활성화하는 플러그인 등 다양한 종류의 플러그인을 지원한다.

2.3.2 RabbitMQ의 동작 과정 [6]

RabbitMQ는 메시지에 따라 다르게 라우팅 할 수 있다. 다음 그림 1은 RabbitMQ 동작 과정을 나타낸다. Producer는 메시지를 송신하는 Application을 의미하며, Consumer는 메시지를 수신하는 Application을 의미한다.



(그림 1) RabbitMQ 동작 과정

1) Producer가 Exchange로 메시지 전달
2) Exchange 타입에 따라 메시지를 어떤 Queue로 보낼 것인지 결정
3) Exchange에 따라 Queue에 메시지 전달
4) Queue에서 Consumer로 메시지 전달

RabbitMQ에서 메시지는 Producer에서 Queue로 직접 전달되지 않고 Exchange 과정을 거친 후 Queue로 전달된다. Exchange 과정에서 메시지는 Exchange 타입에 따라 메시지를 특정 Queue에 보낼지, 여러 개의 Queue에 보낼지, 또는 버릴지를 결정하게 된다. 각 Queue는 특정 바인딩키를 가지며 각 메시지는 라우팅키를 가진다. 다음 표 1은 Exchange 타입을 나타낸다.

(표 1) Exchange 타입

타입	설명
Fanout	알려진 모든 큐에 메시지 전달
Direct	메시지의 라우팅키와 일치하는 바인딩키를 가진 큐에만 메시지 전달
Topic	지정된 토픽과 일치하는 큐에만 메시지 전달
Headers	헤더에 포함된 key=value의 일치 조건에 따라 메시지 전달

3. RabbitMQ 메시지 퍼징 적용

메시지 브로커 중 RabbitMQ는 Erlang 언어로 쓰인 프로그램이다. Erlang 언어는 erl 소스 코드가 바이트 코드로 변환되어 Erlang 가상머신 BEAM을 통해 실행된다. 이로 인해 RabbitMQ는 실행 파일이 바이너리 파일이 아닌 스크립트 파일을 통해 실행되도록 구현되어있다. 따라서 바이너리 퍼징 도구로 알려진 C, C++, Object C를 지원하는 AFL(American Fuzzy Lop) 또는 바이너리 퍼저인 Melkor을 통한 퍼징의 적용이 어렵다. 따라서 본 논문에서는 RabbitMQ 프로그램의 바이너리에 대한 퍼징이 아닌 메시지 퍼징을 적용한다. 메시지 퍼징은 메시징 프로토콜의 메시지 내용을

퍼징하여 RabbitMQ에 전달하여 퍼징하는 기법이다. RabbitMQ에서 적용 가능한 메시지 프로토콜은 AMQP, MQTT, STOMP이다. AMQP의 경우 기본적으로 RabbitMQ가 지원하며, MQTT와 STOMP 또한 플러그인을 활성화하여 사용할 수 있다.

3.1 mqtt_fuzz [7]

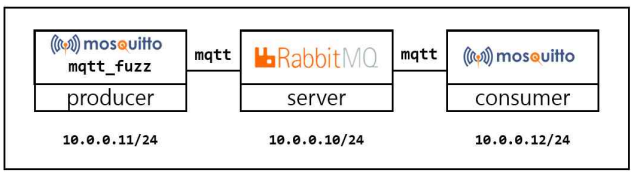
MQTT를 지원하는 퍼저인 mqtt_fuzz는 F-Secure 사에서 개발한 오픈소스 메시지 퍼징 툴이다. 해당 툴은 파이썬 스크립트로 되어 있으며, 입력 값을 퍼징 된 값으로 출력하는 Radamsa와 이벤트 기반 네트워킹 엔진인 Twisted 파이썬 라이브러리를 사용한다. 해당 퍼저는 프로토콜을 구현하여 동작하는 대신 기존 내장된 MQTT 제어 패킷들을 입력 값으로 넣어 Radamsa를 통해 출력으로 나온 퍼징된 제어 패킷을 특정 호스트로 보내 퍼징을 시도한다. 기본으로 내장된 제어 패킷들은 CONNECT, CNNACK, PUBLISH, PUBACK, SUBSCRIBE, PUBCOMP, PUBREL, PUBREC, DISCONNECT이다. 또한 내장된 제어 패킷 외에도 패킷 캡쳐 프로그램인 와이어샤크를 통해 MQTT 패킷 레이어를 raw 파일로 캡처 후 저장하여 제어 패킷을 추가할 수 있다. 또한, 명령 옵션은 표 2와 같다.

(표 2) mqtt_fuzz.py 사용 명령

python mqtt_fuzz.py [대상 호스트] [대상 포트] [옵션]	
옵션	**설명**
-ratio	정상 10 패킷 중 퍼징 할 개수(생략 시 3개)
-delay	제어 패킷 송신 지연 밀리 초 시간(생략 시 50 ms)
-validcases	제어 패킷 디렉토리 지정(생략시 기본 valid-cases)
-fuzzer	퍼저 위치 지정(생략 시 기본 radamsa)

3.2 환경 구축

mqtt_fuzz를 통해 RabbitMQ에 메시지 퍼징을 적용하기 위해 3대의 VM 환경을 구성한다. 각 VM의 운영체제는 Ubuntu 18.04 LTS 64bit이며, 각각의 호스트는 producer, server, consumer가 된다. 그림 2는 VM 환경 구성도를 나타낸다.



(그림 2) VM 환경 구성도

server에는 “sudo apt install rabbitmq-server” 명령을 통해 rabbitmq-server 3.6.10 버전을 설치한다. 또한 “rabbitmq-plugins enable” 명령을 통해 Management, MQTT, STOMP 플러그인을 활성화하여 AMQP 외의 프로토콜을 사용할 수 있게 한다. 메시지를 주고받을 때 사용할 애플리케이션 계정을 위해 RabbitMQ 관리 명령인 “rabbitmqctl”을 통해 RabbitMQ 관리자 계정과 일반 계정을 생성한다. 계정 생성 명령은 코드 1과 같다. 유저를 추가한 뒤에는 유저에 대한 태그를 부여하고 권한을 설정한다. “set_permissions” 명령 옵션에서 vhost는 virtual host를 의미하며, 기본설정은 “/” 이다. 권한은 구성 권한, 쓰기 권한, 읽기 권한으로 분류되며, 권한 허가는 “.*”, 권한 없음은 “^$”를 사용한다. 모든 권한을 허가하는 경우 “‘.*’ ‘.*’ ‘.*’”를 옵션으로 지정하면 된다.

```
sudo rabbitmqctl add_user [ID] [PW]
sudo rabbitmqctl set_user_tags [ID] [PW]
sudo rabbitmqctl set_permissions -p [vhost] [ID] [권한]
```

(코드 1) RabbitMQ 계정 설정 명령

producer와 consumer에는 MQTT 프로토콜이 정상적으로 동작하는지 확인하기 위해 mosquitto 클라이언트를 설치한다. mosquitto 클라이언트는 MQTT를 주고받을 수 있는 프로그램이다. MQTT 메시지가 정상적으로 발행, 구독되는지를 확인하기 위해 사용한다. 메시지를 발행하는 쪽은 “mosquitto_pub” 명령을 사용하며, 구독하는 쪽은 “mosquitto_sub” 명령을 사용한다. 두 클라이언트는 같은 토픽과 같은 계정을 통해 메시지를 주고받는다. 각 명령을 사용하여 “key1” 토픽을 사용하는 예제는 코드 2와 같다.

```
mosquitto_pub -h 10.0.0.10 -p 1883 -t key1 -u test1 -P 1234
-m “hello world!” -i “producer
mosquitto_sub -h 10.0.0.11 -p 1883 -t key1 -u test1 -P 1234
-i “consumer
```

(코드 2) mosquitto_pub, mosquitto_sub 명령 예제

해당 클라이언트를 각각 producer와 consumer에서 실행하여 지속적으로 RabbitMQ를 거쳐 메시지를 주고받게 하여 정상적으로 MQTT 메시지를 주고받는 환경을 구성한다. 다음으로 radamsa와 Twisted를 먼저 설치하고 mqtt_fuzz를 내려받는다. 코드 3은 radamsa, Twisted, mqtt_fuzz 설치 명령이다.

```
git clone https://gitlab.com/akihe/radamsa.git
cd radamsa
sudo make OFLAGS=-O1
sudo make install
cd ..
pip install Twisted
git clone https://github.com/F-Secure/mqtt_fuzz.git
```

(코드 3) radamsa, Twisted, mqtt_fuzz 설치 명령

3.3 메시지 퍼징 적용

앞서 구축한 환경에서 mqtt_fuzz를 통해 server에 메시지 퍼징을 시도한다. 하지만 mqtt_fuzz의 내장된 제어 패킷에는 메시지를 주고받을 때 필요한 RabbitMQ 계정 정보가 포함되어있지 않다. 그림 3은 계정 정보를 가지고 있지 않은 상태에서 외부에서 mqtt_fuzz를 수행하는 경우 RabbitMQ 서버 로그이다.

```
=INFO REPORT==== 4-Apr-2019::23:02:29 ===
MQTT vhost picked using plugin configuration or default

=WARNING REPORT==== 4-Apr-2019::23:02:29 ===
MQTT login failed for "guest" access_refused (access must be from localhost)

=INFO REPORT==== 4-Apr-2019::23:02:29 ===
MQTT protocol error connect_expected for connection 10.0.0.11:48722 -> 10.0.0.10:1883

=INFO REPORT==== 4-Apr-2019::23:02:29 ===
MQTT vhost picked using plugin configuration or default

=WARNING REPORT==== 4-Apr-2019::23:02:29 ===
MQTT login failed for "guest" access_refused (access must be from localhost)
```

(그림 3) mqtt_fuzz를 외부에서 실행 시 서버 로그

퍼징을 시도하면 server에서 메시지에 RabbitMQ 계정 정보가 없어 guest 계정으로 인식하고 localhost가 아닌 곳으로부터 송신된 메시지를 모두 drop 한다. 따라서 mosquitto 클라이언트를 통해 계정 정보를 옵션으로 주어 전달된 제어 패킷을 캡처하여 valid_cases

에 저장 후 mqtt_fuzz를 실행해야 외부에서 사용할 수 있다. 캡처한 패킷 파일을 메시지 제어 타입에 맞게 valid_cases에 넣고 실행하면 외부에서 메시지 퍼징이 가능해진다. 그림 4에서 기존 내장된 패킷과 추가한 계정 정보가 추가된 패킷을 확인할 수 있다. mqtt_fuzz가 정상적으로 동작하는 경우 화면은 그림 5과 같다.

```
greendot@server:~/mqtt_fuzz/valid-cases/connect$ sudo hexdump -C mqtt.connect.85
.raw
00000000  10 16 00 04 4d 51 54 54  04 02 00 00 00 0a 6d 79  |....MQTT......my|
00000010  63 6c 69 65 6e 74 69 64                           |clientid|
00000018
greendot@server:~/mqtt_fuzz/valid-cases/connect$ sudo hexdump -C mqtt.connect.77
7.raw
00000000  10 29 00 06 4d 51 49 73  64 70 03 c2 00 3c 00 0e  |.)..MQIsdp...<..|
00000010  67 72 65 65 6e 64 6f 74  63 6c 69 65 6e 74 00 05  |greendotclient..|
00000020  74 65 73 74 31 00 04 31  32 33 34                 |test1..1234|
0000002b
```

(그림 4) 내장 패킷과 계정 정보가 추가된 패킷

```
1554378576:Reconnecting
1554378576:27b38e21-3bc5-4a41-8cdc-55f72132996d:Connected to server
1554378576:27b38e21-3bc5-4a41-8cdc-55f72132996d:Sending valid connect
1554378576:27b38e21-3bc5-4a41-8cdc-55f72132996d:Fuzzer -> Server: ECkABk1RSXNkcA
PCADwADmdyZWVuZG90Y2xpZW50AAV0ZXN0MQAEMTIzNA==
1554378576:b0b765ee-b199-4a54-8dcc-9ced6db148a7:Sending valid publish-complete
1554378576:b0b765ee-b199-4a54-8dcc-9ced6db148a7:Fuzzer -> Server: cAIAAQ==
1554378576:e5218bcb-fcb5-436d-8cd6-67283affd051:Sending valid disconnect
1554378576:e5218bcb-fcb5-436d-8cd6-67283affd051:Fuzzer -> Server: 4AA=
1554378576:27b38e21-3bc5-4a41-8cdc-55f72132996d:Sending valid subscribe
1554378576:27b38e21-3bc5-4a41-8cdc-55f72132996d:Fuzzer -> Server: ggYAAgABIwA=
1554378576:b0b765ee-b199-4a54-8dcc-9ced6db148a7:Sending valid disconnect
1554378576:b0b765ee-b199-4a54-8dcc-9ced6db148a7:Fuzzer -> Server: 4AA=
1554378576:e5218bcb-fcb5-436d-8cd6-67283affd051:End of session, initiating disco
nnect.
1554378576:27b38e21-3bc5-4a41-8cdc-55f72132996d:Sending valid publish
1554378576:27b38e21-3bc5-4a41-8cdc-55f72132996d:Fuzzer -> Server: NQwACFRvcGljQS
9DAAk=
```

(그림 5) mqtt_fuzz 실행화면

server에서는 모니터링을 지원하는 management 플러그인 웹페이지 “localhost:15672”에 접속하여 퍼징된 메시지가 들어오는 것을 확인하고 “/var/log/rabbitmq/rabbitmq@호스트이름.log” 로그를 “tail -f” 명령을 사용하여 실시간으로 특정 에러가 발생하는지 확인한다. 그림 6는 출력되는 로그 화면이다.

```
=INFO REPORT==== 4-Apr-2019::20:51:05 ===
MQTT vhost picked using plugin configuration or default

=INFO REPORT==== 4-Apr-2019::20:51:05 ===
accepting MQTT connection <0.7877.37> (10.0.0.11:42468 -> 10.0.0.10:1883)

=INFO REPORT==== 4-Apr-2019::20:51:05 ===
MQTT protocol error qos2_not_supported for connection 10.0.0.11:42468 -> 10.0.0.10:1883

=INFO REPORT==== 4-Apr-2019::20:51:05 ===
MQTT vhost picked using plugin configuration or default

=INFO REPORT==== 4-Apr-2019::20:51:05 ===
accepting MQTT connection <0.7901.37> (10.0.0.11:42470 -> 10.0.0.10:1883)
```

(그림 6) “/var/log/rabbitmq/rabbitmq@호스트이름.log” 내용 화면

4. 결론

본 논문에서는 취약점 발견을 위해 취약점 분석 기법 중 메시지 퍼징을 메시지 브로커인 RabbitMQ에 적용하였다. 널리 사용되는 메시지 브로커 중 하나인 RabbitMQ는 Erlang 언어로 작성되어 있어 기존의 바이너리 퍼징 툴을 통한 취약점 분석에 대해 어려움이 있었으나, 메시지 퍼징 툴인 mqtt_fuzz를 통해 MQTT 프로토콜 메시지 퍼징 수행이 가능함을 확인하였다. 추후 연구로는 MQTT 메시지 퍼징을 지속해서 수행하여 RabbitMQ의 취약점을 발견하고 분석하여 파이썬(Python) 언어를 통해 익스플로잇 툴(Exploit Tool)을 구현할 예정이다.

Acknowledgement

이 성과는 2019년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2018R1A4A1025632).

참 고 문 헌

[1] 이성민, 오준석, 최진영 : "소프트웨어의 잠재적 오류 가능성 및 보안취약점 비교분석 연구." 한국정보과학회 학술발표논문집, 37.1D, 2010. pp.106-109.

[2] BACCHELLI, Alberto; BIRD, Christian : Expectations, outcomes, and challenges of modern code review. In: Proceedings of the 2013 international conference on software engineering. IEEE Press, 2013. pp.712-721.

[3] SCHWARTZ, Edward J.; AVGERINOS, Thanassis; BRUMLEY, David : All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In: 2010 IEEE Symposium on Security and Privacy. IEEE, 2010. pp.317-331.

[4] Liang, Hongliang, et al. : Fuzzing: State of the art. IEEE Transactions on Reliability 67.3, 2018. pp.1199-1218.

[5] Cadar, Cristian, and Koushik Sen : Symbolic execution for software testing: three decades later, Commun. ACM 56.2, 2013. pp.82-90.

[6] RabbitMQ Server Document “https://www.rabbitmq.com/admin-guide.html”, 2019년 4월 방문

[7] F-Secure/mqtt_fuzz github “https://github.com/F-Secure/mqtt_fuzz”, 2019년 4월 방문