

# 진행 사항 슬라이드

- 보안 취약점 분석 및 익스플로잇 개발 -



**지도교수 : 이종혁**

Captain : 201621571 손상진

Sailors : 201621110 권순홍

201621136 서민지

201621173 최서윤

# 목차

---

## 1. 현재 상황

- 문제
- 대책

## 2. 진행 사항

- 논문 분석
- Melkor fuzzer 분석 및 문서화
- AFL 분석 및 문서화
- MQTT\_fuzz 분석 및 문서화
- RabbitMQ 문서 한글화

## 3. 추후 계획



QBQB

# 1. 현재 상황

## • 문제

- 취약점을 찾기 위한 fuzzer인 Melkor, AFL을 적용 시도 중 rabbitmq-server가 바이너리 파일이 아님을 확인
  - rabbitmq-server는 셸 스크립트에 의해 짜여져 있음을 확인
  - 스크립트 내에서 erlang으로 컴파일 된 바이트코드를 Beam(Erlang virtual machine)에 의해 실행되는 것을 확인
  - 기존 Melkor과 AFL로 단순하게 퍼징이 불가능함을 확인
  - erlang에 특화된 fuzzer를 찾고 있으나 현재까지 발견하지 못함

```
greendot@greendot-virtual-machine:~$ file /usr/sbin/rabbitmq-server
/usr/sbin/rabbitmq-server: symbolic link to ../lib/rabbitmq/bin/rabbitmq-script-wrapper
greendot@greendot-virtual-machine:~$ file /usr/lib/rabbitmq/bin/rabbitmq-script-wrapper
/usr/lib/rabbitmq/bin/rabbitmq-script-wrapper: POSIX shell script, ASCII text executable
greendot@greendot-virtual-machine:~$
```

```
greendot@greendot-virtual-machine:~$ file /usr/lib/rabbitmq/bin/rabbitmq-server
/usr/lib/rabbitmq/bin/rabbitmq-server: symbolic link to ../lib/rabbitmq_server-3.6.10/sbin/rabbitmq-server
greendot@greendot-virtual-machine:~$ file /usr/lib/rabbitmq/lib/rabbitmq_server-3.6.10/sbin/rabbitmq-server
/usr/lib/rabbitmq/lib/rabbitmq_server-3.6.10/sbin/rabbitmq-server: POSIX shell script, ASCII text executable
```

# 1. 현재 상황

## • 문제

```
start_rabbitmq_server() {  
    # "-pa ${RABBITMQ_SERVER_CODE_PATH}" should be the very first  
    # command-line argument. In case of using cached HiPE-compilation,  
    # this will allow for compiled versions of erlang built-in modules  
    # (e.g. lists) to be loaded.  
    ensure_thread_pool_size  
    check_start_params &&  
    RABBITMQ_CONFIG_FILE=$RABBITMQ_CONFIG_FILE \  
    ERL_MAX_ETS_TABLES=$ERL_MAX_ETS_TABLES \  
    exec ${ERL_DIR}erl \  
        -pa ${RABBITMQ_SERVER_CODE_PATH} ${RABBITMQ_EBIN_ROOT} \  
        ${RABBITMQ_START_RABBIT} \  
        ${RABBITMQ_NAME_TYPE} ${RABBITMQ_NODENAME} \  
        -boot "${SASL_BOOT_FILE}" \  
        ${RABBITMQ_CONFIG_ARG} \  
        +W w \  
        +A ${RABBITMQ_IO_THREAD_POOL_SIZE} \  
        ${RABBITMQ_SERVER_ERL_ARGS} \  
        +K true \  
        -kernel inet default_connect_options "[{nodelay,true}]" \  
        ${RABBITMQ_SERVER_ADDITIONAL_ERL_ARGS} \  
        ${RABBITMQ_LISTEN_ARG} \  
        -sasl errlog_type error \  
        -sasl sasl_error_logger "$SASL_ERROR_LOGGER" \  
        -rabbit error_logger "$RABBIT_ERROR_LOGGER" \  
        -rabbit sasl_error_logger "$RABBIT_SASL_ERROR_LOGGER" \  
        -rabbit enabled_plugins_file "\"$RABBITMQ_ENABLED_PLUGINS_FILE\" \" \" \  
        -rabbit plugins_dir "\"$RABBITMQ_PLUGINS_DIR\" \" \" \  
        -rabbit plugins_expand_dir "\"$RABBITMQ_PLUGINS_EXPAND_DIR\" \" \" \  
        -os_mon start_cpu_sup false \  
        -os_mon start_disk_sup false \  
        -os_mon start_memsup false \  
        -mnesia dir "\"${RABBITMQ_MNESIA_DIR}\" \" \" \  
        ${RABBITMQ_SERVER_START_ARGS} \  
        ${RABBITMQ_DIST_ARG} \  
        "$@"  
}
```

```
● rabbitmq-server.service - RabbitMQ Messaging Server  
   Loaded: loaded (/lib/systemd/system/rabbitmq-server.service; enabled; vendor preset: enabled)  
   Active: active (running) since Tue 2019-03-12 02:46:23 KST; 1 day 16h ago  
 Main PID: 25664 (rabbitmq-server)  
    Tasks: 88 (limit: 2294)  
   CGroup: /system.slice/rabbitmq-server.service  
           └─25664 /bin/sh /usr/sbin/rabbitmq-server  
             └─25669 /bin/sh /usr/lib/rabbitmq/bin/rabbitmq-server  
               └─25928 /usr/lib/erlang/erts-9.2/bin/epmd -daemon  
                 └─25978 /usr/lib/erlang/erts-9.2/bin/beam.smp -W w -A 64 -P 1048576 -t 5000000 -stbt db -zdbbl 32000 -K true -B i -- -root /usr/lib/erla  
                   └─26086 erl_child_setup 65536  
                     └─26149 inet_gethost 4  
                       └─26150 inet_gethost 4
```

# 1. 현재 상황

---

- 대책

- 기존 직접적인 실행파일이 아닌 fuzzed 메시지를 통해 rabbitmq 취약점 분석
  - mqtt\_fuzz를 통해 퍼징
  - AMQP 관련 퍼징 툴 혹은 rabbitmq client API를 통해 제작 후 퍼징
- 발견된 취약점이 많이 나타나는 Management UI 부분, RabbitMQ Plugin 부분 취약점 분석
  - Management UI 사용에 따른 로그 파일 분석

## 2. 진행 사항

---

- 요약

- 논문 분석

- 한글 신규 취약점 발견을 위한 fuzzing 기법 개발에 관한 연구
    - Fuzzing: State of the Art
    - Coverage-based Greybox Fuzzing as Markov(진행중)

- Melkor 분석 및 문서화

- mqtt\_fuzz 분석 및 문서화

- AFL 분석 및 문서화

- RabbitMQ 문서화 및 분석(진행중->중단)

- erlang rabbitmq-server 컴파일



# 2. 진행 사항

- 논문 분석 2편(1/2)

- Fuzzing: State of the Art

- 퍼징의 기본 프로세스(1/3)

- 대상 프로그램 (Target Program)

- 테스트 중인 프로그램으로 바이너리 또는 소스 코드

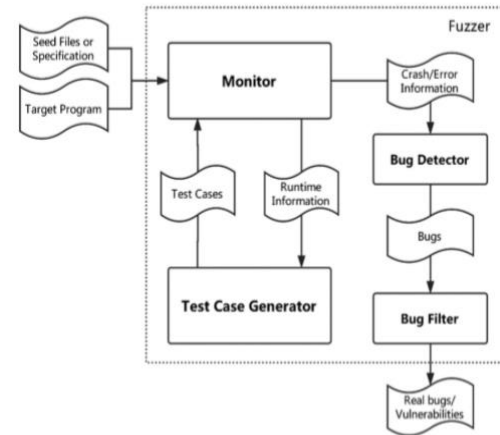
- 실제 소프트웨어의 소스 코드는 일반적으로 쉽게 액세스 할 수 없기 때문에 fuzzer는 대부분의 경우, 바이너리 코드를 대상으로 함

- 모니터 (Monitor)

- 일반적으로 화이트 박스 또는 그레이 박스의 fuzzer에 있음

- 코드 계측(code instrumentation), 오염 분석(taint analysis) 등과 같은 기술을 활용

- 코드 적용 범위(code coverage), 데이터 흐름(data flow) 또는 대상 프로그램의 기타 유용한 런타임 정보를 수집



# 2. 진행 사항

- 논문 분석 2편(1/2)

- Fuzzing: State of the Art

- 퍼징의 기본 프로세스(2/3)

- 테스트 케이스 생성기 (Test case generator)

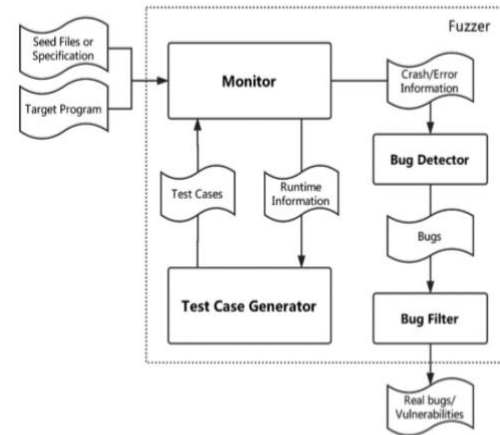
- fuzzer가 테스트 케이스를 생성하는 주요 방법

- 변이 기반(mutation-based, dumb fuzzing)

- well-formed seed 파일을 무작위로 돌연변이화 하거나 런타임 중에 수집 된 target-program-oriented 정보를 기반으로 조정할 수 있는 입력을 생성

- 문법 기반(grammar-based, smart fuzzing)

- 시드 파일이 필요하지 않음
- 문법과 같은 사항들로부터 입력을 생성
- 퍼징의 테스트 케이스는 일반적으로 초기 구문 분석 단계를 통과할 만큼 유효하고 대상 프로그램의 심층 논리에서 버그를 유발할 만큼 무효한 "반유효" 입력 값





# 2. 진행 사항

- 논문 분석 2편(1/2)

- Fuzzing: State of the Art

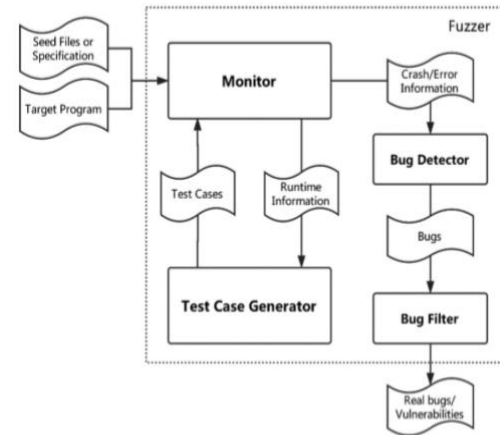
- 퍼징의 기본 프로세스(3/3)

- 버그 탐지기 (Bug detector)

- 사용자가 대상 프로그램에서 잠재적 버그를 발견 할 수 있도록 설계되고 구현됨
- 대상 프로그램이 충돌하거나 오류를 보고하면 버그 감지기 모듈은 관련 정보를 수집하고 분석하여 버그가 있는지 판단
  - e.g., 스택 추적

- 버그 필터(Bug filter)

- 보고 된 모든 버그로부터 악용 가능한 버그를 필터링
  - 일반적으로 수동적으로 이루어짐



## 2. 진행 사항

---

- 논문 분석 2편(1/2)
  - Fuzzing: State of the Art
    - 퍼징 방식에 따른 분류(1/3)
      - 블랙 박스 퍼징
        - "블랙 박스 무작위 테스트"라고도 함
        - 대상 프로그램이나 입력 형식에서 정보를 요구하는 대신 제대로 된 형식의 시드 파일을 무작위로 돌연변이화 시켜 무효 입력 생성
          - 돌연변이화: 비트 플립, 바이트 카피, 바이트 제거 등
        - 단점
          - 코드 커버리지가 낮음
      - 대표적 툴
        - Fuzz, Trinity

## 2. 진행 사항

---

- 논문 분석 2편(1/2)
  - Fuzzing: State of the Art
    - 퍼징 방식에 따른 분류(2/3)
      - 화이트 박스 퍼징
        - 블랙박스 기법의 단점을 해결하기 위해 제안
        - 대상 프로그램의 내부 논리 지식을 기반으로 함
        - 대상 프로그램의 모든 실행 경로를 탐색
        - 동적 기호 실행 (concolic execution이라고도 함) 및 커버리지 최대화 검색 알고리즘을 사용하여 대상 프로그램을 철저하고 빠르게 검색 할 수 있음
    - 대표적 툴
      - Fuzz, Trinity

## 2. 진행 사항

---

- 논문 분석 2편(1/2)
  - Fuzzing: State of the Art
    - 퍼징 방식에 따른 분류(3/3)
      - 그레이 박스 퍼징
        - 코드 계측(code instrumentation) 방식 사용
          - 코드 계측
            - 소스코드의 정확성 및 오류사항을 함께 탐지
      - 대상 프로그램의 모든 실행 경로를 탐색
      - 화이트 박스 퍼징 방식과 차이점
        - 대상 프로그램의 런타임 정보 (예 : 코드 커버리지), 오염 데이터 흐름(taint data flow) 등) 을 사용하여 어떤 경로를 탐색했는지 결정
      - 동적 기호 실행 (concolic execution이라고도 함) 및 커버리지 최대화 검색 알고리즘을 사용하여 대상 프로그램을 철저하고 빠르게 검색 할 수 있음
    - 대표적 툴
      - BuzzFuzz

# 2. 진행 사항

- 논문 분석 2편(1/2)
  - Fuzzing: State of the Art
    - 일반적인 목적의 퍼저(1/2)

TABLE V  
SUMMARIES OF THE TYPICAL FUZZERS

Name	Birth Year	Targets	Key Techniques	Platforms	Availability
Peach	2004	General purpose	Black-box mutation/generation fuzzing	Linux, Windows, MacOS	Open-source
Trinity	2004	OS Kernels	Input knowledge based black-box generation fuzzing	ARM, i386, x86-64, etc.	Open-source
beSTORM	2005	General purpose	Black-box generation fuzzing	Linux, Windows	Commercial
jsfunfuzz	2007	JavaScript engine in Firefox	Grammar-based black-box generation fuzzing, Differential testing	Linux, MacOS, Windows	Open-source
SAGE	2008	Large Windows applications	White-box generation fuzzing	Windows	Microsoft internal
Sulley	2009	Network protocols	Input knowledge based black-box generation fuzzing	Windows	Open-source
IOCTL fuzzer	2009	Kernel drivers	Input knowledge based black-box generation fuzzing	Windows	Open-source
Csmith	2011	C compilers	Grammar-based black-box generation fuzzing, Differential testing	Linux, FreeBSD, MacOS, Windows	Open-source
LangFuzz	2012	Language-agnostic interpreters (JavaScript, PHP)	Grammar-based black-box generation/mutation fuzzing	Linux	Closed-source
AFL	2013	Applications	Coverage-guide gray-box fuzzing, Genetic algorithm	Linux, FreeBSD, Open/NetBSD, MacOS, Solaris	Open-source
YMIR	2013	ActiveX control	Generation of fuzzing grammars using API-level concolic testing	Windows	Closed-source
vUSBf	2014	USB drivers	Input knowledge based black-box generation fuzzing	Linux	Open-source
CLsmith	2015	Many core openCL compilers	Grammar-based black-box generation fuzzing, Random differential testing and EMI testing	Linux	Open-source
Syzkaller	2016	OS Kernels	Coverage-guide gray-box generation/mutation fuzzing	Linux	Open-source
TLS-Attacker	2016	Network protocols	Grammar-based black-box mutation fuzzing, using modifiable variables and two-stage fuzzing	Linux, MacOS, Windows	Open-source
QuickFuzz	2016	Applications	Grammar-based black-box generation/mutation fuzzing	Linux	Open-source
CAB-FUZZ	2017	OS Kernels	gray-box fuzzing, using concolic testing	Linux, MacOS, Windows	Unknown
kAFL	2017	OS Kernels	Coverage-guide gray-box fuzzing, using hypervisor and Intel's Processor Trace technology	Linux, MacOS, Windows	Open-source

# 2. 진행 사항

- 논문 분석 2편(1/2)
  - Fuzzing: State of the Art
    - 일반적인 목적의 퍼저(2/2)

TABLE VI  
TYPICAL FUZZERS AND THEIR PROBLEM DOMAINS

Name	Seeds generation and selection	Input validation and coverage	Handling crash-inducing test cases	Leveraging runtime information	Scalability in fuzzing
Peach	✓	✓	○	×	✓
Trinity	/	✓	○	×	✓
beSTORM	/	✓	○	×	✓
jsfunfuzz	/	✓	○	×	✓
SAGE	/	✓	○	✓	✓
Sulley	/	✓	○	×	✓
IOCTL fuzzer	/	✓	○	×	✓
Csmith	/	✓	×	×	✓
LangFuzz	/	✓	○	×	○
AFL	○	×	×	✓	✓
YMIR	/	✓	○	×	○
vUSBf	/	✓	○	×	○
CLsmith	/	✓	○	×	○
Syzkaller	/	✓	○	✓	✓
TLS-Attacker	/	✓	○	×	✓
QuickFuzz	/	✓	×	×	✓
CAB-FUZZ	/	×	○	✓	○
kAFL	×	○	×	✓	○

The meaning of the symbols in table is as follows:

1) Seeds generation and selection

✓: The fuzzer can automatically generate seeds or adopt some seed selection algorithm.

○: The fuzzer provides some high-quality seeds for testers to choose (usually for mutation-based fuzzers).

×

/: The fuzzer does not require seeds in random (usually for grammar-based fuzzers).

2) Input validation and coverage

✓: The fuzzer utilizes input grammars or other knowledge to generate test cases, or adopts some ways to pass through input validation.

○: The fuzzer uses some methods to mitigate the problem caused by input validation.

×

3) Handling crash-inducing test cases

✓: The fuzzer can analyze the found bugs automatically and generate a detailed bug report.

○: The fuzzer can provide some useful information, like log file, to help subsequent bug analysis.

×

4) Leveraging runtime information

✓: The fuzzer uses runtime information to guide test case generation.


×

5) Scalability in fuzzing

✓: The fuzzer can test real-world applications effectively and has found plenty of bugs.

○: The fuzzer is in its experiment period and applied to some real-world programs.

×

  
QBQB

14

## 2. 진행 사항

---

- 논문 분석 2편(2/2)

- 한글 신규 취약점 발견을 위한 fuzzing 기법 개발에 관한 연구
  - 윈도우에서의 아래 한글 2010 프로그램에 입력되는 HWP 문서 파일을 통한 fuzzing에 대해 설명
  - 퍼징 도구로 Peach Fuzzing Framework를 사용하여 fuzzing 툴을 작성
  - 퍼징(Fuzzing)(1/2)
    - 퍼징을 이용한 프로그램 취약점 발견 과정
      1. 프로그램 식별
      2. 입력 데이터 식별
      3. 무작위 데이터 생성
      4. 무작위 데이터 입력
      5. 프로그램 상태 감시
      6. 공격 가능성 판단

## 2. 진행 사항

---

- 논문 분석 2편(2/2)
  - 한글 신규 취약점 발견을 위한 fuzzing 기법 개발에 관한 연구
    - 퍼징(Fuzzing)(2/2)
      - 분류
        - 퍼징 대상의 이해 유무에 따른 분류
          - 덤(Dumb), 스마트(Smart)
        - 데이터 처리 방법에 따른 분류
          - 생성(Generation), 변형(Mutation)
      - 도구 분류
        - FileFuzz
        - beSTORM
        - Peach Fuzzing Framework



## 2. 진행 사항

---

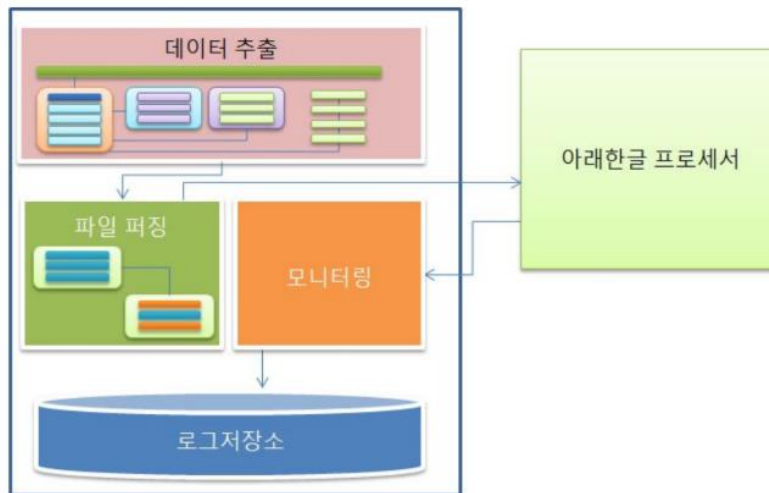
- 논문 분석 2편(2/2)

- 한글 신규 취약점 발견을 위한 fuzzing 기법 개발에 관한 연구
  - 데이터 레코드 기반 퍼징(1/2)
    - 한글의 파일 구조는 Microsoft의 Compound Document에 기초
      - 전체적인 구조는 스토리지와 스트림으로 구분
        - 하나의 스트림에는 바이너리 or 레코드 구조로 데이터 저장
        - 저장 옵션에 따라 압축/암호화
        - 문서를 읽을 시 압축 상태 플래그에 따라 해제해서 읽음
      - 문서 전체 파일은 헤더와 섹터로 구분되며, 각 섹터는 512byte로 나뉘어져 있음
      - 레코드의 구조가 깨지면 인식할 수 없는 파일 포맷으로 예외처리
    - 사용된 퍼징 기법
      - 데이터 변형 방법
        - 레코드간 위치 변경
        - 레코드 데이터 위치 변경

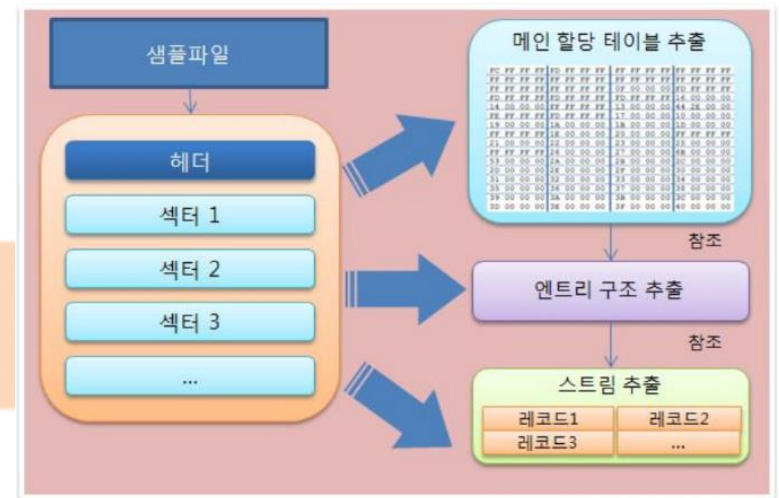
## 2. 진행 사항

- 논문 분석 2편(2/2)

- 한글 신규 취약점 발견을 위한 fuzzing 기법 개발에 관한 연구
  - 데이터 레코드 기반 퍼징(2/2)
    - 퍼징 도구의 구조



[그림 3-13] 레코드 변형에 기반한 퍼저의 구조



[그림 3-14] 데이터 추출기의 동작

## 2. 진행 사항

- 논문 분석 2편(2/2)

- 한글 신규 취약점 발견을 위한 fuzzing 기법 개발에 관한 연구
  - 실험 및 결과
    - 하나의 정상적인 샘플파일을 대상으로 Peach를 통한 퍼징 도구로 3일간 수행
    - 퍼징을 통해 1433개의 크래시를 발견



<표 4-1> 퍼징 결과

크래시 종류	Peach를 이용한 퍼징		구현 도구를 이용한 퍼징	
	덤퍼징	스마트퍼징 파일 길이	레코드 위치변화	레코드 내 데이터 위치변화
EXPLOITABLE	0	6	0	5
PROBABLY EXPLOITABLE	0	12	0	7
PROBABLY NOT EXPLOITABLE	6	505	0	233
UNKNOWN	201	1685	39	1188

- 결론

- 해당 결과를 통해 아래 한글의 신규 취약점 발견 및 소프트웨어의 안정성에 기여 할 것이라 생각

## 2. 진행 사항

- Melkor fuzzer 분석 및 문서화(1/2)

- 예제를 통한 테스트 문서화

- 버퍼 오버플로우 가능성이 있는 바이너리 생성
- 해당 바이너리에 대해 1337번 퍼징

- rule에 따라 퍼징 진행

- readelf를 통해 특정 orc\_0119 파일을 확인

- 현재 세그먼트의 파일 크기가 메모리 사이즈보다 크다는 에러 발생 확인

```
user@ubuntu:~/Melkor_ELF_Fuzzer$ readelf -SW orcs_test/orc_0119
There are 29 section headers, starting at offset 0x1988:

Section Headers:
 [Nr] Name              Type              Address            Off    Size  ES Flg Lk Inf Al
 [ 0] .interp            PROGBITS          0000000000000000 000000 000000 00  0  0  0
 [ 1] .note.ABI-tag      NOTE             0000000000000238 000238 00001c 00  0  0  1
 [ 2] .note.gnu.build-id NOTE             0000000000000254 000254 000020 00  A  0  0  4
 [ 3] .note.gnu.build-id NOTE             0000000000000274 000274 000024 00  A  0  0  4
 [ 4] .gnu.hash          GNU_HASH         0000000000000298 000298 00001c 00  A  5  0  8
 [ 5] .dynsym            DYNSYM          0000000004142434 0002b8 0000d8 18  A  5  1  8
 [ 6] .dynstr            STRTAB          0000000000000390 000390 0000a4 00  A  0  0  1
 [ 7] .gnu.version       VERSYM          0000000000000434 000434 000012 02  A  5  0  2
 [ 8] .gnu.version_r     VERNEED         0000000000000448 000448 000030 00  A  6  1  8
 [ 9] .rela.dyn          RELA            0000000000000478 000478 0000c0 18  A  5  0 4097
[10] .rela.plt          RELA            0000000000000538 000538 000048 18  AI  5  22  8
[11] .init              PROGBITS          0000000000000580 000580 000017 00  MAX  0  0  4
readelf: Warning: [12]: Expected link to another section in info field [12]: .plt          PROGBITS          00000000000005a0 0005a0 000040 10  MAX 0x510GTCxxxxxxxxxlp 0  0 16
[13] .plt.got           PROGBITS          00000000000005e0 0005e0 000008 08  AX  0  0  8
[14] <corrupt>          PROGBITS          00000000000005f0 0005f0 0001f2 00  AX  0  0 16
[15] .finl              PROGBITS          00000000000007e4 0007e4 000009 00  AX  0  0  4
[16] <corrupt>          PROGBITS          00000000000007f0 0007f0 000004 04  AX 0x510GTCxxxxxxxxxlp 0  0 4
[17] .eh_frame_hdr     PROGBITS          00000000000007f4 0007f4 00003c 00  A  0  0  4
[18] .eh_frame         PROGBITS          0000000000000830 c100728c 000108 00  A  0  0  8
[19] .init_array        INIT_ARRAY        00000000000020d8 000d8 000008 00  WA  0  0  8
[20] .fini_array        FINI_ARRAY        00000000000020db0 000db0 000008 08  WA  0  0  8
[21] .text              DYNAMIC           00000000000020db8 000db8 0001f0 10  WA  0  0  8
[22] .got               PROGBITS          00000000000020fab 000fab 000058 08  WA  0  0  8
[23] .data              PROGBITS          00000000000021000 001000 000010 00  WA  0  0  8
[24] .bss               NOBITS            00000000000021010 001010 000000 00  WA  0  0  1
[25] .comment            PROGBITS          0000000000000000 001010 00002a 01  MS  0  0  1
[26] .symtab            SYMTAB            0000000000000000 001040 000618 18  27 43  8
[27] .strtab            STRTAB            0000000000000000 001058 000232 00  0  0  1
[28] .shstrtab          STRTAB            0000000000000000 00108a 0000fe 00  0  0 4097

Key to Flags:
 W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
 L (link order), O (extra OS processing required), G (group), T (TLS),
 C (compressed), x (unknown), o (OS specific), E (exclude),
 l (large), p (processor specific)
readelf: Error: Unable to read program interpreter name
readelf: Error: no .dynamic section in the dynamic segment
```

## 2. 진행 사항

- Melkor fuzzer 분석 및 문서화(2/2)
  - rule 문서화
    - Melkor은 rule이라는 규칙에 따라 퍼징을 실행
    - 해당 rule에 대해 문서화

```
[+] Malformed ELF 'orc_1337':  
  
[+] Fuzzing the relocations section .rela.dyn with 8 SHT_RELA entries  
SHT[9] REL[1] rule [02] executed  
  
[+] Fuzzing the relocations section .rela.plt with 3 SHT_RELA entries  
SHT[10] REL[1] rule [01] executed  
  
[+] Fuzzing the Symbol Table .dynsym with 9 entries  
SHT[5] SYM[3] rule [15] executed  
SHT[5] SYM[4] rule [04] executed  
  
[+] Fuzzing the Symbol Table .symtab with 65 entries  
SHT[26] SYM[0] rule [04] executed  
SHT[26] SYM[2] rule [10] executed  
SHT[26] SYM[6] rule [04] executed  
SHT[26] SYM[9] rule [03] executed  
SHT[26] SYM[12] rule [04] executed  
SHT[26] SYM[12] rule [15] executed  
SHT[26] SYM[13] rule [04] executed  
SHT[26] SYM[14] rule [05] executed  
SHT[26] SYM[15] rule [04] executed  
SHT[26] SYM[21] rule [15] executed  
SHT[26] SYM[26] rule [15] executed  
SHT[26] SYM[30] rule [15] executed  
SHT[26] SYM[35] rule [15] executed  
SHT[26] SYM[36] rule [05] executed  
SHT[26] SYM[41] rule [10] executed  
SHT[26] SYM[43] rule [04] executed  
SHT[26] SYM[46] rule [15] executed  
SHT[26] SYM[47] rule [15] executed  
SHT[26] SYM[54] rule [15] executed  
SHT[26] SYM[64] rule [05] executed  
  
[+] Fuzzing the Dynamic section .dynamic with 31 entries  
SHT[21] DYN[9] rule [05] executed  
  
[+] Fuzzing the Note section .note.ABI-tag with 32 bytes  
  
[+] Fuzzing the Note section .note.gnu.build-id with 36 bytes  
  
[+] Fuzzing the String Table .dynstr with 164 bytes  
  
[+] Fuzzing the String Table .strtab with 562 bytes  
STRS[27] rule [03] executed  
  
[+] Fuzzing the String Table .shstrtab with 254 bytes
```

```
[+] Fuzzing the String Table .shstrtab with 254 bytes  
  
[+] Fuzzing the Section Header Table with 29 entries  
SHT[1] rule [05] executed  
SHT[1] rule [25] executed  
SHT[3] rule [01] executed  
SHT[15] rule [30] executed  
SHT[16] rule [10] executed  
SHT[18] rule [08] executed  
SHT[19] rule [34] executed  
SHT[21] rule [32] executed  
SHT[23] rule [10] executed  
SHT[23] rule [15] executed  
SHT[23] rule [34] executed  
SHT[25] rule [05] executed  
  
[+] Fuzzing the Program Header Table with 9 entries  
PHT[0] rule [01] executed  
PHT[0] rule [07] executed  
PHT[2] rule [03] executed  
PHT[3] rule [01] executed  
PHT[4] rule [13] executed  
PHT[8] rule [03] executed  
  
[+] Fuzzing process finished  
[+] Orcs (malformed ELF) saved in 'orcs_test/'  
[+] Detailed fuzzing report: 'orcs_test/Report_test.txt'
```



## 2. 진행 사항

- AFL(American fuzzy lop) 문서화
  - 예제를 통한 테스트 문서화
    - 화이트 박스 테스트
      - strcmp 취약점을 가진 소스코드 생성
      - testcase 생성
      - afl-gcc를 통한 취약 소스코드 컴파일
      - afl-fuzz를 통해 퍼징
      - 크래시 확인
    - 블랙 박스 테스트
      - strcmp 취약점을 가진 바이너리 파일
      - testcase 생성
      - afl-fuzz를 통해 퍼징
      - 크래시 확인

```
greendot@server:~/work/afl_test$ ./test
ID : root
PW : toor
로그인 성공 .
```

```
american fuzzy lop 2.52b (test)

process timing
  run time : 0 days, 0 hrs, 2 min, 26 sec
  last new path : 0 days, 0 hrs, 2 min, 1 sec
  last uniq crash : 0 days, 0 hrs, 2 min, 23 sec
  last uniq hang : none seen yet
cycle progress
  now processing : 0* (0.00%)
  paths timed out : 0 (0.00%)
stage progress
  now trying : splice 3
  stage execs : 15/16 (93.75%)
  total execs : 690k
  exec speed : 4823/sec
fuzzing strategy yields
  bit flips : 0/896, 0/890, 0/878
  byte flips : 0/112, 0/106, 0/94
  arithmetics : 0/6260, 0/3760, 0/3137
  known ints : 0/522, 0/1839, 0/2615
  dictionary : 0/0, 0/0, 1/108
  havoc : 1/280k, 2/388k
  trim : 18.80%/30, 0.00%
overall results
  cycles done : 133
  total paths : 6
  uniq crashes : 2
  uniq hangs : 0
map coverage
  map density : 0.00% / 0.01%
  count coverage : 1.71 bits/tuple
findings in depth
  favored paths : 3 (50.00%)
  new edges on : 3 (50.00%)
  total crashes : 63.1k (2 unique)
  total tmouts : 5 (3 unique)
path geometry
  levels : 3
  pending : 0
  pend fav : 0
  own finds : 2
  imported : n/a
  stability : 100.00%

[cpu000:107%]

+++ Testing aborted by user +++
[+] We're done here. Have a nice day!
```

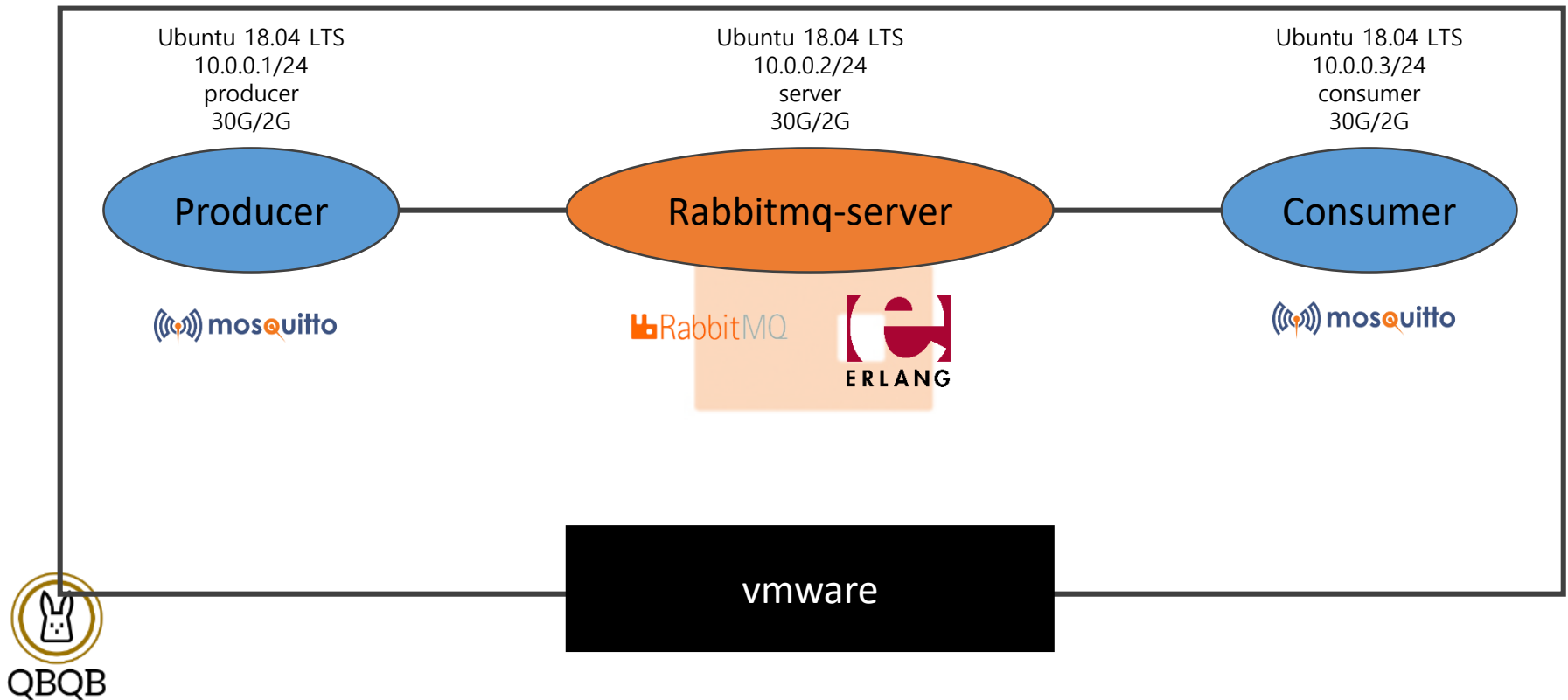
```
greendot@server:~/work/afl_test$ ls -al res/crashes
total 20
drwx----- 2 greendot greendot 4096 3월 13 21:40 .
drwxrwxr-x 5 greendot greendot 4096 3월 13 21:40 ..
-rw----- 1 greendot greendot 599 3월 13 21:40 README.txt
-rw----- 1 greendot greendot 81 3월 13 21:40 id:000000,sig:11,src:000002,op:havoc,rep:64
-rw----- 1 greendot greendot 217 3월 13 21:40 id:000001,sig:11,src:000002+000001,op:splice,rep:64
```

```
greendot@server:~/work/afl_test$ ./test < res/crashes/id:000000,sig:11,src:000002,op:havoc,rep:64
ID : PW : 실패.
Segmentation fault
```



## 2. 진행 사항

- MQTT\_fuzz 분석 및 문서화(1/3)
  - 테스트 베드 구축



## 2. 진행 사항

- MQTT\_fuzz 분석 및 문서화(2/3)

- 사용 방법 문서화

- 설치
    - 사용법

```
greendot@greendot-virtual-machine:~/work/mqtt_fuzz$ python mqtt_fuzz.py
usage: mqtt_fuzz.py [-h] [-ratio fuzz_ratio] [-delay send_delay]
                  [-validcases validcase_path] [-fuzzer fuzzer_path]
                  target_host target_port
mqtt_fuzz.py: error: too few arguments
```

- 테스트

- consumer 없이 producer와 server로 만 퍼징
    - 8시간 동안 동작하다 큐에 36만개의 메시지를 담고 메모리 경고 발생후 이후 메시지 모두 Drop하는 것을 확인
  - consumer-server-producer로 퍼징
    - 2시간 동안 동작 시켰으나, 특이사항 없었음



QBQB



- MQTT\_fuzz 분석 및 문서화(3/3)



## 2. 진행 사항

---

- RabbitMQ 문서화
  - Server Documentation 부분 문서화
  - 문서화가 완료된 부분
    - TLS Support
    - Production Checklist
    - Distributed RabbitMQ
    - Alarms
    - Clustering
    - Virtual Hosts
    - Access Control
    - Authentication Mechanisms
    - Lazy Queues
    - Firehose(Message Tracing)



# 3. 추후 계획

---

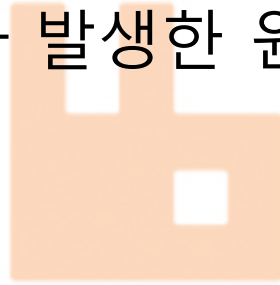
- 계획

- 메시지(AMQP, MQTT, STORM)를 통한 퍼징을 수행하여 크래시를 찾을 계획
  - 해당 크래시를 통해 익스플로잇 툴 개발 진행
- RabbitMQ Management 플러그인의 웹 취약점 분석 계획
- erlang fuzzer를 계속 찾아 rabbitmq-server에 적용 계획

# # 추가 사항

---

- RabbitMQ 로그 파일을 통한 계정 정보 취약점 발견
  - RabbitMQ 사용 중 Management UI에 접속하여 사용하던 중 Queue 설정 페이지가 로드가 갑자기 안되어 rabbitmq 로그 확인
  - 해당 로그파일은 other도 읽을 수 있는 권한으로 외부 사용자가 접근 시 Management UI에 로그인 할 수 있음을 확인
  - 하지만 해당 크래시가 발생한 원인을 찾지 못함



- RabbitMQ 로그 파일을 통한 계정 정보 취약점 발견

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

QB