

2022 IEEE/ACIS

FloTFuzzer

: Response-Based Black-Box fuzzing for IoT Devices

2023.05.03.

송실대학교 대학원 석사과정 서영재

INDEX

- 01.** Introduction
- 02.** Background and related work
- 03.** Design
- 04.** Implementation and Evaluation
- 05.** Discussion and Future work
- 06.** Conclusion

고도로 구조화된 메시지 프로토콜

대부분의 IoT 디바이스에서 사용되는
메시지 프로토콜은 고도로 구조화되어 있어서,
일반적인 테스트 방법으로는 처리하기 어려움

취약점 노출

54%의 보안 카메라가 취약점을 가지고 있으며,
이는 카메라를 해킹하여 사이버 범죄자가 공격을
수행하고 기업 네트워크에 접근할 수 있음

펌웨어 업데이트 부재

펌웨어 업데이트가 부재하거나,
업데이트를 수동으로 진행해야 함
보안 패치가 제대로 이루어지지 않아서 취약점이
지속적으로 존재할 수 있음.

인증 및 권한 부여 문제

디바이스에 대한 외부 공격을 용이하게 만들 수 있음

**보안
문제**

01 Introduction

Our Approach

응답 기반 블랙박스 퍼징 방식인 FIoTFuzzer 제안



- IoT 장치에서 발생할 수 있는 다양한 취약점을 탐지하기 위해 사용되는 블랙박스 퍼징 도구인 **Snipuzz**의 한계를 극복하여 방법론 개선
 - 메시지 세그먼트 분류가 너무 엄격하면 응답 유형이 너무 분할되어 취약점을 탐지하기 어려움
 - 제한된 구조화된 퍼징 입력만 생성할 수 있으며, 이로 인해 더 깊은 코드 위치를 도달하여 더 많은 취약점을 발견하는 것에 한계
 - 펌웨어의 내부 세부 정보를 알 수 없기 때문에 응답 코드를 분석하여 입력의 민감한 바이트를 식별하지만, 모든 취약점을 탐지하기에 충분하지 않음



- IoT 디바이스에서 사용되는 메시지 프로토콜의 형식을 처리할 수 있음
- 메시지 어댑터를 사용하여 원래의 통신 패킷에서 프로토콜, 형식, 인코딩 등의 정보를 식별
- 응답에 기반하여 메시지 세그먼트를 분할하여 문법 기반 퍼징 전략을 개선함

> IoT 디바이스에서 발생하는 심층 기능 구성 요소까지 도달할 수 있으며, 새로운 취약점을 탐지하는 데 매우 유용함

Contribution

New Technique

메시지 생성 및 전달을 제어하여 Fuzzer가 실행 중에 효과적인 IoT 디바이스 퍼징을 구현할 수 있도록 하는 메시지 어댑터 제안

New Framework

변이 시 정보 형식과 데이터 인코딩을 처리하기 위한 프레임워크 도입
원본 데이터 처리 과정에서 포맷을 벗어난 메시지 세그먼트를 변이 시킨 후 변이데이터를 인코딩

- 메시지 세그먼트가 과하게 세분화되는 것을 방지하기 위해 언듈레이터 설정

Implementation & findings

파이썬으로 FloTFuzzer을 구현하고 다른 블랙박스 퍼징 도구와 비교
8개의 디바이스에서 9개의 알려진 취약점 발견, 효율성 보여줌

- Undulator : 메시지 세그먼트화 문제를 해결하기 위해 특정 임계값을 설정하여 FloTFuzzer가 유사한 응답들을 하나의 카테고리로 묶어서 세그먼트화를 수행하는 기능

02

Background and Related work

IoT용 Fuzzer

- 네트워크 프로토콜 레벨에서 수행하는 테스트
- 시뮬레이션 환경을 기반으로 하는 테스트

Tab. 1. Fuzzers used for IoT devices

Fuzzer Name	Fuzz Layer	Hardware Support	Fuzzing Type
SRFuzzerr	Administration layer	SOHO router	Grey-box
Boofuzz	Internet interface	Bare-metal	Black-box
FIRM-AFL	Application	Emulation	Grey-box
IoTfuzzer	Internet interface	Bare-metal	Black-box
RPFuzzer	SNMP protocol	Router	Black-box

생성 기반 퍼징 도구

입력 구문 형식과 동작 상태를 모델링하여 테스트 케이스를 생성

- 고도로 구조화된 입력을 처리하는 소프트웨어 프로그램에 적합
- 현재 문법 및 상태 모델을 수동으로 구성하는 오버헤드가 높음
- 알려지지 않은 문법 규칙을 다루기가 어려움

02

Background and Related work

IoT용 Fuzzer

- 네트워크 프로토콜 레벨에서 수행하는 테스트
- 시뮬레이션 환경을 기반으로 하는 테스트

Tab. 1. Fuzzers used for IoT devices

Fuzzer Name	Fuzz Layer	Hardware Support	Fuzzing Type
SRFuzzerr	Administration layer	SOHO router	Grey-box
Boofuzz	Internet interface	Bare-metal	Black-box
FIRM-AFL	Application	Emulation	Grey-box
IoTfuzzer	Internet interface	Bare-metal	Black-box
RPFuzzer	SNMP protocol	Router	Black-box

변이 기반 퍼징 도구

미리 제공된 SEED 케이스에 변이 작업 세트를 적용하여 수정함으로써 새로운 테스트 케이스를 생성

- 간결하고 구현하기 쉽고 대규모 프로그램을 처리할 수 있음
- 실제로 널리 사용됨

02

Background and Related work

IoT용 Fuzzer

- 네트워크 프로토콜 레벨에서 수행하는 테스트
- 시뮬레이션 환경을 기반으로 하는 테스트

Tab. 1. Fuzzers used for IoT devices

Fuzzer Name	Fuzz Layer	Hardware Support	Fuzzing Type
SRFuzzerr	Administration layer	SOHO router	Grey-box
Boofuzz	Internet interface	Bare-metal	Black-box
FIRM-AFL	Application	Emulation	Grey-box
IoTfuzzer	Internet interface	Bare-metal	Black-box
RPFuzzer	SNMP protocol	Router	Black-box

IoTfuzzer

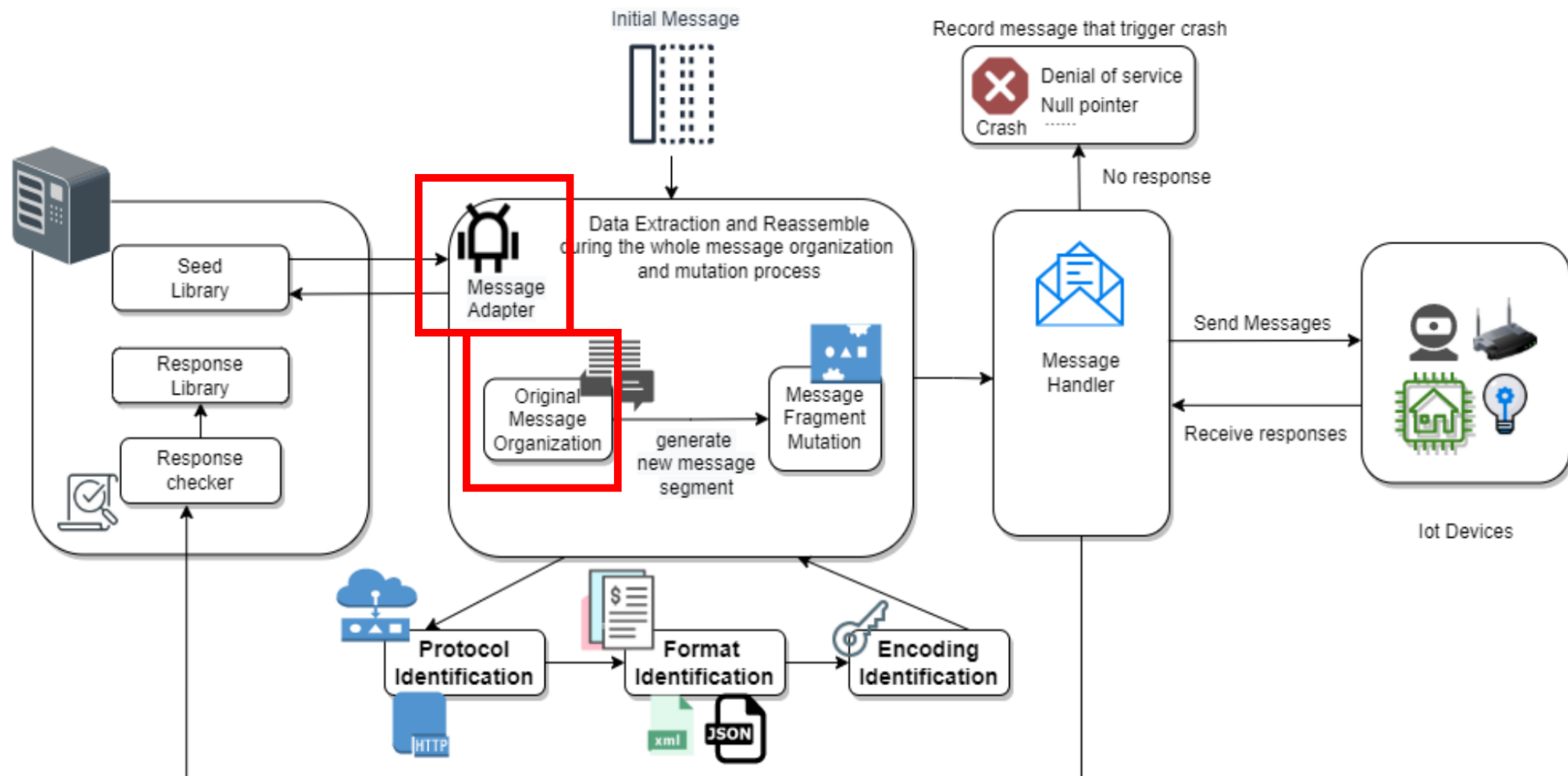
앱을 통해 장치에 테스트 메시지를 보내고
라이브니스 분석을 기반으로 시스템 상태를 관찰하고
IoT 장치의 메모리 관련 오류 감지를 실현하는 앱 기반 블랙박스 퍼징 도구

RPFuzzer

IoT 장치에 대량의 패킷을 전송하고 CPU 사용량을 모니터링하고
시스템 로그를 확인하여 IoT 장치의 라우팅 프로토콜 취약점을 탐지하는
블랙박스 퍼징 도구

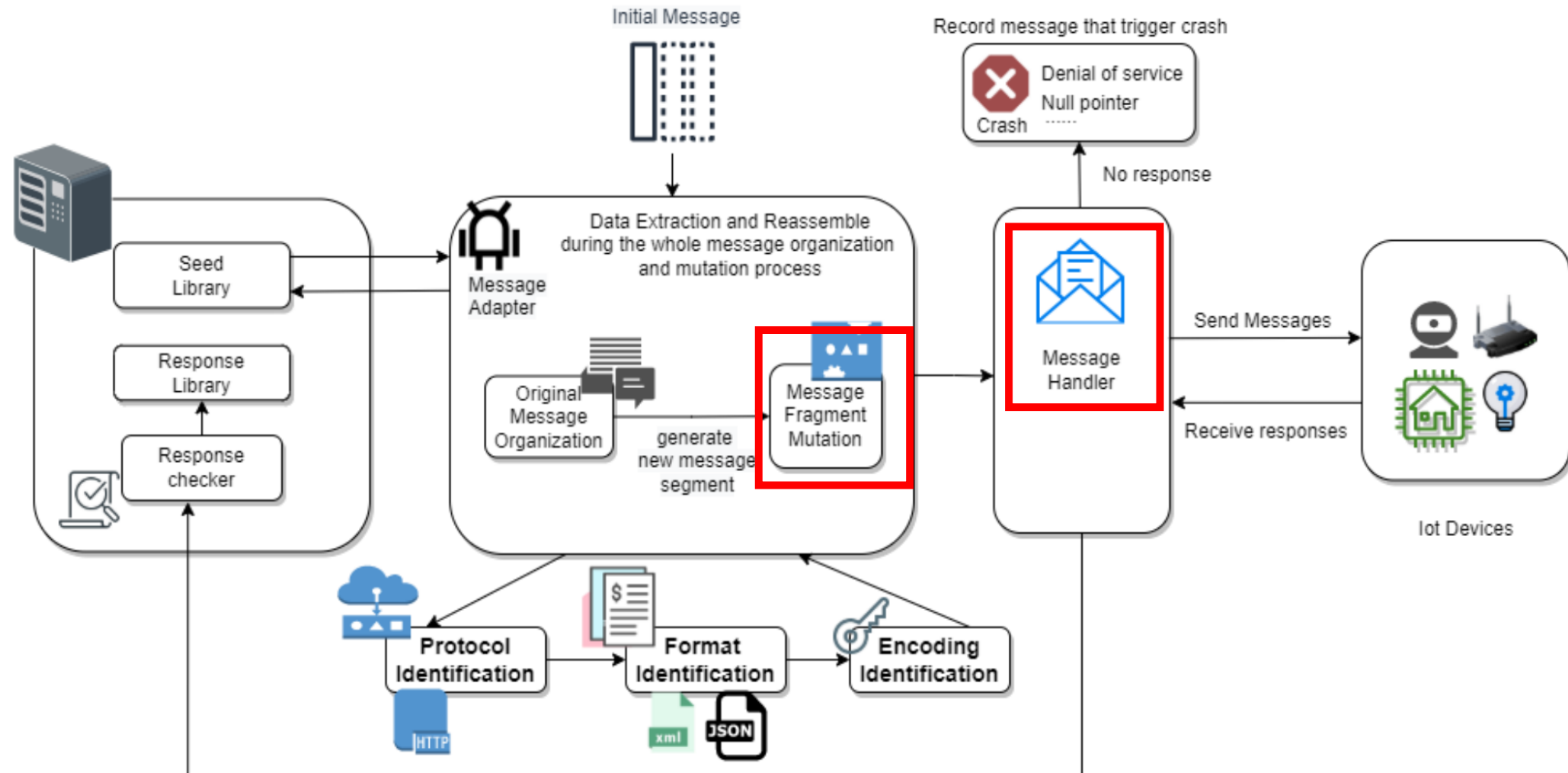
Design

03 FloTFuzzer Overview



Design

03 FloTFuzzer Overview



03 Design

Message Adapter

데이터 추출뿐만 아니라 전체 퍼징 과정에서 나타날 데이터 재조립(변이된 데이터를 원래 형식과 인코딩으로 복원)함

1 프로토콜 식별

응용 계층 프로토콜에 초점을 맞춤

애플리케이션 계층에는 세부 표준을 공개하는 **표준 네트워크 프로토콜**과 프로토콜 표준을 공개하지 않는 **독점 프로토콜**이 있음

원본 메시지를 얻은 후 타사 라이브러리를 사용하여 해당 형식을 구문 분석 > 구문 분석이 성공하면 해당 표준 네트워크 프로토콜에 대한 시맨틱 기능을 추출

2 형식 식별

예상된 형식을 제공하지 않는 메시지는 장치에서 즉시 폐기되므로 심층 기능 구성 요소에 도달 할 수 없음

코드 적용 범위를 확장하기 위해 메시지 어댑터는 통신 프로세스 중에 전달되는 데이터의 형식을 식별함

3 인코딩 식별

내장된 휴리스틱 인코딩 인식기를 사용함

base64 또는 URL 인코딩과 같은 일부 일반적인 인코딩은 일부 요청 헤더 또는 본문에 나타남

03 Design

Message Adapter

데이터 추출뿐만 아니라 전체 퍼징 과정에서 나타날 데이터 재조립(변이된 데이터를 원래 형식과 인코딩으로 복원)함

Input : string


Output : jsonstruct

1. for nextchar in string do
2. if nextchar = '' then
3. parse the string
4. if nextchar = '{' then
5. parse the string object
6. if nextchar = '[' then
7. parse the array
8. if nextchar = 'n' and next word is 'null' then
9. return None
10. return jsonstruct

Algorithm 1. Scan JSON object

string

```
{ "name": "John", "age": 30, "city": "New York" }
```



Jsonstruct

```
{  
    "name": "John",  
    "age": 30,  
    "city": "New York"  
}
```

03

Design

Message Adapter

base64 데이터 인코딩 콘텐츠가 포함된 패킷

인코딩된 데이터의 무작위 변이는 원본 콘텐츠를 무의미하게 만들고 초기 구문 분석기에서 장치에 의해 폐기될 수 있음

휴리스틱 인코딩 인식기는 이전 단계에서 형식 확인으로 구분된 각 필드의 값에 대한 인코딩을 확인함

인코딩 형식이 인식되면 이를 디코딩한 후 처리를 위해 다음 단계에서 메시지 세그먼트 조직에 전달함

POST /cgi-bin HTTP/1.1

Host: 192.168.2.2

Content-Length: 69

Content-Type: application/json

Authorization: Basic

YWRtaW46YCB3Z2V0IGh0dHA6Ly8xOTIuMTY4LjIuMTQ0MDAwL3Rvb2xzL21zZiAtTyAvbXN
mXG5jaG1vZCA3NzcgL21zZ1xuL21zZmA=

{"profile": {"name": "Lee", "age": 30}}

Design

03 Original Message Organization

- Snipuzz와 유사한 방식을 사용하여 각 메시지 **Snippet Determination**
- Seed Corpus에 속하는 각 message에 대한 프로브 메시지 생성 후, IoT 장치에 전송(응답 메시지 수집)
 - 프로브 메시지 p_i : message m 에서 특정 바이트를 삭제하여 생성하는 것
- Heuristic algorithm을 사용하여 각 메시지의 응답 메시지를 대략적 초기 snippet으로 나눔
 - 각 프로브 메시지의 응답 메시지를 분류하여 **스니펫** 생성
- 두 개의 응답이 동일한 카테고리(category)에 속하는지 비교하기 위해서 **Edit distance** 방법 사용
 - **Edit distance** : 두 시퀀스(sequence) 간의 유사성(similarity)을 계산하는 데 사용
- 동일한 메시지 조각을 1초에 두 번 전송하고 메시지 조각의 자체 유사도 s_{ii} 점수 계산 후, 두 응답이 같은 범주에 속하는지 여부 결정
 - 서로 다른 두 개의 응답 r_i 와 r_j 에 대해 두 응답 간의 유사도 s_{ij} 계산
 - $s_{ij} \geq s_{ii}$ 또는 $s_{ij} \geq s_{jj}$ 라면, 두 응답이 같은 category에 속하는 것으로 간주

$$s_{ij} = 1 - \frac{\text{edit_distance}(r_i, r_j)}{\max_len(r_i, r_j)} \quad (1)$$

- r_i, r_j : 응답 메시지
- $\text{edit_distance}(r_i, r_j)$: 두 응답 메시지 간의 수정될 수 있는 최소 횟수
- $\max_len(r_i, r_j)$: 두 응답 사이에 긴 응답 메시지 선택

03 Design

Original Message Organization

Tab. 2. Different parameters cause different response content and similarity score

	Message	Response	Similarity Score	Type
패킷 파라미터	<code>{"id":1,"method":"set_rgb","params": [652,"smooth",300]}</code>	<code>{"method":"props","params":{"hue":239,"rgb":652}}</code>	$S_{11}=1.000$	1
'2' 삭제	<code>{"id":1,"method":"set_rgb","params": [65,"smooth",300]}</code>	<code>{"method":"props","params":{"hue":240,"rgb":65}}</code>	$S_{12}=0.939$	1
's' 삭제	<code>{"id":1,"method":"set_rgb","params": [652,"mooth",300]}</code>	<code>{"method":"props","params":{"hue":239,"rgb":652}}</code>	$S_{13}=1.000$	1

Design

03 Original Message Organization

극단적인 경우 응답 수보다 응답 유형이 더 많아 메시지 조각화 정도가 증가할 수 있는 문제

- s_{jj} , s_{ii} 와 s_{ij} 를 비교할 때 특정 임계값을 설정하는 기능을 하는 언듈레이터를 설정
- 메시지 조각화 정도를 줄이고 작업의 일반화 능력을 향상시킴

메시지 조각을 더 세분화하기 위해서는?

- 순위가 매겨진 클러스터를 coacervating 과정에서 메시지 추출 (Snipuzz 의 계층적 클러스터 방식 채택)
- 하나의 클러스터는 프로세스 후에 남아 있으며 현재 단계에서 가장 유사한 두 개의 클러스터가 지속적으로 병합되고 종료됨

연속 압축의 핵심 아이디어 : 기존 coacervating 클러스터를 벡터화하고 메시지 세그먼트에서 대표 기능을 추출하는 것

- 클러스터가 하나만 남을 때까지 유사한 coacervating 클러스터가 병합됨(Bottom-up 방식)
 - 새로 생성된 모든 메시지 세그먼트는 초기 메시지와 함께 다음 단계의 메시지 조각 변형에 사용됨

03 Design Message Fragment Mutation

Offutt[15]가 제안한 22가지 돌연변이 전략 중 5가지를 결합하면 돌연변이 테스트가 단기간에 최대 분기 커버리지를 얻을 수 있음

플립 비트

- 펌웨어의 파서에 대한 챌린지
- 메시지 조각의 모든 바이트를 비트 단위로 뒤집음

바이트 삭제

- 대상 세그먼트를 무작위로 삭제
- 펌웨어가 데이터 필드를 제대로 확인하지 않으면 충돌이 발생할 수 있음

특수문자

- 일반적인 인코딩의 형식 문자열 삽입 및 교체하고 문자를 자름
- 구문 분석기가 메시지 필드 내용을 처리할 때 버퍼 오버플로 및 문자열 잘림과 같은 문제가 있을 수 있음

사전 설정

- 메시지 세그먼트를 사전의 콘텐츠로 바꿀 수 있는 값으로 사전을 설정함
- 좋은 사전도 퍼징 성공률에 중요한 역할을 함

바이트 복사

- 많은 콘텐츠를 반복하면 펌웨어에서 메모리 필터링이 발생할 수 있음
- 메시지 조각을 여러 번(예: 1000번) 반복

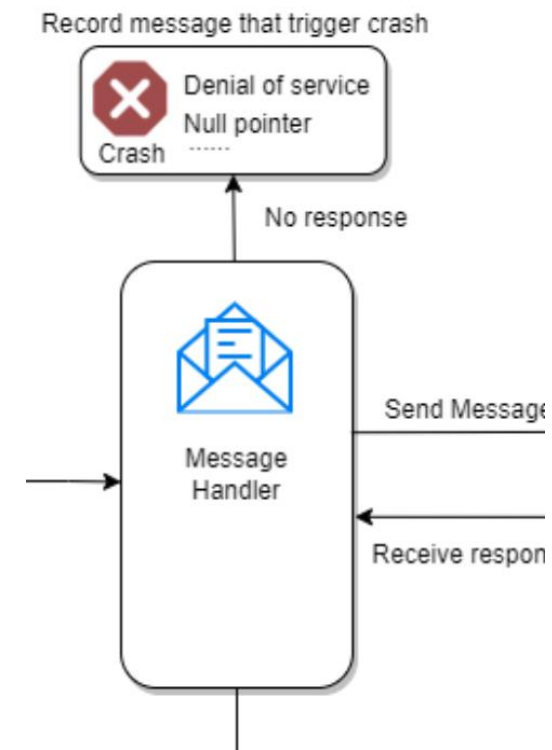
03 Design Message Handler

전송 계층의 TCP, UDP 및 애플리케이션 계층의 HTTP 프로토콜과 같이 원래 테스트 스위트와 IoT 장치 간에는 여러 가지 통신 방법이 있을 수 있음

➤ 정보를 송수신하는 기능을 보다 널리 사용할 수 있도록 소켓을 사용하여 통신함

특정 신호가 수신되거나 여러 연속 요청에 대한 응답 수신 > 충돌이 발생했을 수 있음

➤ FloTFuzzer는 충돌과 해당 패킷을 기록함



04

Implementation and Evaluation

프레임워크 구현

초기 설정

- 테스트 환경에서 베어 메탈(bare-metal) 및 시뮬레이션된 스마트 디바이스를 사용할 수 있음
- Wireshark를 사용하여 테스트 스위트와 IoT 디바이스 간의 통신 데이터를 캡처함

메시지 수집

- Wireshark를 사용하여 캡처한 원시 데이터를 수집하고 후속 퍼징에 사용됨

메시지 어댑터

- 메시지 어댑터는 전체 메시지 조직 및 변형을 담당함
- 메시지의 비-포맷 부분을 세그먼트로 분할하고 변형함
- 메시지 어댑터는 원래 인코딩 형식을 복원하고, 처리를 위해 메시지 핸들러에 전달함

메시지 핸들링

- 메시지 핸들러는 변형된 메시지를 받아들여 처리함
- 이 단계에서 실제 퍼징이 발생할 수 있음
- 메시지 핸들러는 취약점이나 예외 동작을 탐지하는 로직을 포함할 수 있음

결과 분석

- FloTFuzzer는 퍼징 작업의 결과를 분석함
- 취약점이나 예외 동작을 감지하고, 이를 보고할 수 있음

04

Implementation and Evaluation

실험 환경 설정

Intel Core i5 - 8265U × 1.60GHz CPU

8G RAM

Windows10 PC

- Boofuzz 와 Sulley를 벤치마크로 사용
- 12개의 장치에서 14개의 취약점을 포함하는 데이터 세트 구축

일부 평가 대상 디바이스에 대한 세부 정보

Tab. 3. Evaluated Devices

Device ID	Vendors	Models	Device Types	Firmware Versions
1	Tenda	AC9	Wi-Fi router	V15.03.05.19 (6318)
2	Tenda	AC15	Wi-Fi router	V15.03.05.19(6318)_CN
3	TP-link	TL-WR941	Wi-Fi router	V6_150312
4	TP-link	TP-Link WR841N	Wi-Fi router	(EU)_V14_180319
5	TP-link	TL-IPC42C-4	Wireless camera	1.0.6 Build 201126 Rel.62706n
6	D-Link	DIR-822	Wi-Fi router	FW v1.03
7	MikroTik	CHR	Wi-Fi router	6.40.5
8	Xiaomi	MJDP003	Smart bulb	1.4.4_0031
9	Mercury	MIPC372-4	Camera	1.0.1
10	Amcrest	IP2M841	Camera	V2.800.0000000.1R

04

Implementation and Evaluation
취약점 탐지의 효율성

FloTFuzzer 에서 발견한 알려진 취약점

취약점 탐지 효율성 및 기능을 비교하기 위해 24시간동안 실행함
충돌 발생 시 자동으로 디바이스를 재설정하여 테스트 재개

9개의 취약점 발견

- 버퍼 오버플로우 3개
- DOS 6개

FloTFuzzer가 메시지 어댑터 메커니즘을 기반으로
디바이스의 취약점을 자동으로 탐지할 수 있음을 보여줌

Tab. 4. List of discovered known vulnerabilities

Exploit ID	Vulnerability Type	FloTFuzzer	Boofuzz	Sulley
CVE-2020-8423	Buffer Overflow	150s	384s	286s
CVE-2019-6258	Buffer Overflow	145s	264s	N/A
CNVD-2021-30168	Buffer Overflow	63s	55s	N/A
CNVD-2021-35879	DOS	113s	N/A	N/A
CNVD-2021-81533	DOS	66s	N/A	N/A
CNVD-2021-35790	DOS	71s	136s	N/A
CNVD-2021-24948	DOS	134s	N/A	156s
CNVD-2022-16280	DOS	59s	N/A	74s
CNVD-2020-67555	DOS	34s	78s	52s

04 Implementation and Evaluation

최적화 효과

Tab. 5. Statistics of message fragmentation

Device ID	Message fragmentation		
	Threshold=0.6	Threshold=0.3	Threshold=0
1	3	5	8
2	1	7	15
3	5	8	9
4	3	4	4
5	6	8	12
6	2	6	13
7	2	7	19
8	1	1	1
9	3	5	8
10	4	9	12
11	6	11	20
12	4	8	13
Average	3	7	11
Total	40	79	134

하나의 장치에 대해 해당 개수의 응답 세그먼트를 얻음

임계값을 설정하면 메시지 조각화 정도를 효과적으로 제어할 수 있음을 증명함

05 Discussion and Future work

Special Content

특수 분야를 인식하는 능력을 향상시켜야 함
일부 패킷에는 일부 특정 필드가 있음
IoT 장치는 특수 필드의 값을 감지하며, 형식이 올바르지 않으면 변형이 유효하지 않음

Data Encryption

암호화된 데이터 처리에 제한이 있음
일부 장치는 전송되는 콘텐츠를 암호화하고 클라우드 서버는 전송 프로세스에 참여함
클라우드 서버는 사용자의 지시를 IoT 장치에 전달하거나 IoT 장치의 반응에 대한 피드백을 사용자에게 전달하는 중개자 역할을 함
> 이 경우 FloTFuzzer는 암호화된 필드를 분석하기 어려움

Assign Reason

성공적으로 트리거된 취약성의 징후는 라우터 장치가 충돌 응답을 보내는 것이므로 취약성의 원인을 사용자에게 직접 보고할 수 없음
FloTFuzzer는 취약점을 트리거하는 메시지를 기록할 수 있지만 특정 문제 원인을 확인하려면 수동 작업이 필요함

06 Conclusion

- 본 논문에서는 IoT 장치에 숨어 있는 취약성을 발견하는 효과를 향상시키기 위해 설계된 블랙박스 퍼징 프레임워크 FloTFuzzer를 제안
 - 다른 블랙박스 퍼징 테스트와 달리 FloTFuzzer는 메시지 어댑터를 사용하여 데이터를 추출하고 재구성하여 IoT 장치의 고도로 구조화된 형식 요구 사항을 극복함
 - FloTFuzzer는 장치에서 반환된 응답 메시지를 사용하여 Fuzz 변환 프로세스를 안내하는 피드백 메커니즘을 설정하는 Snipuzz의 방법을 개선함
 - 임계값을 설정하면 메시지 세그먼트의 조각화를 더욱 줄이고 변환 효율을 향상시킬 수 있음
 - FloTFuzzer는 우리가 준비한 데이터 세트에서 실험을 수행하여 Boofuzz와 Sulley보다 더 많은 취약점을 발견했으며, 이는 FloTFuzzer의 효과를 보여줌

Q&A