[ACM CCS 2021] SNIPUZZ : Black-box fuzzing of IoT Firmware via Message Snippet Inference

1) Initial Seed 획득

IoT 시장의 성장과 보급이 확대가 되었는데, 일반적으로 장치 펌웨어 내의 구현 결함으로 IoT 취약점 발생함

대부분 제조업체는 제품 펌웨어를 공개하지 않으므로 코드 기반 동적 분석 방법을 수행하기 어려움

IoT 장치는 인터넷과 통신할 수 있는 기능을 포함하여 네트워크 통신을 통해 장치 펌웨어를 테스트 할 수 있음

-실행 중 충돌이 발생하거나, 비정상적인 메시지를 재전송하도록 장치가 PUSH되는 경우

네트워크 통신을 사용하여 IoT 장치의 펌웨어 Fuzzing의 어려움이 존재함

1) 장치에서 내부 실행정보를 얻을 수 없음

- Mutaion Seed 선택이 Random이라, Mutation 전략 최적화가 어려움

2) 입력에 대해 엄격한 문법을 적용

- Random Mutation에 의해 생성된 대부분의 메시지는 입력 구문 규칙 위반 처리가 됨
- 실행되기 전, 펌웨어의 구문 유효성 검사 중 거부가 되는 것임

3) 비표준 통신 형식 사용

- 문법 기반 Mutation 전략은 입력 요구 사항을 충족하는 메시지를 생성할 수 있음

Challenge

1) FeedBack Mechanism 부족

- 펌웨어의 내부 실행 정보를 얻는 것은 거의 불가능
- IoT 장치에서 피드백을 얻고 메시지 생성 프로세스를 최적화하기 위한 경량 솔루션 필요함

2) 다양한 메시지 형식

- 다양한 장치에 적용되기 위해서 원시 메시지에서 형식을 유추해야 함

3) Randomness of Response

- 응답 메시지에는 타임스탬프/토큰과 같은 임의의 요소가 포함될 수 있음
- 이는 동일한 메시지에 다른 응답을 생성하고 퍼징 효율성을 감소시킴

IoT 펌웨어의 취약점을 탐지하기 위한 블랙박스 IoT Fuzzing인 Snipuzz를 제안함

- IoT 장치의 피드백으로 **응답 메시지를 활용하여 Snippet 기반 Mutation Strategy**를 구현함
    - 메시지의 바이트를 하나씩 변경하여 Probe Message를 생성하고, 장치에서 수집된 해당 응답을 분류함

    - Message에서 동일한 역할을 하는 인접한 바이트는 Mutation의 기본 단위인 Initial Message Snippet을 형성함
- **유사성 점수 및 계층적 클러스터링 전략**을 활용함
    - Mutation Stragtegy를 최적화, Randomness of Response Message 및 펌웨어 내부 메커니즘으로 인한 Category의 오분류 감소
- 따라서, IoT 장치의 문법 규칙 및 내부 실행 정보의 지원 없이 IoT 장치의 펌웨어를 효과적으로 테스트할 수 있음

Contribution

1) Message snippet inference mechanism

- IoT 장치의 응답은 펌웨어 코드 실행 경로와 관련, 응답을 기반으로 message snippet과 펌웨어의 코드 실행 경로 간의 관계 추론

2) More effective IoT fuzzing

- 응답 카테고리의 수는 펌웨어의 코드 실행 경로 수와 양의 상관관계

3) Implementation and vulnerability findings

- Snipuzz의 프로토타입 구현 및 최신 퍼징 도구와 비교
    - 20개 IoT 장치 중 5개의 장치에서 null 포인터 예외, 서비스 거부 및 알 수 없는 충돌을 포함하여 5개의 제로데이 취약점을 발견
        - 그 중 3개는 snipuzz에서만 노출

Background

대부분의 IoT 장치는 유사한 High-level 통신 아키텍처를 구현함

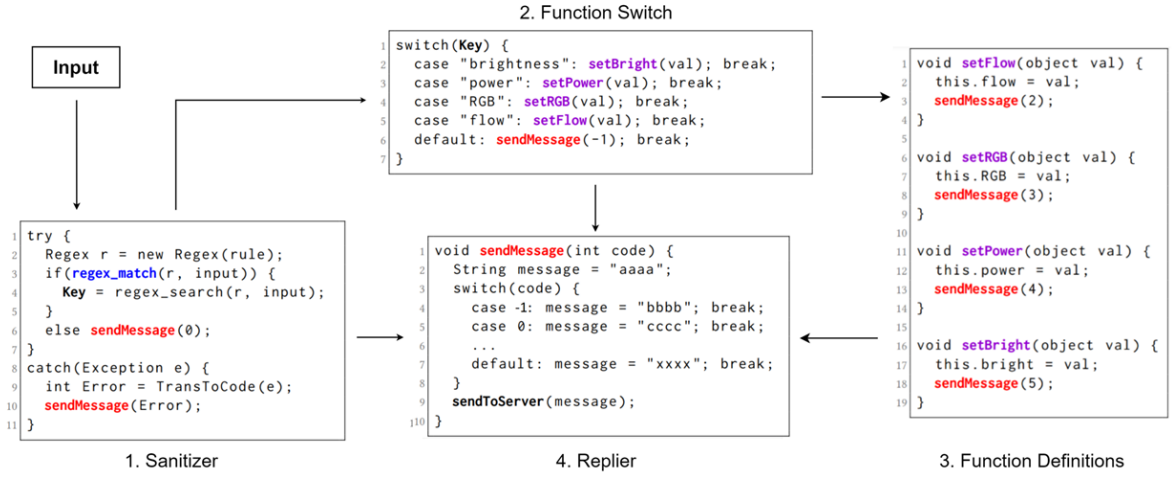


Figure 1: Interaction with IoT Firmware

1) Sanitizer : 외부 입력을 수신 시, 구문 분석 및 Regular Matching 수행

2) Function Switch: 입력에서 추출된 속성 키 및 해당 값에 따라 해당 Function으로 제어를 전송

3) Function Definitions: 각 펌웨어마다 정의된 기능들

4) Replier: 클라이언트에 응답을 보내는 역할

Motivation

Response-Based Feedback Mechanism

- 네트워크 기반 IoT Fuzzer는 장치의 실행 상태를 직접 얻을 수 없으며 feedback mechanism을 설정하기 어려움

  → 입력 메시지에 대한 응답 메시지를 피드백 메커니즘으로 사용

  - 펌웨어에서 반환된 응답 메시지의 내용을 통해 **펌웨어에서 실행된 코드 블록을 유추**할 수 있음
  - 펌웨어는 다른 function을 실행하더라도 동일한 응답 메시지를 반환할 수 있음
  - **서로 다른 두 입력이 서로 다른 응답 메시지를 받으면, 서로 다른 펌웨어 코드 실행 경로로 가는 것으로 추론할 수 있음**

Message Snippet Inference

- 대부분의 IoT 장치는 입력 구문이 다양하고 비표준을 사용하는 경우가 많음

- Grammar-Based Mutation 전략을 위한 통신 형식을 대규모로 다루는 Train Dataset을 구축하는 것은 어려움

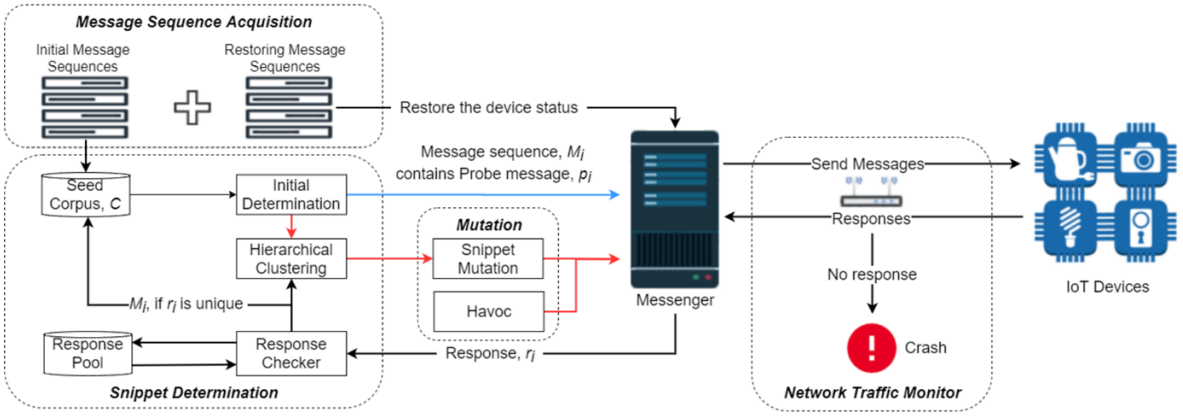
  → Message Snippet을 추론함

  - 유효한 메시지를 바이트 단위로 변경하며 다양한 응답을 수집함
  - **동일한 응답을 가진 연속 바이트를 하나의 Snippet으로 병합할 수 있음**

Methodology

SNIPUZZ는 IoT 장치에서 특정 작업을 요청하기 위해 Message Sequence M을 보내는 클라이언트 역할임

- 모든 메시지는 IoT 장치에 특정 기능이나 작업을 요청함



Message Sequence Acquisition(메시지 시퀀스 획득)

초기 Seed들의 품질은 Fuzzing 결과에 상당한 영향을 미칠 수 있으므로, Highly-Structured Formats을 준수하는 High-Quality 초기 Seed를 얻어야 함

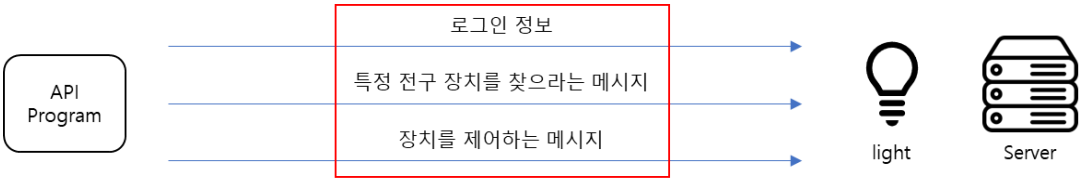
Wireshark를 사용하여 API와 IoT 장치 간의 통신 패킷을 감지 및 기록

1) Initial Seed 획득

- 통신 패킷의 대부분 구조 정보 및 프로토콜은 API 프로그램에서 Message Payload로 정의됨
  - 구조 정보 : Header, Message Contents ..

- 프로토콜 : HTTP, MQTT ..

- Target IoT 장치와 통신하는 동시에 Message Sequence를 초기 Seed로 추출함
  - 예시) API 프로그램을 사용하여 전구를 키고 끌 때,



* Snipuzz는 IoT 장치의 기능을 트리거하는 message sequence를 캡처(**initial seed**)

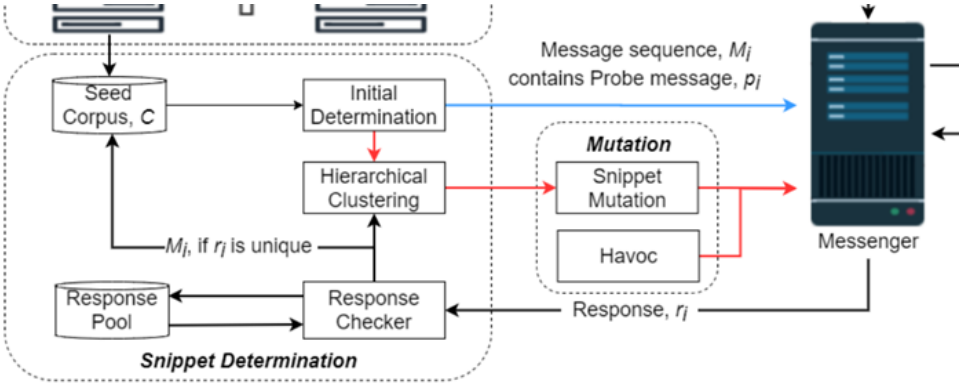
2) Restoring Message Sequence Acquisition(메시지 시퀀스 획득 복원)

- IoT 장치의 Crash 분류를 위한 Test Case를 재연하기 위해 테스트 중인 장치가 각 Test Round에서 동일한 초기 State를 가지도록 함
- 장치에서 Message Sequence 를 전송한 후, 장치를 사전 정의된 상태로 재설정하기 위해 Restoring Message Sequence를 전송함

3) Manual Efforts

- IP 주소 및 로그인 정보 설정과 같은 Test Suites 에서 프로그램을 수동으로 구성함(IoT 장치당 한번만 구성)
- Message Sequence를 동적으로 캡쳐하기 위해 Network Traffice Monitor에서 특정 형식과 프로토콜을 수동으로 정의
- Fuzzing Process를 잘못 이끌 일부 Message Sequence를 필터링
  - 예시) 일부 API 프로그램은 장치를 자동 업데이트하거나 다시 시작할 수 있는 작업을 제공
    - 이와 같은 작업은 장치를 중지하므로 응답이 전송되지 않음
    - 무응답 실행을 충돌로 간주하기 때문에 false-positive 충돌이 발생
- Manual work는 장치당 약 5명이며, Snipuzz의 message sequence acquisition 단계에서만 필요
- IoT 장치의 물리적 재시작 기능을 구현하기 위해 스마트 플러그 사용

Snippet Determination



1) Initial Determination

- Seed Corpus에 속하는 각 message에 대한 probe message 생성 후, IoT 장치에 전송(response messages 수집)
  - "Probe message "p_i ": " $Message\ m$"에"서 특정 바이트를 삭제하여 생성하는 것
- Heuristic algorithm을 사용하여 각 메시지의 response messages를 대략적 initial snippet으로 나눔
  - 각 Probe messages의 response messages를 분류하여 message snippet을 결정

Table 2: Examples of probe messages and corresponding response messages.

Messages	Content	Responses	Content	Category
Message m	{"on":true}	Response r_0	{"success":"/lights/1/state/on":true}	0
Probe message p_1	"on":true}	Response r_1	{"error":{"type":2,"address":"/lights/1/state","description":"body contains invalid json"}}	1
Probe message p_2	{on":true}	Response r_2	{"error":{"type":2,"address":"/lights/1/state","description":"body contains invalid json"}}	1
Probe message p_3	{"n":true}	Response r_3	{"error":{"type":6,"address":"/lights/1/state/n","description":"parameter, n, not available"}}	2
Probe message p_4	{"o":true}	Response r_4	{"error":{"type":6,"address":"/lights/1/state/o","description":"parameter, o, not available"}}	3
Probe message p_5	{"on:true}	Response r_5	{"error":{"type":2,"address":"/lights/1/state","description":"body contains invalid json"}}	1
Probe message p_{11}	{"on":true	Response r_{11}	{"error":{"type":2,"address":"/lights/1/state","description":"body contains invalid json"}}	1

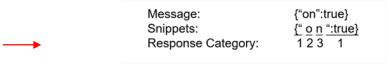


Figure 3: An example of snippet determination.

- Response message에 타임스탬프/토큰 등(Randomness)이 포함될 경우, 의미상 동일한 Response messages가 다른 카테고리로 분류됨
- Response category를 결정하기 위해 문자열의 유사성을 계산하는 Edit Distance(Similarity score) 사용
  - $If(S_ik >= S_ii) or (S_ik >= S_kk)$ "satisfies, responses " r_i" and " r_k"will be considered belonging to the same category".
  - $Self - similarity(S_ii)$: "1" 초 간격으로 p_i를 두 번 전송, response message(r_i, r_i') 대상으로 Similarity score 계산

- 새로운 response message(r_i)에 대해 Response Pool의 모든 response message와의 Similarity score를 비교
  - r_i 이 기존 category에 속하지 않는 경우, (p_i, r_i)가 Response Pool에 추가

$$s_{kt} = 1 - \frac{edit\_distance(r_k, r_t)}{max\_len(r_k, r_t)}, \qquad (1)$$

- r_k, r_t: a response message
- *edit_distance*(r_k, r_t) : counts the minimum number of operations
- *max_len*(r_k, r_t) : selects the longer string between the two responses

1. {"error":{"type":2, "address":"/lights/1/state", "timestamp":23032**9**", "descriptions":"body contains invalid json"}}
2. {"error":{"type":2, "address":"/lights/1/state", "timestamp":23032**8**", "descriptions":"body contains invalid json"}}

   “timestamp” 값의 마지막 자리만 다름(9→8 교체)

   edit_distance(1, 2) = 1

2) Hierarchical Clustering

- Similarity score를 이용하여 randomness of responses를 완화하지만, responses는 여전히 다른 범주로 잘못 범주화 될 수 있음
  - Response messages에 probe message에서 추출되거나 복사된 내용이 포함된 경우 발생
- Agglomerative hierarchical clusters 사용
  - 하나의 클러스터만 남을 때까지 가장 유사한 클러스터를 계속 병합(bottom-up 방식)
  - Vector: (self-similarity score, length of the response, number of alphabetic segments, number of numeric segments, number of symbol segments)

Table 2: Examples of probe messages and corresponding response messages.

Messages	Content	Responses	Content	Category
Message m	{"on":true}	Response r_0	{"success":"/lights/1/state/on":true}	0
Probe message p_1	"on":true}	Response r_1	{"error":{"type":2,"address":"/lights/1/state","description":"body contains invalid json"}}	1
Probe message p_2	{on":true}	Response r_2	{"error":{"type":2,"address":"/lights/1/state","description":"body contains invalid json"}}	1
Probe message p_3	{"n":true}	Response r_3	{"error":{"type":6,"address":"/lights/1/state/n","description":"parameter, n, not available"}}	2
Probe message p_4	{"o":true}	Response r_4	{"error":{"type":6,"address":"/lights/1/state/o","description":"parameter, o, not available"}}	3
Probe message p_5	{"on:true}	Response r_5	{"error":{"type":2,"address":"/lights/1/state","description":"body contains invalid json"}}	1
Probe message p_{11}	{"on":true	Response r_{11}	{"error":{"type":2,"address":"/lights/1/state","description":"body contains invalid json"}}	1

- $S_{34} = 0.979$
- $S_{33} = 1, S_{44} = 1$
- 같은 범주에 속하기 위한 조건: $(S_{ik} >= S_{ii})$ or $(S_{ik} >= S_{kk})$

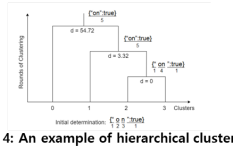


Figure 4: An example of hierarchical clustering.

Mutation

Snippet Mutation

- 메시지의 단일 바이트 대신 전체 Snippet에서 Mutation Schemes 수행
- **Mutation Schemes**
  - **Empty**
    The empty of a data domain may crash the firmware if the data domain is not properly checked.
    Delete an entire snippet to empty the data domain.
  - **Byte Flip**

    To detect bugs in both the syntax parsers and the functional code, Snipuzz flips all bytes in a snippet.
    Byte Flip changes the values of data domains to examine firmware.
  - **Data Boundary**

    To detect the out-of-bound bugs that occur during assignment, Snipuzz modifies the values of numeric data to some boundary values.
  - **Dictionary**

    For the scheme of Dictionary, Snipuzz replaces a snippet with a pre-defined string such as “true” and “false”, which may directly explore more code coverage.
  - **Repeat**

    In order to detect bugs in syntax parsers, Snipuzz repeats a snippet for multiple times.

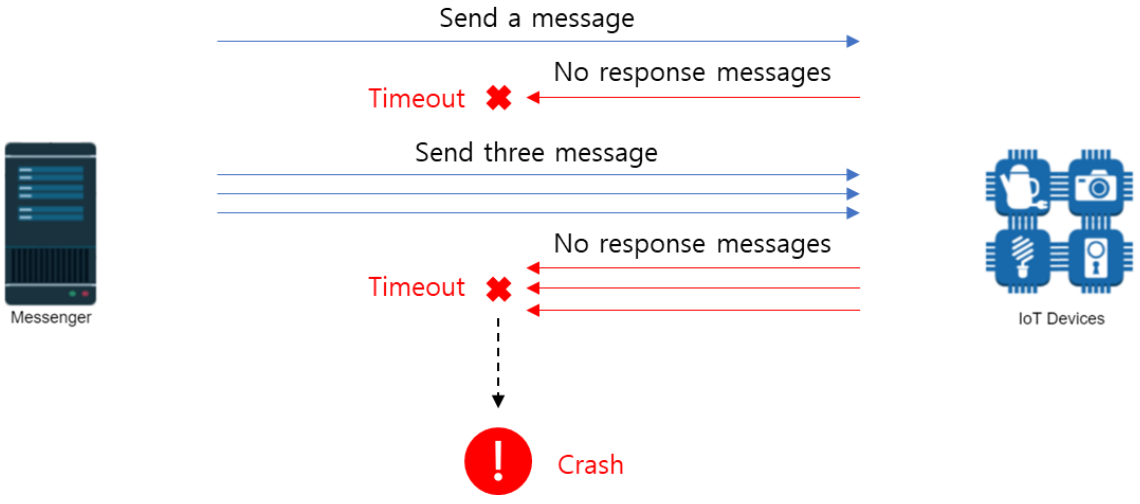
The repetition of data domain can detect defects caused by out-of-boundary problems.

- **Havoc**
  - 버그를 유발하기 위해 동일한 메시지에서 다른 데이터 도메인을 수정해야 할 수 있음
  - 메시지에서 임의의 snippet을 임의로 선택하고 선택한 snippet 각각에 대해 mutation schemes 수행
  - Havoc mutation은 새로운 response category를 찾거나 대상 IoT 장치가 충돌할 때까지 멈추지 않음

Network Traffic Monitor

Network Communication Monitoring는 장치에 전송된 모든 메시지를 기록, 장치가 응답하지 않을 때 장치에 메시지 재전송

- Input Message로 인해 IoT 장치가 충돌했는지 여부를 결정하기 위해 **TimeOut**을 설정함
  - 동일한 Message Sequence를 3번 연속 전송
    - TimeOut의 원인이 장치 충돌이 아닌, 네트워크 변동일 수 있기 때문에 3번 연속 전송함
    - 다시 TimeOut 발생 시, IoT 장치의 충돌로 간주하며, 해당 Message Sequence로 기록함



Experimental Evaluation

실험 Setup

- IoT 장치를 초기화하기 위해 제조업체에서 제공하는 응용프로그램을 사용하여 페어링
- 네트워크 통신을 모니터링하기 위해 테스트 중인 모든 장치는 로컬 라우터에 연결
    - PC: Intel Core i7 six-core x 3.70 GHz CPU, 16GB RAM
- IoT Devices
    - Philips, Xiaomi, TP-Link 및 Netgear와 같은 다양한 유명 브랜드 포함하여 전세계 베스트셀러 20개 IoT 장치 선택
- Benchmark tools
    - 충돌 및 메시지 분할을 찾는 Snipuzz의 성능 확인을 위해 7가지 퍼징 방법론을 사용
        - IoTFuzzer(NDSS, 2018): application에 대한 정적분석, 특정 변수의 값을 mutation
        - Nemesys(Usenix, 2018): 네트워크 메시지 분석을 위한 프로토콜 리버스 엔지니어링 도구, 데이터 도메인의 경계 값 추론
        - BooFuzz-Default, Byte, Reversal: 네트워크 프로토콜, human-guided message segment strategy
        - Doona: 네트워크 프로토콜의 버퍼 및 형식과 관련된 취약점 탐지 도구, 패킷을 seed로 사용하지 않음, 테스트케이스는 각 장치/프로토콜에 대해 미리 정의되어야 함
        - Snipuzz-NoSnippet: Snippet determination component 없이 메시지의 byte mutation

실험 결과

- **SNIPUZZ에서 5개의 IoT 장치에서 13개의 충돌 감지**

Table 3: Experiment Results. Snipuzz discovers the most number of categories and exposes the most number of bugs.

#	Devices	Snipuzz			IoTFuzzer			Doona		BooFuzz-Default		BooFuzz-Byte		BooFuzz-Reversal		Nemesys		Snipuzz-NoSnippet	
		T	C	10/24	T	C	10/24	C	10/24	C	10/24	C	10/24	C	10/24	C	10/24	C	10/24
1	YLDP05YL	UC	3*	**46/71**	UC	1*	31/33	NA	NA/NA	0	11/17	0	11/41	0	11/22	0	26/61	0	21/69
2	YLDP13YL	UC	2*	**35/76**	UC	1*	20/24	NA	NA/NA	0	8/18	0	8/42	0	8/22	0	18/62	0	22/70
3	A60	DoS	1	**28/41**	/	0	18/22	0	5/16	0	7/13	0	8/33	0	5/21	0	22/36	0	20/39
4	Mini C	/	0	**46/72**	/	0	18/31	0	7/15	0	5/11	0	6/31	0	5/21	0	18/68	0	18/70
5	BR30	/	0	**28/51**	/	0	8/19	NA	NA/NA	0	4/11	0	4/31	0	4/20	0	13/40	0	13/48
6	Hue	/	0	**65/110**	/	0	29/36	0	4/11	0	7/11	0	9/31	0	7/25	0	34/110	0	22/99
7	Base Station	/	0	**34/51**	/	0	29/33	0	7/16	0	6/9	0	9/17	0	7/13	0	19/38	0	23/50
8	HS100	NPD	3	**24**/64	/	0	20/27	NA	NA/NA	0	6/13	0	6/31	0	6/22	0	20/64	0	19/**71**
9	HS110	NPD	4	**24/79**	/	0	17/22	NA	NA/NA	0	6/14	0	9/33	0	6/22	0	20/62	0	19/78
10	F7C027au	/	0	**13/21**	/	0	7/10	0	6/14	0	8/12	0	6/18	0	6/15	0	8/14	0	12/21
11	MSS310	/	0	**42/61**	/	0	15/17	0	8/16	0	5/11	0	8/45	0	8/21	0	30/59	0	20/61
12	B25AUS	/	0	**19/42**	/	0	8/13	0	7/19	0	7/14	0	11/17	0	7/11	0	16/36	0	9/41
13	Mini US	/	0	**25/61**	/	0	8/41	NA	NA/NA	0	7/16	0	7/35	0	7/22	0	9/55	0	8/49
14	SP4L-AU	/	0	**37/43**	/	0	18/32	0	5/11	0	5/17	0	7/32	0	5/23	0	23/40	0	17/40
15	R6400	/	0	11/37	/	0	**20**/24	0	4/13	0	3/12	0	4/24	0	4/18	0	6/30	0	6/**41**
16	WL100	/	0	**53/81**	/	0	38/44	NA	NA/NA	0	8/16	0	8/46	0	8/27	0	41/70	0	29/76
17	Alro Pro 2	/	0	**25**/36	/	0	16/22	0	10/14	0	8/13	0	14/22	0	10/17	0	18/22	0	13/**41**
18	F19821W	/	0	**39**/75	/	0	36/33	0	7/13	0	5/11	0	7/23	0	7/14	0	27/65	0	21/**76**
19	T-131P	/	0	**36**/80	/	0	9/22	0	7/16	0	7/20	0	9/42	0	7/35	0	21/65	0	20/**91**
20	RM mini 3	/	0	**14/36**	/	0	9/30	NA	NA/NA	0	10/17	0	14/31	0	10/23	0	6/30	0	5/35

UC: Unknown crash. **NPD**: Null pointer dereference. **DoS**: Denial of service. **T**: Vulnerability type. **C**: Number of crashes. **10/24**: Number of response categories (10 minutes/24 hours).
*: Remotely exploitable. **NA**: Since Doona is only applicable to some network protocols, devices that cannot be tested are represented by 'NA'.

- **취약점 식별**

  1) Null pointer dereferences

  - Message에 대한 response messages 없이 몇 분 후 장치가 자동으로 다시 시작되고 initial state로 복구됨
  - 해당 취약점이 발견된 IoT 장비 모두 JSON 구문으로 mutation message에 의해 트리거됨
    - 중괄호 및 콜론과 같은 일부 형식 또는 message의 일부가 변경되면 메시지의 구문 구조 및 의미가 손상됨

  2) Denial of service

  - Philips의 공식 application은 IoT 기기를 정상적으로 관리하지 못함
    - App에서 장치를 찾을 수 없으며 추가 메시지에 대한 response은 해당 장치를 장치 그룹에 바인딩하도록 요청하고 더 이상 상호작용 불가능
    - 그러나 메시지 패킷이 해당 장치로 직접 전송되면 장치가 정상적으로 작동

      → IoT 장치가 완전히 충돌하진 않지만 application을 통한 서비스가 거부됨

  3) Unknown crashes

  - IoT 장치가 충돌한 다음 약 1분 이내에 자체적으로 다시 시작됨
  - 충돌이 매개변수 무효화와 같은 특정 데이터 도메인의 삭제로 인한 것임을 발견
    - 그러나 펌웨어가 공개되지 않았기 때문에 근본 원인을 확인할 수 없음
    - 구문 분석 프로세스 중에 null 값을 읽는 장치로 인해 할당 중 충돌이 발생한 것으로 추론

- 로컬 네트워크를 사용하는 통신에는 인증이 필요하지 않으므로 로컬 네트워크의 공격자가 장치를 손상시킬 수 있음

  → 따라서 해당 취약점을 원격 악용 가능 취약점으로 간주함

Table 4: Mutated messages of Snipuzz & IoTFuzzer.

Contents of mutated messages	Generated by
{"{"id": 0, "method": "start_cf", "params": ["4, 4, "1000, 2, 2700,100,500 ,1,255,10,5000,7,0,0,500,2,5000,1"]}"	Original Message
{"{"id": 0, "method": "start_cf", "params": ["4, , "1000, 2, 2700,100,500 ,1,255,10,5000,7,0,0,500,2,5000,1"]}"	SNIPUZZ
{"{"id": 0, "method": "start_cf", "params": [", 4, "1000, 2, 270000,100,500 ,1,255,10,5000,7,0,0,500,2,5000,1"]}"	IOTFUZZER

- **실행시간 성능**

  Snipuzz 및 다른 7개 Benchmark Fuzzer가 처음 10분 동안 펌웨어를 탐색하여 response category 분류 결과

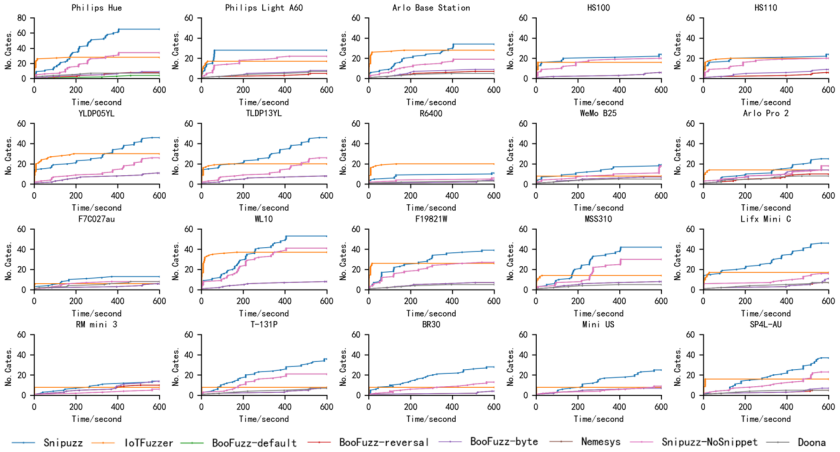


Figure 5: Runtime performance. The number of categories discovered in 10 minutes on all the 20 IoT devices.

Discussion And Limitation

Scalability and manual effort

- Manual works는 공개적으로 사용가능한 first/third party resources에서 얻은 API 프로그램의 패킷을 정리하는데 중점
- Snipuzz를 적용할 때, IoT 장치에 크롤러와 같은 기술을 사용하여 API 프로그램을 자동으로 수집 가능, 스크립트를 통해 키워드를 전처리하여 packet cleaning process를 개선하여 통신 패키지를 자동으로 수집 가능

Threats to validity

- 1) API 프로그램 부재할 때
- 2) 메시지의 암호화
- 3) API프로그램에서 다루는 기능만 검사 가능하기 때문에 펌웨어의 코드 적용 범위는 API 프로그램의 액세스에 따라 달라짐
    - 예시) 전구의 API 프로그램이 전원을 켜는 기능만 지원한다면 전원이 켜진 상태에서 캡처한 메시지를 mutate 하여 밝기 조절하는 것은 불가능

Requirements on detailed responses

- Snipuzz의 탐지 효과는 IoT 장치의 response message에서 얼마나 많은 정보를 얻을 수 있을 지에 따라 message snippet의 품질이 달라짐
    - IoT 장치가 균일한 메시지로 모든 오류를 보고하지 않으면, message snippet을 결정하기 어려울 수 있음
- 많은 IoT 장치에서 디버그 모드에서 advanced error descriptions을 얻을 수 있어 snipuzz의 message snippet determination process를 개선할 수 있었음

Conclusion

- 본 논문에선 IoT 장치에 숨어있는 취약점을 탐지하기 위해 설계된 블랙박스 Fuzzing 프레임워크 Snipuzz를 제안
- 다른 블랙박스 네트워크 Fuzz 테스트와 달리 Snipuzz는 장치에서 반환된 response message를 사용하여 Fuzzing mutation process를 보안하는 feedback

mechanism을 설정

- 장치의 response을 기반으로 메시지의 각 바이트의 문법적 역할을 추론하므로 문법 규칙의 사전 정보 없이 장치의 문법을 충족하는 테스트 사례를 생성할 수 있음
- Snipuzz를 테스트하기 위해 20개의 소비자 등급 IoT 장치를 사용하여 5개의 서로 다른 장치에서 5개의 zero-day 취약점을 발견함