커버리지 기반 CoAP 퍼저 개발 및 취약점 점검

임세창[1], 구형준[2]
[1]성균관대학교 소프트웨어학과 학부생
[2]성균관대학교 소프트웨어학과 교수
dlaqnwk2@skku.edu, kevin.koo@skku.edu

Development of Coverage-based CoAP Fuzzer and Vulnerability Inspection [1)]

Se-chang Lim[1], Hyung-joon Koo[2]
[1]Dept. of Software, Sungkyunkwan University
[2]Dept. of Software, Sungkyunkwan University

요 약

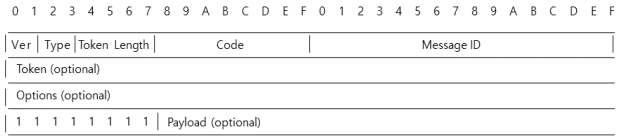
IoT 산업의 규모가 커짐에 따라서 IoT 디바이스 간의 통신 프로토콜에 대한 위협도 증가하고 있으며, 이에 따라 IoT 프로토콜에 대한 보안의 중요성이 대두되고 있다. 하지만 IoT 프로토콜 중 비교적 최근에 등장한 프로토콜인 CoAP 프로토콜에 대한 연구는 아직 충분하지 않으며, CoAP에 대한 기존의 연구는 커버리지를 고려하지 않은 퍼저를 사용하였다. 따라서 이번 연구에서는 커버리지를 고려한 CoAP대상 퍼저를 개발하고 CoAP의 잠재적인 취약성을 점검한다. CoAP의 c 구현체인 libcoap 라이브러리를 대상으로 퍼징한 결과, 총 2개의 힙 버퍼 오버 플로우 취약점을 발견하였다.

1. 서론

IoT (Internet of Things)는 센서, 소프트웨어 등 여러 기술을 내장한 디바이스들이 인터넷을 통해 연결되는 것을 뜻한다. IoT Analytics에 따르면 전 세계에 연결된 IoT 엔드 포인트들은 2021년 122억개에서 2022년 144억개까지 늘어났으며, 2025년에는 270억개까지 늘어날 것으로 전망하고 있다 [1]. IoT 산업이 커짐에 따라서 안전한 IoT 환경을 구축하는 것에 대한 관심도 증가하고 있다. 하지만, IoT를 구성하는 핵심요소 중 하나인 네트워크 프로토콜에 대한 취약성에 대한 연구는 아직 활발하지 않으며, 비교적 최근에 등장한 프로토콜인 CoAP (Constrained Application Protocol)프로토콜 [2]에 대한 연구는 더욱 부족하다. 따라서 이번 연구에서는 CoAP의 c 구현체인 libcoap에 대해서 커버리지를 고려한 그레이박스 퍼저를 제작하고, libcoap의 취약성을 점검한다.

2. CoAP

CoAP은 8비트 마이크로컨트롤러를 사용하거나 적은 양의 메모리를 사용하는 등 제한된 환경에 놓여있는 IoT 노드들 간의 통신을 용이하게 하기 위해서 고안되었다. CoAP은 REST API에 기반한 API를 제공하며 HTTP 프로토콜과 굉장히 유사한 형태를 띄고 있다. HTTP 프로토콜과 다른 점은 CoAP은 패킷의 단편화 현상을 막기 위해 최대한 메시지의 오버헤드를 줄이는 방향으로 설계되었다.



(그림 1) CoAP 메시지 헤더

그림 1은 CoAP 메시지의 헤더를 나타낸다. CoAP 메시지는 4바이트의 고정된 길이의 헤더로 시작한다. 헤더는 버전, 타입, 토큰 길이, 코드, 메시지 아이디로 구성된다. 버전은 2비트로, 항상 01이어야 하며, 타입은 CON, NON, ACK, RST 등의 패킷 타입을 의미한다. 토큰 길이는 요청과 응답 간의 매치에 사용되는 토큰의 길이를 의미하며, 메시지 아이디는 메시지 중복 탐지나 CON/ACK 혹은 NON/RSK간의 매칭에 사용된다. 그 뒤로는 옵션 혹은 페이로드의 존재 여부에 따라 구성된다.

1) 이 논문은 2023년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을받아 수행된 연구임 (No. 2022-0-01199, 융합보안대학원(성균관대학교))

```
1.  static const coap_option_num_t coap_options[COAP_OPT_NUM]  = {
2.  COAP_OPTION_IF_MATCH,  COAP_OPTION_URI_HOST,  ... // coap options
3.  COAP_OPTION_ECHO,  COAP_OPTION_NORESPONSE  };
4.
5.  static const uint16_t coap_option_max_length[COAP_OPT_NUM]  = {
6.  8, 255, 8, 1, ... , 255, 4, 40, 1, 8 }; // coap option lengths
```

(그림 2) CoAP 옵션 과 옵션 별 최대 길이

3. CoAP 퍼저 제작

본 연구에서는 CoAP프로토콜을 c로 구현한 라이브러리인 libcoap [3]를 대상으로 한 퍼저를 제작하였다. 커버리지를 고려하기 위해 퍼저는 libfuzzer [4]를 이용해 제작하였다. libfuzzer는 LLVM 프로젝트의 일부로 개발된 커버리지 기반의 퍼징 테스트 도구로, 자동화된 테스트 케이스 생성을 통해 메모리 버그를 찾는 데 유용하다. CoAP을 대상으로 한 퍼저를 제작한 기존의 연구 [5]나 네트워크 프로토콜을 대상으로 한 기존의 연구 [6]는 코드 커버리지를 고려하지 않고 단순히 변이 기반 퍼저나 생성 기반의 퍼저를 사용하였다. 본 연구에서는 libcoap이 오픈 소스라는 점을 이용해 타겟 함수를 선정한 뒤 libfuzzer를 이용해 퍼징 효율을 극대화하였다.

libcoap 라이브러리는 coap_pdu_t, coap_context_t, coap_session_t 등의 구조체를 이용해 CoAP 메시지와 CoAP 세션 등을 추상화하여 나타낸다. 해당 구조체들을 사용하는 함수들을 리스팅하고 타겟 함수를 정하였다. 표 1은 퍼징 타겟 함수들을 나타낸다.

libfuzzer로 퍼징을 하기 위해서는 퍼저 타겟 함수를 작성하여 생성된 입력값을 타겟 함수의 입력값으로 넣어주어야 한다. 퍼저 타겟 함수의 시드가 되는 코퍼스를 이용해 효율적으로 퍼징을 수행할 수 있는데, CoAP이 HTTP와 비슷하다는 특징을 이용해 coap_send함수에 대한 시드로 HTTP url을 변형하여 사용하였다. 또한 coap_add_option 함수의 옵션 값에 이상한 값이 들어가지 않도록 옵션 배열을 생성해 사용하였다. 그림 2는 해당 옵션 배열을 나타낸다.

4. 퍼징 결과

coap_add_option 함수와 coap_send 함수에 대해서 힙 버퍼 오버플로우 읽기 취약점을 발견하였다.

<표 1> 퍼징 타겟 함수

함수 이름	사용된 구조체
coap_send	coap_session_t, coap_context_t, coap_pdu_t
coap_get_uri_path	coap_pdu_t
coap_add_option	coap_pdu_t

두 취약점 모두 인자로 들어온 길이 정보에 대한 검증이 충분히 이루어지지 않아서 발생한 취약점이었다. 해당 취약점은 libcoap 측에 전달하였으며, 취약한 코드는 수정되었다 [7]. coap_get_uri_path 함수를 대상으로는 일주일 정도 퍼징을 수행했지만 크래시가 발생하지 않았다.

5. 결론

본 연구에서는 IoT 프로토콜 중 하나인 CoAP 프로토콜을 c로 구현한 구현체인 libcoap에 대한 취약성을 커버리지 기반의 퍼저를 이용해 점검하였다. libcoap 라이브러리의 함수들 중 적절한 타겟 함수를 선정하였고, 코퍼스 구성, 퍼저 타겟 함수 제작을 통해 타겟 함수에 대한 퍼징을 수행하였다. 그 결과 총 2개의 취약점을 발견할 수 있었다. 본 연구를 통해 CoAP 프로토콜의 강건성에 대한 연구가 앞으로 더욱 활발하게 이루어질 것으로 기대한다.

참고문헌

[1] “Global IoT market size to grow 19% in 2023 −IoT shows resilience despite economic downturn”, 2023년 2월 7일 수정, 2023년 4월 1일 접속, https://iot-analytics.com/iot-market-size/

[2] CoAP, https://www.rfc-editor.org/rfc/rfc7252

[3] Libcoap, https://github.com/obgm/libcoap

[4] Libfuzzer, https://llvm.org/docs/LibFuzzer.htm

[5] Bruno da S. Melo and Paulo L´ıcio de Geus, “Robustness Testing of CoAP Server-side Implementations through Black-box Fuzzing Techniques” 10.5753/sbseg.2017.19528, November 2017.

[6] V. -T. Pham, M. Böhme and A. Roychoudhury, "AFLNET: A Greybox Fuzzer for Network Protocols," 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), 2020, pp. 460-465, doi: 10.1109/ICST46399.2020.00062.

[7] https://github.com/obgm/libcoap/pull/1065/commits/e7287de3363bf9cb9ac8c4dd48ce2988be1b080d