경량 IoT 를 위한 협력적 펌웨어 퍼징 기법

이진민 [1], 이승은 [2], 김나현 [2], 이일구 [3]
[1] 성신여자대학교 미래융합기술공학과 박사과정
[2] 성신여자대학교 융합보안공학과 학부생
[3] 성신여자대학교 융합보안공학과/미래융합기술공학과 교수

csewa56759@gmail.com, rose6825@gmail.com, 20221079@sungshin.ac.kr, iglee@sungshin.ac.kr

Cooperative Firmware Fuzzing Technique for Lightweight Internet of Things

Jin-Min Lee[1], Seung-Eun Lee[2], Na-Hyun Kim[2], Il-Gu Lee[1,2]
[1]Dept. of Future Convergence Technology Engineering, Sungshin Women's University
[2]Dept. of Convergence Security Engineering, Sungshin Women's University

요 약

IoT(Internet of Things) 기기가 다양한 산업 분야에 활용되면서, 보안과 유지 보수를 위한 관리의 중요성이 커지고 있다. 편리한 IoT 기기 관리를 위해 무선 네트워크를 통한 펌웨어 업데이트 기술인 FOTA(Firmware Over The Air)가 적용되어 있지만, 컴퓨팅 파워가 제한된 경량 IoT 기기 특성상 취약점 탐지를 수행하기 어렵다. 본 연구에서는 IoT 기기들이 퍼징 테스트 케이스를 분할하여 협력적으로 퍼징하고, 노드 간의 퍼징 결과가 다르면 재검증을 수행하는 협력적 퍼징 기법을 제안한다. 실험 결과에 따르면, 중복되는 테스트 케이스를 2 개나 3 개 퍼징하는 협력적 퍼징 기법은 종래 방식 대비 연산량을 최소 약 16%, 최대 약 48% 줄였다. 또한, 종래 퍼징 기법 대비 취약점 탐지 성공률(Success rate of vulnerability detection)을 최소 약 3 배, 최대 약 3.4 배 개선시켰다.

1. 서론

안전, 의료, 교통 등 다양한 산업 분야에 IoT(Internet of Things) 기기가 널리 활용되면서 IoT 기기의 보안 및 유지 보수를 위한 관리의 필요성이 커지고 있다[1]. IoT 기기는 전력이 지속적으로 공급되지 않고 컴퓨팅 자원이 제한된 경량 장치이므로 기기 내부에 강력한 보안 기능을 적용하기 어렵다. 수많은 IoT 기기를 효율적으로 관리하기 위해 Wi-Fi, 셀룰러 등의 무선 네트워크를 통해 펌웨어를 업데이트하는 FOTA(Firmware Over The Air) 기술이 활용되고 있지만, 경량 IoT 기기의 특성상 업데이트되는 펌웨어에 취약점을 탐지하기 위한 퍼징(fuzzing)을 실행하기 어려운 문제가 있다[2]. 이에 따라 IoT 기기에 최적화된 경량 취약점 분석 방식에 대한 연구가 진행되고 있지만, 펌웨어와 IoT 의 특수성을 고려한 연구는 부재한 실정이다[3]. 현재 취약점을 분석하기 위해 퍼징이 활용되고 있지만, 퍼징은 무작위 테스트 케이스를 입력한 후 도출되는 결과값을 통해 취약점 탐지를 진행하므로 IoT 기기에서 펌웨어를 퍼징하기 위해서는 수많은 테스트 케이스를 저장하고 연산해야 하는 문제가 있다[4].

이러한 문제를 해결하기 위해 본 연구에서는 경량 IoT 기기간 협력을 통한 펌웨어 취약점 퍼징 기법을 제안한다. 제안하는 퍼징 기법은 퍼징 테스트 케이스를 분할하여 다수의 IoT 기기가 협력적으로 퍼징을 수행하고, 노드 간 퍼징 결과가 불일치할 경우 재검증을 통해 퍼징 오류를 정정한다.

본 논문의 기여점은 다음과 같다.

- 경량 IoT 기기를 고려한 협력적 퍼징 기법을 제안한다.
- 퍼징 기법의 성능과 효율을 평가 분석하는 프레임워크를 제안한다.

본 논문의 구성은 다음과 같다. 2 장에서는 취약점 탐지와 관련한 선행 연구에 대해 비교 및 분석하고, 3 장에서는 경량 IoT 기기를 위한 협력적 퍼징을 제안한다. 4 장에서는 제안 방식과 종래 방식을 비교한 실험 환경에 대해서 설명하고, 시뮬레이션 기반 실험 결과를 분석한다. 마지막으로 5 장에서는 결론을 맺는다.

2. 관련연구

본 장에서는 경량 IoT 장치에 최적화된 취약점 탐지를 위해 분산 퍼징을 하거나 기기간 협력을 하는 종래 연구의

기여점과 한계점을 분석한다.

Emre Güler 외 7 인은 여러 퍼저의 장점을 조합하여 협력적 퍼징 프로세스의 전반적인 성능을 극대화하는 Cupid 프레임워크를 개발하였다[5]. 그러나 모든 퍼저가 모든 테스트 케이스를 개별적으로 퍼징하여 종래 방식 대비 퍼징 시간이 길고 비효율적이라는 한계점이 있다.

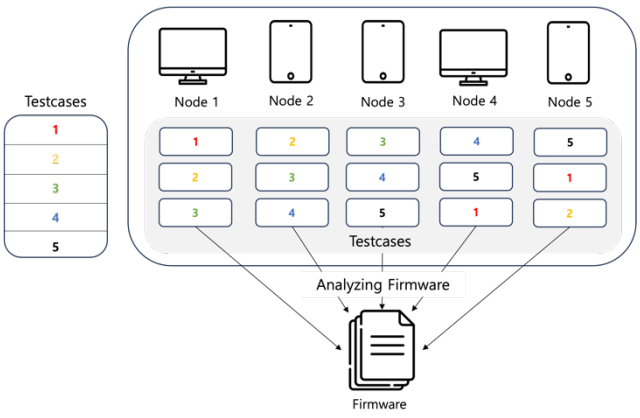
Jin Huang 외 4 인은 PHP 응용 프로그램을 분석하여 자동으로 취약점을 탐지하는 UFuzzer 를 제안하였다[6]. UFuzzer 는 PHP 응용 프로그램을 실험 데이터로 사용하여 제안 방식의 성능을 입증하였다. 그러나 제안 방식의 퍼징을 정확도를 개선하면 퍼징 효율이 감소된다는 한계점이 있다.

Xu Zhou 외 7 인은 중앙 집중식 동적 작업 스케줄링을 통한 분산 퍼징 최적화 퍼징인 UniFuzz 를 제안하였다[7]. 제안 방식은 중복 시드를 처리하고 작업을 분산시킴으로써 종래의 퍼징 방식 대비 성능을 개선했지만, 퍼징 작업을 담당한 기기에 과부하가 발생하면 퍼징 결과를 도출하지 못한다는 한계점이 있다.

Xiaotao Feng 외 7 인은 IoT 장치의 펌웨어 취약점을 탐지하기 위해 메시지 스니펫 추론을 통한 IoT 펌웨어의 블랙박스 퍼징 방식인 Snipuzz 를 제안하였다[8]. 그러나 Snipuzz 는 효율적으로 취약점을 탐지할 수 있는 조건이 제한적이다. 또한, 펌웨어의 업데이트로 인해 이전에 발견했던 취약점이 수정될 수 있다는 것을 고려하지 못했다.

3. 협력적 펌웨어 취약점 분석 방식

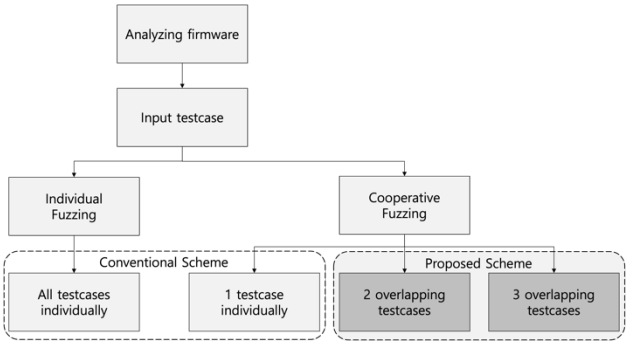
본 장에서는 경량 IoT 기기를 위해 테스트 케이스를 분할하여 퍼징을 수행하는 협력적 퍼징에 대해서 설명한다. 그림 1 은 제안하는 협력적 퍼징을 나타낸 그림이다.



(그림 1) 경량 IoT 를 위한 협력적 펌웨어 퍼징 기법

종래 퍼징 기법은 5 개의 테스트 케이스를 모든 노드에서 퍼징해야 했지만, 제안하는 협력적 퍼징은 테스트 케이스를 분할하여 퍼징을 수행한다. 제안 퍼징 기법은 노드들 간 퍼징 결과가 불일치할 경우 오류가 발생했다고 가정하고 재검증하는 과정을 거친다. 오류는 각 노드가 오동작하거나 전력 제한으로 인해 퍼징을 수행하지 못하는 경우를 의미한다.

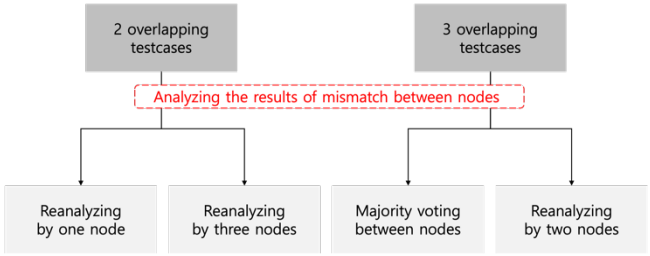
그림 2 는 협력적 퍼징 기법과 종래의 개별적 퍼징 기법으로 퍼징 유형을 분류한 그림이다.



(그림 2) 제안 방식과 종래 방식 비교

그림 2 와 같이 펌웨어 퍼징은 테스트 케이스를 입력하여 취약점이 탐지되는지 확인하는 방식으로 수행된다. 이때, 테스트 케이스를 입력하는 유형에 따라 개별적 퍼징과 협력적 퍼징으로 퍼징 유형을 분류할 수 있다. 기기들이 개별적으로 모든 테스트 케이스를 퍼징하는 방식과 협력적 퍼징에서 각 노드가 하나의 테스트 케이스를 분할하여 퍼징하는 방식이 종래 퍼징 기법에 해당한다. 노드가 하나의 테스트 케이스를 퍼징하는 방식은 오류가 발생해도 알아내기 어렵다. 이에 따라 본 연구에서 제안하는 퍼징 기법은 중복되는 테스트 케이스를 n 개 입력하여 퍼징을 수행한다.

그림 3 은 노드들 간 분석 결과가 불일치할 경우, 대처 방안에 따라 제안하는 협력적 퍼징 기법의 유형을 분류한 그림이다.



(그림 3) 노드들 간 분석 결과 불일치시 대처 방안에 따른 제안 퍼징의 유형 분류

중복되는 테스트 케이스를 2 개나 3 개 입력(n=2, 3)하는 협력적 퍼징 기법은 노드들 간의 분석 결과가 불일치할 경우 대처 방안에 따라 4 가지 방식으로 분류할 수 있다. 2 개의 테스트 케이스를 중복으로 퍼징하는 방식의 경우, 해당 테스트 케이스를 분석하지 않은 하나의 노드가 재검증하거나 나머지 세 노드가 재검증한 후 다수의 결과를 최종 결과값으로 결론짓는다. 3 개의 테스트 케이스를 중복으로 퍼징하는 방식의 경우, 3 개의 결과값이 도출되므로 다수결에 의해 최종 결과값으로 결정하거나 해당 테스트 케이스를 분석하지 않은 두 노드가 재검증한다.

4. 실험환경 및 결과

<표 1> 개별적 퍼징 기법과 협력적 퍼징 기법의 시간 복잡도와 연산량

Cases			Time complexity (seconds)	Computation cost (items, k=3)
Conventional Scheme	Individual fuzzing		5.04	25
	1 testcase individual fuzzing		7.04	5
Proposed Scheme	2 overlapping testcases fuzzing	Reanalyzing by one node	19.24	13(10+1k)
		Reanalyzing by three nodes	24.56	19(10+3k)
	3 overlapping testcases fuzzing	Majority voting between nodes	19	15
		Reanalyzing by two nodes	29.48	21(15+2k)

본 연구에서는 제안하는 방식의 개념을 증명하기 위해 파이썬을 사용하여 시뮬레이션 모델을 구현하고 종래 방식과 비교 및 평가했다.

표 1 은 개별적 퍼징 기법과 협력적 퍼징 기법을 시간 복잡도와 연산량 측면에서 비교한 표이다. 시간 복잡도와 연산량은 실험을 1,000 번 반복한 평균값을 작성하였으며, 연산량은 퍼징에 들어가는 테스트 케이스 수를 기준으로 계산하였다. 1,000 번의 반복 실험 동안 5 개 테스트 케이스 퍼징 중 랜덤한 개수의 오류가 발생한다고 가정하였다. 이에 따라 3 개의 오류가 발생했을 때의 연산량을 계산하였다. 실험 결과에 따르면, 제안 기법은 개별적 퍼징과 하나의 테스트 케이스를 퍼징하는 협력적 퍼징 대비 시간 복잡도가 약 3-5 배 증가하였다. 제안하는 2 개, 3 개의 테스트 케이스를 중복으로 퍼징하는 방식은 재검증하는 노드 수가 증가할수록 시간 복잡도도 증가하였다.

연산량 측면에서의 개별적 퍼징은 5 개의 노드가 5 개의 테스트 케이스를 전부 퍼징하기 때문에 총 25 개로 설정하였고, 하나의 테스트 케이스를 퍼징하는 협력적 퍼징은 총 5 개로 설정하였다. 제안하는 퍼징은 2 개, 3 개의 테스트 케이스를 중복으로 퍼징하기 때문에 각각 기본적으로 10 개, 15 개로 설정하였다. 표에 있는 $k(0 \leq k \leq 5)$는 오류가 발생한 테스트 케이스를 의미하며, k 가 증가할수록 제안하는 퍼징 기법의 연산량도 증가한다. 실험을 통해 도출된 오류 발생 테스트 케이스 수가 평균 3 개이므로, 2 개를 중복으로 퍼징하는 방식에서 1 개 재검증과 3 개 재검증의 연산량은 각 13 개, 19 개로 계산하였다. 또한 3 개를 중복으로 퍼징한 후 2 개 노드가 재검증하는 방식의 연산량은 21 개로 계산하였다. 반면, 다수결에 의해 최종 결과를 결정하는 방식은 추가적인 분석이 없으므로 오류에 따른 연산량 증가가 발생하지 않는다. 이에 따라 제안하는 협력적 퍼징 기법은 개별적 퍼징 기법 대비 연산량을 최소 약 16%, 최대 약 48% 개선된 것을 확인할 수 있다.

하나의 테스트 케이스를 퍼징하는 종래의 협력 퍼징 기법은 퍼징 결과에서 오류가 발생했음을 확인할 수 없다. 퍼징 결과에 오류가 발생한다는 것은 취약점 탐지 실패를 의미한다.

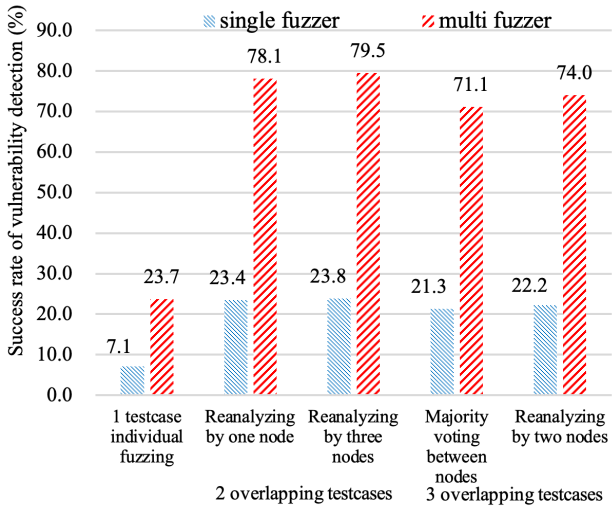
$$\gamma = \varepsilon \times \theta \quad (1)$$

$$\alpha = \frac{\beta + \gamma}{\delta} \times 100 \quad (2)$$

본 연구에서는 취약점 탐지 성공률(Success rate of vulnerability detection)인 α를 식 (2)와 같이 정의한다. β 는 취약점 탐지 성공 횟수(Number of successful vulnerability detections)를 의미하며, γ는 에러 복구를 통한 취약점 탐지 성공 횟수(Number of successful vulnerability detections with error recovery)를 의미하고 δ는 의심스럽다고 탐지하는 모든 경우의 수(Total number of detected suspicious cases)를 의미한다. 제안하는 방식의 취약점 탐지 성공률 α는 종래의 취약점 탐지 성공률과 달리 에러 복구를 통한 취약점 탐지 성공 횟수 γ도 함께 탐지 성공률 계산에 포함된다. 이에 따라, γ는 식 (1)과 같이 취약점 탐지 실패 횟수(Number of failed vulnerability detections)인 ε에 오류 복구 성공률(Success rate of error recovery)을 의미하는 θ을 곱하여 구할 수 있다. 반면, 하나의 테스트 케이스를 퍼징하는 종래의 협력 퍼징 기법은 오류 복구를 수행할 수 없으므로 오류 복구를 통한 취약점 탐지 성공 횟수 γ가 0 으로 계산된다.

그림 4 는 단일 퍼저를 사용할 때와 다중 퍼저를 사용할 때, 하나의 테스트 케이스를 퍼징하는 협력적 퍼징과 중복되는 테스트 케이스를 2 개 혹은 3 개 퍼징하는 협력적 퍼징 기법을 취약점 탐지 성공률 측면에서 비교한 그래프이다.

단일 퍼저를 사용하는 경우, 중복되는 테스트 케이스를 2 개나 3 개 퍼징하는 협력적 퍼징이 하나의 테스트 케이스를 퍼징하는 종래 협력적 퍼징에 비해 취약점 탐지 성공률이 최소 약 3 배, 최대 약 3.4 배 개선되었다. 다중 퍼저를 사용할 경우에도 종래 협력적 퍼징 대비 제안하는 협력적 퍼징에서 취약점 탐지 성공률이 동일한 비율로 개선되었다. 또한, 모든 퍼징 기법에서 다중 퍼저를 사용할 때, 단일 퍼저 대비 취약점 탐지 성공률이 약 3.3 배 증가하였다. 다중 퍼저에서 제안 퍼징 기법을 사용하는 경우, 종래 방식에서 단일 퍼저를 사용하는 것 대비 취약점 탐지 성공률을 최대 11.2 배 향상시켰다. 실험을 통해, 종래 취약점 탐지 방식 보다 더 많은 취약점을 오류없이 탐지할 수 있음을 확인했다.



(그림 4) 단일 퍼저와 다중 퍼저 사용에 따른 협력적 퍼징 기법의 취약점 탐지 성공률

5. 결론

최근 다양한 산업 분야에 IoT 기기가 활용되고 있으나, 보안과 유지 보수를 위한 관리가 제대로 이루어지고 있지 않다. IoT 기기는 무선 네트워크를 통해 펌웨어 업데이트가 진행되지만, 컴퓨팅 파워가 제한되어 펌웨어 취약점을 탐지하기 위한 퍼징을 수행하기 어렵다. 종래 연구에서는 IoT 기기에 최적화된 취약점 분석을 수행했지만, IoT 의 특수성을 고려하지 않았다. 본 연구에서는 경량 IoT 기기의 특수성을 고려하기 위해 펌웨어 퍼징에 사용되는 중복 테스트 케이스를 사용하고 노드 간의 퍼징 결과 불일치시 재검증을 수행하는 협력적 퍼징 기법을 제안했다. 제안하는 협력적 퍼징은 개별적 퍼징 기법 대비 연산량을 최소 약 16%, 최대 약 48% 줄였다. 또한 제안 방식은 테스트 케이스를 중복적으로 퍼징함으로써 오류를 복구하여, 종래 방식 대비 취약점 탐지 성공률을 최소 약 3 배, 최대 약 3.4 배 개선시켰다. 후속 연구에서는 상용 펌웨어와 퍼저를 활용한 테스트베드 환경에서 제안 방식과 종래 방식의 성능을 확인하고 제안 방식의 시간 복잡도를 개선하기 위한 방법을 연구할 예정이다. 또한, 다양한 IoT 기기가 있는 환경을 고려하여 테스트 케이스 수와 중복 테스트 케이스 입력의 최적 값을 도출할 계획이다.

Acknowledgement

본 논문은 2024 년도 정부재원(과학기술정보통신부 여대학원생 공학연구팀제 지원사업)으로 과학기술정보통신부와 한국여성과학기술인육성재단의 지원 (WISET 계약 제 2024-138 호), 산업통상자원부 및 한국산업기술진흥원의 지원(No. RS-2024-00415520)과 과학기술정보통신부 및 정보통신기획평가원의 지원 (No. IITP-2022-RS-2022-00156310)을 받은 연구결과로 수행되었음

참고문헌

[1] Malik Abdul Sami, Tamim Ahmed Khan, “Forecasting failure rate of IoT devices: A deep learning way to predictive maintenance,” Computers and Electrical Engineering, 110, 2023.

[2] Akashdeep Bhardwaj, Keshav Kaushik, Salil Bharany, SeongKi Kim, “Forensic analysis and security assessment of IoT camera firmware for smart homes,” Egyptian Informatics Journal, 24, 4, 2023.

[3] Donglan Liu, Hao Zhang, Rui Wang, Fangzhe Zhang, Lili Sun, Xin Liu, Lei Ma, “A lightweight IoT firmware vulnerability detection scheme based on homology detection,” Journal of High Speed Networks, 28, 1-11, 2022.

[4] Xiaogang Zhu, Sheng Wen, Seyit Camtepe, and Yang Xiang, “Fuzzing: A Survey for Roadmap,” ACM Computing Surveys, 54, 11, 1-36, 2022.

[5] Emre Güler, Philipp Görz, Elia Geretto, Andrea Jemmett, Sebastian Österlund, Herbert Bos, Cristiano Giuffrida, Thorsten Holz, “Cupid: Automatic Fuzzer Selection for Collaborative Fuzzing,” Annual Computer Security Applications Conference, 360-372, 2020.

[6] Jin Huang, Junjie Zhang, Jialun Liu, Chuang Li, Rui Dai, “UFuzzer: Lightweight Detection of PHP-Based Unrestricted File Upload Vulnerabilities Via Static-Fuzzing Co-Analysis,” International Symposium on Research in Attacks, 78-90, 2021.

[7] Xu Zhou, Pengfei Wang, Chenyifan Liu, Tai Yue, Yingying Liu, Congxi Song, Kai Lu, Qidi Yin, “Unifuzz: Optimizing distributed fuzzing via dynamic centralized task scheduling,” arXiv.Org, 2009.06124v1, 2020.

[8] Xiaotao Feng, Ruoxi Sung, Xiaogang Zhu, Minhuo Xue, Shen Wen, Dongxi Liu, Surya Nepal, Yang XiangFeng. “Snipuzz: Black-box fuzzing of Iot firmware via message snippet inference,” Proceedings of the 2021 ACM SIGSAC conference on computer and communications security, 337-350, 2021.