*Journal of The Korea Institute of Information Security & Cryptology* ISSN 1598-3986(Print)
VOL.32, NO.6, Dec. 2022 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2022.32.6.1121

# 유사 HAL 함수 탐색을 통한 펌웨어 퍼징 기법*

정 세 연,[1†] 황 은 비,[1] 조 영 필,[2] 권 태 경[3‡]
[1,3]연세대학교 정보대학원 정보보호 연구실 (대학원생, 교수), [2]한양대학교 (교수)

# Firmware Fuzzing Method through Pseudo-HAL Identification*

Seyeon Jeong,[1†] Eunbi Hwang,[1] Yeongpil Cho,[2] Taekyoung Kwon[3‡]
[1,3]Yonsei University (Graduate student, Professor),
[2]Hanyang University (Professor)

## 요 약

펌웨어 취약점을 찾기 위한 퍼징 기법인 HAL-Fuzz는 MCU 벤더에서 제공하는 하드웨어 추상 계층의 HAL 함수를 이용하는 효율적인 기법이다. 하지만 정확한 HAL 함수를 사용하지 않는 대부분의 펌웨어는 다룰 수가 없다. 본 논문에서는 유사 HAL 함수 탐색이라는 새로운 방식을 제안하고 HAL-Fuzz의 퍼징 가용성을 높이고자 한다. 실험을 통해 HAL 함수뿐만 아니라 개발자 구현 유사 HAL 함수도 탐색하였으며 퍼징이 가능함을 확인하였다.

## ABSTRACT

HAL-Fuzz, a fuzzing technique to find firmware vulnerabilities, is efficient by using the HAL function of the hardware abstraction layer provided by MCU vendors. However, it cannot handle most firmware that unused the exact HAL function. In this paper, we propose a new method for identifying pseudo-HAL functions to increase the fuzzing availability of HAL-Fuzz. In experiments, we identified not only the HAL but also the pseudo-HAL functions, implemented by the developer, and that fuzzing is possible.

**Keyword:** Firmware fuzzing, Firmware emulation, HAL, Pseudo-HAL identification

## I. 서 론

임베디드 장치는 사물인터넷, 항공, 무기 등 다양한 곳에서 사용되고 있다. 2016년 이후 크게 증가하고 있는 펌웨어 취약점은 사회에 치명적인 영향을 줄 수 있기 때문에 펌웨어 퍼징을 통한 동적 분석과 사전 탐지가 매우 중요하다. 하지만 임베디드 장치를 구성하는 MCU(Microcontroller Unit)의 한정된 자원은 온디바이스 퍼징을 어렵게 만든다. 한편 MCU 내부의 다양한 주변장치 구성은 에뮬레이션 기반의 퍼징 또한 어렵게 만든다. 최근 연구 동향을 보면 에뮬레이션 기반 퍼징을 위한 많은 노력이 있었다[2, 3, 4, 5]. Muench 등은 2018년 NDSS에서 펌웨어 퍼징 시 발생되지 않은 크래시에서도 버그가 발생 가능하기 때문에 에뮬레이션이 중요하며, 주변장치 모델링을 통한 부분 에뮬레이션이 합리적인 방안이라 제안했다[1].

HAL-Fuzz[2]는 HAL(Hardware Abstraction Layer) 함수 핸들링을 통한 주변장치 모델링 기반 펌웨어 퍼저이다. HAL은 벤더에서 개발자가 펌웨어를 쉽게 개발할 수 있도록 제공된 라이브러리이며, HAL-Fuzz는 이를 통해 주변장치 모델링을 진행한다. 하지만 개발자가 직접 함수를 작성하거나 기존 HAL 함수를 수정하는 경우가 많은데 (이것을

Received(10. 05. 2022), Modified(11. 28. 2022), Accepted(11. 29. 2022)
* 이 논문은 2021년도 정부(과학기술정보통신부)의 재원으로 한국연구재단 의 지원을 받아 수행된 연구임(NRF-2019R1A2C1088802).
† 주저자, best6653@gmail.com
‡ 교신저자, taekyoung@yonsei.ac.kr(Corresponding author)

우리는 유사 HAL 함수라고 부른다.) 이 경우에는 HAL-Fuzz로 퍼징이 불가하다. 특히 RTOS 기반 펌웨어는 주로 유사 HAL 함수로 구성되기 때문에 유사 HAL 함수 탐색 방법에 대한 연구가 필요하다.

따라서 본 논문에서는 유사 HAL 함수 탐색 방법을 제안하고, HAL-Fuzz의 퍼징 가용성 확장을 도모한다.

## II. 연구 배경

### 2.1 관련 연구 : 펌웨어 퍼징

Muench 등은 주변기기 모델링을 통한 부분 에뮬레이션이 하드웨어에 독립적인 펌웨어 퍼징에 대한 합리적인 방향이 될 수 있다고 제안했고, 이후 관련한 연구들이 제안되어왔다 [1, 2, 3, 4, 5]. Pretender [5]는 머신러닝을 활용해 모델링을 하고 펌웨어만으로 퍼징이 가능하지만, 모델링 시 실제 장치가 필요하다.

$P^2$im [3], μEmu [4]는 레지스터 레벨에서의 주변장치를 모델링 및 퍼징을 제안한다. $P^2$im 은 MMIO(Memory-mapped I/O)에 접근하는 모든 레지스터를 각각 처리하고, μEmu는 기호 실행(Symbolic execution)을 통해 주변장치를 모델링한다. 하지만 레지스터 레벨 주변장치 모델링 방법은 퍼징 속도가 느리다.

HAL-Fuzz [2]는 Pretender와 달리 하드웨어 없이 주변장치 모델링 및 퍼징이 가능하며, 함수를 처리 하는 방식으로 $P^2$im 과 μEmu 보다 빠른 퍼징이 가능하다. 이와 관련한 자세한 내용은 아래에서 설명한다.

### 2.2 유사 HAL 탐색의 필요성

Fig.1.과 같이 HAL은 장치의 하드웨어 부분과 소프트웨어 사이에서 MCU 레지스터에 직접적으로 값을 쓰거나 읽어서 주변 장치를 제어하는 역할을 한다.

HALucinator [3]는 이러한 HAL 기반의 함수레벨 펌웨어 에뮬레이션을 제안했다. 이는 HAL 함수 탐색 프로그램인 Libmatch를 통해 HAL 함수를 탐색한 후, 결과로 나온 리스트의 HAL 함수가 호출될 경우

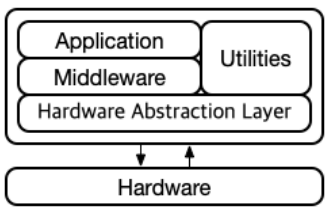


Fig. 1. STM32 firmware architecture

Table 1. Fuzzing performance on Drone firmware(5h)

| Firmware | Speed (exec/s) | | Unique Basic Blocks | |
|---|---|---|---|---|
| | HAL-Fuzz | $P^2$im | HAL-Fuzz | $P^2$im |
| Drone | 29.3938 | 8.6129 | 326 | 1278 |

Table 2. Peripheral related functions in CNC firmware

| Firmware | Pseudo-HAL | HAL |
|---|---|---|
| CNC | usart_putc<br>… | HAL_GPIO_ReadPin<br>… |
| Total (#) | 28 | 20 |

이를 후킹하여 처리한다. HAL-Fuzz는 HALucinator와 동일한 개념에 AFL(American Fuzz Lop)을 연결한 퍼저이다.

우리는 먼저 HAL-Fuzz의 유효성을 간단한 실험을 통해 검증했다. Table 1.은 5장의 실험과 같은 환경에서 드론(Drone) 펌웨어를 5시간씩 5번 수행한 퍼징 성능의 평균을 정리한 결과이다. 이를 통해 HAL-Fuzz가 $P^2$im보다 약 3.4배 더 빠른 것을 알 수 있다. 즉, 함수 처리를 통한 퍼징 방법이 더 효율적이다.

하지만 HAL-Fuzz의 경우 퍼징 가용성 측면에서 두 가지의 한계점이 존재한다. 첫째, HAL이 제공되지 않는 경우 펌웨어에서 함수를 탐색하지 못한다. 유사 HAL 함수는 개발자가 수정하거나 직접 개발한 하드웨어 제어 함수이다. 현재 범용적으로 HAL 함수가 사용되고 있으나, 유사 HAL 함수를 통해 주변장치를 제어하는 펌웨어가 존재한다. Table 2.는 CNC 펌웨어의 주변장치 관련 함수 분포를 보이는데, HAL 함수뿐만 아니라 유사 HAL 함수도 호출 되는 것을 알 수 있다. 이 경우 HAL-Fuzz로는 에뮬레이션이 실패한다.

둘째, HAL 제공과 상관없이 개발자가 직접 함수를 코딩하거나 수정하는 경우 함수 탐색이 되지 않는다. Libmatch는 펌웨어와 동일하게 컴파일된 SDK(Software Development Kit) 파일이 요구되며, 스트립 펌웨어는 함수 탐색이 어렵다. 하지만 상용 펌웨어의 경우 컴파일 정보를 알 수 없고 스트립 펌웨어가 사용되기에 펌웨어 정보에 독립적인 탐색 방법이 필요하다.

따라서 본 논문에서는 위의 한계점을 보완할 수 있는 유사 HAL 함수 탐색 기법을 제안한다. 또한 이를 통해 HAL-Fuzz의 퍼징 가용성의 확장을 도모한다.

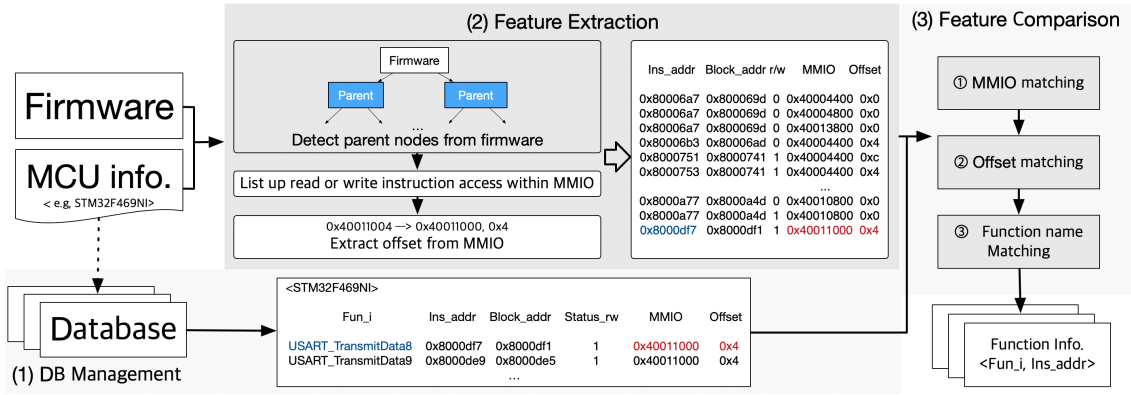


Fig. 2. Pseudo-HAL identification system design

## III. 시스템 설계 및 구현

펌웨어 실행 시 주변장치 함수는 각각의 MMIO 레지스터와 오프셋을 이용해 읽기/쓰기 작업을 한다. 본 장에서는 Fig.2.와 같이 MMIO 오프셋 특징을 이용한 총 3단계의 유사 HAL 함수 탐색 기법을 제안한다.

### 3.1 시스템 설계

**(1) 정보 입력 및 데이터베이스 구성 단계**는 먼저 대상 펌웨어와 이식할 MCU 명을 입력받는다. 해당 MCU 정보는 데이터베이스에서 함수 정보 파일 선택에 사용된다. 데이터베이스 구성은 MMIO 레지스터에 접근하는 최소 단위 함수 특징으로 구성된다. MMIO 범위에서 접근하는 정보를 통해 주변장치를 파악할 수 있다. 따라서 정적분석을 통해 MMIO 범위에 엑세스되는 레지스터 정보를 추출하되, 그 때의 함수이름(*func_i*), 읽기/쓰기 등 상태(*state_rw*)를 추출한다. 퍼징에서 메모리 읽기는 입력요소로서 중요하며, 쓰기는 무시되기에 분류하는 과정이 필요하다. 다음으로 접근했던 MMIO를 통해 주변장치 주소(*peri_addr*)와 오프셋(*offset*)을 추출한다. 데이터베이스는 구성은 〈*func_name, state_rw, peri_addr, offset*〉와 같다.

**(2) 특징 추출 단계**는 입력된 펌웨어에서 먼저 최상위 부모노드를 추출 후 이를 실행 한다. 실행 시 MMIO 범위에 접근하는 모든 명령(instruction)의 *state_rw*를 파악한다. 만약 읽기/쓰기가 발생했다면, 〈*명령주소(ins_addr), 블록주소(block_addr), state_rw, peri_addr, offset*〉 필드를 작성하여 목록화 한다. 이 때, 만약 특정 주변장치의 MMIO의 같은 오프셋에서 읽기와 쓰기가 모두 발생했다면 쓰기로 기입한다. 특정 함수의 경우 레지스터 쓰기를 할 경우 동일한 오프셋을 읽고 쓰기 때문이다.

**(3) 특징 비교 단계**는 특징 추출 결과와 데이터베이스를 비교한다. 먼저 ① 각 주변장치의 MMIO(base address)를 매칭하고, ② 레지스터에 대한 *state_rw*와 *offset*이 일치하는지 여부를 파악한다. 다음으로 ③ 데이터베이스에 있는 함수명으로 매칭하여 유사 HAL 함수 탐색결과를 얻는다. 함수 탐색 결과 필드는 〈*func_i, ins_addr*〉와 같다.

### 3.2 시스템 구현

본 시스템은 Angr를 기반으로 데이터베이스를 구성하고, 유사 HAL 탐색 및 데이터베이스 매칭을 수행하는 972 라인의 파이썬 프로그램으로 만들어졌다. 또한 퍼징은 HAL-Fuzz 기반으로 유사 HAL 함수 탐색결과를 *addr.ymal* 파일로 저장 후 진행한다.

## IV. 실험 및 결과 분석

본 장에서는 제안 방법의 유사 HAL 및 HAL 탐색률과 Libmatch의 HAL 함수 탐색 성능 비교결과를 보여준다. 또한 Grbl 밀링 컨트롤러의 Cortex-M 포트 제공 펌웨어인 CNC 펌웨어를 제안한 기법으로 함수 탐색 후 퍼징을 진행한 결과를 보여준다.

### 4.1 실험 환경

실험은 Intel® CoreTM i7-8700 CPU@ 3.20GHz, 8GB RAM, Ubuntu 18.04.4 LTS (VM) 환경에서 진행됐다. 실험 대상 펌웨어는 STM32F469NI, STM32F

103RB 기반이며, HAL 함수가 호출되는 펌웨어 4개, 유사 HAL함수가 호출되는 펌웨어 10개를 제작하여 실험에 사용했다. 또한 해당 실험에서 사용된 데이터베이스는 총 302개의 주변장치 관련 함수를 포함한다.

### 4.2 유사 HAL 함수 탐색 결과

Table 3.은 유사 HAL 함수가 호출되는 10개의 펌웨어의 주변장치 관련 함수 탐색 결과이다. 해당 펌웨어는 유사 HAL 함수만 호출하도록 제작되었으며, HAL 함수가 호출되지 않았기 때문에 Libmatch로는 함수 탐색이 되지 않았다. 하지만 제안하는 방법으로는 각 펌웨어에서 호출되는 전체 주변장치 함수 대비 최대 92.3%, 평균 68.38%의 높은 탐색률을 보인다.

Table 3. Pseudo-HAL function identification rate (%)

| Firmware | Libmatch | Ours |
|---|---|---|
| Baremetal_I2C | 0 | 68.1 |
| FreeRTOS_I2C | 0 | 63.6 |
| RIOT_SPI_receive | 0 | 64.2 |
| Baremetal_UART | 0 | 64.2 |
| RIOT_I2C_receive | 0 | 60.0 |
| RIOT_I2C_transmit | 0 | 62.5 |
| RIOT_SPI_receive | 0 | 60.0 |
| RIOT_UART | 0 | 64.7 |
| RIOT_SPI | 0 | 92.3 |
| RIOT_I2C | 0 | 84.2 |

### 4.3 HAL 함수 탐색 결과

Fig.3.은 제안하는 방법과 Libmatch의 HAL 함수 탐색률을 보여준다. SPI_receive의 경우 제안하는 방법이 더 높은 탐색률을 보이는 반면, UART_transmit과 UART_receive는 2개, I2C_receive는 1개의 함수가 Libmatch에서 더 탐색됐다. 이는 제안하는 방법으로 NVIC 관련 함수인 *HAL_NVIC_SetPriority*, *HAL_NVIC_SetPriorityGrouping*이 탐색되지 않았기 때문이다. 하지만 위 함수는 함수의 접근 주소가 MMIO 범위가 아니기에 제안 방법의 탐색 범위에 해당하지 않는다. 따라서, 제안 방법이 HAL 함수 또한 Libmatch 만큼 탐색 가능하다는 것을 알 수 있다.

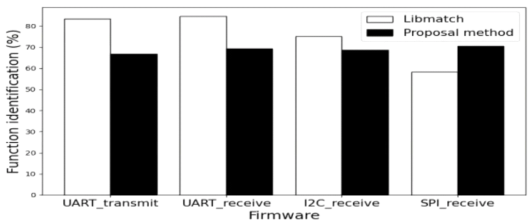


Fig. 3. HAL function identification rate

### 4.4 퍼징 결과

Table 4.은 앞선 유사 HAL 탐색 결과를 기반으로 한 CNC 펌웨어의 퍼징 결과를 보여주는데, HAL-Fuzz로는 실행되지 않았던 CNC 펌웨어가 제안된 방법으로 퍼징이 가능하고 크래시도 발견할 수 있음을 확인했다. 이를 통해 유사 HAL 함수 탐색으로 HAL-Fuzz의 퍼징 가용성을 확장할 수 있음을 알 수 있다.

Table 4. Result of CNC firmware fuzzing (24h)

| | Executions | Total Paths | Crashs |
|---|---|---|---|
| Ours | 4,020,289 | 958 | 6 (Unique) |
| HAL-Fuzz | X | X | X |

## V. 결론 및 향후 계획

본 논문은 HAL-Fuzz의 퍼징 가용성 확장을 위해 유사 HAL 함수 탐색 방법을 제안했으며, 제안된 방법을 통해 펌웨어만으로 유사 HAL과 HAL 함수를 평균 약 68%로 탐색할 수 있음을 확인했다. 또한 펌웨어와 간단한 정보만으로 함수 추출이 가능한 점에서 object가필요하고 컴파일 옵션 등의 정보가 필요한 Libmatch의 단점을 극복했다. 향후 본 논문을 기반으로 보다 다양한 함수를 호출하는 펌웨어를 제작 및 검증을 통해 함수 탐색률을 높여 기법을 발전시키고자 한다.

## References

[1] M. Muench, J. Stijohann, F. Kargl, A. Francillon, and D. Balzarotti. "What You Corrupt Is Not What You Crash: Challenges in fuzzing embedded devices," NDSS, Jan. 2018.

[2] A. A Clements, E. Gustafson, T. Scharnowski, P. Grosen, D. Fritz, C. Krue

gel, G. Vigna, S. Bagchi,and M. Payer.“Halucinator: Firmware re-hosting through abstraction layer emulation,” USENIX 20, pp. 1201 - 1218, 2020.
〔3〕 B. Feng, A. Mera, and L. Lu.“P2im: Scalable and hardware-independent firmware testing via automatic peripheral interface modeling,” USENIX 20, pp. 1237 - 1254, 2020
〔4〕 Wei Zhou, Le Guan, Peng Liu, and Yuqing Zhang. “Automatic firmware emulation through invalidity-guided knowledge inference,” USENIX 21, 2021.
〔5〕 E. Gustafson, M. Muench, C. Spenskyy, N. Redini, A. Machiry, Y. Fratantonio, D. Balzarotti, A. Francillon, Y. RynChoe, C. Kruegel, et al. “Toward the analysis of embedded firmware through automated re-hosting,” RAID, pp. 135 - 150, Sep. 2019

## 〈저자소개〉

정 세 연 (Seyeon Jeong) 정회원
2018년: 대구가톨릭대학교 정보보호학과 학사
2021년: 연세대학교 정보대학원 정보보호 석사
2021년~현재: 슈어소프트테크(주) 재직
〈관심분야〉 소프트웨어 보안, 시스템 보안, 정보 보안

황 은 비 (Eunbi Hwang) 학생회원
2019년: 성신여자대학교 통계학과 학사
2019년~현재: 연세대학교 정보대학원 정보보호 석박통합과정
〈관심분야〉 소프트웨어 보안, 시스템 보안

조 영 필 (Yeongpil Cho) 종신회원
2010년: 포항공과대학교 전자전기공학과 학사
2018년: 서울대학교 전기컴퓨터공학부 박사
2018년~2020년: 숭실대학교 조교수
2020년~현재: 한양대학교 조교수
〈관심분야〉 소프트웨어 보안, 시스템 보안

권 태 경 (Taekyoung Kwon) 종신회원
1992년 2월: 연세대학교 컴퓨터과학과 학사
1995년 2월: 연세대학교 컴퓨터과학과 석사
1999년 8월: 연세대학교 컴퓨터과학과 박사
1999년~2000년: U.C. Berkely Post-Doc
2001년~2013년 8월: 세종대학교 컴퓨터공학과 교수
2007년~2008년 Univ. Maryland at College Park 교환교수
2013년 9월~현재: 연세대학교 정보대학원 교수
〈관심분야〉 암호 프로토콜, 소프트웨어 보안, 시스템 보안, 인공지능 보안