논문 2017-12-40

BadUSB 취약점 분석 및 대응 방안

(Analysis and Countermeasure for BadUSB Vulnerability)

서 준 호, 문 종 섭*
(Jun-Ho Seo, Jong-Sub Moon)

Abastract : As the BadUSB is a vulnerability, in which a hacker tampers the firmware area of a USB flash drive. When the BadUSB device is plugged into the USB port of a host system, a malicious code acts automatically. The host system misunderstands the act of the malicious behavior as an normal behaviour for booting the USB device, so it is hard to detect the malicious code. Also, an antivirus software can't detect the tampered firmware because it inspects not the firmware area but the storage area. Because a lot of computer peripherals (such as USB flash drive, keyboard) are connected to host system with the USB protocols, the vulnerability has a negative ripple effect. However, the countermeasure against the vulnerability is not known now. In this paper, we analyze the tampered area of the firmware when a normal USB device is changed to the BadUSB device and propose the countermeasure to verify the integrity of the area when the USB boots. The proposed method consists of two procedures. The first procedure is to verify the integrity of the area which should be fixed even if the firmware is updated. The verification method use hashes, and the target area includes descriptors. The second procedure is to verify the integrity of the changeable area when the firmware is updated. The verification method use code signing, and the target area includes the function area of the firmware. We also propose the update protocol for the proposed structure and verify it to be true through simulation.

Keywords : BadUSB, Integrity, Bootloader, Firmware, Code signing

Ⅰ. 서 론

Universal Serial Bus (USB)는 컴퓨터와 전자 장치간의 연결 및 통신, 전원 공급을 위한 표준 규격이며 키보드나 마우스, 대용량 저장 장치 등을 연결하는데 사용된다 [1]. USB 플래시 드라이브는 USB 인터페이스가 통합된 플래시 메모리가 포함된 데이터 저장 장치이다.

USB 통신 규격은 다양한 용도로 사용되지만 상대적으로 USB 장치에 대한 보안은 다른 임베디드 장치에 비해 관심과 중요성이 덜 강조되고 있다. 2014년에는 USB의 설계상 취약점을 이용한 보안 이슈가 대두되었다.

*Corresponding Author (jsmoon@korea.ac.kr)
Received: Aug. 30 2017, Revised: Sep. 25 2017, Accepted: Oct. 27 2017.
J.H. Seo, J.S. Moon : Korea University

ⓒ IEMEK J. Embed. Sys. Appl. 2017 Dec. 12(6) 359-368
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2017.12.6.359

BadUSB 취약점은 Blackhat USA 2014 보안 컨퍼런스에서 Karsten Nohl과 Jakob Lell의 “BadUSB - On accessories that turn evil” 발표를 통해 처음 소개되었다 [2]. 이들은 USB 업데이트 과정을 분석하고 펌웨어를 역공학하여 찾아낸 USB 플래시 드라이브의 설계상 취약점을 이용하여 펌웨어를 재프로그래밍 하였다. 이렇게 변조된 펌웨어는 윈도우의 DNS 설정을 변경하거나 키보드로 인식시키는 등의 악의적인 행위가 가능하였다. 그러나 그들은 BadUSB 취약점이 잠재적으로 모든 USB 장치에 적용 가능하기 때문에 파급력이 상당하고 대응책이 없다고 판단하여 소스코드나 제작 방법과 같은 세부적인 내용은 공개하지 않았다.

하지만 DerbyCon 4.0에서 Adam Caudill과 Brandon Wilson은 Phison 2303 마이크로컨트롤러 기반의 USB 플래시 드라이브를 키보드로 인식시키는 BadUSB의 동작을 시연하고 BadUSB 제작에 사용한 소스코드와 도구들을 공개하였다 [3, 4]. 이는

BadUSB 취약점이 USB 장치의 설계상의 취약점이므로 USB 제조사들에게 압력을 가하고 대응책을 마련을 강구하기 위함이었다.

BadUSB 취약점은 키보드에뮬레이션 (Keyboard Emulation)을 이용하여 권한 상승을 일으키거나 DNS 설정을 바꾸는 등의 컨퍼런스에서 발표된 공격 시나리오 외에도, 호스트 파일을 변조하거나 중요 파일을 유출시킬 수 있고 사용자의 입력을 가로채는 키로깅 (Keylogging)을 통해 사용자 입력 정보를 획득하는 행위가 가능하다. 또한 악성 사이트에 접속하여 멀웨어 (Malware)를 다운로드 하거나, 설정되어 있는 망분리를 무력화시킬 수 있다.

이와 같이 BadUSB 취약점을 이용하여 값을 입력하거나 정보를 유출시킬 수 있으며 호스트의 설정을 바꾸는 행위가 가능하지만 이러한 행위들로 수반되는 보안 위협은 안티 바이러스에 의한 탐지나 USB 포맷으로 해결이 어려우며, 제시한 내용 이외의 보다 다양하고 수준 높은 보안 위협을 초래할 수 있다.

본 논문에서는 호스트에서 탐지가 어려운 BadUSB 취약점에 대하여, 분석을 통해 설계 단계의 대응책을 제시한다. 이러한 설계 단계의 대응책은 USB 장치를 부팅시키지 않기 때문에 악의적인 코드가 실행되지 않아 보안 위협을 방지할 수 있다.

본 논문의 구성은 다음과 같다. Ⅱ장에서 BadUSB 취약점 관련 연구 및 BadUSB 취약점의 발생 원인을 분석하며, Ⅲ장에서 이에 대하여 부팅 시 USB 부트로더 및 펌웨어의 무결성 검증을 방식과 펌웨어 업데이트 프로토콜을 제안한다. Ⅳ장에서는 제안하는 내용에 대해 시뮬레이션을 통해 검증하고, Ⅴ장에서 결론을 맺는다.

Ⅱ. 관련 연구

이 장에서는 먼저 기존의 BadUSB 취약점 관련 연구를 소개하고, 일반 USB 장치가 BadUSB로 변조되는 부분을 살펴본다.

1. BadUSB 관련 연구

기존의 BadUSB 관련 연구의 경우 BadUSB가 수행되는 호스트 기반에서의 관련 연구와 BadUSB가 변조되는 장치 기반의 연구로 나눌 수 있다.

1.1 호스트 기반의 BadUSB 관련 연구

호스트 기반의 관련 연구는 안티바이러스 제품이나 보안 프로그램을 통해서 BadUSB의 키보드 에뮬레이션 동작을 차단하는 방식으로 진행되었다. 해외 안티 바이러스의 경우 USB 장치가 키보드로 인식될 때, 화상 키보드나 키보드 입력을 이용하여 사용자가 입력을 통해 인증한 후에 USB 장치를 키보드로 사용할 수 있도록 한다 [5]. 보안 프로그램을 통한 연구는 입력 메시지를 후킹하거나 문자열 입력간의 시간 차이를 분석하거나, 특정 문자열에 대하여 차단하는 방법으로 진행되었다 [6]. 이러한 방식들은 BadUSB 취약점을 이용한 키보드 에뮬레이션에 대해서만 방어가 가능하며, 호스트에 기반을 두는 보안 대책이기 때문에 다양한 운영체제에 대해 각각 적용해야한다는 단점이 있다. 또한 악의적으로 삽입한 코드가 검증되지 않은 장치 드라이버 설치를 유도하여 악의적인 행위를 할 수 있기 때문에, 설치되는 드라이버의 무결성을 검증하는 연구도 진행되었다 [7]. 하지만 이러한 BadUSB 취약점에 대해 드라이버의 무결성을 검증하는 연구의 경우 각 호스트의 운영체제 따른 제약사항과 무결성 비교 가능한 데이터베이스에 대한 한계가 있다. 또한 호스트 기반의 BadUSB 검증 연구는 실제 변조된 USB 장치의 펌웨어의 악의적인 코드에 대해서는 직접 검증하지 못하는 한계를 지닌다.

1.2 장치 기반의 BadUSB 관련 연구

장치 기반의 관련 연구는 장치 내의 펌웨어가 변조되었는지에 대해 데이터 무결성을 검증하는 방식으로 진행된다. 데이터 무결성이란 데이터의 정확성 및 일관성을 보장하고 유지하는 것으로 [8], 데이터가 정당하게 생성되어 위·변조되지 않음을 보장해주는 성질이다. BadUSB 검증 방식에 대하여 코드 서명 (Code Signing)을 이용한 해결 방안이 제시되었다 [2, 9]. 코드 서명 방식은 서명 값을 이용하여 해당 정보가 정당한 생성자에 의해 생성된 것이며 위·변조되지 않음을 검증할 수 있다. 특히 이 방식의 경우 장치에 저장되는 공개키 보관에 대한 안전성과 검증 방식의 정확성이 고려되어야 한다. 이러한 코드 서명 기법과 암호 기술 관련 美 표준 기술에 기반하여 BadUSB 악용가능성을 감소시키는 방안이 제시되었지만 [9], 성능이나 경제성 등을 고려하여 현실적으로 제한될 수 있다.

본 논문에서는 실제 BadUSB를 분석하여 변조되는 부분을 도출하고, 암호학적 해시 함수와 코드 서명 기법을 이용하여 BadUSB 취약점에 대한 상세한 대응방안을 제시한다. 또한 시뮬레이션을 통해 제안하는 대응 방안에 대해 검증한다.

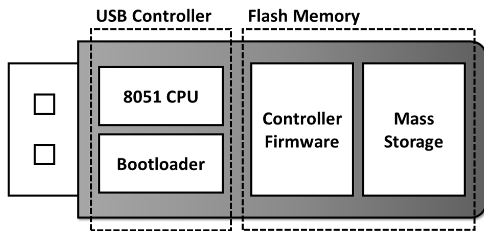

그림 1. USB 플래시 드라이브 구조
Fig. 1 The structure of USB flash drive

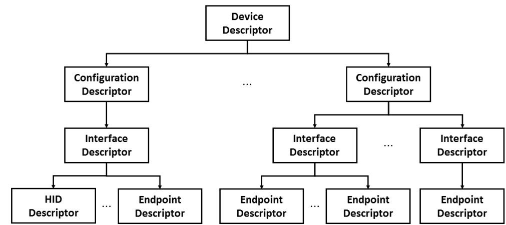

그림 2. 디스크립터의 계층적 구조
Fig. 2 The hierarchy of descriptors

2. 분석

USB 장치의 구조와 전송 방식, 자료 구조 등의 일반적인 USB 플래시 드라이브에 대한 분석을 통해 BadUSB 장치로 변조되는 부분을 분석한다.

2.1 USB 플래시 드라이브 구조

그림 1은 Blackhat USA 2014에서 Karsten Nohl과 Jakob Lell이 발표한 내용의 USB 장치 구조이다 [2]. 내부 컨트롤러의 경우 데이터 영역과 메모리 영역이 분리된 하버드 구조로 8051 프로세서를 사용하며, USB 장치의 동작을 제어하는 기능을 수행한다. 플래시 메모리의 경우 컨트롤러가 실행할 코드 및 데이터가 저장되어 있는 펌웨어와 사용자가 데이터를 저장할 수 있는 스토리지가 존재한다. BadUSB 장치는 사용자가 호스트를 통해 데이터를 저장하는 스토리지가 아닌 펌웨어를 변조한 것이기 때문에, 스토리지 영역을 검사하는 안티 바이러스를 통해 BadUSB 취약점을 탐지하기 어려우며 USB를 포맷하더라도 변조된 부분을 정상적으로 복구하기 힘들다.

2.2 USB 전송 방식

USB 전송 방식은 호스트와 USB 장치 간의 데이터 교환을 의미한다. USB 전송 방식에는 제어 전송과 벌크 전송, 인터럽트 전송, 등시성 전송이 존재한다 [10].

제어 전송은 USB 장치가 호스트와 연결 시 가장 먼저 일어나며 장치의 종류, 전송속도 등을 결정한다. 이러한 장치 인식 과정을 열거(Enumeration) 과정이라고 부르며, 이때 장치 정보를 나타내는 디스크립터 (Descriptor)라는 자료구조를 교환한다. 이 자료구조는 '2.3 디스크립터'에서 상세히 다루며, BadUSB 장치는 제어전송을 통해 기존의 USB 장치 이외의 다른 장치로 인식시킨다.

벌크 전송은 대용량 저장 장치에서의 데이터 교환이나 호스트가 USB 장치의 동작을 제어하기 위한 명령 전달 시 사용된다. 인터럽트 전송은 사용자 인터페이스를 담당하는 키보드나 마우스 같은 Human Interface Device (HID)에서 주기적인 데이터 전송을 위해 사용된다 [11]. 등시성 전송은 실시간성이 필요한 데이터 전송 시에 사용된다. 만약 BadUSB 장치가 키보드와 같은 HID 장치로 인식되어 데이터를 전송할 경우 인터럽트 전송이 사용된다.

2.3 디스크립터

디스크립터는 장치의 종류, 기능과 같은 정보를 정해진 양식으로 표현하는 자료구조이다. 앞서 설명한 제어 전송을 이용한 열거과정에서 디스크립터를 교환하며, 디스크립터를 변조하여 BadUSB 장치로 인식시킬 수 있다. 디스크립터는 그림 2와 같이 계층적인 구조를 가진다 [12].

장치 디스크립터 (Device Descriptor)는 단 하나만 존재하며 호스트에 가장 먼저 전달된다. 장치의 버전과 제조사, 제품에 대한 정보 등을 가진다.

컨피규레이션 디스크립터 (Configuration Descriptor)는 장치 기능 등의 설정 정보를 가지며, 인터페이스 디스크립터 (Interface Descriptor)는 대용량 저장장치나 HID 장치와 같이 장치의 실제기능과 관련된 정보를 가진다. 각 상위의 디스크립터는 하위 디스크립터의 개수를 가지고 있다.

가장 하위의 디스크립터는 엔드포인트 디스크립터, HID 디스크립터, 리포트 디스크립터 등이 있다. 엔드포인트 디스크립터 (Endpoint Descriptor)는 데이터의 전송 방향이나 전송 형식, 최대 패킷 사이즈 등을 포함하며, 상위 인터페이스 디스크립터에서 개수를 명시한 엔드포인트를 이용한 전송 방식에 대한 정보를 가진다. HID 장치임을 의미하는 HID 디스크립터는 하위 디스크립터의 종류와 크기를 결정하며, 특히 하위의 리포트 디스크립터의 경우 HID 장치가 호스트와 주고받는 가변적인 길이의 데이터를 포함한다.

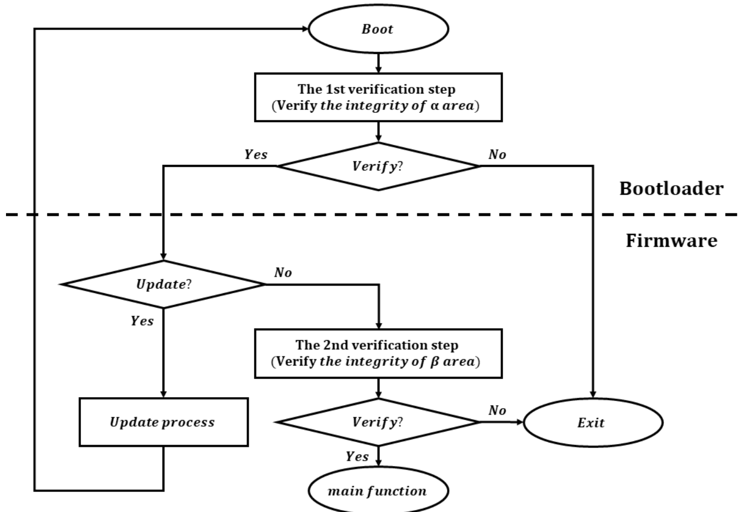

그림 3. 제안하는 검증 과정을 포함한 부팅 과정
Fig. 3 The boot process with suggested verification method

2.4 변조

BadUSB 장치는 기존의 USB 장치로써의 기능이 아닌 다른 기능으로 동작하는 장치로, 기능을 변경하거나 새로운 기능을 추가할 수 있다. 앞서 언급한 디스크립터나 내장 함수를 수정하여 기능을 수정할 수 있다.

디스크립터를 수정하는 경우 공격자는 장치의 기능에 대한 정보를 갖는 인터페이스 디스크립터를 추가하거나 수정한다. 이로 인해 기존의 제조사가 설정한 USB 장치의 기능 이외의 HID와 같이 다른 용도를 추가시킬 수 있다. 이때 상위의 컨피규레이션 디스크립터에서 인터페이스 디스크립터의 개수를 수정하며, 하위의 엔드포인트 디스크립터나 HID 디스크립터, 리포트 디스크립터 등을 추가하여 원하는 공격 유형의 BadUSB 장치로 변조 가능하다.

또한 펌웨어 내의 함수 영역에 공격자가 원하는 함수를 추가하거나 기존의 함수를 수정하여, 기존 USB 장치에 악의적인 행위가 가능하도록 한다.

Adam Caudill과 Brandon Wilson은 BadUSB를 제작할 수 있는 도구들을 공개하였으며 [4], 해당 도구를 이용하여 디스크립터나 내장 함수를 수정한 펌웨어를 생성할 수 있다. 이렇게 생성한 펌웨어를 USB에 주입하여 BadUSB 장치를 제작할 수 있다.

디스크립터와 내장 함수는 모두 플래시 드라이브 내의 펌웨어에 저장되어 있다. 변조된 디스크립터와 내장 함수가 수행되는 이유는 BadUSB 장치가 동작할 때, 펌웨어가 제조사에 의해 정당하게 생성되어 위・변조되지 않았는지에 대한 무결성 검증이 이루어지지 않기 때문이다.

Ⅲ. 제안 방법

BadUSB 장치는 디스크립터나 내장함수의 변조되는 내용에 따라 다양한 악의적인 행위가 가능하며, 안티 바이러스 제품이나 보안 프로그램으로 BadUSB 취약점을 이용한 모든 공격을 방어하기에는 한계가 있다. 또한 사용자는 다양한 플랫폼의 운영체제를 사용하기 때문에 모든 호스트를 고려하여 호스트 기반의 무결성을 검증 방식을 통한 해결 방식에는 어려움이 있다. 따라서 본 논문에서는 USB 장치에서의 펌웨어의 무결성 검증 방법을 제시한다.

USB 펌웨어는 업데이트를 통해 수정 가능한 부분이 있지만, 업데이트 되더라도 디스크립터와 같이 수정되지 않아야하는 부분이 있다. 본 논문에서는 변하지 않아야 하는 영역을 α영역 (고정 영역)이라 부르며, 업데이트를 통해 수정 가능한 영역을 β영역 (가변 영역)이라고 부른다. 각 영역에 대해 순차적인 두 개의 단계로 나눠서 검증한다.

전체적인 검증을 포함하는 부팅 프로세스는 그림 3과 같으며, 먼저 부트로더에서 펌웨어의 α영역을 검사한다. 이후 검증을 통과하면, 업데이트과정인지 여부에 따라 펌웨어 업데이트를 수행하거나 펌웨어 부팅을 위해 2단계 무결성 검증을 수행한다.

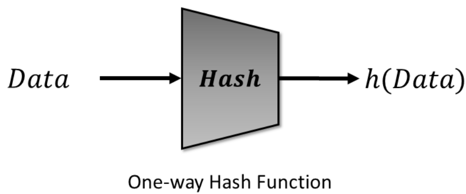

그림 4. 일 방향 해시 함수
Fig. 4 One way hash function

업데이트를 진행하는 함수와 2단계 무결성 검증에 사용되는 함수, 키 값은 1단계 무결성 검증 과정을 통과된 영역으로 신뢰할 수 있는 작업을 수행한다. 각각의 1, 2단계 무결성 검증 방식 및 펌웨어 업데이트 프로토콜은 아래에서 상세하게 살펴본다.

1. 1단계 펌웨어 고정 영역 무결성 검증

1단계 무결성 검증 방식의 경우 수행 주체는 부트로더이며, 부팅 시 가장 먼저 해시 값을 이용하여 펌웨어의 α영역을 검증한다. α영역은 앞서 설명한 디스크립터와 같이 제조사의 올바른 펌웨어 업데이트 후에도 수정되지 않아야하는 고정 영역이다. 해당 영역은 디스크립터와 펌웨어 업데이트 함수, 2단계 무결성 검증 (가변 영역 무결성 검증)을 수행하는 함수와 이때 사용되는 키 값이 있다.

USB 장치 제조 시 제조사는 부트로더에 α영역에 대한 해시 값을 저장한다. α영역을 검증하는 함수는 부팅 시 펌웨어의 해당 영역의 값을 읽어 해시 함수에 넣고 저장된 해시 값과 비교한다. 비교에 실패하면 장치는 부팅되지 않도록 한다. 이 때 사용되는 암호학적 해시 함수는 그림 4와 같이 일 방향 해시 함수의 일종으로 역상 저항성, 제 2 역상 저항성, 충돌저항성을 만족시켜 원래의 입력 값과 해시 값이 같은 입력 값을 찾기 어렵게 하는 성질을 만족하는 해시 함수이다 [13]. 특히 제 2 역상 저항성은 해시 함수와 입력을 모두 아는 경우에도 해시 값이 같은 입력을 찾는 것이 계산적으로 불가능한 성질이다. 이러한 성질은 공격자가 역공학을 통해 해당 해시 함수를 알아내더라도 해당 해시 값과 같은 펌웨어 영역을 만들어 낼 수 없음을 보장한다.

이러한 암호학적 해시 함수는 실제 임베디드 분야에서 펌웨어를 검증하기 위해 사용되거나 펌웨어의 무결성 검증 연구 분야에 적용되며 [14, 15], 펌웨어 업데이트 시에도 데이터 무결성을 위하여 사용된다 [16].

하지만 부트로더에 저장된 펌웨어의 해시 값에 대한 안전성이 보장되지 않으면, 공격자는 펌웨어 수정과 동시에 해시 값 또한 수정하여 1단계 무결성 검증을 우회할 수 있다. 본 연구 분석에 사용된 8051 계열의 마이크로컨트롤러 외에도 대부분의

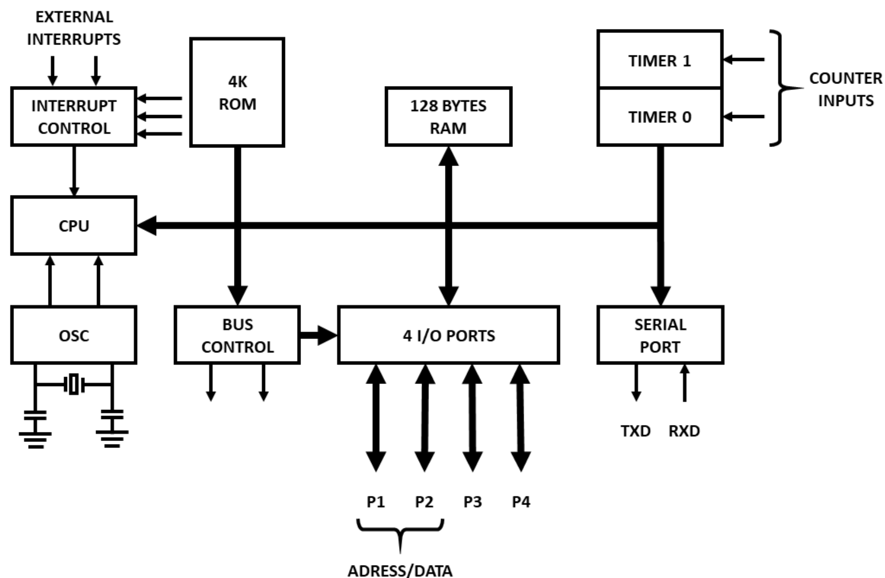

그림 5. 8051 코어의 블록 다이어그램
Fig. 5 Block diagram of the 8051 core

그림 6. 코드 서명을 통한 서명 값 생성
Fig. 6 Creating the signature for code singing

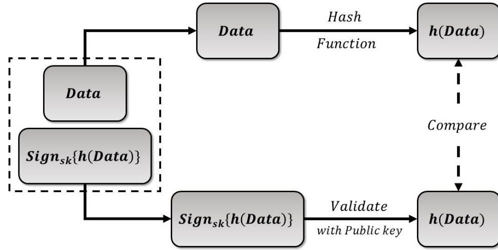

그림 7. 코드 서명을 통한 서명 값 검증
Fig. 7 Validating the signature for code signing

마이크로컨트롤러 (MCU)는 그림 5와 같이 내부에 Central Processing Unit (CPU)나 Random Access Memory (RAM), 부트로더를 저장할 Read Only Memory (ROM)를 가진다 [17]. 마이크로컨트롤러 내부의 ROM은 Erasable Programmable Read-Only Memory (EPROM) 이외에도 한 번 데이터를 기록하면 수정할 수 없는 마스크롬 (Mask ROM)을 사용할 수 있다. 본 논문에서 제안하는 부트로더와 해시 값은 마이크로컨트롤러 내부의 마스크롬을 사용한 ROM에 저장하도록 하여, 제조 시 저장한 내용이 수정될 수 없도록 무결성을 보장하도록 한다.

α영역 검증 함수의 의사코드는 알고리즘 1과 같으며, 본 논문에서 제안하는 1단계 무결성 검증 함수와 펌웨어의 α영역에 대한 해시 값을 추가하는 것 이외의 기존의 부트로더 영역은 변경하지 않는다.

2. 2단계 펌웨어 가변 영역 무결성 검증

2단계 무결성 검증 방식의 경우 수행 주체는 펌웨어이며, 코드 서명 기법을 이용하여 펌웨어 업데이트 시 변경되는 β영역에 대하여 검증한다. 코드 서명은 검증하고자하는 정보에 대해 해당 부분의 서명 값을 이용하여 생성자가 정당한지 증명할 수 있으며, 정보가 위·변조되지 않음을 보장해주는 기법이다 [18].

서명 값 생성과정은 그림 6과 같다. 서명하고자 하는 정보에 대해 해시 함수를 통해 해시 값을 생성한 뒤, 개인키를 이용하여 서명한다. 검증 과정은

Pseudo Code 1 Verify the integrity of α area

```
1: function verify α()
2:    target = read(update_firmware_function
              | verify_β_function |
              descriptors | public key)
              in firmware
3:    h_alpha = read the hashed data (of α
              area) in bootloader
4:    if h_alpha ≠ h(target) then
5:       exit booting
6:    else
7:       jump to firmware (ask whether it is
         a process for updating firmware)
8:    end if
9: end function
```

알고리즘 1. α영역 무결성 검증 의사코드
Algorithm 1. The pseudo code for verifying the integrity of α area

Pseudo Code 2 Verify the integrity of β area

```
1: function verify β()
2:    target = read(header | functions area)
3:    h_target = hash(target)
4:    h_beta = validate the signature (in
             firmware) using the public
             key
5:    if h_target ≠ h_beta then
6:       exit booting firmware
7:    else
8:       jump to main function in functions
         area
9:    end if
10: end function
```

알고리즘 2. β 영역 무결성 검증 의사코드
Algorithm 2. The pseudo code for verifying the integrity of β area

그림 7과 같이 검증자가 검증할 정보와 해시 함수를 이용하여 해시 값을 생성하고, 공개키를 이용하여 서명 값에서 해시 값을 추출한 뒤 두 값을 비교한다. 값이 같을 경우 코드 서명을 통해 해당 정보에 대한 무결성이 보장된다.

코드 서명은 암호학적 해시 함수와 함께 펌웨어의 데이터 무결성 검증과 업데이트 시 검증을 위해 사용된다 [14, 16].

본 논문에서 펌웨어 업데이트 시 수정되는 β영역

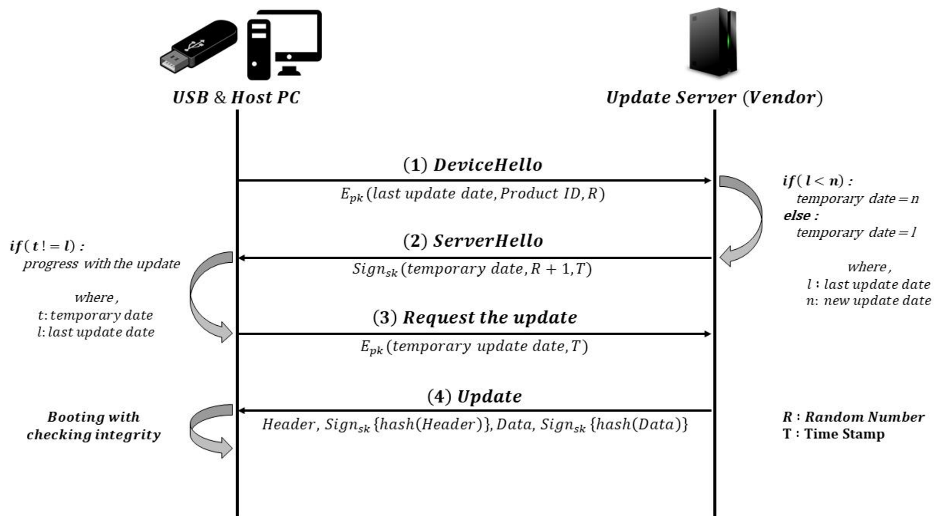

그림 8. 제안하는 펌웨어 업데이트 프로토콜
Fig. 8 The proposed firmware update protocol

은 업데이트를 위한 펌웨어 정보를 내장하는 헤더 영역과 USB 장치의 기능이 구현된 내장 함수 영역이다. 또한 제조사는 업데이트 시 β영역에 대해 해시 함수와 제조사의 개인키를 이용하여 생성한 서명 값을 함께 업데이트한다.

부트로더에서 1단계 무결성 검증을 마친 후, 펌웨어 부팅 시 2단계 무결성 검증 함수는 β영역을 읽어 해시 함수에 넣은 해시 값과 공개키를 이용하여 서명 값으로부터 추출한 해시 값을 비교한다. 비교한 값이 다르면 펌웨어 부팅을 종료하며, 비교한 값이 같으면 기존의 내장 함수 영역의 메인 함수로 이동한다.

β영역 검증 함수의 의사코드는 알고리즘 2와 같으며, 본 논문에서 제안하는 헤더 영역과 펌웨어 업데이트 함수, 2단계 무결성 검증 함수, 공개키, 업데이트 시 함께 업데이트되어 검증 시 사용되는 β영역의 서명 값을 추가하는 것 이외의 기존의 펌웨어 영역은 변경하지 않는다.

본 논문에서는 제조 시 값이 고정되어야 하는 고정 영역과 업데이트 시 수정되는 가변 영역에 대해 순차적으로 서로 다른 주체가 무결성을 검증하는 두 단계의 무결성 검증 방식을 제시하였다.

본 논문에서 마이크로컨트롤러의 ROM은 마스크롬으로 설정하도록 제시하였다. 마스크롬은 최초로 내용을 기록한 후 내용을 변경할 수 없기 때문에 마스크롬 내에 저장되어 있는 부트로더의 무결성을 보장한다 [19]. 또한, 펌웨어에 저장되어 있는 디스크립터와 같이 USB 장치의 기능을 포함한 내용은 값이 변경되면 의도치 않은 장치의 기능을 수행할 수 있다. 따라서 디스크립터와 같은 펌웨어의 고정 영역에 대한 해시 값과 1단계 무결성 검증을 수행하는 함수는 부트로더에 저장하여 1단계 무결성 검증을 수행한다.

펌웨어 업데이트 시 수정되는 가변 영역 또한 제조사에 의해 정당하게 수정되었음을 보장하여야 한다. 공격자는 수정되는 내장하는 함수 영역을 변조하여 악의적인 행위를 수행할 수 있다. 따라서 본 논문에서 해당 영역의 무결성 검증을 코드 서명 기법을 이용하여 두 번째 단계에 수행한다. 가변 영역은 펌웨어 업데이트 시마다 변경되어야하기 때문에 검증에 사용되는 서명 값은 업데이트 시 가변 영역과 함께 펌웨어에 저장한다. 두 번째 단계에서 사용되는 공개키와 무결성 검증을 수행하는 함수는 1단계 무결성 검증을 통해 검증된 영역이다.

따라서 본 논문에서는 1단계와 2단계의 무결성 검증 방식을 제시하였다.

3. 펌웨어 업데이트 프로토콜

본 논문에서 제안하는 펌웨어 구조의 경우 공개키가 존재하기 때문에 업데이트 과정 역시 공개키 사용이 가능하다. 제안하는 업데이트 프로토콜은 공개키를 이용한 단방향 인증을 통해 정당한 서버로부터 업데이트를 진행할 수 있는 프로토콜이며, 그림 8과 같이 총 4단계로 진행된다.

먼저 1단계의 경우 업데이트 서버에 최신 업데이트를 확인하기 위한 과정이다. USB 장치는 펌웨어의 헤더에 저장되어 있는 마지막 펌웨어 업데이트 날짜, USB의 장치 ID와 난수를 생성하여 공개키로 암호화한 뒤 호스트를 통해 업데이트 서버로 전송한다.

2단계에서 업데이트 서버는 1단계에서 받은 정보를 제조사의 개인키로 복호화한다. 마지막 업데이트 날짜를 확인하고 최신 업데이트 날짜보다 이전일 경우 임시 날짜 변수 (Temporary date)에 새로운 업데이트 날짜를 채우거나 업데이트가 필요 없는 경우 장치로부터 받은 날짜를 채운다. 해당 임시 날짜와 난수에 1을 더한 값, 현재 시간 값을 제조사의 개인키로 서명한 뒤 전송한다.

3단계에서 USB 장치는 이전 단계에서 받은 값에 대해 공개키를 이용하여 정보를 추출한 뒤 임시 날짜를 확인하여, 업데이트가 필요한 경우 요청하고자하는 최신 업데이트 날짜와 시간 값을 공개키로 암호화하여 업데이트 서버에 다시 전송한다.

마지막으로 업데이트 서버는 3단계에서 장치로부터 받은 정보를 복호화하여 시간 값이 현재 시간의 임계값 안에 포함될 경우, 헤더 정보와 헤더 정보에 대한 서명 값, 업데이트할 데이터와 서명 값을 보낸다. 장치는 헤더의 서명 값을 통해 헤더 정보를 검증한 뒤, 검증이 완료되면 업데이트를 수행한다. 업데이트 후에는 본 논문에서 제안하는 검증 과정을 거쳐 부팅하도록 한다.

Ⅳ. 시뮬레이션 및 검증

본 논문에서 제시한 USB 무결성 검증 방법과 USB 펌웨어 업데이트 내용에 대하여 리눅스 환경에서의 시뮬레이션을 진행하여 검증을 수행한다.

USB 장치의 펌웨어를 수정할 수 있는 도구가 공개된 것과 달리 부트로더는 제조사 이외에 수정이 불가능하다. 따라서 본 논문에서 실제 USB 장치가 아닌 리눅스 환경에서 부트로더와 펌웨어를 자체적으로 제작하여 시뮬레이션 검증을 수행한다. 해당 호스트는 3.30GHz intel core i5, DDR3 1GB RAM, 우분투 (Ubuntu) 운영체제를 사용하였다. 해시 함수는 MD5, 코드 서명을 위한 알고리즘은 임베디드 장치를 고려하여 ECDSA 방식을 사용하였으며, 키 길이는 160비트를 사용하였다.

무결성 검증 시뮬레이션은 정상적인 부팅 과정을 수행한 뒤, 디스크립터와 함수 영역을 각각 수정하여 변조된 펌웨어 부팅 과정에 대해 시뮬레이션

```
system@ubuntu:~/Desktop/badUSB$ ./start_bootloader firmware
=============================================================
IN BOOTLOADER
-------------------------------------------------------------
1st Verification (Verify alpha area in firmware.)
        -> Success verifying alpha area
        -> Boot Firmware

=============================================================
IN FIRMWARE
-------------------------------------------------------------
Update the firmware? (1 : yes, 2 : No! boot the firmware)? 2
-------------------------------------------------------------
2nd Verification(Verify beta area in firmware.)
        -> Sucess verifying beta area
        -> Booting Firmware

=============================================================
Executing USB device
```

그림 9. 정상적인 USB 장치 부팅 결과
Fig. 9 The result of booting normal USB device

```
system@ubuntu:~/Desktop/badUSB$ ./start_bootloader firmware
=============================================================
IN BOOTLOADER
-------------------------------------------------------------
1st Verification (Verify alpha area in firmware.)
      -> Fail & Exit
```

그림 10. 디스크립터 변조 시 USB 부팅 결과
Fig. 10 The result of booting USB device with modified descriptors in the firmware

```
system@ubuntu:~/Desktop/badUSB$ ./start_bootloader firmware
=============================================================
IN BOOTLOADER
-------------------------------------------------------------
1st Verification (Verify alpha area in firmware.)
        -> Success verifying alpha area
        -> Boot Firmware

=============================================================
IN FIRMWARE
-------------------------------------------------------------
Update the firmware? (1 : yes, 2 : No! boot the firmware)? 2
-------------------------------------------------------------
2nd Verification(Verify beta area in firmware.)
        -> Fail
```

그림 11. 내장 함수 변조 시 USB 부팅 결과
Fig. 11 The result of booting USB device with modified functions in the firmware

을 진행한다.

펌웨어 업데이트 시뮬레이션은 세 가지 악의적인 업데이트 상황에 대해 시뮬레이션을 진행한다.

1. 무결성 검증 시뮬레이션

펌웨어가 변조되지 않은 USB 장치가 부팅 시 제안하는 무결성 검증 과정을 수행하는 과정은 그림 9와 같다. 펌웨어의 α영역을 부트로더에서 검증 (1단계 무결성 검증)한 뒤 펌웨어를 부팅하기 전에 업데이트 여부를 결정할 수 있다. 업데이트 과정이 아닌 부팅과정인 경우 펌웨어를 부팅하여 β영역의 무결성 검증 (2단계 무결성 검증)을 수행한다.

펌웨어를 변조하여 BadUSB 장치를 만드는 경우 그림 10과 그림 11과 같이 본 논문에서 제안하는 무결성 검증 방식에 의해 장치는 부팅되지 못한다. α영역 (고정 영역)인 디스크립터를 수정하는

```
system@ubuntu:~/Desktop/badUSB$ ./start_bootloader firmware
================================================================
IN BOOTLOADER
----------------------------------------------------------------
1st Verification (Verify alpha area in firmware.)
        -> Success verifying alpha area
        -> Boot Firmware

================================================================
IN FIRMWARE
----------------------------------------------------------------
Update the firmware? (1 : yes, 2 : No! boot the firmware)? 1
================================================================

[UPDATE PROCESS]

================================================================
IN BOOTLOADER
----------------------------------------------------------------
1st Verification (Verify alpha area in firmware.)
        -> Success verifying alpha area
        -> Boot Firmware

================================================================
IN FIRMWARE
----------------------------------------------------------------
Update the firmware? (1 : yes, 2 : No! boot the firmware)? 2
----------------------------------------------------------------
2nd Verification(Verify beta area in firmware.)
        -> Fail
```

그림 12. 업데이트 상황에서의 실험 결과
Fig. 12 The result of the experiment on update process

경우 그림 10과 같이 부트로더의 부팅 중 1단계 무결성 검증 과정에서 탐지되며, β영역 (가변 영역)인 내장 함수 영역을 수정하는 경우 그림 11과 같이 2단계 무결성 검증 과정에서 탐지되어 장치의 실행이 종료된다.

2. 업데이트 내용 시뮬레이션

업데이트 과정은 1단계 무결성 검증 과정을 통과한 펌웨어 내의 업데이트 함수를 통해 수행된다. 업데이트 시뮬레이션 과정은 다음 세 가지 상황에 대해 검증을 수행한다.

- Case 1. 서버 위장 : 펌웨어 업데이트를 수행하는 서버를 위장하여, 업데이트 내용을 진행한다.
- Case 2. 중간자 공격 (MITM) : 업데이트 과정의 중간에 위치하여, 업데이트 파일을 수정하여 장치에 펌웨어를 업데이트한다.
- Case 3. 펌웨어 강제 수정 : 호스트에서 재프로그래밍을 통해 펌웨어를 직접 수정한다.

본 논문에서 제시한 업데이트 내용은 정당한 서버인지를 확인하는 단방향 인증이다. 서버를 위장(Case 1)하여 업데이트 내용을 수행하려는 경우 서버의 개인키를 알지 못하기 때문에 USB 장치로부터 받은 내용을 확인하여 응답 메시지를 보낼 수 없다. 또한 서명 값을 생성할 수 없어 임의의 펌웨어를 업데이트할 수 없다.

중간자 공격 (Case 2)을 통해 업데이트 과정 중 4단계에서 임의의 변조된 펌웨어를 업데이트 시킬 수 있다. 하지만 해당 펌웨어는 그림 12와 같이 업데이트 후 부팅 시 본 논문에서 제안하는 무결성 검증 과정을 통과하지 못한다. 그림 12의 경우 중간자 공격을 통해 내장 함수 영역을 수정한 펌웨어가 부팅되는 결과이다. 이는 공격자가 부트로더를 수정하지 못하며, 개인키를 알지 못해 수정한 펌웨어에 대해 서명 값을 생성하지 못하기 때문에 2단계 무결성 검증 단계에서 장치의 실행이 종료된다.

펌웨어를 강제로 수정 (Case 3)한 경우 그림 10과 그림 11과 같이 부팅 시 제안한 무결성 검증 단계에서 탐지되어 장치 실행이 종료된다.

Ⅳ. 시뮬레이션 및 검증

BadUSB 취약점의 경우 본 논문에서 분석한 USB 플래시 드라이브뿐만 아니라 데이터를 저장할 수 있고 USB 규격을 사용하는 모든 장치에 적용가능하다. 또한 안티 바이러스에서 탐지하는 스토리지 영역이 아닌 펌웨어를 변조하기 때문에 제조사의 설계상의 연구와 대응책이 요구된다. 본 논문에서 연구하고 적용한 시뮬레이션은 실제 USB 장치가 아닌 리눅스 환경에서 수행하였기 때문에 추가적으로 실제 USB 장치에 적용해야할 필요가 있다. 또한 제안하는 대응 방안은 설계 단계에서 적용 가능한 방법이므로 이미 시중에서 사용하는 수많은 USB 장치에 적용하기 어려운 점이 있다. 하지만 USB 규격은 다양한 용도로 많이 사용되어 BadUSB 취약점에 대한 파급효과가 크기 때문에, 본 논문을 통해 USB의 보안에 대해 활발한 연구와 인식이 제고되기를 기대한다.

References

[1] S. L. Garfinkel, “USB Deserves More Support,” The Boston Globe, pp. C4, 1999.
[2] K. Nohl, J. Lell, “BadUSB-On Accessories That Turn Evil,” Black Hat USA, 2014.
[3] A. Caudill, B. Wilson, “Making BadUSB Work for you,” Derbycon, 1994.
[4] A. Caudill, B. Wilson, “Phison 2251-03 (2303) Custom Firmware & Existing Firmware Patches (BadUSB),″ Available: https://github.com/adamcaudill/Psychson/tree/master/firmware, 2014.

[5] Kaspersky Lab, BadUSB Attack Prevention, "https://help.kaspersky.com/keswin/10sp2/en-us/97194.htm".
[6] B.H. Choi, T.W. Suh, "A Security Program To Protect Against Keyboard-Emulating BadUSB," Journal of the Korea Institute of Information Security and Cryptology, Vol. 26, No. 6, pp. 1483-1492, 2016 (in Korean).
[7] Y.S. Lee, H.J. Lee, K.R. Lee, K.B. Yim, "Cognitive Countermeasures Against BAD USB," Proceedings of Springer International Conference on Broadband and Wireless Computing, Communication and Applications, pp. 377-386, 2016.
[8] J.E. Boritz, "IS Practitioners' Views on Core Concepts of Information Integrity," International Journal of Accounting Information Systems, Vol. 6, No. 4, pp. 260-279, 2005.
[9] J. Cho,. "Countermeasures for BadUSB Vulnerability," Journal of the Korea Institute of Information Security and Cryptology, Vol. 25, No. 3, pp. 559-565, 2015.
[10] Axelson, Jan, "USB Complete: the Developer's Guide," Lakeview research LLC, 2015.
[11] Bus Universal Serial, "Device Class Definition for Human Interface Devices (HID)," 2001.
[12] Logic. Beyond, "USB in a NutShell," 2014.
[13] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone, "Handbook of applied cryptograph," CRC press, 1996.
[14] Atmel Corporation, "Atmel AT02333: Safe and Secure Booltloader Implementation for SAM3/4", 2013.
[15] Y. Li, J. M. McCune, A. Perrig, "VIPER: Verifying the Integrity of PERipherals' Firmware.", Proceedings of the 18th ACM Conference on Computer and Communications Security, 2011.
[16] TEXAS INSTRUMENTS "Application Report : Secure In-Field Firmware Updates for MSP MCUs", 2015.
[17] Intel Company, "MCSr 51 Microcontroller Family User's Manual.", Intel, 1994.
[18] MSDN Microsoft, "Introduction to Code Signing," https://msdn.microsoft.com/en-us/library/ms537361(v=vs.85).aspx
[19] F. Vahid, T. Givargis, "Embedded System Design: a Unified Hardware/software Introduction," New York Wiley, 2002.

Jun-Ho Seo (서 준 호)

He is received B.S. degree in Department of Electronics and Information Engineering from Korea University in 2016. He is currently enrolled in Graduate school of information security at Korea University. His research interests include system security, network security, embedded system security and machine learning.

Email: tigerjh92@gmail.com

Jong-Sub Moon (문 종 섭)

He is received B.S. and M.S. degrees in computer science from Seoul National University, Korea, in 1981 and 1983, respectively. He also received the Ph.D. degree in computer science from the Illinois Institute of Technology, Illinois, USA, in 1991. He is currently a professor with Department of Electric and Information Engineering of Korea University, Korea. His current research interests include system security, network security, pattern recognition and neural network.

Email: jsmoon@korea.ac.kr