논문번호 12-04-17

Journal of IKEEE.Vol.16,No.4,389~394,December 2012
http://dx.doi.org/10.7471/ikeee.2012.16.4.389
107

보안 실행 환경을 위한 소프트웨어 기반의 암호화 패턴 부트스트랩

Software-based Encryption Pattern Bootstrap for Secure Execution Environment

최 화 순 *, 이 재 흥*★

Hwa-Soon Choi*, Jae-Heung Lee*★

Abstract

Most current systems have ignored security vulnerability concerned with boot firmware. It is highly likely that boot firmware may cause serious system errors, such as hardware manipulations by malicious programs or code, the operating system corruption caused by malicious code and software piracy under a condition of no consideration of security mechanism because boot firmware has an authority over external devices as well as hardware controls. This paper proposed a structural security mechanism based on software equipped with encrypted bootstrap patterns different from pre-existing bootstrap methods in terms of securely loading an operating system, searching for malicious codes and preventing software piracy so as to provide reliability of boot firmware. Moreover, through experiments, it proved its superiority in detection capability and overhead ranging between 1.5 % ~ 3 % lower than other software security mechanisms.

요 약

현재 대부분 시스템은 부트 펌웨어에 대한 보안 취약점이 무시되어 왔다. 부트 펌웨어는 하드웨어 제어 권한과 다른 외부 장치의 권한을 가지고 있기 때문에 보안 메커니즘이 고려되지 않은 상태에서는 악의적인 프로그램이나 코드에 의해 하드웨어가 제어되고 악의적인 코드에 의해 운영체제 손상, 프로그램 도용과 같은 심각한 시스템의 오류를 초래할 가능성이 높다. 본 논문에서는 부트 펌웨어에 대한 신뢰성을 제공하기 위해 악의적인 코드 탐색과 프로그램 도용방지, 운영체제의 보안 로드를 위하여 기존 부트스트랩 방식에 벗어난 암호화된 부트스트랩 패턴을 가지는 소프트웨어 기반의 구조적 보안 메커니즘을 제안한다. 또한 실험 결과를 통해 다른 소프트웨어 보안 메커니즘 비해 적은 1.5~3% 사이의 오버헤드와 검출능력의 우수함을 입증한다.

Key words :secure boot, embedded security, secure bootstrap, secure execution environment

* Dept. of Computer Engineering, Hanbat University
(010-3893-4237 , yic08c@nate.com)
Corresponding author
(010-3404-6954 , jhlee@hanbat.ac.kr)
※ Acknowledgment
This research was financially supported by the Ministry of Education, Science Technology (MEST) and National Research Foundation of Korea(NRF) through the Human Resource Training Project for Regional Innovation
Manuscript received Dec. 9, 2012; revised Dec. 17, 2012; accepted Dec 17. 2012

Ⅰ. 서론

반도체 기술의 끊임없는 소형화 및 집적화에 따라 점차 많은 기능을 요구하고 개발되고 있으며 시스템은 점차 실생활과 여러 산업 분야에 널리 쓰이고 있다. 하지만 이러한 기술이 발전함에 따라 기존의 시스템뿐만 아니라 임베디드 시스템에서도 다양한 형태의 보안에 대한 심각한 피해가 우려되고 있다. 임베디드 보안 공격은 기존에 데스크탑 컴퓨터 시스템이

소프트웨어적, 원격적인 위협이 많은 것과는 달리 공격 자체가 하드웨어로 직접적으로 공격할 수 있어 임베디드 시스템에서의 보안 기법이 많이 연구되어 가고 있다[1]-[3].

대부분의 임베디드 시스템 환경은 자원에 대한 많은 제약 사항을 가지고 있으며 임베디드 시스템에 있어 보안은 점점 기능적 보안에서 시스템 구조, 하드웨어 측면의 보안으로 고려되고 있다[4].

임베디드 기기 환경에서 공격자가 소프트웨어 구성을 변경하면 이를 검색하고 탐지하는 메커니즘이 필요하며 많은 연구가 진행되고 있지만 컴퓨터 시스템의 신뢰성의 문제는 여전히 미해결 문제로 남아 있다. 이 중에서도 부트 펌웨어에 대한 보안 취약점은 무시되어 왔다. 컴퓨팅 장치는 하드웨어가 초기화되고 운영체제가 로드되는 일련의 단계를 가지게 되고 이러한 부트 펌웨어는 하드웨어 대한 제어 권한을 가지고 있기 때문에 악의적인 프로그램이나 코드에 의해 하드웨어가 제어가 될수 있다. 이러한 악의적인 부트 펌웨어는 하드웨어 장치의 손상이나 운영체제 손상 또는 프로그램 도용이 일어나 시스템에 심각한 오류를 초래할 기능성이 높다.

임베디드 시스템 상의 공격은 보안 메커니즘과 암호 알고리즘의 약점을 이용한다. 이들의 약점 보안을 우회하거나 약화 시킨다. 임베디드 시스템에서의 취약성의 주요 과제는 다음과 같다[5].

• 복잡성 : 더 많은 코드는 버그와 보안 취약성의 가능성을 증가 시킨다. 임베디드 시스템이 여러 장치와 통합되고 여러 가지 기능이 추가됨에 따라 코드와 소프트웨어가 더욱 복잡해지고 있다. 이러한 문제는 버퍼 오버플로우와 같은 공격을 막을 수 없는 C와 C++의 사용으로 더욱 악화되고 있다.

• 확장성 : 자바나 닷넷과 같은 소프트웨어 시스템은 소프트웨어 확장이 쉽다. 운영체제는 드라이버와 모듈을 통해 확장성을 지원하며 임베디드 시스템은 이러한 확장성을 제공한다. 하지만 이러한 확장성은 시스템 보안으로부터 더 많은 위험을 야기 시킬 수 있다.

• 연결성 : 점점 임베디드화된 유비쿼터스 네트워크 시스템을 지원함에 따라 이 네트워킹의 연결성으로 인한 더 많은 위협이 존재한다.

현재 시스템은 이러한 보안기술의 문제의 해결을 위해 ASICs(Application-Specific Integrated Circuit devices), ASIP(Application-Specific Instruction Set Processors), FPGA 등의 기술이 주로 사용되어 물리적 보안 기법들이 많이 연구되어 가고 있으다. 대표적인 기술로는 TCG(Trusted-Computing-Group)의 TPM(Trusted-Platform-Module)와 AEGIS(Architecture EnGines for Information Security)가 있다.[2] 위 두 기술은 단일 칩 프로세서의 구조의 메커니즘을 제공하며 소프트웨어적 기법에 비해 암호화 보안 수준을 향상시켜 안정성이 장점으로 작용하고 있지만 기존 시스템에 적용하기가 힘들고 비용문제가 단점으로 작용하고 있다.

하드웨어 기반의 보안 단점을 보완하고자 나타난 소프트웨어 기반의 보안 메커니즘으로는 Shepherding와 MIDES(Middleware-based Intrusion Detection for Embedded System) 메커니즘이 대표적이다. 위 두 기술은 소프트웨어 기반으로 주로 커널 계층, 미들웨어 및 응용 프로그램계층에서의 암호화 기술이 주를 이룬다. Shepherding은 커널 단계에서 코드의 실행을 제한하는 임베디드 응용프로그램을 내장시켜 제어흐름을 모니터링한다[9]. 그리고 MIDES도 또한 Shepherding과 비슷하게 임베디드 시스템에서 주기적으로 프로파일 메서드를 호출하여 응용프로그램을 모니터링하고 그 결과를 정상적인 값과 비교하는 구조로 되어 있다[10]. 이러한 소프트웨어 기반의 보안 메커니즘은 하드웨어 기반 메커니즘에 비해 높은 오버헤드를 가지고 있으며 대부분이 기능적 암호화 기술로써 구현되어 있어 치명적인 하드웨어 오류를 발생 시킬 수 있다.

따라서 본 논문에서는 부트 펌웨어에 대한 신뢰성을 제공하기 위해 부트 펌웨어가 로드되는 과정에서 모듈이 자동 탐색과 보안 기능을 포함하여 악의적인 코드 탐색과 도용 방지, 운영체제의 보안 로드를 위한 새로운 방식의 부트스트랩 방식을 기술한다. 또한 임베디드 시스템에서의 최대의 관심사는 비용이다. 이러한 보안 메커니즘의 장치 비용 증가는 대량 생산에 있어 비용이 크게 증가하게 되고 기존 시스템에 적용이 힘들어지게 된다. 본 논문에서는 기존 시스템에 적용이 쉬우며 비용 증가의 문제가 없는 새로운 시스템 보안을 위한 소프트웨어 기반의 보안 기법을 제안한다.

II. 본 논문의 제안방법

1. 전체 구성

전체 구성은 그림1과 같이 크게 ROM 쓰기 부분과 RAM 로드 부분을 나누어져 있다.

ROM 쓰기 부분은 부트로더나 운영체제가 보관되어 있는 램에 프로그램을 쓰기하는 과정으로 일반적으로 이미지 파일을 롬 쓰기 한다. 이 과정에서 시스템은 개발자가 가지고 있는 고유한 키 값을 가지고

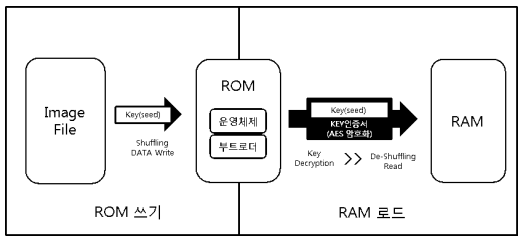

Fig. 1. Overall Diagram
그림 1. 전체 구성도

이미지 파일과 함께 쓰기 하게 되는데 이 키 값을 가지고 데이터들의 주소의 패턴을 뒤섞기하게 되고 뒤섞기의 시드(seed) 값이 키 값으로 쓰이게 된다.

RAM 로드 부분은 롬에 있는 부트로더를 부트스트랩하여 로드하는 과정으로 롬에 있는 데이터를 디코딩 하여 램에 로드하게 된다. 이 과정에서 디코딩을 하기 위해 개발자가 쓰기를 위해 쓴 키 값을 가지고 수행을 하게 되는데 AES(Advanced Encryption Standard)로 암호화된 키를 복호화 하여 나온 시드 값을 가지고 다시 원래의 순차적인 프로그램을 램에 로드하는 과정이다.

2. 부트 스트랩

부트스트랩은 컴퓨터의 전원을 키거나 리셋을 누르는 동작으로 시스템을 시동하는 일로써 롬에 저장되어 있는 소프트웨어를 램으로 로드하고 소프트웨어를 실행하는 일련의 행동이다. 본 논문에서 제안하는 부트스트랩 구성은 그림2에 나타내고 있다.

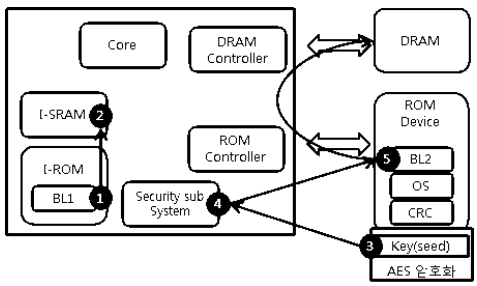

Fig. 2. Bootstrap Flowchart
그림 2. 부트스트랩 순서도

시스템의 부팅과정을 설명하면 다음과 같다.

(1) IROM에 있는 BL1 부트로더를 iSRAM에 로드

(2) 로드된 BL1은 저 레벨의 CPU 초기화, 부트 모드 설정이 실행

(3) BL2를 로드하기 위해 암호화되어 있는 데이터를 읽음

(4) 암호화 키를 보안 하드웨어 엔진인 보안 서브시스템을 통해 복호화 하고 시드값을 얻음

(5) 전 단계에서 얻은 시드값을 가지고 롬에 있는 BL2 데이터를 순차적으로 변환한 뒤에 램에 로드

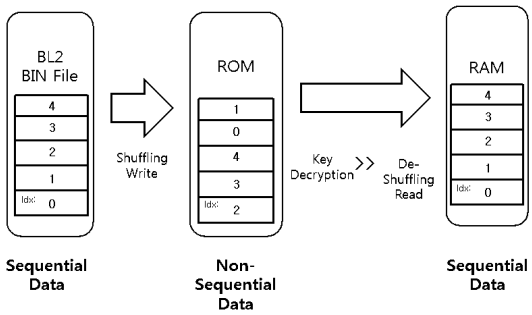

Fig. 3. Bootstrap Diagram
그림 3. 부트스트랩 구성도

그림3은 롬에서의 읽기, 쓰기 과정에서 패턴 뒤섞기를 이용하여 롬에 대한 악의적인 코드나 악의적인 실행을 방지하기 위해 순차적인 부트 펌웨어를 롬에서 비순차적으로 주소를 배열함으로써 프로그램 실행시에 이를 순차적으로 디코딩하는 과정에 문제가 있으면 온전한 프로그램이 실행이 안 되는 시스템이다. 프로그램을 비순차적으로 재배치하는 과정에서 패턴 뒤섞기를 사용하며 이를 온전하게 순차적인 프로그램으로 로드하기위해 디코딩을 사용하게 된다. 패턴 뒤섞기나 디코딩을 하기위해서는 시드값을 통해 실행될 수 있다.

3. 암호화 패턴

롬에 올릴 이미지 파일을 보안성을 향상 또는 강화하기 위해 패턴 뒤섞기 방법을 사용하였다. 이렇게 패턴 뒤섞기를 이용하면 순차적인 프로그램을 롬에 비순차적으로 배열하기 때문에 프로그램 도용과 코드 삽입을 막을 수가 있다. 패턴 뒤섞기 방법에서는 서로 다른 패턴을 가지는 것이 중요하다. 패턴이 분석이 되었다 하더라도 서로 다른 시스템에서는 이 패턴이 유지 되지 않도록 하기 위해서 고유한 키 값에 의해 패턴을 뒤섞는 것이 필요하다.

그림4에서 나타난 것과 같이 패턴 뒤섞기는 데이터를 뒤섞기하는 것이 아니라 주소를 뒤섞기하여 롬의 읽기, 쓰기를 보안하는 방법이다. 랜덤함수를 이용하여 주소를 뒤섞기하지만 이 랜덤주소는 시드값에 의해 일정한 패턴을 가지게 된다. 그리고 이러한 패턴이 중복되지 않아야 되는 것이 중요한데, 이 방법으로는 1부터 N의 사의의 숫자를 임의의 순열을 생성

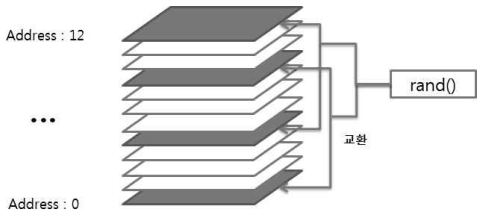

Fig. 4. Pattern Shuffling Structure
그림 4. 패턴 뒤섞기 구조

하고 재배치 한다.

이와 같은 단계가 완료가 되면 비순차적인 순열로 만들어 지게 된다. 순차적인 프로그램을 각 주소를 비순차적으로 바꿀 수 있으며 랜덤 함수의 시드 값 조정으로 각 각 다른 패턴의 비순차적인 결과를 보일 수 있다.

4. 공격 검출

악의적인 프로그램과 도용을 방지하기 위해 비순차적인 롬 구조를 가지는 부트스트랩은 키 값을 암호화하는 것은 방지만 할 수가 있고 악의적인 프로그램인지 코드가 변경된 것인지를 알 수 가 없다. 그래서 이러한 프로그램을 검출하는 부분이 추가적으로 필요하며 본 논문에서는 32비트 CRC(Cyclical Redundancy Check)를 사용 한다. 본 논문의 시스템에 적용하여 ROM에 저장된 악의적인 코드삽입으로 인한 CRC 에러 오류로 찾을 수 있고 또한 악의적인 프로그램으로 인한 디코딩 불일치로 인한 CRC 에러 오류로 찾을 수 가 있다.

모듈은 크게 CRC를 생성하는 과정과 CRC를 체크하는 과정으로 나누어 진다. BL2 프로그램에 대한 CRC를 생성하고 난 뒤 BL2를 ROM에 쓰기 할 때 CRC 코드 또한 패턴 뒤섞기한 데이터로 롬에 같이 쓰기를 한다. 이렇게 하여 롬에서 램에 로드하기 전에 CRC 체크를 하여 에러가 발생할 경우 악의적인 프로그램 도용 여부를 판단한다.

III. 실험 및 결과

본 논문의 실험 방법은 임베디드 시스템 환경에서 악의적인 코드의 변경의 검출 능력과 기존 시스템과의 오버헤드를 측정해 보았다.

실험 구성 환경은 다음과 같다.

- SoC : Samsung S5PV210
- OS : Linux 2.6.32
- Bootloader : U-boot 1.3.2

위와 같은 실험환경에서 악의적인 코드를 생성하기 위해 실험에서는 몇 가지의 명령어 조작을 통해 결함을 주입하였다. 결함 주입 방법으로는 소스 코드 또는 ELF(Executable & Linking Format)을 이용하여 파일에 직접적으로 결함을 주입하여 실행하였다.

Table. 1. Types of Malicious code
표 1. 악의적 코드의 유형

공격유형	설명
포인터 결함	메모리 명령의 주소를 변경
인터페이스 결함	전달된 매개 변수 대신 레지스터에 있는 기존 값을 사용
분기 결함	분기 명령을 삭제
루프 결함	루프 명령어의 종료조건을 반전
플립 결함	명령어 비트 플립
NOP 결함	명령어를 제거

표1에서와 같이 6가지 공격을 악의적인 코드의 유형의 코드 변환이라 가정하고 결함 코드를 랜덤으로 각 유형별로 1000번 발생시켜 실험하였다. 버퍼 오버플로우의 경우에 디버거 접근 같은 경우는 JTAG 에뮬레이터를 이용하여 실험 하였다.

Table. 2. Attack Detection Check
표 2. 공격 검출 체크

구분	악의적 코드	디버거 접근	형변환
보안 부트스트랩	○	×	○
AEGIS	△	×	○

표2에서와 같이 시스템은 악의적인 코드와 형변환에서는 모두 검출한 결과를 나타내었다. 하지만 버퍼 오버플로우는 악의적인 코드로 인한 버퍼 오버플로우는 감지하였지만 직접적으로 램에 접근하여 공격하는 오버플로우는 검출을 하지 못하며 디버거 접근은 프로그램 코드에 의한 것이 아니라 디버그 모드로 접근

하는 방식이기 때문에 모든 하드웨어 또는 소프트웨어 공격에 취약성을 실험 결과를 통해 알 수가 있었다. AEGIS는 어플리케이션에서 발생할 수 잇는 악의적인 코드와 버퍼오버플로우등을 검출하지 못하며 미들웨어 혹은 외부 메모리에 존재하는 공격에 대해서만 검출이 가능한 단점이 있다.

본 실험을 통하여 악의적인 프로그램과 악의적인 코드를 99% 검출이 가능하였다.

악의적 코드는 ELF파일을 조작하여 랜덤적으로 주소의 포인트를 바꾼다거나 명령어를 조작하여 결함을 주입하였다. 암호화된 키의 알고리즘이 우회되었거나 암호화 메커니즘의 신뢰성을 전제로 한 실험 결과이다. 기존 시스템과 보안 시스템의 적용시의 오버헤드를 측정 해보았으며 측정 결과는 그림5와 같다.

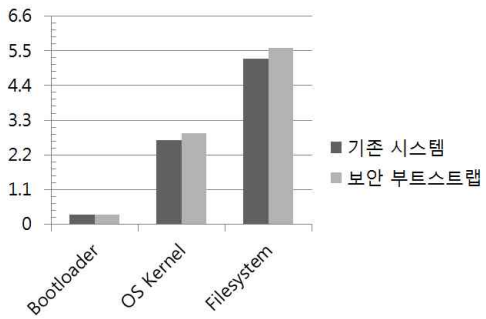

Fig. 5. Time Measurement Graphs of Encryption Bootstrap
그림 5. 보안 부트스트랩의 시간 측정 그래프

이와 같이 보안 부안 부트스트랩을 적용 시에는 PAGE당 읽기, 쓰기에 있어 표2와 같이 1.5%의 오버헤드율을 보였으며 시스템 부팅과정에 있어서 보안 부트 시스템의 적용 시에 전체적인 오버헤드율이 평균 3%로 시스템 적용 시에도 많은 오버헤드율을 보이지 않았다.

Table. 3. Overhead Compare of Security Mechanisms
표 3. 보안 메커니즘의 오버헤드 비교

	MIDES	Shepherding	보안 부트스트랩
오버헤드	2.5~6.9%	1.9~29%	1.5~3%

MIDES(Middleware-based Instruction Detection for Embedded System)[10]와 Shepherding[9] 과 비교해보아도 낮은 오버헤드율을 가지고 있으며 위 두 가지 다른 메커니즘은 검출의 단계를 제외한 오버헤드율을 나타내고 있다.

본 논문의 보안 부트스트랩은 부팅과정에서 딱 한번 실행되기 때문에 시스템에 많은 영향이 끼치지 않으며 운영체제와 파일시스템간의 어플리케이션 보안 측면까지 고려된다면 시스템의 오버헤드율은 단점으로 작용할 수 가 있다.

Ⅳ. 결론

본 논문에서는 임베디드 시스템에서 부트 펌웨어의 무단 도용과 악의적인 코드 삽입을 방지하기 위해 소프트웨어 기반의 보안 부트스트랩 방법을 제안하였다. 유비쿼터스 컴퓨팅 시스템으로 들어서게 되면서 여러 임베디드 기기들이 활용하게 되었고 보안이 취약한 임베디드 기기를 기반으로한 여러 주제들이 연구되어가고 있지만 아직도 보안에 대한 취약점이 남아있는 상태이다.

본 논문의 소프트웨어 기반의 보안 방법이나 다른 연구의 보안 방법은 언제나 취약점이 존재하며 이것을 해결해 나아가기 위해서는 프로세서의 구조적 측면이나 하드웨어와 소프트웨어의 결합된 솔루션이 구비되어야 한다. 향후 하드웨어와 소프트웨어 결합된 구조적 측면의 결합된 솔루션에 대해 연구할 계획이다.

References

[1] Trusted Computing Group, “TCG TPM Specification, Version 1.2, Revision 103,” https://www.trustedcomputinggroup.org/specs/TPM, 2007.

[2] W. A. Arbaugh, D. J. Farber, and J. M. Smith, “A Secure and Reliable Bootstrap Architecture,” in IEEE Symposium on Security and Privacy. IEEE, 1997, pp. 65 - 1.

[3] K. Shimizu, “The Cell Broadband Engine Processor Security Architecture,” http://www.ibm.com/developerworks/power/library/ pa-cellsecurity/, Apr. 2006.

[4] Paul Kocher, Ruby Lee, Gary McGraw, Anand Raghunathan and Srivaths Ravi, "Security as a New Dimension in Embedded System Design", ACM, pp. 753-760, June 2004.

[5] S. Rabi, A. Raghunathan, and S. Chakradhar. "Temper Resistance Mechanisms for Secure Embedded Systems", in Proc. int. Conf. ,Jan. 2004.

[6] Eric James Lorden. “A Secure Software Platform

for Real-Time Embedded Systems", Bradley Departm ent of Electrical and Computer Engineering Blacksbu rg, Virginia December 15, 2006
[7] Vulnerability notes database, CERT coordination center: http://www.kb.cert.org/vuls/, 2006.
[8] Black, Paul E. F, "Fisher - Yates shuffle", Diction ary of Algorithms and Data Structures, National Insti tute of Standards and Technology, Retrieved 2007-08 -09.
[9] Vladimir Kiriansky, Derek Bruening, Saman Am arasinghe. "Secure Execution Via Program Shepherdi ng." MIT Press, 2002, pp. 191 - 206.
[10] E. Naess, D. A. Frincke, A. D. McKinnon, and D. E. Bakken, "Configurable middlewarelevel intrusio n detection for embedded systems," International Wor kshop on Security in Distributed Computing Systems, vol. 02, pp. 144 - 151, 2005

BIOGRAPHY

Choi Hwa-Soon(Student Member)

2011 : BS degree in Computer Engineering, Hanbat National University.
2011 ~ Present : MS degree in Computer Engineering, Hanbat National University.

Lee Jae-Heung (Member)

1983 : BS degree in Electrical Engineering, Hanyang University.
1985 : MS degree in Electrical Engineering, Hanyang University.
1994 : PhD degree in Electrical Engineering, Hanyang University.
1989 ~ Present : Professor in Dept. of Computer Engineering, Hanbat University