Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.25, NO.6, Dec. 2015 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2015.25.6.1449

IoT 기기를 위한 경량의 소프트웨어 제어 변조 탐지 기법*

박 도 현,† 이 종 협‡
가천대학교

A lightweight detection mechanism of control flow modification for IoT devices*

Dohyun Pak,† JongHyup Lee‡
Gachon University

요 약

IoT환경에서는 소프트웨어의 안전성과 무결성이 중요하다. 하지만 IoT의 제한된 성능 때문에 소프트웨어 검증에 음영지역이 발생한다. 제한된 제어 변조 공격은 이러한 음영지역을 목표로, 대상 프로그램의 핵심 분기문을 변조하여 동작을 조작한다. 본 논문에서는 이러한 공격에 대응하여 프로그램에 marker를 직접 주입하고 확인함으로써 공격을 탐지하는 효율적인 기법을 제안한다.

ABSTRACT

Constrained IoT devices cannot achieve full coverage of software attestation even though the integrity of software is critical. The limited modification attacks on control flow of software aim at the shadow area uncovered in software attestation processes. In this paper, we propose a light-weight protection system that detects modification by injecting markers to program code.

Keywords: Software Attestation, IoT devices

I. 서 론

인터넷에 항상 연결되어 있는 IoT 장비들은 소프트웨어 업데이트를 통해 장비의 기능을 다양화할 수 있지만 이러한 업데이트가 오히려 IoT 장비에 대한 공격의 수단으로 이용되고 있다. 따라서 IoT 장비 내에서 실제 수행되고 있는 소프트웨어의 주기적인 검증이 중요하다. 하지만 IoT 장비의 제한된 성능 탓에 높은 통신량이나 연산량을 요구하는 검증 기법을 적용하기 어려운 실정이다.

성능이 제한된 기기의 소프트웨어 무결성을 검증(attestation)하기 위하여 다양한 방법이 제안되었다. 대표적으로 랜덤한 패턴을 통하여 소프트웨어 코드 자체나 메모리 영역에 대한 체크섬(checksum)을 계산하는 방법들[1,2]이 제안되었으나, 무결성 검증 과정에서 많은 반복연산을 필요로 하기 때문에 에너지를 제한하는 경우에는 음영지역이 발생하는 문제점을 가지고 있다[3]. 공격자는 이러한 음영지역을 목표로 하여 최소한의 변조를 통하여 소프트웨어의 동작을 유의미하게 변경하고자 하는 '제한된 제어 변조 공격'을 시도하게 된다. 이 공격은 소프트웨어 업데이트나 물리적 접촉을 통해서 수행될 수 있으며 핵심 분기문(key branch)에 대한 변조를 목표로 하는 기존 소프트웨어 크래킹(cracking)과 유사하게 나타난다[4].

Received(09. 03. 2015), Modified(10. 28. 2015), Accepted(11. 03. 2015)
* 이 논문은 2012년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2012R1A1A1044693)
† 주저자, dhmpak@gachon.ac.kr
‡ 교신저자, jonghyup@gachon.ac.kr(Corresponding author)

본 논문에서는 제한된 변조 공격에 대하여 효과적으로 대응하기 위한 기법을 제안한다. 제안하는 기법은 핵심 분기문 코드에 marker를 직접 주입하여 변조를 손쉽게 탐지할 수 있는 소프트웨어 바이너리 코드를 생성한다. 특히 바이너리 코드를 직접 수정하는 기존의 기법들의 한계점을 해결하기 위하여 소스 코드에 slot을 주입하고 바이너리 코드에서 주입된 slot에 marker를 적용하는 2단계의 접근방법을 제안한다.

II. 제한된 제어 변조 공격의 문제점

2.1 소프트웨어 검증의 음영지역

실행파일의 변조를 확인하기 위하여 프로그램 코드 자체에 대한 checksum을 수행하는 방식은 무선 센서네트워크와 같이 제한된 성능을 가진 환경에 적합하다[1,2]. Checksum 결과의 spoofing을 방지하기 위하여 랜덤하게 전체 영역을 순회하는 기법이 일반적으로 제안되었지만, coupon collector의 문제[5]에서 지적된 것처럼 대상이 n개의 checksum 넓이로 되어 있을 때 전체 영역을 커버하기 위하여 $O(n\ln n)$의 과도한 반복을 필요로 하는 문제를 가지고 있다. 즉 8 바이트 체크섬 연산을 이용하여 24 kB 크기의 프로그램 코드를 음영지역 없이 순회하기 위하여 25,752번의 체크섬 연산이 필요하게 되는 셈이다. 랜덤 순회방식의 에너지 비효율성 때문에 IoT 장비에서 에너지 보전을 위하여 검증의 빈도를 조절하거나 체크섬 연산의 횟수를 조절하게 하는데, 이 과정에서 검증의 음영지역이 발생하게 된다. [3]의 연구에서는 업데이트 이후 attestation의 비효율성을 지적하고 명령어의 중요도에 따라 순회하는 prioritized checksum 기법을 제안하였다. 본 논문에서는 제한된 변조 공격에 의한 집중하여, 공격자가 변조할 수 없는 marker 주입을 통한 효율적 탐지 기법을 제안한다.

2.2 제한된 제어 변조 공격

소프트웨어 크래킹과 같이 최소한의 변조를 통해 대상 소프트웨어의 동작을 변경하고자 시도하는 공격을 '제한된 제어 변조 공격'이라고 부른다. 주로 특정한 서브루틴(subroutine)의 실행 여부를 변경하는 것을 목표로 하며, 프로그램 코드내에서 핵심 분기문을 찾아 수정하는 방식으로 이루어진다. 조건 분기문인 경우에는 opcode를 직접 변조하여 조건을 뒤집거나 (x86에서 jg와 jle 명령어의 opcode는 1 bit 차이) 무조건 분기문인 경우에는 분기문을 nop으로 덮어씀(x86에서 jump의 경우 일반적으로 2바이트)으로서 분기를 무효화하는 변조 공격을 수행한다. 본 논문에서는 제한된 변조 공격을 4바이트 이내의 수정을 통해 프로그램 변조를 시도하는 공격으로 가정한다.

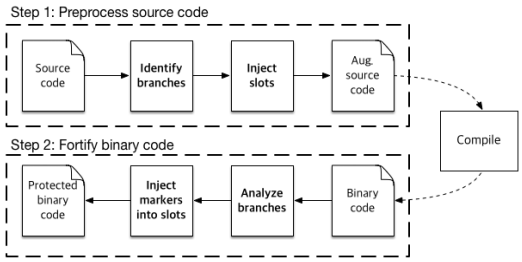

Fig. 1. Process of the proposed system

2.3 Binary rewriting을 통한 방어 기법의 어려움

많은 연구와 소프트웨어의 보호 기법들[6]이 바이너리를 직접 수정하는 binary rewriting의 접근방법을 사용해왔다. 일반적으로 이러한 방식에는 바이너리 코드에 명령어나 심볼들을 삽입하여 공격 탐지나 보호를 수행하는데, 새롭게 추가된 코드들 때문에 기존의 프로그램 코드의 위치(주소)가 밀리거나 틀어지게 되어 해당 프로그램 내의 많은 분기문들의 목표 주소값들을 재조정해야 하는 문제가 발생하게 된다. 특히 직접(direct) 분기문에 대해서는 모든 주소를 찾아내어 변경하면 되지만 jump table이나 함수 포인터와 같은 간접(indirect) 분기문에 대해서는 값을 찾아서 재조정된 주소값으로 변경해야 하는 어려움을 가지게 된다. 따라서 binary rewriting 기법들은 큰 효용에 비하여 제한적으로 사용되는 실정이다.

III. 경량의 제어 변조 탐지 기법

제한된 변조 공격을 방어하기 위하여 본 논문에서는 소스 코드와 바이너리 코드의 2단계의 과정을 통한 탐지 기법을 제안한다. IoT장비의 프로그램을 탑재할 때에는 검증자가 별도의 키를 생성하고 이후에

변조를 탐지할 수 있는 marker를 생성하여 주입한다. [Figure 1]은 제안하는 시스템의 marker 생성 및 주입 절차를 나타낸다. 우선 소스 코드에 대한 분석을 통하여 분기문이 존재하게될 곳들을 찾는다. 그리고 각 분기문 전후에 2단계에서 marker를 위한 slot을 소스코드에 삽입한다. 이렇게 처리된 소스코드가 컴파일 되고 난 이후에 바이너리 코드를 함수별로 분석하여 1단계 삽입된 slot에 marker값을 계산하여 주입한다. 이때 이미 준비되어 있는 slot만으로 binary rewriting이 한정되기 때문에 2.3절의 문제가 발생하지 않는다. 또한 이렇게 변형된 바이너리는 이후 검증자가 maker에 대한 확인만으로 변조를 판단할 수 있는 장점을 가지게 된다.

3.1 1단계 - 소스 코드 분석 및 slot 주입

1단계에서는 소스 코드의 프로그램을 분석하고 분기문이 발생하는 지역을 찾아 slot을 추가한다. Slot을 추가하는 경우는 다음과 같다.

1) 함수마다 시작위치에 function slot을 삽입한다.

2) 조건 분기문에 해당하는 명령어의 경우에는 (if문, while문, for문) 조건 분기에 해당하는 명령어 직전에 slot을 추가하고, 참과 거짓에 조건에 해당하는 코드의 시작점에 slot을 각각 추가한다.

3) 무조건 분기에 해당하는 경우에는 (goto문, 함수 직접 호출) 분기 전과 분기 후에 slot을 추가한다.

임의로 설정한 32 bits 크기의 marker를 담을 수 있는 slot을 위하여 다음의 asm 명령어를 이용한다.

```
asm("prefetchnta 0x01020304");
```

prefetchnta는 prefetch를 위한 명령어이지만 일반적으로는 기능을 수행하지 않고 side effect가 없어 CFI[6]와 같은 binary rewriting 기법에서 사용되는 명령어이다. 또한 asm 지시어를 이용하기 때문에 위의 명령어는 컴파일 후에도 바이너리에 그대로 남게 되어 2단계 marker의 값을 주입하는데 사용된다.

3.2 2단계 - 바이너리 코드에 marker 주입

2단계에서는 컴파일된 바이너리 코드를 함수별로 disassemble 후 marker를 slot에 주입한다. 공격자가 marker를 임의로 생성하는 것을 방지하기 위해서 프로그램마다 비밀키 k를 생성하고 marker를 slot에 주입할 때에는 k와 함께 hash한 값을 이용한다.

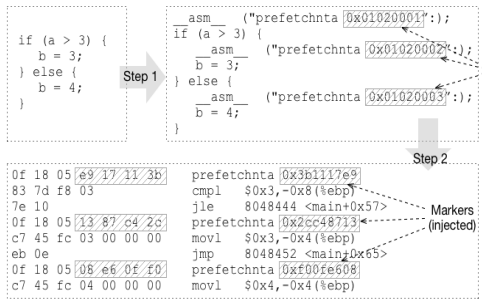

Fig. 2. Example of the proposed system

1) 함수 내의 분기문의 수(n)를 모두 세어 함수의 function slot에 표기한다. 이를 위해서 $H(k||n)|_{32}$의 값을 계산하여 slot의 위치에 덮어쓴다. 여기서 H()는 hash 함수, ||는 연접(concatenation), $|_{32}$는 최상위의 32 bit를 잘라내었(truncate)음을 의미한다. 이후 분기문을 순차적으로 돌며 marker를 주입한다.

2) i번째 분기문이 조건 분기문의 경우에는 분기 이전 slot에 marker로 기록($H(k||i)|_{32}$) 하고 참(true)분기에 해당하는 slot에는 i+1을 거짓(false)분기에 해당하는 slot에는 i+2의 값을 각각 기록한다. 즉 참분기 slot에는 $H(k||i+1)|_{32}$를 거짓분기 slot에는 $H(k||i+2)|_{32}$를 기록한다.

3) i번째 분기문이 무조건 분기문의 경우에는 분기문 이전과 이후 slot에 같은 값($H(k||i)|_{32}$) 을 기록한다. 하지만 function call의 경우에는 하나의 function을 다양한 위치에서 호출할 수 있으므로 해당되는 분기문의 이전/이후 slot들을 통일된 marker 값으로 표기한다.

[Figure 2]는 제안하는 기법이 적용된 예이다. 조건 분기에 해당하는 if문을 1단계에서 발견하게 되면 if문 이전과 then, else에 해당하는 블록의 시작위치에 slot을 삽입하여 컴파일한다. 2단계 과정에서는 컴파일된 바이너리의 slot위치에 조건에 맞게 계산된 marker를 주입한다. 검증자가 운용 중인 프로그램을 직접 검증할 때에는 저장하고 있는 k값을 이용하여 marker들의 정합성만을 빠르게 확인할 수 있다. 원격 검증의 경우에는 노드가 대상 프로그램의 marker와 주변에 대해서만 checksum을 계산하여 검증자에게 제공하는 것으로 수행이 가능하다. 또한

base station이 소프트웨어 업데이트에서의 변조를 탐지하기 위하여 전파할 소프트웨어에서 marker 생성에 사용한 k값을 미리 전달하여 노드가 전달받은 프로그램을 설치 이전 변조 유무를 효율적으로 판별할 수 있다.

IV. 구현 및 분석

4.1 구현

제안하는 시스템의 1단계는 CIL[7]을 기반으로 구현하였으며 2단계는 python을 이용하여 구현하였다. 소스 코드에서 특정 패턴을 찾아 분석 또는 변형을 할 수 있도록 하는 CIL의 visitor를 이용하여 소스 코드내 분기문 패턴에 대하여 slot을 추가하였다. 컴파일 이후에는 disassemble된 정보와 생성된 slot에 대한 바이너리 시퀀스를 찾아 marker를 삽입하였다.

4.2 안전성 및 성능 분석

2.2절에서 정의된 제한된 제어 변조 공격을 수행하는 공격자는 비밀값 k를 알지 못하기 때문에 정당한 marker를 생성할 수 없다. 또한 marker가 hash의 결과값이기 때문에 일부분을 수정하여 원하는 형태로 수정할 수 없다. 따라서 분기문의 조건을 뒤집거나 존재하는 분기문을 nop으로 삭제하는 공격은 광범위한 변조를 하는 경우에만 가능하다. 즉 4 byte 이내의 변조를 통하여 핵심 분기문에 대한 제어를 변경할 수 없기 때문에 제한된 제어 공격으로부터 안전하다.

제안하는 기법은 주입된 marker에 의해 프로그램 크기를 증가시킬 수 있다. 이를 측정하기 위해 NIST Software Assurance Reference Dataset (SARD)[8]의 Juliet Test Suite 중 control과 관련된 취약점의 category(Untrusted search path, Trapdoor, Logic time bomb, Dead code, Unchecked loop condition)에서 test case 18개의 소스 코드를 대상으로 실험을 수행하였다. 제안 시스템을 적용하였을 경우에는 평균적으로 46.4 KBytes에서 50.7 KBytes로 10.16%의 크기가 증가한 것을 확인할 수 있었다. 검증과정에서 변조 탐지를 위해서는 주입된 marker(prefetchnta에 해당하는 opcode는 제외)와 프로그램 내의 모든 분기문 명령어(e.g., jmp, je, jne, jl, je, js, ...)에 대한 연산이 이루어지는데 이부분은 평균 4.3 KBytes로 전체 프로그램 코드의 8.51%에 해당한다. 따라서 제안된 기법은 해당 부분에 대한 검증만으로도 전체 코드에 대한 제한된 변조 공격을 방어할 수 있어 효율성이 높은 기법이라 할 수 있다.

V. 결 론

본 논문에서는 IoT 환경이 제어 변조 공격을 효율적으로 탐지하기 위한 기법을 제안하였다. 제한된 변조 공격에 대한 방어를 위하여 소스 코드와 바이너리 코드에 적용하는 2단계 방법을 사용한다. 제안된 기법은 프로그램은 제어권에 집중하여 핵심 분기문에 대한 보호를 목적으로 하고 있지만 프로그램 전반에 대하여 확장이 가능하다.

References

[1] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla, "SWATT: Software-based attestation for embedded devices," Proceedings of IEEE Symposium on Security and Privacy. pp. 272 - 282, May. 2004.

[2] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla, "SCUBA: Secure Code Update By Attestation in sensor networks," Proceedings of ACM workshop on Wireless security, pp. 85-94, Sep. 2006.

[3] JongHyup Lee, LeeHyung Kim, and Taekyoung Kwon, "Energy-Efficient Software Integrity Checks to Build Secure Industrial Wireless Active Sensor Networks," IEEE Transactions on Industrial Informatics, (early access), Aug. 2015.

[4] Paul Craig, Mark Ron. Software Piracy Exposed - Secrets from the Dark Side Revealed, Andrew Williams, 2015.

[5] Gunnar Blom, Lars Holst, Dennis Sandell, Problems and Snapshots from

the World of Probability, Springer-Verlag, 1994.
〔6〕 M. Abadi, M. Budiu, Ú. Erlingsson, and J. Ligatti, "Control-flow integrity principles, implementations, and applications," ACM Transactions on Information and System Security, vol. 13, no. 1, pp. 4:1-40, Oct. 2009.
〔7〕 G. Necula, S. McPeak, and S. Rahul, "CIL: Intermediate language and tools for analysis and transformation of C programs," Compiler Construction, pp 213-228,, Springer, Jan. 2002.
〔8〕 NIST Software Assusrance Reference Dataset, http://samate.nist.gov/SARD/

〈저자소개〉

박 도 현 (Dohyun Pak) 정회원
1995년 2월: 연세대학교 수학과 졸업
2001년 5월: 미국 아이오와주립대 산업공학과 석사
2005년 4월: 미국 미시간대학교 금융공학 박사 (산업공학과)
2005년~2008년: 한국투자증권 퀀트 팀장
2008년~2011년: 하나대투증권 파생운용 팀장
2011년~2012년: IBK 투자증권 파생담당 이사
2012년~2013년: 미래에셋증권 파생상품운용팀장
2015년 3월~현재: 가천대학교 금융수학과 부교수
〈관심분야〉 금융공학, 금융보안

이 종 협 (JongHyup Lee) 종신회원
2002년 2월: 연세대학교 기계전자공학부 졸업
2004년 2월: 연세대학교 컴퓨터과학과 석사
2009년 8월: 연세대학교 컴퓨터과학과 박사
2009년~2012년: 미국 카네기멜론 대학교, CyLab, Postdoc. 연구원
2012년~2015년: 한국교통대학교 소프트웨어학과 조교수
2015년 3월~현재: 가천대학교 금융수학과 조교수
〈관심분야〉 금융보안, 소프트웨어 보안