인증코드를 통한 프로그램 무결성 보장에 관한 연구

조신영*, 박민우*, 정태명**
*성균관대학교 전자전기컴퓨터공학과
**성균관대학교 정보통신공학부
e-mail: {sycho, mwpark}@imtl.skku.ac.kr
tmchung@ece.skku.ac.kr

A Study on Program Integrity with Authentication Code

Shin-Young Cho*, Min-Woo Park*, Tai-Myoung Chung**
*Dept. of Electrical and Computer Engineering, Sungkyunkwan Univ.
**School of Information Communication Engineering, Sungkyunkwan Univ.

요 약

오늘날 인터넷의 발달과 스마트 기기들의 발달로 인해 우리의 일상생활 깊숙이까지 정보화가 이루어졌다. 최근 스마트폰을 사용하는 사람들의 수가 1,000만 명이 넘어서고 있으며 스마트폰을 통해 필요한 프로그램을 다운받는 수도 급격하게 증가하고 있다. 그런데 프로그램을 제작 후 인터넷을 통해 배포할 시 보안상 안전하지 않다는 문제가 있다. 악의적인 제 3자가 배포 중인 프로그램에 악성코드를 삽입하여 악성코드도 같이 배포되도록 할 수 있기 때문이다. 이를 대비하기 위해 프로그램의 무결성을 보장하는 PAC(Program Authentication Code)을 제안한다. PAC은 기존 악성코드 정보를 가진 데이터베이스가 없어도 프로그램에 악성코드가 삽입되는 것을 탐지할 수 있다. 또한 해시함수를 사용하여 고정된 길이의 인증코드를 생성함으로 네트워크로 전송하거나 한정된 메모리에서 보관하기에 용이하다.

1. 서론

오늘날 인터넷의 발달로 인해 우리는 시간과 장소에 구애받지 않고 인터넷을 사용할 수 있게 되었다. 최근 스마트폰이 등장하면서 일상생활 안에서 인터넷 활용이 더욱 다양해졌다. 스마트폰의 사용자는 20009년 11월에 아이폰이 국내에 출시된 것을 계기로 본격적으로 확산되기 시작했다. 아이폰이 처음 출시 될 때만해도 스마트폰 가입자 수가 80만 명에 불과 했었다. 하지만 2010년부터 가입자 수가 급격한 상승세를 보이더니 2011년 초에는 1,000만 명을 넘어서고 있으며, 2012년 말에는 2,000만 명을 넘어설 것으로 전망하고 있다[1].

스마트폰의 사용자 수가 급증함에 따라 스마트폰을 통해 할 수 있는 다양한 서비스에 대한 사용자들의 요구도 증가하였다. 이러한 요구에 따라 어플리케이션 개발자들은 교육, 게임, 사무 등 다양한 영역의 앱(App, Application의 줄임말)을 개발하여 애플에서는 앱 스토어, 안드로이드는 안드로이드 마켓에 업로드 하며, 사용자들은 마켓에 올라와 있는 앱을 인터넷을 통해 필요에 따라 다운 받을 수 있다. 최근 통계에 따르면 2011년 3월 기준으로 앱 스토어와 안드로이드 마켓에 약 70만 개의 앱이 등록되었다고 한다[1]. 또한 앱의 다운로드 건 수가 2011년 초에 앱 스토어는 100억 건을 돌파 하였고, 안드로이드마켓은 27억 건의 다운로드 수를 기록할 만큼 앱의 다운로드 양이 급증하고 있다.

사용자는 앱 스토어 혹은 안드로이드 마켓을 통해 프로그램을 다운 받을 경우, 보안상 위험에 노출될 수 있다. 프로그램이 제작된 후 배포과정에서 악의적인 제 3의 공격자가 프로그램에 악성코드를 삽입할 수 있기 때문이다. 공격자는 인터넷을 통해 전송 중인 프로그램에 악성코드를 삽입하거나 프로그램을 저장하고 있는 서버를 감염시켜서 악성코드를 삽입 할 수 있을 것이다. 악성코드에 감염된 프로그램이 사용자에게 전달되고, 사용자가 아무 의심 없이 프로그램을 실행 하게 되면 악성코드는 사용자의 컴퓨터에 설치되어 악의적인 행위를 하게 된다.

프로그램에 삽입된 악성코드는 백신 프로그램의 검사를 통해 탐지될 수 있다. 하지만 신종 악성코드의 경우 해당 악성코드에 대한 정보가 데이터베이스에 없기 때문에 악성코드를 탐지하는 데에 어려움이 있다. 최근 악성코드와 악성코드 변종들이 하루에도 몇 천 개씩 새로 등장하고 있어서 해당 악성코드와 관련 정보의 양도 늘어나고 있다. 이로 인해 데이터베이스의 크기도 커지게 되었다. 이러한 이유로 오용탐지에 기반하고 있는 방식은 한계가 있다.

프로그램의 안전한 실행을 보장하는 방법들 중 Proof Carrying Code(이하 PCC)가 있다. 최근에도 연구되고 있는 PCC는 제 3자에 의존하지 않고도 프로그램의 안정성을 보장할 수 있다는 장점이 있지만 프로그램의 안정성 확인을 위해 프로그램 제공자 측에서 제공하는 정보인 증명(Proof)의 크기가 너무 커서 실용화하기에는 비효율적이

라는 문제가 있다. 이러한 증명의 크기를 줄이기 위한 노력이 지금까지 계속되고 있다[2][4][5][6].

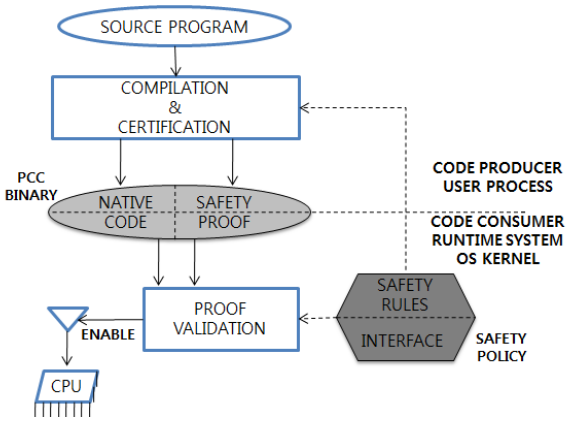
본 논문에서는 증명의 크기 문제와 오용 탐지 방식으로 인한 문제점을 해결하기 위해 비정상 탐지 방식을 기반으로 하여 프로그램의 무결성을 보장하는 메커니즘을 제안하려고 한다. 제안하는 메커니즘은 증명의 크기를 축소하기 위해 해시함수를 사용하며 악성코드가 삽입되는 위치에 대한 연구를 거쳐 어떠한 정보를 사용하여 증명을 생성할 지를 언급한다. 우리가 제안하는 메커니즘은 앱 스토어 및 안드로이드 마켓을 사용하는 사용자들이 인터넷을 통해 프로그램을 다운 받을 경우, 전송 도중에 악성코드가 삽입되었더라도 이를 빠르게 탐지할 수 있어 효율적으로 사용될 것으로 기대된다.

2. 관련 연구

본 장에서는 제안 메커니즘 제안에 앞서 기존 연구인 PCC에 대한 설명을 한다.

2.1 PCC(Proof Carrying Code)

Proof Carrying Code(이하 PCC)는 호스트 시스템이 검증할 수 없는 외부에서 제공하는 프로그램을 호스트 시스템에서 안전하게 수행할 수 있는가를 알 수 있도록 해주는 메커니즘이다. 다음 (그림 1)은 PCC의 전체적인 개요이다[2][3].



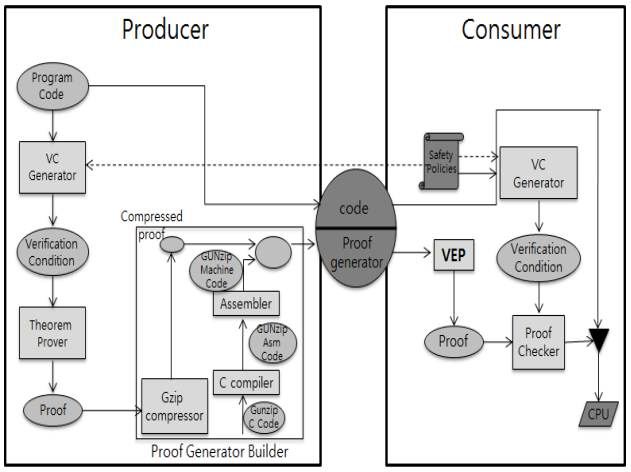
(그림 1) Proof-Carrying Code 구조

프로그램을 생성하여 상대방에게 전송하는 사람을 코드 생성자(Code Producer)라고 하고 프로그램을 전송받아 실행하는 사람을 코드 소비자(Code Consumer)라고 명명한다. 코드 소비자는 안전정책(Safety Policy)을 코드 생성자에게 제공하고 코드 생성자는 이를 기반으로 프로그램 코드의 증명(Proof)를 생성하여 원시코드에 붙여서 코드 소비자에게 보낸다. 그러면 코드 소비자는 증명 검증(Proof Validation)을 통해서 검증하고 이상이 없다고 판단되면 프로그램이 안전하다는 것이 증명되어 코드 소비자 측에서 프로그램을 실행하게 된다[10][11][12].

PCC는 실시간으로 프로그램의 실행을 체크하지 않아도 되서 비용 및 동작 측면에서 효율적이고, 공개키 암호기법과 같이 제 3자에 의존하지 않아도 되서 보다 안전하다는 장점이 있다. 하지만 모든 프로그램 코드를 일차 술어 논리를 기반으로 하는 논리 프로그래밍을 이용하여 증명(Proof)을 생성하므로 그 크기가 원시코드보다 커지게 된다. 크기가 큰 증명은 프로그램을 네트워크를 통해 전송하거나 보관할 때 매우 비효율적이라는 단점이 있다.

2.2 Extended PCC

PCC에서 증명(Proof)의 크기를 줄이기 위해 증명을 압축하거나 다른 논리 프로그래밍 방식을 사용하여 증명을 생성하는 등 여러 가지 방식이 연구되었으나 증명의 크기를 효과적으로 줄이지 못했기 때문에 Heidar Pirzadeh[6]는 Extended Proof Carrying Code(이하 EPCC)를 제안한다. EPCC는 증명대신에 증명을 생성하는 프로그램을 제작하여 보내는 방식으로 증명의 크기 문제를 해결한다. 다음 (그림2)는 EPCC의 전체적은 framework이다.



(그림 2) EPCC의 Framework

증명을 생성하고 검증하는 과정은 원래의 PCC와 같다. EPCC에서 추가된 블록은 증명 생성기 제작기(Proof Generator Builder)와 VEP(Virtual Machine for EPCC)이다. 증명 생성기 제작기에서는 Kolmogorov complexity에 따라 증명을 증명 생성기(Proof Generator)로 만든다. EPCC에서는 증명 대신에 증명 생성기를 원시 코드에 붙여서 코드 소비자에게 전송하면, 코드 소비자는 안전성이 보장된 가상 머신인 VEP에서 증명 생성기를 실행시켜 실제 증명을 생성한다. VEP은 자체적으로 설계한 보안 시행(Security Enforcement)을 이용하여 증명 생성기도 프로그램임으로 발생할 수 있는 문제를 해결한다[7][8][9].

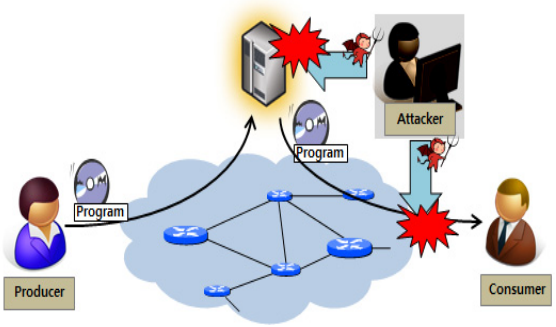
EPCC는 증명의 크기를 줄임으로써 증명 보관 문제를 해결하고, 네트워크 통신 시에 발생하는 어려움을 해소하였으며, 코드 소비자 측에 존재하는 VC Generator와 Proof Checker으로 구성된 TCB(Trusted Computing

Base)의 크기를 줄이는데 기여하였다. 하지만 프로세스가 매우 복잡하다는 단점이 있다.

3. Program Authentication Code(PAC)

본 논문에서는 프로그램이 제작된 후 배포 과정에서 제 3자의 악의적인 사용자에 의해 변조되지 않음을 증명하여 안전한 설치를 할 수 있도록 하기위한 메커니즘을 제안한다.

제안하는 PAC이 적용되는 환경은 다음 그림과 같다. 프로그램이 배포된 후에 인터넷을 통해 전송되는 도중이나 프로그램이 보관되는 서버가 감염된 경우 등의 요인으로 인해 공격자에 의해서 프로그램에 악성코드가 삽입될 수 있다. 이러한 프로그램은 사용자의 컴퓨터에서 실행되어 오버플로우나 루트 권한을 비정상적으로 얻는 등의 악성 행위를 수행할 수 있다. 이러므로 프로그램의 무결성을 보장하는 메커니즘이 필요하다.

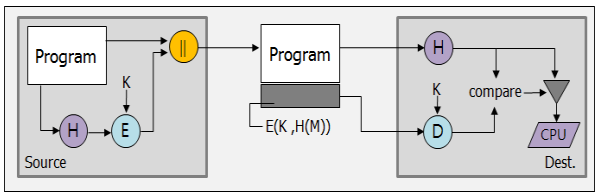


(그림 3) 메커니즘 적용 환경

오용탐지 기법 기반의 메커니즘은 처음 등장한 악성코드의 경우 탐지하기 어렵고, 이미 탐지되었더라도 해당 악성코드에 대한 정보들이 데이터베이스에 모두 저장 되어 있어야 탐지 할 수 있음으로 시간이 지날수록 데이터베이스의 크기가 커진다는 단점이 있다. 본 논문에서 제안하는 PAC은 비정상탐지 기법을 기반으로 하여 근본적으로 악성코드가 프로그램에 삽입되는 것을 탐지하여 프로그램의 무결성을 보장한다.

비정상탐지 기법 중 PCC가 있는데, PCC의 경우에는 탐지를 위한 증명의 크기가 너무 커서 비효율적인 반면에 PAC은 증명 연산 시 해시 함수를 사용하여 증명의 크기를 최소화 하며 증명 생성 시 사용하는 정보들을 축소화 하였다. 그래서 스마트폰과 같은 모바일의 제한된 용량에서도 증명의 보관이 가능하며 네트워크를 통해서 프로그램과 증명을 전송 할 때에도 효율적이다.

프로그램을 처음 제작하는 제작자는 신뢰할 수 있다고 가정하며, 제작자는 코드 상의 특정 위치의 정보들로 해시 연산 하여 PAC을 생성한다. 프로그램을 배포하는 배포자도 신뢰할 수 있다고 가정하며, 이에 해당하는 것이 앱 스토어 및 안드로이드 마켓이다. 인증코드를 생성할 때는 송신자와 수신자만 알고 있는 비밀코드 값을 함께 연산에 사용하여 제 3자가 똑같이 만들 수 없도록 한다. 이렇게 생성된 인증코드와 프로그램을 수신자에게 전송하면 수신자는 비밀코드 값과 함께 같은 연산을 수행하여 전송받은 인증코드와 비교하게 된다. 비교하여 올바를 경우 프로그램을 실행하게 되고 그렇지 않으면 실행하지 않는다. 다음 그림은 제안하는 메커니즘의 동작과정이다.



(그림 4) 제안하는 메커니즘

PAC에서 해시함수의 입력 값으로 사용하는 정보들과 메커니즘의 요구사항은 다음에 설명한다.

2.1 안전 요소

프로그램의 특정 정보들을 해시 연산하여 인증 코드를 생성하는데, 이러한 특정 값들을 안전 요소(Safety Factor)라고 부른다. PCC에 관한 논문[2][5][6]을 보면 안전 정책(Safety Policy)을 지정하고 있는데, 여기서 안전성을 체크하는 정보들은 다음과 같다. 우리는 이러한 정보들에서 제안하는 메커니즘의 안전 요소를 도출할 것이다.

<표 1> 안전 요소

구분	Safety Factor
Proof-Carrying Code	memory safety forward, backward, branches, loops encoding control over resource usage preservation of data-abstraction boundaries
An Extended Proof-Carrying Code Framework for Security Enforcement	Code Size, Stack Size, Heap Size, Length of Execution, Program counter, Number of operand, Type of operands, Legal range of operands, Legal code destination, Legal stack destination, Stack overflow, Heap overflow
Foundational Proof-Carrying Code	machine state (register band + memory) general registers floating-point registers, condition codes, program counter

2.2 Requirement

PAC은 스마트 폰과 같이 성능 면에서 제한된 환경에서도 사용 가능해야 한다. PAC은 위조 불가능해야 하며 안전해야 한다. 이러한 PAC의 요구사항은 다음과 같다.

○ Simplicity : 자원이 제한되어 있는 모바일 환경을 기반으로 하기 때문에 속도적인 측면에서 효율성을 위해 인증코드를 생성하는 과정과 안전요소의 수를 간소화 시켜야 한다. 이때 전체가 아닌 특정 일부 보안요소만을 사용해도 전체를 사용하는 만큼 안전하다는 것을 증명해야 한다.

○ Solidity : 다른 프로그램 중 똑같은 해시 인증 코드를 만들 수 없도록 충동을 최소화해야 하며 인증코드가 견고해야 한다. 이를 위해 여러 해시 함수들 중 가장 안전한 해시 함수를 찾고 해당 해시 함수를 사용해야 한다.

○ Security : 송신자와 수신자는 인증코드를 만들기 위해 특정한 키 값을 공유하게 되는데, 이때 키는 안전하게 분배해야 하며 그 키 값은 안전하게 보관해야 한다.

4. 결론 및 향후 연구

본 논문에서는 프로그램의 인증코드를 제공함으로써 신뢰할 수 없는 인터넷을 통해 전송되는 프로그램을 호스트 시스템에서 실행해도 안전하도록 프로그램의 무결성을 보장해주는 PAC(Program Authentication Code)을 제안하였다. 제안하는 메커니즘은 스마트폰과 앱 스토어 및 안드로이드에서 발생한다고 상황을 가정하였고, 배포자 및 앱 제작자는 신뢰할 수 있다고 가정하였다. PAC은 해시함수를 사용하여 프로그램의 안정성을 증명하는 값의 크기가 비효율적으로 커지는 것을 해결하였으며, 비정상탐지에 기반하여 악성코드가 삽입되는 것을 근본적으로 탐지할 수 있도록 하였다.

향후 연구 주제는 제안하였던 요구사항들에 대해서 적당한 해시함수는 무엇인지, 특정 일부 요소만 사용해도 안전한지에 대한 성능평가 및 구체적인 프로토타입 설계가 필요하다.

ACKNOWLEDGEMENT

본 연구는 지식경제부 및 한국산업기술평가관리원의 산업융합원천기술개발사업(정보통신)의 일환으로 수행하였음. [KI001810039260 , 개인 및 기업 맞춤형 서비스를 위한 개방형 모바일 클라우드 용 통합개발환경 및 이기종 단말-서버 간 협업 기술 개발]

참고문헌

[1] 2011 국가정보화 백서 pp40

[2] George C. Necula, “Proof-Carrying Code”, POPL ‘97, 1997

[3] Geroge C. Necula, “Safe Kernel Extensions Without Run-Time Checking”, 1996

[4] George C. Necula, Rahul, S.P., “Oracle-based checking of untrusted software”, 2001

[5] Appel, A.W., “Foundational proof-carrying code”, 2001

[6] Heidar Pirzadeh, Danny Dube, and Abdelwahab Hamou-Lhadj, "An Extended Proof-Carrying Code Framework for Security Enforcement", 2010

[7] Li, M., Vitnyi, P.: An Introduction to Kolmogorov Complexity and its Applications, vol. 3. Springer Publishing Company, Heidelberg(2008)

[8] Heidar Pirzadeh, Danny Dube, “VEP: a Virtual Machine for Extended Proof-Carrying Code”, VMSEC ‘08, 2008

[9] Heidar Pirzadeh, Danny Dube, and Abdelwahab Hamou-Lhadj, "An Extended Proof-Carrying Code Framework for Security Enforcement", 2010

[10] George C. Necula, "The Design and Implementation of a Certifying Compiler",

[11] Andrew W. Appel, Edward W. Felten, Zhong Shao, "Scaling Proof-Carrying Code to Production Compilers and SEcurity Policies", 1999

[12] Goerge C. Necula, Peter Lee., "Safe, Untrusted Agents using Proof-Carrying Code"