2000년 한국정보처리학회 추계 학술논문발표집 제7권 제2호

분기기법에 기반한 개선된 Proof-Carrying Code에 관한 연구

이정은*, 정태명*
*성균관대학교 전기전자 및 컴퓨터공학부
e-mail:{jelee, tmchung}@rtlab.skku.ac.kr

A Study on Enhanced Proof-Carrying Code based on Branching Method

Jung-Eun Lee*, Tai-Myoung Chung*
*School of Electrical and Computer Engineering
Sungkyunkwan University

요약

네트워크가 복잡해지고 에이전트를 이용한 분산 시스템이 증가함에 따라 악성코드의 수와 종류도 현저한 증가 추세를 보이고 있다. 코드의 안전한 실행을 위해서는 그 동안의 수동적인 대응책을 넘어선 능동적인 방법이 요구되는 시점에서 Proof-Carrying Code라는 능동적인 대응책이 제시되었지만 현실적인 문제에 봉착해 있다. 따라서 본 논문에서는 분기를 기반으로 한 Proof-Carrying Code의 변환을 시도해 기존의 Proof-Carrying Code 방식보다 효율적이고 능동적으로 코드의 안전한 수행을 보장하는 방법을 제시한다.

1. 서론

컴퓨터 사용의 보편화와 인터넷 어플리케이션이 증가함에 따라 컴퓨터 시스템은 사용자나 시스템 관리자, 프로그램 작성자의 의도에 반(反)하도록 강요하는 요소들에 쉽게 노출되어 있고 그 범위 또한 방대해져 피해가 날로 심각해지고 있다. 이러한 현 추세에서 네트워크 상에 유입된 실행 코드의 변형 가능성이 예전에 비해 명시적으로 높아졌다는 사실을 부인 할 수 없다[10]. 비합법적 요소들에 대한 코드의 노출은 시스템의 노출로 이어지며, 이는 자원의 손상, 파괴 등의 크고 작은 피해를 초래하기 때문에 비합법적 수행의 시도를 검출해내어 악성코드에 따른 피해에 대응하여야 할 필요성이 크게 요구된다. 특히 악성코드의 탐지는 액티브 네트워크(Active Network)환경과 에이전트 기반의 시스템이 점차로 현실화됨에 따라 더욱 중요한 기술이 될 전망이므로 이에 대한 연구가 활발히 이루어지고 있다[13].

그 동안 이루어졌던 악성코드의 검출에 대한 많은 시도와 방법들은 대부분 이미 피해를 준 사례가 있는 악성코드의 분석을 통해 획득한 특성을 기반으로 한 수동적인 방법들이었다[11,12]. 그러나 악성코드의 수가 기하 급수적으로 증가하고, 그 방법 또한 다양해질 뿐만 아니라 지능화 되고 있는 추세이기 때문에 기존의 수동적인 방법으로는 새롭게 변형된 악성코드를 검출하여 피해에 대응하기에는 역부족인 것이 현실이다.

이러한 현실을 극복하기 위해 대두된 방법들도 급변하고 있는 현실에 대응하지 못하는 한계에 부딪힌다[12]. 수동적인 방법으로는 가속화되고 있는 크고 작은 피해를 막을 수 없는 시점에서 능동적인 악성코드의 검출 방법이 더욱 요구된다. 따라서 본 논문에서는 코드의 안전한 실행을 능동적으로 보장하는 Proof-Carrying Code(PCC)에 대한 설명과 여기에 나타나는 문제점에 대한 해결책의 일환으로 EPCC(Enhanced PCC based on Branching)를 제시한다.

2장에서는 기존의 PCC에 대한 설명과 PCC의 문제점에 대해 분석하고 3장에서는 좀더 효율적인 PCC인 EPCC를 제시한다. 4장에서는 본 논문의 결

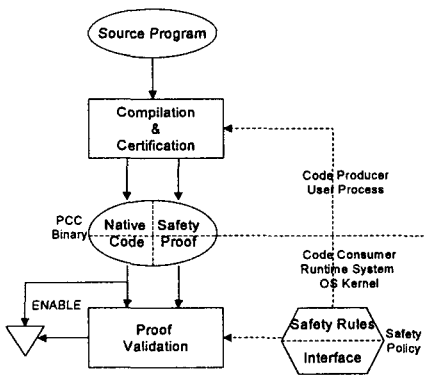
론과 향후 연구 계획을 기술한다.

2. 관련연구

2.1 Proof-Carrying Code

PCC는 호스트에서 안정성이 확인되지 않은 코드 실행의 안정성 여부를 결정하도록 하는 소프트웨어 메커니즘이다. 이 메커니즘은 코드 작성자(Code Producer)가 PCC로 불리는 특별한 형태로 된 바이너리를 생성하도록 규정한다[1].

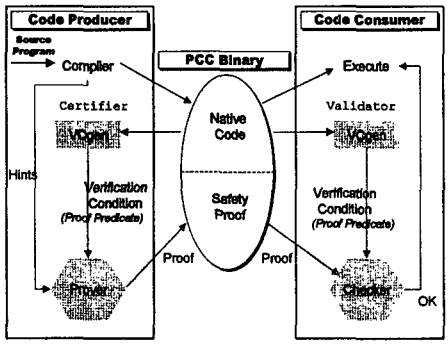
[그림 1]은 PCC를 생성하는 전형적인 메커니즘을 나타낸다. 전체적인 흐름은 안전 정책(Safety Policy)을 중심으로 이루어진다. 이 정책은 외부 프로그램을 안전하게 실행할 수 있는 조건을 코드 실행자(Code Consumer)에 의해 구체적으로 정의되고 표면화된다. 안전 정책은 인증된 수행과 그에 관련된 안전 조건들을 기술하는 안전 규칙(Safety Rules)과 코드 실행자와 외부 프로그램간의 호출 프로시져를 규정하는 인터페이스(Interface)의 두 요소로 구성된다[1].



[그림 1] Proof-Carrying Code

[그림 1]의 PCC 시스템에서 코드 실행자는 안전 정책을 정의하고 코드 작성자는 소스 프로그램을 컴파일한 원시 코드(Native Code)와 안전 정책에 준수하여 생성한 안전 증거(Safety Proof)로 PCC 바이너리를 구성한다. PCC 바이너리를 전송 받아 실행시킬 코드 실행자는 PCC의 원시 코드로부터 PCC가 위조되지 않음을 확인하고 안전 증거를 검사하여 안정성을 증명함으로써 코드의 안전한 수행을 가능하도록 한다[1,2,4,5].

이러한 수행을 가능하게 하는 인증 컴파일러(Compilation&Certification)의 구조를 고려하여 PCC를 재구성하면 [그림 2]와 같은 메커니즘을 얻을 수 있다. VCgen(Verification Condition Generator)에 의해 안전 정책에서 정의한 속성을 생성해서 최종적으로 안전 증거를 도출해내는 것이다. 안전 증거는 전형적으로 소스 프로그램의 형검사(TypeChecking)나 자료 흐름(Data flow)의 속성에 기반을 두고 있기 때문에 안전 증명 모듈(Prover)에 제공되는 정보(Hints)는 형과 자료 흐름에 대한 것이다. 따라서 PCC는 어떤 특별한 컴파일러의 사용을 요구하지는 않는다[2, 6].



[그림 2] Proof-Carrying Code 사용 방법

2.2 Proof-Carrying Code 평가

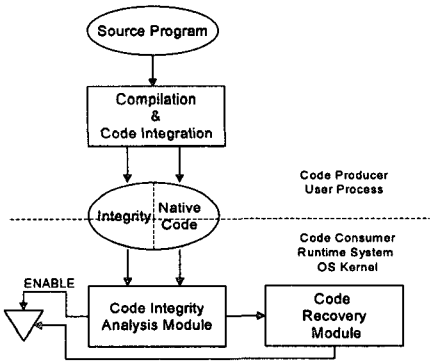
코드의 안정성 확인을 위해 제시된 PCC는 안전 정책을 정의하는데 특별한 제한을 두고 있지 않기 때문에 총체적이다[4]. 뿐만 아니라 전체적으로 코드 실행자는 빠르고 간단하게 검사하고, 쉽게 신뢰할 수 있다는 장점이 크게 부각된다. 또한 코드의 속성에 기반한 검사를 수행하기 때문에 별도의 암호화와 제3의 인증기관의 확인 절차가 필요하지 않으며 안정성을 정적으로 미리 컴파일 시 생성함으로써 런타임(Run-time) 검사를 하지 않아 여타의 수동적인 방법에 비해 실행 속도측면에서 이득을 볼 수 있다[1,3,4]. 그렇지만 안전증거의 크기가 원시 코드보다 훨씬 증가될 우려가 있으며 안전 정책의 확립과 안전 증거의 생성 등의 방법적인 측면에서 여러 가지 해결해야 할 문제가 존재한다[1].

3. Branching에 기반한 PCC의 변형된 방법

비합법적인 수행으로부터 코드를 보호하기 위해 제시된 PCC도 앞에서 설명한 바와 같은 문제에 직

면하고 있다. 본 논문에서 제시되는 방법은 PCC에서 제시된 방법에 변형을 취한 방법이며 PCC가 직면한 문제들의 해결점을 제시한다. EPCC는 안전 정책을 정의하는 단계의 고려가 불필요하기 때문에 안전 정책 확립에 관계된 문제의 해결을 꾀한다. 뿐만 아니라 특정한 코드 실행자에 종속되어 있는 안전 정책에 따른 안전 증명 모듈과 안전 검사 모듈(Checker)이 이종의 안전 정책에서는 변경이 요구되는 반면에 본 시스템은 단일하고 일관된 안전 증명 모듈과 안전 검사 모듈로써 안정성 확인이 가능하므로 표준화가 용이하다. 또한 안전 증거의 크기 문제에서도 EPCC를 도입함으로써 해결점에 귀착할 수 있다.

[그림 3]은 본 논문에서 제시하는 시스템의 안정성을 검증하는 전형적인 프로세스를 나타낸다.

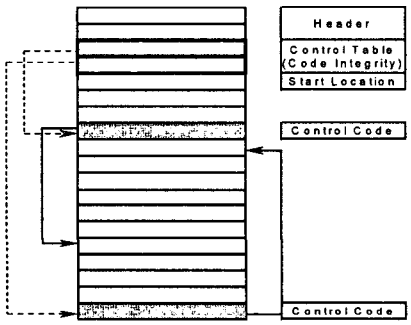


[그림 3] EPCC

[그림 3]은 핵심적인 요소로 간주되는 안전 정책을 중심으로 진행되는 [그림 1]과는 달리 안전 정책에 대한 고려를 배제시켰다. 이 시스템에서는 특별히 안전 정책을 분기에 기반하여 확립한다. 악성코드의 비합법적 수행은 코드가 처음 실행될 때 얻게 될 제어권을 가로채어 코드 사용자나, 작성자가 의도하지 않은 결과를 도출해낸다[9,10]. 코드의 제어권을 가로채기 위해서는 코드의 분기 부분이 자신의 코드를 지시하도록 변경해야 한다는 악성코드의 속성에 착안하여 EPCC 시스템은 안전 정책으로 코드 내의 분기의 이미지를 이용하여 악성코드의 코드 변경 시 이를 검출해내는 기법을 도입한다.

[그림 3]에서 코드 작성자는 컴파일하여 생성된 원시 코드와 원시 코드의 분기들을 이미지화함으로써 생성된 코드 무결성(Integrity)으로 EPCC 바이너리를 완성시킨다. [그림 3]의 코드 실행자는 실행 코드내의 코드 무결성과 원시 코드내의 분기의 일치성 여부가 확인되면 원시 코드를 실행시키는 메커니즘으로 시스템이 유동한다. 코드 작성자가 컴파일 후 정상 코드의 분기에 대한 이미지를 그대로 유지하고 있기 때문에 비록 악성 코드에 의한 변경이 있어도 정상의 원시 코드로 복원이 가능하다.

코드 실행자에 의해 정의될 안전 정책을 제거함으로써 코드 실행자에 종속적인 인증 컴파일러와 인증 확인 모듈이 일반화 될 수 있어 표준화가 훨씬 수월해진다. 본 시스템에서 코드의 내부적인 구성도는 [그림 4]와 같다.



[그림 4] EPCC로 만들어낸 실행 코드

실행 코드는 일반적으로 헤더와 각 코드 세그먼트, 그리고 해당 참조 데이터로 구성되는데[8], EPCC에 의해 생성한 EPCC 바이너리에는 코드 무결성을 확보하기 위한 제어 테이블(Control table, [그림 5])이 첨가된다. 코드 실행자는 이 제어 테이블을 참조하여 코드의 안정성을 확인하게 된다. 예를 들면 ELF(Excutable Linkage File) 실행 코드의 파일 형식에 따르면 코드 실행의 제어권이 양도될 주소가 명시되어 있다[8]. 만약 이 주소에 대한 정보의 사전 통보가 이루어져 있으면 비록 악성코드에 의한 제어권 양도에 대한 코드의 변경이 있어도 이를 검출하고 악성코드가 시도한 방향의 회진(回進)이 가능하다. 또한 악성코드에 의해 변경된 주소 정보는 제어 테이블을 참조하여 정상 코드 생성 초기의 분기로 복구가 가능하다. 다음 [그림 5]는 EPCC 바이너리가 소유하게 될 제어 테이블의 구성이다.

이러한 구성은 본 시스템이 추구하는 능동적인 악성 코드로부터의 대응에 부합하며 초기의 PCC시스템이 가지고 있는 안전 정책의 생성에 대한 어려

움이 해결된다. 또한 초기 PCC의 안전 증거의 크기 문제에서도 크게 해결점을 찾을 수 있다. 또한 정적인 무결성 검사를 하기 때문에 속도의 향상도 기대할 수 있다.[7].

Address	Code Identifier	Code Operand
Address	jump	Jump address
Address	Code Identifier	Code Operand
Address	jump	Jump address
Address	Code Identifier	Code Operand
Address	jump	Jump address
Address	Code Identifier	Code Operand

[그림 5] Control Table(Code Integrity)

4. 결론 및 향후 계획

네트워크가 복잡해지고 사용자의 수가 급격히 늘어나고 있는 추세와 더불어 악성코드의 비합법적 수행도 가속화되고 있는 추세에 기존에 제시되고 사용되었던 방법으로는 더 이상 실행 코드의 안정성을 보장 할 수가 없다. 또한 암호의 사용이나 제 3의 인증기관으로의 위탁도 개인적인 결함을 내포하고 있기 때문에 좀 더 효율적인 대응 방식이 필요하다[1]. 이러한 시점에서 PCC는 능동적으로 코드의 안정성을 보장해 줄 수 있는 시스템으로 제시되었다.

PCC가 안정성이 보장되고 효율적인 시스템이지만 여전히 구현상의 현실적인 문제를 안고 있다. 이를 해결하기 위한 EPCC는 안전 정책 확립 문제와 안전 증거 크기 문제를 해결한다. 또한 안정성과 효율성을 보장하면서 현실적인 문제에 더욱 접근한 시스템으로 앞으로도 연구할 가치가 높은 시스템이다. 정상 실행 코드의 변경을 요구하는 악성코드의 대부분은 실행 코드의 분기 부분의 변경을 감행하여야 가능하므로 본 시스템은 이러한 악성코드로부터의 능동적인 대응이 가능하다[10].

악성코드를 대응해서 등장한 기존의 방법들은 코드의 스캔기법(Scan)과 같은 수동적인 방법이 사용되고 상용화되는 것이 현실이다. 이러한 방법들은 악성코드 속성에 대한 정보가 제공되어야 제작이 가능하다[11]. 다시 말하면 악성코드에 누군가 피해를 입은 후에야 대응이 가능한 것이다. 더 이상은 이러한 수동적인 방법으로는 코드와 시스템을 보호할 수 없는 시점에서 본 시스템에 대한 연구를 진행시켜 악성코드로부터 더욱 효율적으로 대응할 필요가 있다. 현재 EPCC시스템에서 무결성을 생성해 내기 위한 컴파일러 부분의 모듈 추가와 코드 실행자의 무결성 검사 모듈 추가의 구현이 시행될 계획이다.

6. 참고 문헌

[1] George C. Necula, Peter Lee. *Proof-Carrying Code.* In The 24th Annual ACM Symposium on Principles of Programming Languages(Jan. 1997), ACM.

[2] George C. Necula, Peter Lee, *The Design and Implementation of a Certifying Compiler.*

[3] George C. Necula, Peter Lee, *Safe Kernel extensions without run-time checking.* InSecond Symposium on Operation Systems Design and Implementations(Oct, 1996), Usenix.

[4] George C. Necula and Peter Lee. *Safe, Untrusted Agents using Proof-Carrying Code.*

[5] R. A. Riemenschneider. *Checking the Correctness of Architectural Transformation Steps via Proof-Carrying Architectures.*

[6] Andrew W. Applel, Edward W. Felten, Zhong Shao. *Scaling Proof-Carrying Code to Production Compilers and Security Policies.* (Jan, 1999)

[7] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman. *Compilers. Principles, Techniques, and Tools.*

[8] TIS(Tool Interface Standards), *ELF:Executable and Linkable Format.* Portable Formats Specification, Version 1.1.

[9] AntiViral Toolkit Pro the Standard in internet Virus Protection.
http://www.avpve.com/virused/classification/index.html

[10] Fred Cohen. *Computer Viruses-Theory and Experiments*(1984)

[11] Eugene V. Kaspersky. Need the best virus protection(1998)

[12] W. T. Polk, L. E. Bassham. *A Guide to the Selection of Anti-Virus Tools and Techniques.* National Institute of Standards and Technology Computer Security Division(Dec, 1992)

[13] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. *A Survey of Active Network Research* (Jan, 1997).