情 報 保 護 學 會 誌
第 16 卷 第 5 號, 2006. 10

프로그래밍 언어에 기반한 정보흐름 보안

이 은 영

요 약

컴퓨터가 다루고 있는 정보의 기밀성(confidentiality of information)을 보호하는 일은 매우 중요하며, 그 중요성은 날이 갈수록 증가하고 있다. 그럼에도 불구하고, 실제로 정보의 기밀성과 무결성(integrity)을 완벽하게 보호해 주는 컴퓨터 시스템은 매우 찾기가 어렵다는 것이 또한 알려진 사실이다. 기존에 존재하는 이론적인 프레임워크들은 기밀성을 표현하기에는 부적절하며, 기밀성 보호를 위한 실제적인 시스템들 역시 이론적인 면에서 충분히 만족스럽지 못하다. 이와 같은 현실에서 타입시스템을 가지는 프로그래밍 언어가 정보 기밀성 보호를 위한 효과적인 방법으로 새로운 관심을 받고 있으며, 상당한 연구가 이 방향으로 진행되고 있다. 본 논문에서는 이제까지 진행되었던 연구들을 중심으로 프로그래밍 언어를 이용한 정보 기밀성 보호의 연구 동향과 주요 과제들을 소개하고자 한다.

Ⅰ. 서 론

정보 시스템들이 네트워크를 통해서 연결되어 있고, 인터네트워킹이 빠른 속도로 확대되고 있는 현재의 컴퓨팅 환경에서 기존의 보안 메커니즘들이 정보 기밀성을 보호하는데 그리 효과적이지 못하다는 것이 이미 알려진 사실이다. 군사, 의료, 재정 시스템뿐만 아니라, 웹 서비스에 기반을 둔 이메일이나 인터넷 쇼핑, B2B(business-to-business) 전자상거래에 이르기까지 현존하는 많은 시스템들에서 사용자의 사생활 보호가 문제로 제기되고 있는 반면, 그에 대한 만족할만한 해결책은 알려져 있지 않은 것도 또한 사실이다. 이와 같은 상황에서 프로그래밍 언어를 이용한 정보의 기밀성 보호는 그 대안으로 관심을 끌고 있다.

기밀성이라는 특성을 분석하는 것은 그리 쉽지 않다. 그 이유는 대부분의 기밀성 침해가 시스템을 디자인하거나 구현하는 과정에서 의도하지 않은 실수로 일어나는 경우가 많기 때문이다. 또 요즘의 정보 시스템들은 여러 개의 호스트나 외부에서 만들어진 코드를 합쳐서 만들어지는 경우가 대부분인데, 외부 호스트나 외부 코드가 신뢰성에 대해서 검증되지 않고 사용되는 경우, 최종적으로 만들어진 시스템의 기밀성을 보장하는 것은 매우 어려워진다.

액세스 제어(access control)는 기밀 정보를 보호하는 가장 일반적인 방법이다. 액세스 제어를 사용할 경우, 사용자는 기밀 정보를 포함하고 있는 파일이나 객체에 접근하기 위해서 특별한 권한(privilege)을 가지고 있어야 한다. 액세스 제어를 이용한 방법의 문제점은 액세스 제어가 기밀 정보의 획득은 제어할 수 있지만, 이미 획득된 정보의 전파는 막을 수 없다는 데 있다. 일단 프로세스가 적법한 절차에 따라서 기밀 정보를 획득한 후, 실수이든 고의이든, 획득한 정보를 누설할 수 있다는 것이다. 전자서명을 이용한 검증이나 안티바이러스 프로그램을 이용한 보호 역시 검증받은 프로그램이 획득한 정보를 누설하는 것을 막을 수는 없다. 적법한 절차를 걸쳐서 획득한 정보가 일정한 기밀성 보호 정책에 따라서 제대로 유지되고 있는지를 검증하는 방법은 정보를 이용하는 프로그램 내부의 정보 흐름(information flow)을 분석하는 것이 최선의 방법이다. 그렇지만 대부분 컴퓨팅 시스템의 복잡도와 크기를 고려해 볼 때, 프로그램의 정보 흐름을 손으로 분석한다는 것은 불가능하다.

프로그래밍 언어의 타입 시스템을 이용하여 정보 흐름을 분석하는 시도가 새로운 분야로 각광받는 것도 이 때문이다. 이미 자신의 타입 시스템 내부에서 보안상의 특성을 보장하는 프로그래밍 언어들이 제안되고 구현되고 있다.[1]-[13] 본 논문에서는 이런 타입 시스템을 가진 프로그래밍 언어를 *보안타입 언어*(security-

* 동덕여자대학교 정보과학대학 컴퓨터학과 (elee@dongduk.ac.kr)

typed language)라고 부르기로 하겠다. 보안타입 언어에서 모든 프로그램 변수와 연산식에는 기존의 타입뿐만 아니라, 보안과 관련된 타입이 부과된다. 보안타입 언어를 이용한 프로그램은 주어진 프로그램이 가지는 보안상의 특성에 대한 검증이 타입 검증(type checking)을 통해서 컴파일 시에 이루어지기 때문에, 별도의 런타임 체크가 전혀 필요 없거나, 아주 최소한도로만 요구된다는 장점을 가진다. 또한 타입 시스템을 가지는 일반적인 언어와 마찬가지로 보안타입 언어 역시 조합이 가능하다는 장점을 가지고 있다. 다시 말해서 보안타입 언어로 만들어진 서브시스템이나 컴포넌트를 이용해서 만들어진 정보 시스템은 서브시스템들 사이의 타입 서명(type signature)이 일치하는 한 타입 시스템이 보장하는 보안성이 조합의 결과로 만들어진 최종 시스템에서도 유지된다는 것이다.

II. 배경 연구

1. 언어를 기반으로 둔 보안

프로그래밍 언어를 보안성 보장에 이용하는 것은 단지 기밀성을 보장하는 데에만 그치지 않는다. 보안성 보장에 프로그래밍 언어를 이용한 가장 대표적인 예로는 자바 런타임 시스템을 들 수 있다. 자바 런타임 시스템에서 제공하는 바이트코드 검증기(bytecode verifier), 자바 애플릿, 샌드박스 모델(sandbox model), 혹은 스택 검열(stack inspection) 등이 자바가 제공하는 대표적인 보안 기재들이다. 이 중에서 바이트코드 검증기만이 정적인 프로그램 분석기법을 사용할 뿐, 나머지들은 모두 런타임 분석에 의존하고 있다. 그렇지만 이들이 제공하는 보안성 보호가 자바라는 언어를 통해서 제공된다는 점에서 위에 나열한 기법들은 모두 프로그래밍 언어에 기반을 두고 있다고 말할 수 있을 것이다. 그렇지만 자바 언어가 제공하는 기법들은 어느 것도 정보흐름 제어를 염두에 두고 설계되지 않았기 때문에 정보기밀의 보호에는 적합하지 않다는 한계점을 가지고 있다. 자바 타입 시스템에서 바이트코드 검증기는 객체 주소가 변경되거나 외부에서 개인(private) 필드에 접근하는 것을 차단할 수 있다. 개인 필드에 대한 접근을 차단하는 것은 기밀성 보호에서 매우 중요한 요소 중에 하나이기는 하지만, 바이트코드 검증이 컴파일 타임에 정적으로만 이루어지기 때문에 그리 강력하지 못하다는 한계점을 가지고 있다. 샌드박스 모델은 자바 애플릿이 사용할 수 있는 클래스를 제한함으로써 애플릿을 사용하는 클라이언트 시스템을 보호하는 방법이다. 그렇지만 이 경우 악의적인 애플릿이 자신이 작성된 (혹은 다운로드된) 호스트와의 통신을 통하여 기밀정보를 유출하는 것은 막을 수는 없다는 단점을 가지고 있다. 마지막으로 스택 검열 역시 동적인 액세스 제어를 통하여 정보의 무결성을 보호하는데 사용되지만 기밀성 유지는 고려하지 않고 있다.

프로그래밍 언어를 이용한 기법들은 계속 연구가 진행되고 있는 분야로, 모바일 코드에 의한 공격으로부터 시스템을 보호하고자 하는 시도나 보다 일반적인 보안정책을 프로그래밍 언어로 구현하고자하는 시도가 진행되고 있다.[15]-[21]

2. 은닉 채널

컴퓨터 시스템 내부에서 채널은 시스템 사이에서 혹은 시스템 내부에서 정보를 정하는 통로가 된다. 은닉 채널이란 본래 채널로서 사용되지 않는 기재를 통해서 정보를 전달하거나 유출하는 방법을 말한다. 은닉 채널을 이용한 정보의 유출을 막는 것은 중요한 정보가

〔표 1〕 은닉채널 분류표

분 류	특 징
함축적 흐름 (*implicit flow*)	프로그램의 제어구조를 이용하여 정보를 유출
종료 채널 (termination channels)	프로그램이 정상적으로 종료하는지의 여부로 정보를 유출
타이밍 채널 (timing channels)	특정한 액션 (프로그램의 종료 등)이 일어나는 시간을 이용하여 정보를 유출
확률적 채널 (probabilistic channels)	관찰 가능한 데이터의 확률적 분포를 변환시켜서 프로그램의 행동을 관찰하고 정보를 유출
리소스 사용 채널 (resource exhaustion channels)	리소스 (메모리, 디스크 공간 등)의 사용량을 이용하여 정보를 유출
전력 채널 (power channels)	전력 사용량을 이용하여 정보를 추측

새어나가는 것을 막는 중요한 열쇠가 된다. 은닉 채널은 그 특성에 따라 〔표 1〕처럼 분류될 수 있다.

3. 강제적인 액세스 제어

강제적인 액세스 제어 기법에서 각각의 데이터는 기밀성 보호정책에 의해서 자신의 보안 레벨을 할당받게 된다. 그리고 프로그램 내부에서 연산이 수행될 때 일반적인 연산 외에도 해당하는 데이터의 보안 레벨에 대한 연산도 함께 수행된다. 이와 같은 보안 레벨에 대한 연산은 정보흐름에 대한 힌트를 주고, 정보의 유출을 찾아내는 키로서 작용한다. 강제적인 액세스 제어는 미국 국방성이 발행한 오렌지북에 명시된 방법으로 방법 자체는 간편하지만, 지나치게 제약적이여서 국방용이 아닌 일반적인 용도로 사용하기에 어렵다는 한계를 가지고 있다.[22]

강제적 액세스 제어의 명백한 단점은 함축적 흐름(implicit flow)을 제대로 찾아낼 수 없다는 점이다. 명시적 흐름 (explicit flow)이 기밀 정보가 공개된 변수 (public variable)에 할당되는 것을 말한다면, 함축적 흐름은 프로그램의 제어 구조에 따라서 정보가 보이지 않게 유출되는 현상을 말한다.

〔그림 1〕은 함축적 흐름이 생길 수 있는 프로그램의 예를 보여주고 있다. 편의상 프로그램의 모든 데이터는 2개의 보안 레벨, *high*와 *low*로 나누어진다고 가정하자. 〔그림 1〕의 프로그램에서 변수 *l*이 보안 레벨 *low*를 가지고, 변수 *h*가 보안 레벨 *high*를 가진다고 가정하면, 위의 프로그램은 보안상 안전하다고 볼 수 없다. 명시적 흐름이 생기지는 않았다 하더라도 프로그램의 실행결과 보안 레벨이 낮은 변수 *l*을 관찰함으로써 보안 레벨이 높은 기밀정보인 변수 *h*의 값에 대한 정보를 얻을 수 있기 때문이다.

위의 예제 프로그램에서 보인 것과 같은 함축적 흐름을 통한 정보 유출을 피하기 위해서는 변수의 보안 레벨을 프로그램의 실행 도중에 변경시키는 방안이 고려되어야 한다. 다시 말해서 〔그림 1〕에서 if문이 실행되고 난 후 변수 *l*의 보안 레벨을 *high*로 변경시키고 외부에서 관찰할 수 없도록 하는 것이 그 방법이다. 그렇지만 이런 경우 프로그램이 수행됨에 따라 높은 수준의 보안 레벨을 가진 데이터가 단조적으로 증가한다는 문제가 생긴다. 또한 프로그램 수행 도중에 보안 레벨의 강등을 허용할 경우에는 함축적 흐름을 야기하는 보안상의 허점이 생길 수 있다는 점도 이미 밝혀진 문제점이다.

```
1.    l := 0;
2.    if h = 1  then l := 1
3.              else skip
```

〔그림 1〕 함축적 흐름

4. 정적인 정보흐름 제어

Denning 등은 그들의 논문에서 정적인 프로그램 분석이 정보의 흐름을 제어하는데 쓰일 수 있으며, 정적 분석이 정확도를 증가시키면서도 런타임의 오버헤드를 줄일 수 있는 방법임을 보였다.[27] 정보 흐름의 정적인 특성들은 대부분 정리 증명기(theorem provers)를 사용하여 분석되었다. 정보 흐름은 타입 검증을 통해서 분석되기도 하는데, 그 대표적인 예가 바로 Jif 컴파일러이다.[6]

타입 검증을 이용한 방법에서, 프로그램에서 사용되는 모든 연산식들은 2가지의 타입을 가지게 된다. 하나는 일반적으로 사용되던 타입(예를 들면, int, double 등)이며, 다른 하나는 값이 어떻게 사용되느냐에 대한 레이블이다. 강제적인 액세스 제어기법과 달리 연산식에 부과되는 모든 레이블들은 순수하게 정적인 방식으로 부여된다. 레이블들은 정해진 정보흐름 정책에 따라 주어진 데이터가 어떻게 사용되는지를 설명한다. 컴파일러는 레이블이 붙여진 프로그램을 읽어들여서 타입 검증을 한다. 프로그램의 타입 검증이 무사히 끝났다는 것은 주어진 프로그램을 실행시켰을 때 정보흐름 정책에 반하는 어떠한 정보의 흐름도 없다는 것을 의미하게 된다.

정보흐름의 정적인 분석이 가지는 또 다른 장점은 정보의 함축적 흐름을 찾아내는데 유리하다는 점이다. 프로그램-카운터 레이블(program-counter label, pc)은 함축적 흐름을 정확히 판단하고 현재 프로그램 내부에서 흐르고 있는 정보 사이의 의존관계를 추적하기 위해서 사용된다. 예를 들어 대입문의 경우, 대입에 사용되는 변수의 보안레벨은 적어도 대입문이 수행되는 문맥의 *pc*보다 더 제한적이어야 안전한 명령문으로 간주된다.

Ⅲ. 언어기반 정보흐름 제어

1. 의미론을 이용한 제어

이 장에서는 간단한 보안타입 언어를 이용하여 어떻

$C ::=$ skip | $var := exp$ | $C_1; C_2$
| if exp then C_1 else C_2
| while exp do C

〔그림 2〕 명령어 문법

게 기밀성 보장이 타입 시스템을 이용하여 구현될 수 있는지 알아보도록 하겠다. 〔그림 2〕는 이 장에서 사용될 언어의 문법을 보여주고 있다. 이 언어는 간단하게 skip과 대입문, 명령문의 연속적 조합, 조건문, 그리고 while 반복문으로 이루어져 있다. 앞 장에서와 마찬가지로 변수 h와 l은 보안 레벨 *high*와 *low*를 가진다고 가정하며, 연산식 exp는 변수와 상수에 대한 산술 연산에 의해서 만들어진다.

프로그램의 불간섭성(noninterference)은 다음과 같이 정의될 수 있다.

기밀인(보안레벨이 high인) 입력 값에 주어진 변화는 공개된(보안레벨이 low인) 출력 값에 아무런 영향을 끼치지 않는다.

이와 같은 개념적인 정의는 프로그램인 언어에서 정교하고 이론적으로 표현될 수 있다. 만약 주어진 프로그램이 입력상태 $s=(s_h, s_l)$로 시작되고, 마지막에 출력상태 $s'=(s'_h, s'_l)$로 끝난다고 가정하자. 여기에서 s_h, s_l은 각각 보안레벨이 *high*인 변수들과 보안레벨이 *low*인 변수들로 이루어진 집합이다. 그렇다면 프로그램 C에 대한 의미론(semantics) $[\![C]\!]$는 다음과 같은 함수로 정의된다.

$$[\![C]\!] : S \rightarrow S_\perp \; (S_\perp = S \cup \{\perp\} \text{ and } \perp \notin S)$$

이 함수는 $s \in S$인 입력상태에 대해서 $[\![C]\!]s \in S$인 출력상태를 반환하거나 프로그램이 제대로 종료하지 않은 경우 $\perp$를 반환한다. 프로그램 상태에 대해서 정의되는 동치관계 $=_L$는 2개의 상태가 동일한 보안레벨 *low* 변수를 가질 때 성립한다 ($s =_L s'$ iff $s_l = s'_l$). 프로그램의 수행 행동은 또 다른 관계인 $\approx_L$로 표현된다. 2개의 출력상태가 공격자의 의해서 전혀 구분될 수 없을 때 관계 $\approx_L$는 성립하게 된다. 정의된 프로그래밍 언어의 의미론에 의해서 프로그램 C에서 간섭 현상이 일어나지 않는다는 것은 다음과 같은 식으로 정의될 수 있다.

$$\forall s_1, s_2 \in S. s_1 =_L s_2 \;\Rightarrow\; [\![C]\!]s_1 \approx_L [\![C]\!]s_2 \qquad (1)$$

다시 말해서 2개의 입력상태가 같은 *low* 변수를 가지고 있다면, 프로그램의 실행 후에도 공격자에게 보이는 상태는 동일하다는 것이다. 공격자에게 보이는 상태를 어떻게 선택할 것인가의 결정은 주어진 보안 모델에서 어떤 보안 특성을 사용하느냐에 따라서 달라진다. 대부분의 경우 관계 $\approx_L$가 대칭관계이고 반사관계인 것이 요구되며, 경우에 따라서는 동치관계인 것이 요구되기도 한다. 〔수식 1〕에 의해서 코드 ($h := l+4$)는 안전한 프로그램으로 판명된다. 왜냐하면 프로그램에 의해서 *high* 변수의 값만이 변했을 뿐 *low* 변수의 값은 아무런 영향도 받지 않았기 때문이다. 그렇지만 코드 (*if h = 3 then l := 5 then skip*)는 보안상 안전하지 못한 프로그램이 된다. 예를 들어 변수 l에 저장되었던 값이 5가 아닐 경우, *high* 변수 h의 값에 따라 명령어 수행 후 l에 저장되는 값이 달라지기 때문이다.

2. 보안 타입 시스템

보안타입 시스템은 프로그램 변수나 연산식에 어떻게 적절한 보안타입을 할당하는지를 기술하는 타입 규칙(typing rule)의 집합이다. 일반적으로 $\vdash exp : \tau$는 주어진 연산식 exp가 타입 규칙에 의해서 타입 τ를 가진다는 것을 의미한다. 이처럼 연산식의 타입을 결정하는 행위를 타입 결정(typing judgment)이라고

$\vdash exp : high$ [E1]

$\dfrac{h \notin Vars(exp)}{\vdash exp : low}$ [E2]

$[pc] \vdash \text{skip}$ [C1]

$[pc] \vdash h := exp$ [C2]

$\dfrac{\vdash exp : low}{[low] \vdash l := exp}$ [C3]

$\dfrac{[pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash C_1 ; C_2}$ [C4]

$\dfrac{\vdash exp : pc \quad [pc] \vdash C}{[pc] \vdash \text{while } exp \text{ do } C}$ [C5]

$\dfrac{\vdash exp : pc \quad [pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash \text{if } exp \text{ then } C_1 \text{ else } C_2}$ [C6]

$\dfrac{[high] \vdash C}{[low] \vdash C}$ [C7]

〔그림 3〕 보안타입 시스템

부른다. 마찬가지로 $[pc] \vdash C$는 프로그램 C가 보안 문맥 (security context) pc에서 타입을 결정할 수 있다는 뜻으로 해석된다. 보안 문맥은 원하는 보안 특성에 따라 여러 가지로 다양하게 정의될 수 있겠으나 본 논문에서는 앞 장에서와 마찬가지로 2개의 상태(*low* 혹은 *high*)만을 가진다고 가정하도록 하겠다.

〔그림 3〕의 주어진 타입 규칙들은 앞 절에서 사용한 간단한 프로그래밍 언어를 위한 규칙들이다. 이 타입 규칙들은 Volpano 등이 개발한 타입 시스템과 동일하다.[2] 보안 문맥은 2개의 상태(*low* 혹은 *high*)만을 가지며 모든 연산식 역시 보안 상태에 대한 타입만을 가지게 된다. 타입 규칙 〔E1〕는 모든 상수를 포함한 모든 연산식이 *high* 타입을 가질 수 있다는 것을 보여준다. 이에 비해서 *low* 타입은 상당히 제한적으로 주어지는데 이를 나타내는 타입 규칙이 〔E2〕이다. 규칙 〔E2〕는 연산식 *exp*에 포함되어 있는 변수 중 어떤 것도 *high* 타입을 가지지 않았을 경우에만 연산식 *exp*가 *low* 타입을 가질 수 있다는 것을 보여주고 있다. 이것은 상당히 보수적인 방법의 타입 결정으로 프로그램 내부에서 생길 수 있는 간섭 현상을 최소화하려는 노력이라고 볼 수 있다.

타입 규칙 〔C1〕과 〔C2〕는 명령어 skip과 *high* 타입 변수로의 대입문의 경우 언제든지 현재의 보안레벨을 그대로 유지시키면서 타입을 결정할 수 있다는 것을 서술하고 있다. 그러나 타입 규칙 〔C3〕의 경우는 조금 다르다. *low* 타입 변수로의 대입문의 경우는 r-value가 *low* 타입인 경우에만 타입이 결정된다. 이것은 *high* 타입의 값이 *low* 타입의 변수로 대입되는 것을 막는다.

타입 규칙 〔C5〕와 〔C6〕은 프로그램 제어구조를 이용한 함축적 흐름을 막기 위한 타입 규칙들이다. 만약 조건식의 보안 레벨을 평가한 결과가 *high*라면 if문의 분기 부분이나 while 반복문의 몸체 부분의 보안 레벨도 마찬가지로 *high*여야 한다는 것이다. 마지막으로 타입 규칙 〔C7〕은 포섭관계(subsumption)를 나타낸다. 규칙 〔C7〕에 의하면 만약 어떤 프로그램이 보안문맥 *high*에서 타입 결정이 가능하다면, 같은 프로그램은 보안문맥 *low*에서도 타입 결정이 가능하다. 이를 이용하면 보안 레벨 *high*의 조건문이나 반복문의 수행을 마친 후 보안 문맥을 다시 *low*로 낮추는 것이 가능하고, 이를 이용하여 앞 장에서 서술했던 것처럼 보안 레벨이 프로그램의 진행에 따라 단조적으로 증가하는 현상을 막을 수 있다.

Ⅳ. 현재 동향

1. 보안정책 관련연구

프로그램의 불간섭성을 보장하는 것이 여러 가지 면에서 유익함에도 불구하고, 실제적인 사용에서 적당하지 않은 경우가 있다. 특히 보안 레벨을 낮추어야하는 경우가 이에 해당한다. 첫 번째 경우 기밀 정보를 암호화하고 이를 공개된 매체(예를 들면 네트워크)를 통해서 전송하는 경우이다. 이 경우 기밀 정보임에도 불구하고 암호화된 결과물은 보안 레벨을 낮추어야만 전송이 가능하다. 또 다른 예는 패스워드를 검증하는 프로그램의 경우이다. 시스템 내부에 저장된 패스워드는 최상위의 보안 레벨을 가진다. 그렇지만 패스워드의 검증을 위해서는 사용자가 입력하는 값은 공개적인 경로를 통해서 전해지며, 상대적으로 낮은 보안 레벨을 가지고 있다. 따라서 입력된 패스워드를 검증하기 위해서는 저장된 패스워드의 보안 레벨을 낮추는 것이 불가피하게 된다.

Myers 등은 보안 레이블의 분산화 모델을 주창했는데, 논문에서 그들은 보안의 주체들(principals)의 관계와 프로세스의 권한은 정적으로 분석하는 방법을 사용하였다.[3] 이 경우 보안 레이블은 보안 레벨의 분산화를 위한 별도의 자료 구조를 가지도록 구성되었으며, 이를 통해서 서로 불신하는 주체들 사이에서도 상호 레이블링이 가능하도록 지원하였다.

암호화 프로토콜은 보안 레벨의 하향화를 필요로 하는 또 다른 분야이다. Abadi는 spi-calculus를 주창하였는데, 이것은 기밀성을 보호하는 암호화 프로토콜을 위한 타입 시스템이다.[28] 이 타입 시스템의 경우 낮은 보안 레벨에서는 비밀키와 비밀키의 사용 모두를 관찰할 수 없다.

의도적으로 보안 레벨을 낮추는 경우 프로그래머가 의도하지 않은 방향으로 정보가 유출될 위험성 역시 동시에 증가하게 된다. Zdancewic 등은 보안 레벨을 낮추는 경우에 발생할 수 있는 채널의 사용을 탐색하고 제한하는 방법을 제안하였다.[29] 그들은 공격자를 시스템 동작에 영향을 줄 수 있는 적극적인 공격자(active attacker)와 단지 시스템의 동작을 관찰만 할 수 있는 소극적인 공격자(passive attacker)로 나누고, 자신들의 보안 레벨 하향화 방식을 사용할 경우 적극적인 공격자가 얻어낼 수 있는 시스템 정보는 소극적인 공격자가 얻어낼 수 있는 정보보다 많지 않다는 것을 주장하였다.

Volpano 등은 패스워드를 검증하는 프로그램의 경우 보안 레벨의 하향화가 불가피하다는 것에 초점을 맞추고, 이를 해결하기 위한 타입 시스템을 디자인하였다.[32] 그들의 타입 시스템은 패스워드 쿼리와 비슷한 연산을 제공하면서 보안에 대해서 확률적인 확실성을 보장하였다. 우선 타입 체크된 프로그램에서는 다항식 기간(polynomial time) 안에 어떤 비밀도 유출되지 않으며, 비밀은 무시할 수 있을 만한 작은 확률을 가지고 유출 가능하다는 것이다. 또 이 연구의 연장으로서 Volpano는 편방향(one-way) 함수로 암호화되어 저장된 패스워드를 풀어내는 일은 편방향 함수를 깨는 것과 마찬가지로 어렵다는 것을 증명하였다.

2. 인증가능 컴파일 관련연구

정보흐름을 검증하기 위해서 특정 언어를 기반으로 한 컴파일러를 이용하는 것은 또 다른 취약점을 야기할 수 있다. 다름이 아니라 타입 검증기(type checker)와 코드 발생기(code generator)를 trusted computing base(TCB)에 포함시켜야 한다는 것이다.[23] 현재의 복잡하고 여러 가지 기능을 가진 언어들을 위한 컴파일러는 그 기능만큼이나 크고 복잡한 컴파일러를 가지고 있는 경우가 많다. 따라서 그 복잡성에 비례해서 그 안에 알려지지 않은 버그가 (의도적이든 의도적이지 않던 간에) 있을 확률이 높아진다. 따라서 정보흐름을 분석하는 경우 가능한 한 실행 가능한 기계어와 가까운 형태의 언어로 분석하는 것이 바람직하다. 인증가능 컴파일(certifying compilation)은 컴파일 과정에서 정적인 분석을 하는데 필요한 정보를 추출하고, 중간과정 코드에 이와 같은 정보를 그대로 보전하자는 취지에서 시작되었다.[24] 자바 언어의 바이트코드 검증기와 타입드 어셈블리 언어(typed assembly langauge)가 그 대표적인 예이다.

로우 레벨(low level) 언어의 경우는 보안 정보 흐름을 분석하는 데 그리 유용하지 못하다고 알려져 있다. 대표적인 이유는 정보흐름을 분석하는데 사용되는 프로그램의 구조물들이 컴파일 과정에서 대부분 소멸되어 버리기 때문이다. Zdancewic 등은 논문을 통해서 타입 시스템으로 로우 레벨 프로그램의 불간섭성을 보장하는 것은 프로그램이 컨티뉴에이션(continuation)만을 제어 연산자로 사용하는 경우에만 가능하다는 것을 증명하였다.[11]

인증가능 컴파일의 또 다른 가능성은 기계어 코드의 안전성을 정보흐름에 적용하는 경우이다. 그 대표적인 예가 타입드 어셈블리 언어와 증거운반 코드 (proof-carrying code)이다. 타입드 어셈블리 코드는 생성된 기계어 코드가 타입 안전성(type safety)을 위배하지 않는다는 것을 보장하는 타입 시스템을 제공한다.[16] 이에 비해서 증거운반 코드는 프로그램이 정해진 보안정책을 만족시킨다는 증명을 코드를 배포할 때 배포자(프로그램 작성자 혹은 판매자)가 함께 배포하고, 그 증명은 프로그램을 실행하기 전에 검증되는 형식으로 이루어진다.[18][33]

3. 동적정책 관련연구

프로그래밍 언어를 바탕으로 정보흐름을 분석하는 경우에 있어서 정보흐름에 관한 정책이 정적으로 컴파일 시간에 정해질 것이라는 믿음이 일반적이다. 그러나 정적으로 정해지는 보안 정책은 대상 시스템의 규모가 큰 경우에는 그렇게 바람직하지 않다. 예를 들어서 대규모의 파일 시스템이라면 파일에 대한 접근권한이 보안 정책을 이룰 것이다. 따라서 어떤 사용자가 자신의 파일의 접근권한을 변경했다면, 이 변경사항은 파일 시스템의 보안 정책에 동적으로 반영되어야 한다.

이와 같은 요구는 프로그래밍 언어를 염두에 둔 경에서 동적인 보안 정책을 관리하는 방법이 제안되었으며,[3] Jif 컴파일러로 구현되었다.[6] Jif 시스템에서 모든 타입에는 보안 레벨을 뜻하는 레이블이 덧붙여진다. 보안 레벨을 뜻하는 타입 *label*은 first-class 타입으로도 사용될 수 있고, 다른 변수에 대한 레이블로도 사용될 수 있다. 의존적 타입 시스템(dependent types)은 런타임에 계산되는 값에 따라서 그 타입이 변하는 타입 시스템을 말한다. 의존적 타입 시스템은 프로그래밍 언어 분야에서 오랜 기간에 걸쳐 연구가 이루어지고 있는 주제이며, 최근에 괄목한만한 성과를 얻고 있다.[24],[25]

4. 보안을 위한 정적분석

제어 분석과 데이터 흐름 분석은 프로그램 분석에서 이미 널리 알려진 기술이다.[26] 이 기술을 또한 프로그램 내부에서 보안성의 의존관계를 분석하는 데에도 유용하게 쓰일 수 있다. 대부분의 타입 시스템은 직관적이고 이해하기 쉽도록 디자인되어 있다. 그러나 제어 분석이나 데이터 흐름 분석이 타입 시스템과 함께 폭넓게 쓰이고 있다. 그 이유는 어떤 언어의 경우에는 기본 타입(principal type)이 부족하거나 제대로 정의

되지 않아서 결과적으로 변수나 연산식에 부여되는 타입이 부정확한 경우가 있기 때문이다. 이와 같은 경우 타입 시스템만으로는 충분한 정보를 표현할 수 없다.

Bodei 등은 제어흐름 분석을 이용하여 Bell-Lapadula 보안성을 π-calculus로 표현하였다.[30] Nielson 등은 엠비언트(ambients)를 이용하여 방화벽을 서술하고, 이를 통해서 올바른 패스워드를 알지 못하는 공격자의 접근을 방화벽이 어떻게 정적으로 차단하는지를 증명하였다.[31]

Clark 등은 제어흐름 분석을 통하여 보다 정확한 보안 분석을 이루어 냈다.[34] 특히 그는 프로그램 전체의 제어흐름을 분석해서 사용함으로써 다른 어느 타입 시스템보다 정확한 결과를 얻어내었다. 다음의 프로그램은 보안상 안전한 프로그램의 예이다.

$$(\text{if } h = 1 \text{ then } l := 1 \text{ else } l := 0);\ l := 0$$

위의 프로그램은 *high* 변수인 h의 값에 상관없이 *low* 변수인 l의 값이 정해지고, 또한 변수 h값의 변화가 변수 l에 전혀 반영되지 않기 때문에 앞 장에서 정의된 [수식 1]에 의해서 보안상 안전한 프로그램이다. Clark의 타입 시스템은 위의 프로그램이 안전하다는 것을 판별해 낼 수 있지만, 기존의 일반적인 타입 시스템들은 위의 프로그램을 기각한다. 왜냐하면 기존의 타입 시스템은 프로그램의 전체적인 제어흐름을 추적하기 보다는 프로그램 문장 단위를 처리를 하며, 이때 예제 프로그램의 if문은 보안 레벨이 높은 조건식으로부터 보안 레벨이 낮은 분기문으로 이어지는 구조를 가지고 있기 때문이다.

5. 은닉채널 관련연구

앞 장에서 이미 논의했듯이 함축적 흐름은 은닉 채널의 대표적인 예이다. 그렇지만 [표 1]에 나와 있는 것처럼 그 외에도 다양한 종류의 은닉채널이 존재하며, 다양한 종류의 은닉채널을 통한 정보의 유출은 굉장히 알아내기 어렵다. 이 절에서는 함축적 흐름을 제외한 다른 종류의 은닉채널을 탐색하고 방지하기 위한 연구들에 대해서 소개하도록 하겠다.

만약 어떤 공격자가 프로그램의 종료를 관찰할 수 있는 위치에 있다고 가정하자. 이런 경우 프로그램 *(while h =1 do skip)*은 더 이상 안전하지 않다. 이처럼 불간섭성은 프로그램의 종료에 영향을 받으며, 이를 막기 위해서 [수식 1]에서 사용되는 관계 $\approx_L$는 아래와 같이 정의될 수 있다.

$$s \approx_L s' \text{ iff } (s,s' \in S \wedge s =_L s') \vee (s = s' = \perp)$$

즉 2개의 프로그램 종료 상태는 모두 같은 *low* 레벨 상태로 끝나거나, 혹은 모두 멈추지 않을 때 관계 $\approx_L$를 만족시킨다. 타입 시스템에서 보안레벨이 *high* 인 조건식을 가지는 반복문을 허락하지 않고, 보안레벨이 *high*인 조건문은 분기문에 반복문을 사용할 수 없게 한다면, 프로그램의 종료여부를 이용한 공격을 막을 수 있다.

실제 컴퓨터 시스템에서 종료하지 않는 프로그램과 종료할 때까지 아주 오랜 시간이 걸리는 프로그램은 구별이 쉽지 않다. 따라서 프로그램의 종료여부를 이용한 은닉채널은 시간차를 이용한 은닉채널의 일부로도 간주될 수 있을 것이다. 타이밍 은닉채널은 특히 병렬 처리되는 프로세스가 존재하는 시스템 내부에서 치명적일 수 있다. 코드 (if $h=1$ then C_{long} else *skip)*는 시간차를 이용해서 어떻게 높은 보안 레벨의 값이 유출될 수 있는지를 보여준다. 변수 h가 1인 경우 프로그램은 아주 긴 계산을 하게 되고, 그렇지 않은 경우에는 바로 종료하게 된다. 따라서 공격자는 프로그램이 끝날 때까지의 시간을 관찰함으로써 높은 보안 레벨을 가진 변수 h의 값을 유추할 수 있게 되는 것이다. 타이밍 은닉채널을 방지하기 위해서 [수식 1]의 관계 $\approx_L$는 다음과 같이 강화되어야 한다.

관계 $\approx_L$를 만족시키기 위해서 2개의 프로그램은 모두 종료하거나 모두 종료하지 않아야하며, 종료하는 경우에도 낮은 레벨에서 관찰할 수 있는 실행 단계의 수가 같아야 한다.

또한 타입시스템도 high 레벨의 조건식을 가진 조건문의 경우 분기문에 반복문이 없어야함은 물론, 조건문 자체도 실행이 원자화(atomic)되어야 한다.

V. 결 론

점점 복잡해지고 다양해지는 컴퓨팅 환경에서 내부의 기밀 정보를 보호할 수 있는 컴퓨터 시스템에 대한 요구가 갈수록 증가하고 있다. 기밀성의 보호를 위해서는 기본적인 액세스 제어부터 시작하여 암호화, 방화벽, 전자서명, 그리고 안티바이러스 프로그램까지 많은 시스템들이 만들어지고 사용되고 있다. 그러나 이 중의 어느 시스템도 컴퓨터 시스템 내부의 정보 흐

름을 추적하고 제어한다는 근본적인 문제를 만족할만한 수준으로 해결하고 있는 것처럼 보이지는 않는다.

컴퓨터 시스템에 흐르는 정보를 추적하기 위해서 시스템을 구성하는 프로그램을 분석하고, 프로그램을 구성하는 프로그래밍 언어를 분석하고 새롭게 디자인함으로써 정보의 흐름과 유출을 막겠다는 시도는 이와 같은 상황에서 유추될 수 있는 당연한 시도라고 할 수 있을 것이다.

본 논문에서는 프로그래밍 언어를 이용한 기밀성 보호에 필요한 배경 연구 및 기본 원리, 그리고 현재의 연구 동향에 대하여 알아보았다. 정보보안의 핵심이라고 할 수 있는 기밀성 보호를 위해서 프로그래밍 언어를 이용한 기법들을 기존의 방법들과 병행함으로써 더욱 근본적이고 확실한 효과를 거둘 수 있을 것이라고 생각한다.

참 고 문 헌

[1] J. Palsberg and P. Ørbæk, "Trust in the λ-calculus," in Proc. Symposium on Static Analysis. Sept. 1995, number 983 in LNCS, pp. 314-329, Springer-Verlag.

[2] D. Volpano, G. Smith, and C. Irvine, "A sound type system for secure flow analysis," J. Computer Security, vol. 4, no. 3, pp. 167-187, 1996.

[3] A. C. Myers and B. Liskov, "A decentralized model for information flow control," in Proc. ACM Symp. on Operating System Principles, Oct. 1997, pp. 129-142.

[4] N. Heintze and J. G. Riecke, "The SLam calculus: programming with secrecy and integrity," in Proc. ACM Symp. on Principles of Programming Languages, Jan. 1998, pp. 365-377.

[5] G. Smith and D. Volpano, "Secure information flow in a multithreaded imperative language," in Proc. ACM Symp. on Principles of Programming Languages, Jan. 1998, pp. 355-364.

[6] A. C. Myers, "JFlow: Practical mostly-static information flow control," in Proc. ACM Symp. on Principles of Programming Languages, Jan. 1999, pp. 228-241.

[7] G. Barthe and B. Serpette, "Partial evaluation and non-interference for object calculi," in Proc. FLOPS. Nov. 1999, vol. 1722 of LNCS, pp. 53-67, Springer-Verlag.

[8] D. Volpano and G. Smith, "Probabilistic noninterference in a concurrent language," J. Computer Security, vol. 7, no. 2-3, pp. 231-253, Nov. 1999.

[9] J. Agat, "Transforming out timing leaks," in Proc. ACM Symp. on Principles of Programming Languages, Jan. 2000, pp. 40-53.

[10] A. Sabelfeld and D. Sands, "Probabilistic noninterference for multithreaded programs," in Proc. IEEE Computer Security Foundations Workshop, July 2000, pp. 200-214.

[11] S. Zdancewic and A. C. Myers, "Secure information flow and CPS," in Proc. European Symposium on Programming. Apr. 2001, vol. 2028 of LNCS, pp. 46-61, Springer-Verlag.

[12] A. Banerjee and D. A. Naumann, "Secure information flow and pointer confinement in a Java-like language," in Proc. IEEE Computer Security Foundations Workshop, June 2002, pp. 253-267.

[13] F. Pottier and V. Simonet, "Information flow inference for ML," in Proc. ACM Symp. on Principles of Programming Languages, Jan. 2002, pp. 319-330.

[14] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, 21(1), 2003.

[15] R. Wahbe, S. Lucco, T. Anderson, and S. Graham. "Efficient softwarebased

fault isolation," in Proc. ACM Symp. on Operating System Principles, Dec. 1993, pp. 203-216.

[16] G. Morrisett, D. Walker, K. Crary, and N. Glew, "From System F to typed assembly language," ACM TOPLAS, vol. 21, no. 3, pp. 528-569, May 1999.

[17] D. Wagner, Static analysis and computer security: New techniques for software assurance, Ph.D. thesis, University of California at Berkeley, 2000.

[18] G. C. Necula, "Proof-carrying code," in Proc. ACM Symp. on Principles of Programming Languages, Jan. 1997, pp. 106-119.

[19] U. Erlingsson and F. B. Schneider, "SASI enforcement of security policies: A retrospective," in Proc. of the New Security Paradigm Workshop, Sept. 1999, pp. 87-95.

[20] D. Evans and A. Twyman, "Flexible policy-directed code safety," in Proc. IEEE Symp. on Security and Privacy, May 1999, pp. 32-45.

[21] F. B. Schneider, G. Morrisett, and R. Harper, "A language-based approach to security," in Informatics-10 Years Back, 10 Years Ahead, vol. 2000 of LNCS, pp. 86-101. Springer-Verlag, 2000.

[22] Department of Defense, Department of Defense Trusted Computer System Evaluation Criteria, DOD 5200.28-STD (The Orange Book) edition, Dec. 1985.

[23] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," Proc. of the IEEE, vol. 63, no. 9, pp. 1278-1308, Sept. 1975.

[24] M. A. Sheldon and D. K. Gifford, "Static dependent types for first class modules," in Proc. Lisp and Functional Programming, June 1990, pp. 20-29.

[25] H. Xi and F. Pfenning, "Dependent types in practical programming," in Proc. ACM Symp. on Principles of Programming Languages, Jan. 1999, pp. 214-227.

[26] F. Nielson, H. Riis Nielson, and C. Hankin, Principles of Program Analysis, Springer-Verlag, 1999.

[27] D. E. Denning and P. J. Denning, "Certification of programs for secure information flow," Comm. of the ACM, vol. 20, no. 7, pp. 504-513, July 1977.

[28] M. Abadi and A. D. Gordon, "A calculus for cryptographic protocols: The Spi calculus," Information and Computation, vol. 148, no. 1, pp. 1-70, Jan. 1999.

[29] S. Zdancewic and A. C. Myers, "Robust declassification," in Proc. IEEE Computer Security Foundations Workshop, June 2001, pp. 15-23.

[30] C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson, "Static analysis of processes for no read-up and no write-down," in Proc. Foundations of Software Science and Computation Structure. Apr. 1999, number 1578 in LNCS, pp. 120-134, Springer-Verlag.

[31] F. Nielson, H. Riis Nielson, R. R. Hansen, and J. G. Jensen, "Validating firewalls in mobile ambients," in Proc. CONCUR'99. 1999, number 1664 in LNCS, pp. 463-477, Springer-Verlag.

[32] D. Volpano and G. Smith, "Verifying secrets and relative secrecy," in Proc. ACM Symp. on Principles of Programming Languages, Jan. 2000, pp. 268-276.

[33] A. W. Appel, "Foundational Proof-Carrying Code," in 16th Annual IEEE Symposium on Logic in Computer

Science (LICS '01), pp. 247-258, June 2001.

[34] D. Clark, C. Hankin, and S. Hunt, "Information flow for Algol-like languages," Journal of Computer Languages, 2002.

〈著者紹介〉

이 은 영 (Eunyoung Lee)
정회원

1996년 2월: 고려대학교 전산과학과 전산학학사
1998년 8월: 고려대학교 전산과학과 전산학석사
2004년 1월: Princeton University, 전산학박사
2005년 3월~현재 동덕여자대학교 컴퓨터학과 전임강사
〔관심분야〕 프로그래밍언어, 코드보안, 컴파일러