通信情報保護學會誌
第8卷 第4號, 1998. 12

정보통신 기반구조 보호를 위한 보안 커널 개발 동향

이 정 효*, 이 철 원*, 박 정 호*, 이 홍 섭*

A Trend of Security Kernel Development for Protection of Critical Information Infrastructure

Jung-Hyo Lee, Cheol-Won Lee, Chung-Ho Park and Hong-Sub Lee

요 약

미국을 비롯한 정보보호 기술 선진국에서는 일반 기업은 물론 정부 차원에서 안전한 운영체제(Secure Operating System) 개발에 적극적으로 나서고 있다. 운영체제 기술 발전의 흐름에 따라 안전한 운영체제 또한 기존의 IK(Integrated Kernel) 방식보다는 MK(Micro Kernel) 방식으로 개발 경향이 변하고 있다. 미국에서는 '97년 6월에 DTOS(Distributed Trusted Operating System) 프로토타입(prototype) 구현을 끝으로 종료된 Synergy 연구 과제의 후속으로 현재 Flask 프로젝트가 진행 중이다. Flask 커널 역시 이러한 MK를 기반으로 하고 있으며 현재 Flask 보안 커널 개발 프로젝트는 Flux 연구 과제의 주요 주제 중 하나로 진행중이다. DTOS 연구 과제의 명맥을 잇는 Flask 프로젝트는 보안 커널에 대한 적극적이고 지속적인 미국 정부차원의 노력인 것이다.

본 논문에서는 주요 정보통신 기반 구조의 보호를 위한 보안 커널(Security Kernel) 개발 동향을 파악하기 위해 현재 미국에서 진행중인 보안 커널 개발 동향 및 연구 내용에 대해 분석한다.

제 1 장 서론

미국의 HPCC(High Performance Computing and Communication) 프로그램 중의 하나로 NSA(National Security Agency)에서 주도한 Synergy 프로젝트(Synergy research program)가 '97년 6월 DTOS(Distributed Trusted Operating System) 프로토타입(prototype) 구현을 끝으로 종료되었다[1]. Synergy 프로젝트는 보안 정책, 보안 메커니즘, 확장성 있는 보안, 진보된 보안 시스템, 새로운 평가 방법론에 대한 유용한 프로토타입의 생성을 그 목표로 진행되었다. Synergy 프로젝트의 일환으로 진행된 DTOS 과제는 Mach 마이크로커널(microkernel)에 기반한 보안 메커니즘을 적용하여 안전한 운영체제 커널 프로토타입의 개발과 정형화된 수학적 방법을 이용하여 마이크로커널 기반 운영체제의 안전성 보증에 대한 연구

*한국정보보호센터

를 주요 목표로 하였다.

이러한 Synergy 프로젝트의 후속으로 현재 진행중인 Flask(Fluke Security Kernel) 프로젝트에는 Utah 대학교, NSA 및 SCC(Secure Computing Corporation) 등이 적극적으로 참여하고 있다. 현재 Flask 프로젝트는 Utah 대학교의 Flux 팀에 의해 주도적으로 이루어지고 있으며 Flux 프로젝트 가운데 핵심 연구 과제중 하나이기도 하다[2].

Flux 과제는 현재 미국 Utah 대학교와 NSA 및 SCC 등 많은 협력 기관 및 업체들과 공동으로 수행중인 연구 과제로써, 다음의 다섯 가지 주제를 주요 연구 대상으로 한다.

- Fluke/Flask high-security kernel and OS
- Flick IDL(Interface Definitioin Language) compiler
- OS Toolkit
- Java-based Systems: Operating systems, Resource Management and active networks
- Khazana infrastructure for distributed shared state

이와 같이 미국에서는 안전한 운영체제가 국가 정보통신 기반구조를 보호하는 주요 수단임을 인식하고 안전한 운영체제를 개발하려는 노력을 지속적이고도 적극적으로 추진하고 있다[3].

본 논문에서는 미국의 보안 커널 개발 노력의 일환으로 프로토타입이 개발 완료된 DTOS 및 Synergy 프로젝트과 그 후속 연구 과제로 현재 진행 중인 Flask 프로젝트에 대하여 고찰한다. 제2장에서는 미국의 정보통신 기반 구조 보호를 위한 주요 연구 및 개발 활동인 DTOS 개발 현황에 대해서 알아보고 제3장에서는 그 후속 과제로 진행중인 Fluke/Flask 보안 커널 개발 현황 및 Flask 보안 구조에 대해서 분석하며 제4장에서 이러한 노력이 갖는 의미를 피력함으로써 본 논문의 결론을 맺고자 한다.

제 2 장 DTOS 개발 현황

1. DTOS 개발 목표

DTOS는 NSA의 Synergy 프로그램 중의 일부로 강력하고 유연성 있는 보안 통제 제공을 목적으로 만든 운영체제의 프로토타입으로 '97년 6월에 개발 완료되었다[4][5]. Synergy 프로젝트는 차세대 운영체제에 강력한 보안 메커니즘을 포함하도록 운영체제 개발자들을 독려하는 장기적인 전략중의 하나이다. DTOS 프로그램의 주목적은 Synergy 프로그램의 기본적인 요구사항을 만족시키기 위한 것으로 Synergy 구조내 최하위수준의 소프트웨어 구성요소에 대한 프로토타입을 개발하고, 보안 요구사항을 만족시키기 위한 보증 문서와 증거의 제공 즉, Synergy 구조를 따르는 전체 시스템의 보안에 대한 증거를 어떻게 다룰 것인가를 연구하는 것이다[4].

2. DTOS 프로토타입의 목적

DTOS는 High assurance multilevel secure 환경에 사용하기 위해 B3 등급을 목표로 설계된 DTMach(Distributed Trusted Mach)를 대체하는 것으로 안전한 분산 운영체제를 위한 고수준의 설계를 하는 것이었다. DTOS 프로토타입 개발은 거의 마이크로커널과 보안서버(Security Server)의 설계로 집중된다.

마이크로커널은 기존의 Mach 3.0을 개선하는 방향으로 설계되었다. 이는 기존의 마이크로커널에서 요구하는 보안에 대한 필요성이 크게 다르지 않다는 것을 나타낸다. Mach 마이크로커널을 선택한 이유는 하나의 통제 메커니즘으로 거의 모든 시스템 오퍼레이션을 통제할 수 있으며, 널리 사용되는 플랫폼(platform)이라는 점과 함께 유닉스 응용을 지원하며, 소스코드를 별도의 동의 절차 없이 사용할 수 있다는 것 등등 다양하다.

보안서버에는 시스템의 보안정책이 포함되는데 보안정책의 주 기능은 마이크로커널과 같은 다른 서비스를 제공하는 구성요소에 의해 시행되는 보안결정(security decision)을 제공하는 것이다.

프로토타입의 설계 목적은 정책의 유연성을 제공하고 Mach와의 호환성을 유지하며 Mach 커널과 유사한 성능을 갖는 시스템을 구현하는 것이다. DTOS는 위와 같은 요구사항을 충족하며 기존의 Mach에서의 여러 가지 문제점을 개선했다. Mach의 문제점과 이를 해결한 DTOS에 대한 비교 내용은 (표 1)과 같다.

(표 1) Mach의 문제점과 DTOS의 개선 방향

구 분	내 용	세 부 내 용
Mach의 문제점	포트 권한 전송에 대한 제한된 통제	· 전송에 대한 적법성 여부 판단에 대한 메커니즘 제공 결여 · 통제는 중간 대리자를 요구
	객체 관련 서비스의 통제 결여	· 포트권한에 대한 소유는 객체 동작에 대한 모든 권한을 소유하는 결과 발생 · 다중 포트에 대한 부분적인 서비스를 요구
	송신 식별자의 결여	· 메시지 전송자에 대한 식별이 불가능 · 지역 인증은 사용자 레벨의 인증 프로토콜을 요구
	메시지 탈취에 대한 보안 부족	· 신뢰성 있는 경로 메커니즘의 제공이 없음 · 커널에서 메시지 수신자 지정을 명시하는 메커니즘이 없음
DTOS 개선 방향	커널 객체에 대한 보안 정보의 결합	· 보안식별자는 작업, 포트, 메모리공간, 장치를 결합 · 특정 보안식별자를 갖는 객체 생성 가능 · 객체로부터 보안식별자 도출 가능 · 타겟 작업 식별자로부터 객체 보안식별자 도출 가능
	개선된 객체 서비스 통제	· 커널 객체 동작에 대해서 접근벡터를 기반으로 통제 · 수신자는 송신측의 접근통제 벡터 정보의 획득 가능
	보안정책서버와의 상호작용	· 커널은 subject SID와 object SID를 제공함으로써 접근벡터 요구 가능 · 접근벡터 캐쉬 관리 기능 추가(flush, wire 등)
	IPC 통제에 대한 확장 기능	· 수신자는 송신 작업으로부터 보안식별자 도출 가능 · 송신자는 수신자 지정을 명기할 수 있는 기능 · 접근벡터에 대한 IPC 허가여부 검사 가능

3. DTOS 프로토타입 개요

3.1 DTOS 프로토타입 구조

DTOS의 접근방법은 모든 커널 오퍼레이션을 통제하는 일반적인 메커니즘을 제공함으로서 커널 엔티티를 통하여 정의된 많은 정책을 허용하는 것이다. 이 방법의 이점은 상위 계층에서 보안 특성이 침투당하더라도 방어하는 다른 계층을 제공할 수 있다는 것이다. 예를 들면, KeyKOS/KeySAFE 및 TMach는 서버계층의 단일 보안 유지가 실패하면 침입자는 금지된 많은 오퍼레이션을 수행할 수 있는 기회를 제공한다. 이것은 전체 보안을 파괴되는 결과를 초래할 수도 있다[9]. 이

이외에도 커널 기반구조를 채택함으로써 얻을 수 있는 이점으로는 안전한 서버와 응용 내에서 보안 기능을 쉽게 개발할 수 있으며, KeyKOS/KeySAFE나 TMach처럼 특정 보안정책에 국한되지 않고 일반적인 모든 커널 엔티티의 보호를 위한 단일 모델을 제공할 수 있다.

보안정책의 유연성과 localization의 지원을 위해 DTOS 구조는 보안정책 시행과 보안정책 의사결정 간에 엄격한 분리를 제공한다. 그리고 DTOS 커널은 모든 커널 내의 동작을 클라이언트의 접근통제에 대한 보안서버로부터 허가 여부를 참조하여 보안정책을 시행한다.

3.2 커널 확장

보안을 위한 커널 확장의 주목적은 보안서버가 모든 커널 동작에 대한 타당성 여부를 결정하는 기능을 제공하는 것이다. 그러나 보안 커널은 클라이언트나 객체의 실제 신분에 대한 정보를 보안서버에게 요구하지 않고 대신에 각 클라이언트와 객체에 대한 SID(security identifier)를 사용한다.

DTOS 커널에 대한 주요 확장 내용은 다음과 같다:

- 커널은 SID와 커널 객체간의 관계를 관리한다.
- 커널은 각각의 커널 동작에 대한 접근통

(표 2) 확장된 커널 내용

내 용	구 현
보안식별자 관리 (SID Management)	· Mach 인터페이스와 차별화된 객체 생성 인터페이스 기능 · 기본적으로 SID는 변경 불가능하게 관리 · 새로운 객체에 대한 클라이언트의 SID 명명 기능
커널 접근통제 (Kernel Access Control)	· 모든 커널 동작에 대한 접근통제가 가능한 단일 모델 제공 · 클라이언트와 접근되어지는 객체간에 단일 허가 여부 Check 기능 (Initial permission checks) · 커널 내부에서 수행되어지는 중에 허가 여부를 판단해야 하는 경우에 필요한 접근통제 기능(Deferred permission checks)
메모리 접근통제 (Memory Access Controls)	· 메모리의 읽기, 쓰기, 실행에 대한 접근통제 기능 지원
타스크 생성 인터페이스 (Task Creation Interface)	· 타스크가 다른 타스크를 생성하는 인터페이스 지원 · 생성자는 하부 타스크를 새롭게 초기화 할 수 있는 인터페이스 지원
IPC 확장	· 수신자가 수신된 메시지를 통해 송신자의 SID를 도출 가능 · 송신자가 송신 메시지에 대한 SID를 명명할 수 있는 기능
DTOS 커널 구현시 발생하는 장애 (Obstacles)	· 시스템 변화에 따른 기존 응용프로그램의 이식성 · 확장 보완된 시스템 변화에 대한 자세한 지식 습득의 어려움 · 하위 레벨에서의 동작 루틴의 부재 · 구현의 기본이 되는 Mach code의 비합리적인 구조 · 소스 코드의 변경 유무에 대한 식별의 어려움 · Mach 코드의 버그로 인한 DTOS 구현시에 발생하는 버그 수정의 어려움 · Mach 시스템 자체에서 발생하는 문제점과 추가 보완된 기능에서 발생하는 문제점 구별의 어려움 · 시스템 변경으로 생기는 예상치 못한 부작용 · 가정으로 시작한 요구사항들의 기능 테스트 · Desired Robustness of the Prototype

제와 커널 동작에 대한 허가 여부를 결정하기 위해 정보를 가져오는 보안서버와의 인터페이스를 제공한다.

· 커널은 보안서버로부터 하드웨어 기반 메모리 접근통제를 정의하기 위해 보안결정을 사용한다.

DTOS의 상세한 커널 확장 내용은 (표 2)와 같다.

3.3 보안서버 설계 및 구현

보안서버는 DTOS 시스템 내의 보안결정을 하는 구성요소로서 사용자 공간에서 일반적인 Mach 타스크처럼 수행된다. 보안서버의 기능은 SID와 보안 속성간의 사상(mapping)을 제공하고, 특별히 요청된 접근 권한이 두 SID 간에 주어질 수 있는가에 대한 판단을 함으로써 보안결정에 대한 응답을 처리한다.

DTOS 프로토타입은 MLS(Multilevel Secure System)와 TE(Type Enforcement) 보안정책을 구현하였다. 이는 요구되는 다양한 정책을 쉽게 적용할 수 있도록 모듈화된 설계를 하였다. 보안서버의 설계와 구현을 위해서 고려되어야 할 주요 요소는 다음과 같다.

· SID의 해독 : 보안서버는 SID와 보안 속성간의 사상(mapping) 관계를 유지한다. 따라서 SID와 보안 속성간에 변형시키기 위한 인터페이스가 제공되어져야 한다. 보안 커널은 명명된 보안 속성에 대해서는 알 필요가 없기 때문이다. SID는 다음과 같은 보안 속성 필드와 연결된다.
  - 분류자(Classifier) : 정책과는 독립적인 필드
  - 시행 속성 타입(domain or type)
  - 계층적인 보안 레벨 (top secret, secret, confidential, unclassified)
  - MLS 정책을 위한 계층 레벨 변수의 범주 집합
  - 인증 문맥 (사용자 이름 등)

초기에 SID는 보안 속성과 직접적으로 연결이 되었으나 최근 사상(mapping) 구조는 보안서버 내에서만 해석 가능하게 되었다. 이것은 보안서버 이외의 응용 개발자들에 의한 불법적인 사용에 대한 위협 요소를 제거하기 위한 것이다.

· 보안결정 생성 : 보안서버의 핵심은 보안결정 생성 기능이다. 보안서버의 기본형은 속성들의 상호 동작에 대해서 정의된 Hard-code 로직과 명명된 속성간에 상호작용과 특성을 허락하는 보안 데이터베이스 등의 조합을 기반으로 보안결정을 생성한다.

· 안전한 데이터베이스 : 보안정책은 안전한 데이터베이스에 저장되어 있다. 그러나 관리자가 거대한 보안 데이터베이스를 관리하기 위한 도구가 매우 부적절하거나 부족하다. 보안 데이터베이스 사용 도구의 부족은 커널 동작에 대한 높은 수준의 통제를 제공하는 DTOS를 관리할 수 없게 되는 문제가 발생한다.

· 사용자 레벨의 정책 시행 : DTOS 커널 입장에서는 보안정책은 클라이언트 SID와 객체 식별자간에 허가 여부를 판단하는 접근 권한의 집합이다. 이것은 매우 간단한 인터페이스로 처리 가능하지만 상위 계층의 클라이언트에 의해 요구되는 보안정책 결정은 언제나 단순하지 않다. 상위 계층의 보안정책 제안은 동작을 수행할 것인지의 여부를 판단하는 것 대신에 어떻게 수행할 것인가에 대한 정의를 담고 있다. 즉 선택된 암호 알고리즘 또는 레이블된 출력 데이터 등 다양한 선택이 가능하다.

제 3 장 Fluke/Flask 보안 커널 개발 현황

1. Fluke 개발 현황 및 프로세스 모델 특징

1.1 Fluke 개발 현황

Fluke(Flux u-kernel Environment) 프로젝트는 Flask 프로젝트와 함께 현재 진행 중인 Flux 연구 과제의 한 주제로써 Flask 프로젝트가 시작되기 이전에 Fluke 프로젝트는 추진되었고 이 Fluke 커널의 새로운 버전이 Flask인 것이다[2].

Fluke 과제는 현재 미국의 유타(Utah)대학교에서 활발하게 진행되어 현재 프로토타입이 완료되었으며 관련 연구 및 협력 기관과 그 연구 내용은 아래와 같다.

- NSA R32
  - High trust system in/on Fluke
  - Formal Security Policy Model
  - Formal Top Level Specs for kernel, virtual memory manager, process manager
  - Formal method
  - Apply to OS Toolkit

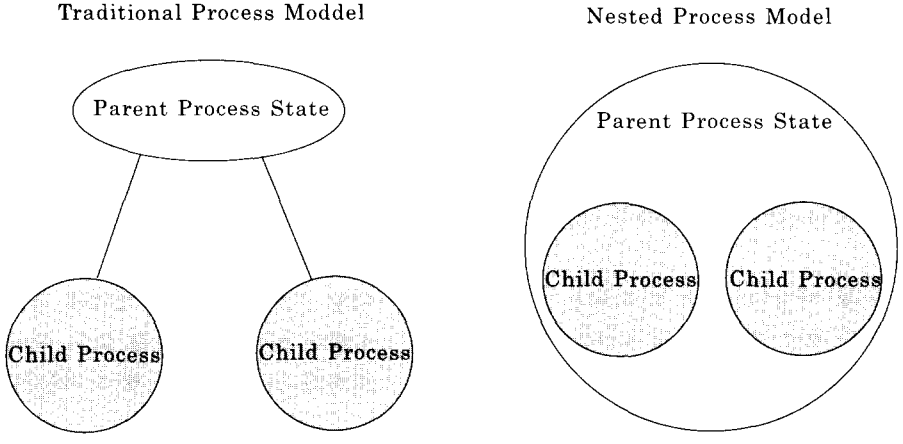
- Secure Computing Corporation
  - Refine PVS composition framework and apply to Fluke nesters
  - Parts of FSPM(Formal Security Policy Model) and FTLS(Formal Top Level Specs)

- 그외 관련 기관으로는 CMU, OGI, CMU RT, DEC SRC, Sandia, CLI 등이 있다.

1.2 Fluke 프로세스 모델 특징

Fluke의 특징 중의 하나는 내포 프로세스 모델(Netsted Process Model)이 적용되었다는 것이다[6]. 기존의 Unix 시스템에서는 부모 프로세스(parent process)와 자식 프로세스(child process) 간의 연계성이 미약하고 자식 프로세스가 소유하는 자원은 부모 프로세스와 독립적으로 운용 가능하게 되어있으나 Fluke에서는 자식 프로세스가 소유하는 모든 자원에 대한 통제권을 부모 프로세스가 가진다. 또한 자식 프로세스에 대한 모든 서비스는 부모 프로세스를 통하여 이루어진다. 부모 프로세스가 통제 가능한 자원은 코드, 데이터, 힙(heap), 스택(stack), 스래드(threads) 등을 포함한다[7].

다음 (그림 1)은 전통적인 프로세스 모델과 비교하여 내포 프로세스 모델(Nested Process



(그림 1) 전통적인 프로세스 모델과 Nested 프로세스 모델의 비교

Model)에서의 부모 프로세스와 자식 프로세스간의 관계를 보여준다.

2. Flask 개발 동향 및 보안 구조

2.1 Flask 개발 동향

미국 Utah 대학교에서는 Flux 프로젝트의 일환으로 Fluke 마이크로커널의 프로토타입을 개발하였으며 Flask는 Fluke 마이크로커널에 보안 기능을 강화한 Fluke 커널의 새로운 버전이다. Flask(Fluke Security Kernel) 연구 과제는 DTOS(Distributed Trusted Operating System) 연구 과제와 DTMach(Distributed Trusted Mach) 연구 과제의 연속성을 띠고 있으며 그 목표와 보안 구조에 있어서 상당한 유사성을 가지고 있다[8].

현재 Flask 연구 과제는 Utah 대학교 Flux 팀과 미 국방부(Department of Defense)의 협력 하에 개발 진행 중이며 DARPA(Defense Advanced Research Projects Agency)에 의해 일부 지원되고 DoA(Department of Army)와 AFRL(Air Force Research Laboratory)에 의해 관리된다[9].

2.2 Flask 목표

Flask는 고수준의 신뢰성 있는 시스템을 지원하는 Fluke(Flux u-kernel Environment)의 한 버전이며 Flask 시스템의 목표는 다양한 보안 정책을 수용할 수 있는 유연성(flexibility)을 지원하는 데 있다.

기본적으로 운영체제가 다양한 보안 정책을 모두 수용한다는 것은 어려운 일이다. 다양한 보안 정책을 수용하기 위해서 다중 보안 정책을 지원한다는 것만으로 해결되지 않는다. 다양한 보안 정책을 수용하기 위해서는 첫째, 보안 정책에 의해 통제되는 high-level functions이 사용하는 low-level objects에 대한 적절한 접근 통제가 이루어져야 한다. 둘째, 접근 권한은 보안 정책과 정확히 일치되어야만 한다. 셋째, 정책은 정적(static)이지 않아야 한다. 정책의 변화 또는 동적 정책(dynamic policy)을 위하여 기존에 허용된 접근 권한에 대한 폐지(revocation)가 이루어져야 한다. 정책 유연성을 지원하는 다른 시스템의 경우 이 세가지 항목중 일부를 적절히 처리하지 못함으로써 문제가 발생하게 된다.

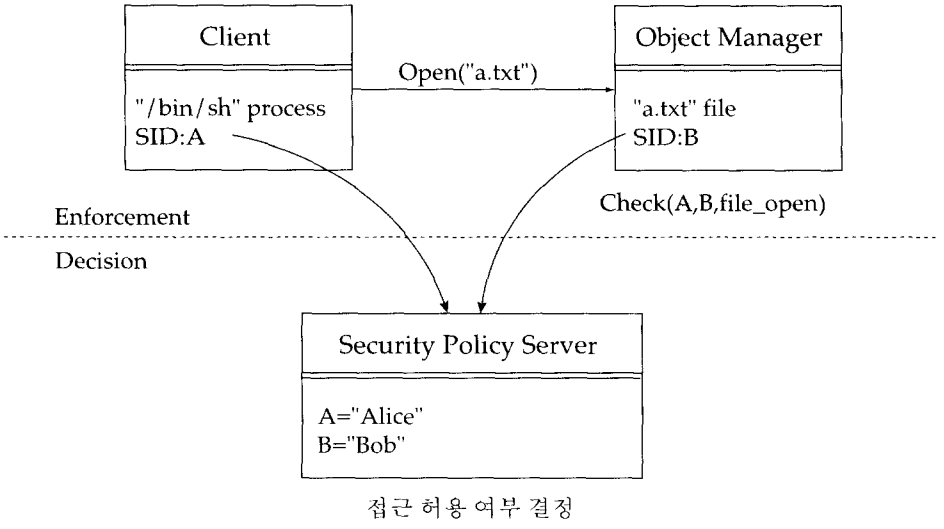
보안 정책의 유연성(Security Policy Flexibility)을 언급하게 되면 우선적으로 다양한 보안 정책의 목록 작성을 떠올릴 수 있을 것이다. 요구되는 다양한 보안 정책들을 목록화 함으로써 필요한 경우 해당 보안 정책을 사용하는 것이다. 그러나 이러한 경우 시스템에서 제공되는 자원의 한계로 인해 현실적으로 요구되는 다양한 보안 정책들을 지원할 수 없게 된다. 그 외에도 정책 유연성 지원에 대한 난제는 산재해 있고 최근에 이러한 보안 정책 유연성을 지원하기 위한 도구 개발 차원에서 Adage [10], ASP(Adaptive Security Policies) [11], Dynamic DTE(Domain Type Enforcement) [12], ARBAC(Administrative Role-based Access Control)[13] 등의 많은 연구가 진행 중에 있다. Flask 연구 과제에서는 정책 수행 메커니즘 자체 및 이 수행 메커니즘과 보안 정책 사이에서 해결되어야할 문제에 중점을 두고 보안 정책의 유연성을 부여하였다.

2.3 Flask 보안 구조

Flask의 보안 구조는 DTOS에서 유래되며 보안 정책 유연성뿐만이 아니고 응용 프로그램 투명성(application transparency), 높은 보안 수준(defense-in-depth), ease of assurance와 관련 정책 변화에 따른 최소한의 코드 변환 등을 그 목적으로 한다. 이와 같은 목적을 제공하기 위하여 보안 구조는 다음 두 가지 요구 사항을 만족해야 한다. 첫째, 객체들(objects)에 대한 모든 접근을 중재할 수 있도록 주체와 객체간의 분리 기능이 있어야 한다. 둘째, 객체에 대한 접근을 시도하는 주체에 대한 안전한 신분확인 기능을 제공하여야

한다.

기본적인 Flask 구조는 파일 관리자, 프로그램 관리자 등과 같은 객체 관리자(object managers)와 보안서버(Security Server)로 구성된다. 이것은 일반적인 마이크로커널(microkernel) 기반의 운영체제와 유사한 구조이다. 객체 관리자는 시스템의 제어 동작을 제공하고 보안서버는 특정 보안 정책에 대한 보안 결정을 담당한다. 객체 관리자는 그러한 보안 결정에 대한 수행을 담당한다. 특정 보안 정책이 요구되어 보안 정책에 대한 변경 사항이 필요하다하더라도 객체 관리자는 보안 정책에 대해 독립적으로 운영되며 보안서버만이 특정 보안 정책을 반영하기 위해 변경될 수 있다. (그림 2)는 이러한 보안 결정 메커니즘의 일례를 나타낸다. 클라이언트(주체)에서 a.txt 파일에 대한 open 접근을 시도하게되면 객체 관리자(이 경우 파일서버)는 보안 정책 서버에게 보안 결정에 대한 질의를 한다. 보안 정책 서버는 주체의 SID와 객체의 SID를 통하여 접근을 요청한 주체의 객체에 대한 허용 여부를 판단하게된다. 그리고 객



(그림 2) 보안 결정 메커니즘의 예

체 관리자는 보안서버의 보안 결정을 근거로 보안 결정에 대한 수행을 담당한다. 여기서 SID는 주체 및 객체에 부여되는 것으로 각 주체 및 객체에 대한 정보를 포함하고 있다.

이러한 기본 구조에 대해 좀 더 복잡한 문제가 발생할 수 있다.

첫째, 통제 허용 여부에 대한 결정은 항상 접근되어지는 객체의 신분(identity)에 의존한다. 따라서 시스템 내의 모든 객체에 대한 정보를 유지하여야 할 필요가 있다. 이러한 정보는 SID(Security Identifier)에 저장되는데 객체 관리자는 이러한 사상(mapping)을 유지하여야 하며 보안서버는 객체가 생성될 때 기본적인 SID를 정의하여야 한다.

둘째, 통제 허용 여부에 대한 결정은 또한 항상 접근을 시도하는 클라이언트(주체)의 신분(identity)에 의존한다. 따라서 마이크로커널(microkernel)은 기본적인 통신 서비스를 제공하여야하며, 이를 통하여 클라이언트와 서버간에 SIDs를 통한 상호 신분 확인 기능을 제공한다.

셋째, 객체 관리자는 접근 허용 여부 결정에 필요한 요구를 최소화하기 위해 사용되는 캐쉬(cache)와 전이된 허용(permission)의 형태(form)에 보안 결정에 대한 지역 복제본을 효과적으로 유지한다. 따라서 보안 정책의 변화는 정책의 표

현(representatioin)이 일관성있게 유지되도록 보안서버와 객체 관리자 사이의 조정(coordination)을 필요로 하는 문제가 발생한다.

2.4 Flask 설계 및 구현

Flask 프로토타입은 Fluke 마이크로커널 기반 운영체제로부터 파생되었다. Flask 구조가 마이크로 커널 기반의 시스템에 국한된 것은 아니지만, 마이크로커널 기반 시스템은 보안, 보증성(Assurability) 및 유연성 관점(flexibility perspective)에 대한 장점이 있다. 마이크로커널에 대한 해결되지 않은 성능 논쟁에도 불구하고 이러한 장점으로 인해 많은 시험이 이루어졌다. 마이크로커널의 보안 기능 수행 메커니즘은 많은 위협들에 대하여 마이크로커널이 직접 처리할 수 있고 서버의 보안 취약점이 발생할 경우에도 강도 높은 보호 수준(defense-in-depth)을 제공한다. 마이크로커널은 각 객체 관리자들이 상호작용을 함에 따라 발생 가능한 보안상의 문제점으로 인해 각 객체 관리자간의 상호신뢰에 제한을 둔다. Flask를 구성하는 각 요소별 기능 및 특징은 다음과 같다.

가. 객체 레이블링(Object Labeling)

보안 정책에 의해 통제되는 모든 객체들은 보안 정책에 의한 보안 문맥(security context)으로 레이블된다. 보안 문맥(security context)의 해석(interpretation)은 정책 의존적이나 사용자 이름, 역할, 보안 레이블(sensitivity labels), 데이터 타입과 같은 특성을 지정할 수도 있다. 객체와 그 객체의 보안 문맥(security context)간의 사상(mapping)은 두 단계를 통해 유지된다. 먼저 보안서버가 정수값인 SID(Security Identifier)를 각 객체의 보안 문맥에 부여하고 객체 관리자는 각 객체와 그 객체 SID간의 사상을 유지한다. SID는 모든 객체 관리자가 보안 문맥의 내용뿐만 아니라 그 형식에도 독립적으로 상호작용 할 수 있도록 한다.

객체가 생성되면 보안 문맥을 표시하는 SID가 부여되는데 이 문맥은 일반적으로 그 객체의 생성을 요청한 클라이언트와 그 객체가 생성된 환경에 종속적이다. 예를 들면, 새롭게 생성된 파일의 보안 문맥은 그 파일이 생성된 디렉토리의 보안 문맥과 해당 객체의 생성을 요청한 클라이언트의 보안 문맥에 종속되는 것이다. 이와 유사하게 전이된 프로세스(transformed process)는 실행 파일의 보안 문맥과 호출 프로세스의 보안 문맥에 의존적이다. 이러한 구조하에서 어떤 보안 관련 사건에 대한 보안 결정은 객체 관리자가 객체의 타입과 관련 객체의 SID를 보안서버에게 제공함과 동시에 보안서버의 보안 결정을 요청하게되고 보안서버는 이러한 SID를 기반으로 보안 결정을 하게된다.

나. 보안 정책 수행(Policy Enforcement)

본 절에서는 위에서 언급한 Flask에서 요구되는 객체 관리자들의 개별적인 특성에 대해 설명한다. 설정된 보안 정책의 수행은 마이크로커널, 가상 메모리 관리자, 파일서버, 프로세서 관리자, 네트워크서버 등을 통해서 이루어진다.

· 마이크로커널(microkernel)

Flask 마이크로커널의 모든 상태는 다음의 여섯 가지 기본적인 객체 형태에 대해서 세 가지 서비스 형태를 띤다.

서비스 형태	객체 형태
수행	스레드(threads)
메모리 관리	주소 공간, 맵핑(mappings), 영역(regions)
IPC	port, port sets

메모리 그 자체가 마이크로커널 내의 객체가 아니라 하더라도 마이크로커널은 각 메모리 세그먼트(segment)에 대한 메모리 관리와 SID를 연계시키는 서비스를 제공한다. 각 객체의 SID는 그

객체와 관련된 메모리 세그먼트의 SID와 동일하다. SID는 일단 설정되면 변경될 수 없다.

마이크로커널에서는 몇 가지 기본적인 서비스에 대한 통제가 이루어진다. 마이크로커널의 객체에 대한 생성, 읽기, 쓰기 오퍼레이션은 각각 Create, Read, Write 권한을 통해 통제되고 DMA(Direct Memory Access)는 Create, Read, Execute 권한으로 통제된다.

마이크로커널 통제에 있어서 가장 두드러진 특징은 객체간의 관계를 고려하는 것이다. 마이크로커널의 한 객체에 대한 상태(state) 설정은 간접적으로 다른 객체의 상태에도 영향을 미칠 수 있다. 예를 들면 한 스래드(thread)의 상태를 설정함으로써 그 스래드가 주소 공간(address space)을 할당받게 되면 그것은 단순히 그 스래드의 주소공간 부여의 의미만으로 끝나는 것이 아니라, 그러한 오퍼레이션은 당연히 주소 공간에 대한 영향을 미치게 되는 것이다. 다시 말하면, 주체가 주소 공간에 자신이 소유한 Write 권한을 역시 자신의 소유인 스래드에 전파하는 것은 지나치게 관대한 권한을 제공하는 것이 될 수도 있으므로 스래드와 주소 공간간의 관계는 명확하게 통제되어야 하는 것이다. 스래드와 주소 공간의 관계처럼 한 객체가 필수적으로 다른 객체에 소유되는 경우에는 마이크로커널은 두 객체가 동일한 SID를 소유하였음을 확인한다. 그 외의 경우 두 객체간의 관계는 명확한 권한(permission)에 의해 통제된다. 예를 들어, 한 주소 공간에서 다른 주소 공간의 기억 사상(memory mapping)을 통제하는 두 주소 공간간의 권한(permission)의 경우, 이 동작은 두 주소 공간간의 수령자(recipient) 주소 공간으로의 Write 권한과 Map 권한의 두 가지 권한(permission)을 요구한다.

· 가상 메모리 관리자(Virtual Memory Manager)

Flask 메모리 관리자 인터페이스는 응용 프로그램을 위한 두 가지의 고수준 추상화(high-level abstractions)를 제공한다. Segment와 Mempool이 그것이며, Segment는 고정된 크기일수도 있고 제한된 크기만큼 커질 수도 있는 메모리의 연속된 조각들이며 Mempool은 계층적인 자원 통제 메커니즘(hierarchical resource control mechanism)이다. Flask 가상 메모리 관리자(VMM)은 사용자 모드(User Mode) 프로세스이다.

Flask 구조에 있어서 가장 중요한 문제 중 하나는 물리적 메모리의 관리이다. 커널은 Segment를 통하여 물리적 메모리의 페이지(page)를 통제한다. 가상 메모리 관리자(VMM)가 Segment에게 SID를 부여하는 경우, 가상 메모리 관리자는 해당 Segment에 발생할 수도 있는 Page Fault를 해결하고 정상적으로 레이블된 물리적 메모리를 획득하기 위해 커널을 호출해야만 한다. 따라서 가상 메모리 관리자의 물리적 메모리는 수많은 다른 물리적 페이지(page)의 그룹으로 나누어져 있다. 그러한 그룹간의 물리적 메모리의 이동은 한 SID의 메모리를 커널에게 돌려주고 다른 SID로써 메모리를 할당해야 하므로 상대적으로 고비용을 요구하게된다. 따라서 그룹간의 적절한 균형과 SID간의 페이지 교체를 최소화하는 것이 중요하다.

· 파일서버(File Server)

Flask 파일서버는 파일 시스템, 디렉토리, 파일, 오픈 파일 등의 네 가지의 레이블된 객체를 제공한다. 모든 파일 시스템은 mounting/unmounting과 같은 오퍼레이션을 제어하기 위해서 뿐만 아니라 파일 시스템 내의 모든 파일의 총체적인 레이블(label)을 나타내기 위해서 레이블되어야 한다. 오픈 파일 객체에 대한 통제는 오픈 파일 객체에 대한 접근의 전파(propagation)가 보안 정책에 의해 통제되어질 수 있도록 그 파일들 자체에 대한 통제와는 분리된다.

대부분의 파일 시스템 오퍼레이션은 읽기, 쓰기 오퍼레이션처럼 개념적으로 단순하지가 않기 때문에 파일서버에 의해 수행되는 독특한 접근허용 모드가 많다. 예를 들면 디렉토리 오퍼레이션

만을 위한 접근허용 모드만도 24가지가 있다. 보안서버는 결정에 대한 요청이 발생하면 이러한 모든 접근허용 여부에 대해서 결정을 할 수 있기 때문에 이러한 독특한 접근허용 모드는 추가적인 성능상 영향력을 최소화하면서 세밀한 접근 통제(fine-grained access control)를 가능하게 한다. 그러나 이러한 것으로 인해 보안 정책의 저수준의 스펙(low-level specification)이 복잡해지는 단점이 있다.

파일은 지속적으로 유지되므로 해당 파일의 레이블 역시 지속적으로 유지되어야 한다. 이것은 파일 시스템에서 사용되는 각 보안 문맥(security context)과 파일 내에 지속적으로 유지되는 SID 사이의 강력한 사상(mapping)을 통해서 이루어진다. 파일서버가 이 사상을 유지한다 하더라도 파일서버는 그 해당 파일과 관련된 보안 문맥을 해석하지는 않는다.

· 프로세스 관리자(Process Manager)

Flask 프로세스 관리자는 POSIX(Portable Operating System Interface for Computer Environment) 프로세스 개념이 구현된 사용자 모드(User Mode) 프로세스이다. Fluke 프로세스의 SID와는 달리 POSIX 프로세스의 SID는 execve를 통해 변경될 수도 있다. 그러한 SID의 전이(transition)는 기존의 SIDs와 새로운 SIDs 간의 Transition 권한에 의해 통제된다. 기본적인 전이는 DOLM(Default Object Labeling Mechanism)에 의해 정의된다.

프로세스 관리자는 가상 메모리 관리자, 파일서버 및 마이크로커널과 연계되어 각 POSIX 프로세스가 안전하게 초기화되도록 한다. 파일서버는 실행영역의 메모리가 해당 파일의 SID로 레이블되도록 한다. 마이크로커널은 Execute access에 대해서 메모리가 실행만을 수행하도록 한다. 또한 프로세스 관리자는 변형된 POSIX 프로세스의 상태에 대한 초기화를 담당한다.

· 네트워크서버(Network Server)

Flask 네트워크서버 역시 사용자 모드 프로세스이며 BSD 소켓(socket) 시스템 호출 인터페이스 기반을 통하여 네트워크 접근을 제공한다. 네트워크서버는 Flux 프로젝트의 하나인 OS Kit으로부터 네트워크 프로토콜(protocol)을 추출하여 사용한다. 네트워크서버는 보안 정책에 의해 모든 IPC(Interprocess Communications)의 정당성 여부를 확인한다.

네트워크서버의 주요 통제 대상은 소켓이다. 네트워크서버는 SID와 각 소켓을 연관시킨다. 소켓에 사용되는 기본적인 SID는 그 소켓의 생성 방식에 따라 다르다. 만약 소켓이 로컬(local) 프로세스에 의해서 생성되었다면 기본적으로 생성 프로세스의 SID를 부여받고 기존에 형성된 소켓을 통해 새로운 연결의 결과로 생성되었다면 기본적으로 기존 소켓의 SID를 부여 받게된다.

2.5 Security Server

보안서버는 보안 정책 결정(security policy decision)을 하고, SIDs와 보안 문맥(security context)간의 사상(mapping)을 유지하고, 새로 생성되는 객체에 SIDs를 부여하고, 객체 관리자 AVC(Access Vector Caches) 관리 및 정책의 적용과 변경 등의 기능을 제공한다. 시스템 내에서 발생하는 많은 오퍼레이션이 보안서버의 결정 결과에 종속적인 경향이 강하기 때문에 많은 경우 보안서버가 결정 사항에 대한 caching을 객체 관리자의 AVC(Access Vector Cache)에 의해 수행되는 caching에게 제공하는 것이 유리하다. 보안서버는 전형적으로 보안서버의 서비스에 대한 정책 수행을 담당한다.

보안서버에 의해 제공되는 보안 정책은 보안서버의 코드와 보안 데이터베이스의 조합에 의해 결정된다. 보안 정책의 변화는 기존의 보안 정책 데이터베이스를 변경함으로써 이루어질 수도 있으나 경우에 따라 보안서버의 일부 코드의 변경 또는 전체 보안서버 자체의 교체를 요하는 경우

도 발생할 수 있다.

Flask 보안서버 프로토타입은 MLS(Multi-level Security), Type Enforcement, 신분 기반 접근 통제(Identity-based Access Control), 동적 역할 기반 접근 통제(Dynamic Role-based Access Control) 등 네 가지 세부 정책의 조합으로 보안 정책을 수행한다. 보안서버에 의해 제공되는 접근 결정은 이 네 가지의 세부 정책을 만족해야만 한다. Flask 보안서버는 MLS의 구현과 동적인 역할 기반 접근 통제의 지원, 신분 기반 접근 통제의 보강된 지원 등의 측면에서 DTOS의 보안서버와는 다르다. 정책 변경 사항이 세부 정책들간의 상호의존성과 일관성 있게 유지되는한 보안서버내의 세부 정책의 변경은 간단하게 이루어질 수 있다.

MLS 정책은 다중 등급 주체(multi-level subjects)와 주체간의 상호작용(subject-to-subject interactions)을 지원하는 BLP(Bell & LaPadual) 모델의 한 형태이다. 보안 정책 데이터베이스를 통해서 사용자와 해당 MLS 범위(ranges)를 결정할 수 있다. Source MLS 범위와 Target MLS 범위간의 관계를 결정함으로써 MLS 접근 결정이 이루어진다.

TE(Type Enforcement) 정책은 타입에 기반한 접근 행렬(matrix)을 명확하게 정의한다. 전형적인 TE 표현과는 달리 TE 세부 정책에서는 도메인(domain)과 타입간의 구분이 없다. 정책 데이터베이스는 사용자와 관련된 타입을 정의하고 또한 각 Source type과 Target type쌍에 대한 사용이 허용된 모든 AVP(Access Vector Permission)를 정의한다. 이러한 TE 접근 규칙은 전통적인 TE의 DDT(Domain Definitoin Table)와 DTT(Domain Transition Table)를 포함한다. Source type와 Target type 쌍의 허용된 권한을 통해 TE 접근 결정은 이루어진다.

신분기반 접근통제 정책은 사용자 신원의 특성(attributes)에 기반하여 개개의 접근 결정이 이루어진다. 이러한 구속력은 모든 접근 결정에 부여될 수도 있고 접근 결정을 제한할 수도 있다. 또한 권한의 허가 여부를 결정하기 위해 Source User Identity와 Target User Identity 사이에 규정되어야 하는 관계를 지정하기도 한다.

동적 역할 기반 접근 통제 역시 유사한 강제성을 띠고 있으나 이것은 보안 문맥의 역할 특성(role attribute)에 기반 한다. 보안서버는 각 SID 별로 역할 특성을 연관시킨다. 보안 정책 데이터베이스는 역할의 집합과 역할 기반의 규칙에 대해 파셜오더(Partial Order)를 정의한다. 역할 기반 규칙은 역할간의 관계가 파셜오더에 기반 하였다는 것만을 제외하면 신분 기반 규칙과 유사하다.

제 4 장 결 론

본 논문에서는 미국 NSA와 Utah대학교에서 주도적으로 추진중인 DTOS 및 Flask 프로젝트에 대해서 소개하였다. 미국을 비롯한 정보보호 기술 선진국의 경우 단일 시스템 내부에서의 보안성 강화는 물론 네트워크, 데이터 베이스 등의 정보자원 보호에 대한 총체적인 솔루션(Total Solution)을 제공하기 위해 노력하고 있으며 그 노력은 일반 기업체는 물론 정부 차원에서도 활발하게 진행되고 있다. 이러한 노력은 향후 국가적 정보통신 기반구조의 보호를 위한 주요한 수단으로 활용될 것이다.

앞에서도 언급되었듯이 미국의 경우, 정보보호의 기반이 되는 운영체제 보안 기술은 Synergy 연구 과제의 핵심인 DTOS 개발의 명맥을 이어 Flask 프로젝트의 형태로 그 노력은 지속적이고 활발하게 이루어지고 있다.

정보보호 제품 및 기술, 특히 안전한 운영체제 개발 기술은 여타의 제품 및 기술과 달리 국가간 상호 배타적 성격을 지니고 있으며 기술 자체를 무기화 할 수 있는 특성으로 인해 자국의 안보와도 직결된다고 할 수 있다. 따라서 21세기 국가 경쟁력 강화는 물론 자국의 안보 차원에서 정보보호 기술의 핵심이 되는 안전한 운영체제 기술의 적극적인 연구와 개발 활동이 시급히 요구된다고 사료된다.

참고 문헌

[1] http://www.sctc.com/randt/HTML/dtos.html

[2] Jay Lepreau, "The Flux Project" Presentation materials, Univ. of Utah, December 11, 1996.

[3] http://beta.missilab.com, MISSI Introduction, 1996.

[4] 김학범 외, "운영체제 보안 기술동향", 통신정보보호학회지, 제8권 제2호.

[5] Secure Computing Corporation, "DTOS Lessons Learned Report", 27. June, 1997.

[6] Bryan Ford, Mike Hibler, Jay Lepreau, Patrick Tullmann, "Microkernels Meet Recursive Virtual Machines", Univ. of Utah, October, 1996.

[7] Patrick Tullmann, Jay Lepreau, Bryan Ford, Mike Hibler, " User-level Checkpointing Through Exportable Kernel State", Univ. of Utah, October 28, 1996.

[8] Ray Spencer, Stephen Smalley, Peter Loscocco, Mike Hibler, David Andersen, Jay Lepreau, " Flask Security Architecture : System Support for Diverse Security Policies", Univ. of Utah Technical Report UUCS-98-014, August, 1998.

[9] http://www.cs.utah.edu/projects/flux/fluke.html

[10] M. E. Zurko and R. Simon, "User-Centred Security", In proc. of the New Security Paradigms Workshop, Sept. 1996.

[11] M. Carney and B. Loe, "A Comparision of Methods for Implementing Adapative Security Policies", In Proc. of the Seventh USENIX Security Symp., Jan. 1998.

[12] T. Fraster and L. Badger, "Ensuring Continuity During Dynamic Security Policy Reconfiguration in DTE", In Proc. of the 1998 IEEE Symp. on Security and Privacy, May 1998.

[13] S. G. Ravi Sandhu, Venkata Bhamidipati and C. Youman, "The ARBAC97 Model for Role-Based Administration of Roles: Preliminary Description and Outline", In Proc. of the Second ACM Workshop on Role-Based Access Control, Nov. 1997.

□ 著者紹介

이 정 효

1995년 2월 경북대학교 전자공학과 (공학사)
1995년 1월~1997년 5월 (주)삼성전자 연구원
1997년 7월 ~ 현재 한국정보보호센터 연구원

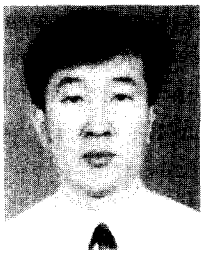
※ 주관심분야: 컴퓨터 · 네트워크 보안, 안전한 운영체계(Secure OS), 침입탐지

□ 著者紹介

이 철 원

1987년 2월 충남대학교 수학과 (학사)
1989년 8월 중앙대학교 대학원 전산학과(석사)
1989년 9월~1996년 6월한국전자통신연구소 선임연구원
1996년 6월 ~ 현재 한국정보보호센터 선임연구원

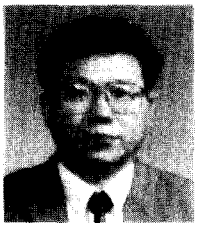
※ 주관심분야: 컴퓨터 · 네트워크 보안, 정보보호시스템 평가체제, 정보보호기술 표준화



박 정 호

1984년 한양대학교 산업공학과 (공학사)
1986년 한양대학교 산업공학과 대학원 (공학석사)
1990년 7월 ~ 1992년 9월 (주)데이콤 연구원
1992년10월 ~ 1996년 5월 한국전산원 선임연구원
1996년 5월 ~ 현재 한국정보보호센터 선임연구원

※ 주관심분야: 컴퓨터 · 네트워크 보안



이 홍 섭

1979년 2월 한양대학교 (학사, 전자공학)
1985년 2월 한양대학교 (석사, 전자공학)
1980년 ~ 1996년 한국전자통신연구소, 연구원 ~ 책임연구원 , 실장
1996년 ~ 현재 한국정보보호센터 연구개발부장, 기술본부장
정보통신기술협회 정보보호분과위원회 의장
한국통신정보보호학회 상임이사

※ 주관심분야 : 시스템 및 네트워크 정보보호