안전한 임베디드 시스템을 위한 E-BLP 보안 모델의 구현

(Implementation of the E-BLP Security Model for Trusted Embedded Systems)

강 정 민 † 남 택 준 † 장 인 숙 † 이 진 석 †
(Jungmin Kang) (Taekjun Nam) (Insook Jang) (Jinseok Lee)

요 약 E-BLP(Extended BLP) 보안모델은 시스템 내에서 실질적인 주체가 되는 프로세스에 대한 신뢰성을 고려한 BLP 모델의 확장이다. 본 논문은 임베디드 시스템의 보안을 위하여 E-BLP 보안모델을 구현한 내용을 다룬다. 구현된 EBSM(E-BLP Based Security Module)은 식별인증, 접근통제 및 프로세스의 동적 신뢰성 검사를 위한 DRC(Dynamic Reliability Check) 부분으로 구성된다. EBSM의 접근통제에 의해 악의적인 프로세스의 중요객체에 대한 접근 시도가 차단되는 것과, 정상적인 내부사용자가 불법적인 루트권한 탈취를 위해 버퍼 오버플로우 공격을 하는 경우 DRC에 의해 차단되는 것을 보인다. 또한 EBSM을 적용한 임베디드 시스템의 성능 오버헤드에 대해 소개한다.

키워드 : 임베디드 시스템, 식별인증, 접근통제, 버퍼 오버플로우

Abstract E-BLP security model considers the reliability of the processes that are real subjects in systems. This paper deals with the implementation of the E-BLP model for secure embedded systems. Implemented EBSM(E-BLP Based Security Module) consists of three components: identification and authentication, access control and DRC(Dynamic Reliability Check) that checks the process behavior dynamically. Access Control of EBSM ensures unreliable processes not to access the sensitive objects and the DRC detects the buffer overflow attack by normal user. Besides, the performance overhead of the embedded system applying the EBSM is introduced.

Key words : Embedded System, Identification and Authentication, Access Control, Buffer Overflow

1. 서 론

정보화의 발달은 개인에게 여러 정보기기의 소유를 가능하게 하였으며, 아울러 임베디드 시스템의 발전을 가능하게 했다. 임베디드 시스템에서의 보안 노력은 현재 전개되고 있는 유비쿼터스 환경을 안전하게 하는 초석이 될 것이다. 본 논문의 구성은 다음과 같다. 2장에서는 임베디드 보안에 대한 관련연구를 살펴보고, 3장에서는 E-BLP 보안 모델에 대해 설명한다. 4장에서는 임베디드 시스템에 E-BLP 모델을 구현한 내용과 성능에 대해 기술한다.

† 정 회 원 : 국가보안기술연구소 연구원
jmkang@etri.re.kr
tjnam@etri.re.kr
jis@etri.re.kr
jinslee@etri.re.kr
논문접수 : 2005년 3월 10일
심사완료 : 2005년 6월 10일

2. 관련연구

특정한 목적을 위해 전용 하드웨어와 이를 제어하기 위한 프로그램으로 구성된 시스템을 임베디드 시스템이라고 한다. 본 장에서는 이러한 임베디드 시스템에서의 보안에 대해서 살펴본다.

2.1 임베디드 보안

초창기에는 임베디드 시스템이 매우 제한적인 목적으로 사용되었기 때문에 임베디드 시스템에서 사용되는 프로그램의 기능이 매우 단순하였다. 그러나 임베디드 시스템에서 사용되는 프로그램의 기능이 많아지고 복잡해짐에 따라 이를 위한 운영체제가 도입되었다. 이후 상대적으로 공격에 노출될 위험이 높아지게 되었는데, 특히 유·무선 등 네트워킹 기능을 제공하는 임베디드 프로그램은 공격에 매우 취약하고 치명적인 약점을 가지고 있다. 이러한 약점을 바이러스와 해킹 공격과 같은 위협으로부터 보호하기 위한 노력이 필요하며, 이를 위

해서 임베디드 시스템에 사용하는 운영체제는 식별인증, 접근통제, 침입방지, 자체보호, 보안감사, 보안관리와 같은 보안기능도 고려해서 개발해야 된다. 또한 각 임베디드 프로그램의 목적에 따라서 방화벽, 침입차단, 바이러스 탐지 및 차단과 같은 기능을 용이하고도 효과적으로 수용할 수 있도록 임베디드 보안 미들웨어 플랫폼 및 임베디드 보안 프로그램에 대한 기술 개발의 필요성도 매우 높아지고 있다[1-3].

현재 정부에서는 무선인터넷 보안을 강화하기 위해서 휴대폰에 백신 탑재를 의무화하려는 움직임이 있고, 안철수 연구소, 하우리, 시만텍 등 국내외 안티바이러스 업체들도 모바일 백신 개발에 나서고 있다.

2.1.1 임베디드 시스템에서의 보안 고려 사항

[4]에서는 그림 1과 같이 계층별로 임베디드 시스템에서 제공해야 될 보안 고려사항을 제시하고 있다. 표 1은 제시된 보안 고려사항들에서 몇 가지 항목을 정리한 것이다.

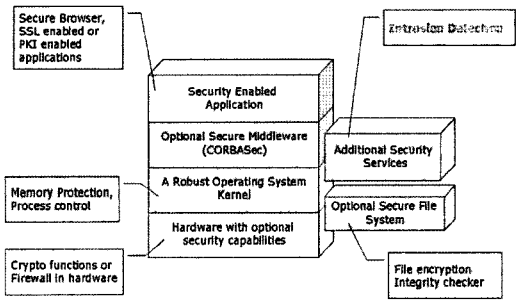


그림 1 임베디드 시스템을 위한 보안 구조

표 1 임베디드 시스템에서의 보안 고려 사항

종 류	내 용	기 법
접근 통제	임베디드 시스템으로 다운로드 되고 실행되는 많은 코드들이 악의적으로 사용될 수 있으므로 이러한 객체들에 알맞은 등급을 부여하고, 그 등급 밖에 있는 주체에 의한 시스템 자원과 같은 객체들에 대한 접근을 통제하는 기술	BLP, RBAC
다양한 환경	여러 복잡한 프로토콜이 사용되면서 그들 사이의 보안대책을 세우기 어려운 문제점	사용자 인증
리소스 보호	객체 재사용, 제한된 메모리 보호, 프로세스 생성 제한, 보안처리를 위한 컴퓨팅 파워의 소모, 보안처리를 위한 전력 소모	객체 재사용 방지 및 관리
파일 시스템 보호	임베디스 시스템의 이동성에 의해서 발생할 수 있는 분실에 대비하기 위해서 분실 시에도 파일 시스템에 있는 기밀 자료를 보호하는 기술	암호화, 개인 토큰 사용

2.1.2 임베디드 시스템에서의 침해 사례

임베디드 기기 중 전용 악성코드가 가장 먼저 등장한 것은 개인휴대정보단말기(PDA)이다. 그러나 PDA는 단순한 기능만을 수행하기 때문에 이를 기반으로 한 침해사례 역시 미미한 단계이다.

휴대폰에 바이러스가 처음 등장한 것은 2000년 노르웨이에서 특정 문자메시지가 노키아 휴대폰에 전달될 때 작동을 정지시킨 사건이다. 일본에서는 2001년 수백대의 휴대폰에 경찰 응급전화번호로 전화를 걸도록 하는 휴대폰 악성코드가 발견되기도 하였다. 2003년 12월에 4건의 휴대폰에 대한 바이러스 형태의 공격이 보고되었고, 팜 계열의 모바일 장비에 3건의 바이러스 또는 트로이 목마와 유사한 공격이 보고 되었다. 이러한 휴대폰에서의 악성코드도 휴대폰의 오작동을 유발하는 수준을 벗어나지는 못했다.

그러나 최근 차세대 휴대폰 운영체제인 심비안(Symbian)을 탑재한 스마트 폰에 카비르(Cabir) 웜이 등장하면서 휴대폰에서의 보안에 대한 인식이 달라지고 있다. 이 웜은 심비안을 탑재한 단말기의 블루투스 장치를 이용하여 다른 휴대폰으로 침투 할 수 있는 기능을 가지고 있다. 브라질 프로그래머가 카비르 소스를 인터넷에 공개해서 다양한 변종이 2005년 들어서 계속 발견되고 있다. 임베디드 기기중 가장 많이 사용되고 퍼져있는 휴대폰에 치명적인 바이러스가 유포된다면 종전의 인터넷 대란보다 더욱 치명적인 사회문제가 발생할 수도 있을 것이다.

2.1.3 임베디드 시스템에서의 접근통제 구현사례

아직껏 임베디드 환경에서 접근통제를 구현한 사례는 거의 없다. 여기서는 최근 E-RBAC 접근 통제 모델을 임베디드 환경에 구현한 사례를 소개한다[5].

E-RBAC은 커널이 실행할 연산(operation) 각각의 적법성 여부를 검사할 뿐만 아니라, 연산들 사이의 연관관계를 바탕으로 연산집합 전체의 수행이 시스템에 어떤 영향을 주는지를 검사하므로, 시스템 공격을 보다 효율적으로 막는 접근통제 기법이다. 이를 위해 E-RBAC 커널은 연산들 사이의 연관관계 정보를 절차(procedure)의 형태로 저장하며, 특정 연산의 수행이 어떤 절차의 실행에 해당하는 지를 파악해야 한다. E-RBAC의 효과적이고 체계적 구현을 위하여 CCPN(Constrained Coloured Petri Net) 모델을 구현하였다. CCPN은 E-RBAC의 정형화를 위하여 Coloured Petri Net 모델을 확장해 제안된 수학적 모델이며, E-RBAC의 절차적 요소의 표현을 위한 상태기계적 특성과 접근통제행렬 표현을 위한 집합명세적 특성을 동시에 가지는 일종의 혼합정형기법이다.

구현은 국내업체에서 개발한 IFC-ETK100 임베디드

보드에서 진행되었으며, MMU가 없는 하드웨어적 특성을 고려하여 uCLinux 커널을 이용하였다. 또한 CCPN의 절차적 요소로 인하여 커널 수행에 성능부하가 발생할 것이 예상되어, 이를 최소화하기 위해 다음과 같은 방법을 선택하였다. 첫째, 커널을 직접 수정하였다. 둘째, 구현 시 포함될 상태기계의 상태공간 탐색 오버헤드를 줄이기 위하여 자료구조 표현을 간소화 하였다. 접근 규칙 및 절차는 배열과 벡터연산으로 표현하며, 규칙의 매칭과정은 벡터의 비트연산으로 표현하였다. E-RBAC을 구현한 임베디드 타겟 보드 상에서의 연산의 실행시간 측정 결과는 약 10%의 부하가 발생한다.

3. E-BLP 보안 모델

E-BLP 보안 모델은 BLP 모델을 확장하여 비 신뢰적인 프로세스에 의한 중요 객체로의 접근을 차단하고, 프로세스의 악의적인 행위를 차단하기위해 본 저자들에 의해서 제안된 모델이다[6-10].

3.1 구성요소

3.1.1 사용자: U

- **U**: 사용자의 집합
- **User Labels** = User Levels × P(Categories)
  단, P(x)는 임의의 집합 x에 대한 멱집합.
- **User Levels** = {Top Secret, Secret, Confidential, Classified, Unclassified, Anonymous}
- **Categories**: 사용자 혹은 객체가 속하는 부서의 집합으로 예를 들면 {dept A, dept B, dept C, ...}이다.

임의의 사용자를 위한 보안 레이블은 하나의 보안 등급(level)과 0개 이상의 부서(category)정보, 즉 0인 경우는 Ø이 되며 그렇지 않은 경우에는 이를 포함하는 부서의 집합이 된다. 사용자의 부서정보가 없을 때 보안 레이블 정보는 보안 등급 정보만을 갖게 되며 부서정보는 Ø이 되어 (User Level, Ø)로 표시된다. 사용자 보안 레이블 정보는 보안 관리자에 의해 정의되어 SKDB (Security Kernel Data Base)에 안전하게 저장되며, 사용자 로그인 허용유무 결정을 위한 정보로서 사용되고, 사용자가 로그인 시 제출한 보안 레이블 정보는 프로세스에 상속되어 접근 통제를 위한 정보로서 사용된다. E-BLP 보안 모델에서의 ***Anonymous*** 등급은 시스템 내의 등급화 된 사용자가 아닌 외부의 사용자들을 위해 존재하며, 공유(Shared) 객체만을 접근하도록 제한된다.

3.1.2 프로그램: PG

- **PG** = Common ∪ Public : 프로그램의 집합
  단, Common ∩ Public = Ø
- PG_E: 실행중인 프로그램, 즉 프로세스의 집합.
  Common 영역의 프로그램들은 Anonymous 사용자를 제외한 사용자들에 의해서 실행되며 기존 BLP 모델의 접근 규칙을 적용받게 된다. 반면 Public영역의 프로그램들은 공유 객체로의 접근만을 허용한다. 또한 두 영역의 프로그램들은 실행 시 프로세스의 신뢰성을 보장하기 위해 ***DRC(Dynamic Reliability Check)***에 의해 검사된다. 정적으로 분류되는 두 영역의 프로그램들은 실행 시 예기치 않은 악의적인 행위를 할 수 있다. 다음 절에서는 프로세스의 신뢰성에 대한 언급을 한다.

3.1.3 프로세스 신뢰성: C

- **C** = **{Trust, Untrust}**: 프로세스 신뢰성

동적 프로세스의 행위, 즉 신뢰성 판단은 알려져 있는 취약점 (예: 스택버퍼오버플로우 취약점 등)정보를 기반으로 임의의 프로세스가 수행 중 악의적인 행위를 한다면 C의 값은 untrust가 되며, 그렇지 않다면 trust가 된다. 예를 들어 스택 오버플로우를 일으켜 스택 버퍼에 삽입된 쉘을 실행하여 루트권한을 탈취하는 프로세스의 동작은 untrust로 간주된다.

3.1.4 접근 주체: S

- **S** = **U** × PG_E: 접근주체의 집합

접근 통제 시 사용자와 동적인 주체가 되는 프로세스를 주체로서 정의한다. 이는 동적인 프로세스의 행위를 고려하기 위함이다.

3.1.5 접근 객체: O

- O: 접근 객체의 집합
- **Object Labels** = Object Levels × P(Categories)
- **Object Levels** = {Top Secret, Secret, Confidential, Classified, Unclassified, ***Shared***}

사용자의 보안 레이블 정보와 마찬가지로 객체의 보안 레이블 정보는 등급과 부서정보로서 구성이 되며 보안 관리자에 의해 결정된다. 가용성과 중요한 객체들 (Top Secret ~ Unclassified)의 보호를 위해 Public 영역 프로세스의 접근은 Shared 객체로 제한된다.

3.1.6 접근 동작: A

- A = { r, w, a, e }
- r: 객체 정보 읽기.
- w: 객체 정보의 읽기/쓰기.
- a: 객체 정보를 읽을 수 없고, 새로운 정보 추가.
- e: 실행 가능 접근 객체를 실행.

접근 주체의 접근 객체에 대한 접근 요구는 위의 동작들 중 하나에 의해서 발생 한다

3.2 시스템 상태

- **V** = **(B, M, F)**는 시스템 상태의 집합을 나타낸다.
- **B** = **S** × **O** × **A**는 접근 주체 S가 접근 객체 O를 A의 접근 동작으로 접근하는 것을 의미한다.
- **M**: 접근 주체(사용자)가 접근 객체에 대해 허가된 동작을 명시하는 접근행렬로서 행은 접근 주체(사용자)

를, 열은 접근 객체를 나타낸다. 단 M의 각 요소는 M[u, o]로 표시한다.

- $\mathbf{F}$= (f_u, f_c, f_o, f_p, f_r) : 주체/객체의 등급 검사 함수들 (f_u, f_c, f_o), 프로세스의 영역 검사 함수(f_p) 및 동적 신뢰성 검사 함수(f_r)이다.
- f_u: U → User Levels : 사용자의 최고 보안등급을 검사하는 함수이다.
- f_c: U → User Levels : 사용자의 현재 로그인 등급을 검사하는 함수이다.
  단, ∀u ∈ U, fu(u) ≥ fc(u).
- f_o: O → Object Levels : 접근 객체의 보안등급을 검사하는 함수이다.
- f_p: PG_E → {common, public} : 프로세스의 영역을 검사하는 함수이다.
  임의의 p ∈ PG_E에 대해
  f_p(p) = common, if p ∈ Common.
  f_p(p) = public, if p ∈ Public.
- f_r: PG_E → C : 프로세스의 신뢰성을 검사하는 함수이다.

3.3 시스템

- System = (V, R, T, v_0)

시스템은 상태의 집합 V, 초기 상태(initial state)인 v0, 주체에게 객체에 대한 접근 권한을 부여하는 give access 및 부여된 권한을 제거하는 rescind access 같은 접근 요구의 집합 R, 그리고 Transition function T: (V × R) → V로 구성 된다.

3.4 보안규칙

- ss-property
  if M[u, o] = r, then $f_c(u) \geq f_o(o)$
- *-property
  if M[u, o] = w, then $f_o(o) \geq f_c(u)$
- ds-property
  if (s, o, a) ∈ B, then a ∈ M[u, o]
- path-property
  임의의 p ∈ PG_E에 대해
  - if f_p(p) = common, then ($f_c(u) \neq$ Anonymous) ∧ ($f_o(o) \neq$ Shared) ∧ (f_r(p) = Trust)
  - if f_p(p) = public, then $f_o(o)$ = Shared ∧ (f_r(p) = Trust)

Path-property는 추가된 보안 규칙으로서 common과 public으로 분류되어 수행중인 프로세스들이 접근 할 수 있는 객체를 명시하며, 또한 프로세스가 실행 시 신뢰적이어야 함을 의미한다.

4. E-BLP 구현 및 성능시험

4.1 설계

그림 2는 전체적인 시스템의 구성을 보여준다. 응용계층은 신뢰적인 프로세스인 Common과 비 신뢰적인 Public 프로세스로 나뉘며, 이들은 시스템 호출에 의해 시스템 서비스를 요구하며, 요구는 참조모니터와 각 보안 모듈(EBSM: E-BLP Based Security Module: 식별인증, 접근통제, DRC)들에 의해 객체에 대한 접근 허가/불허가 결정된다.

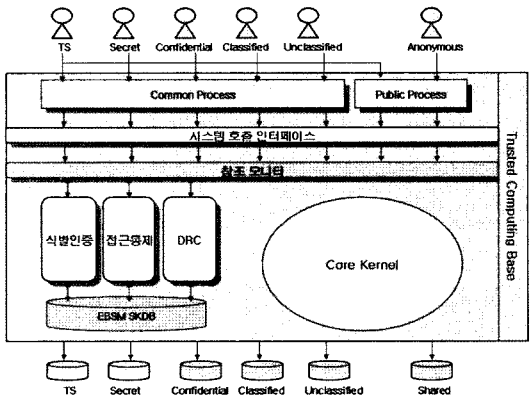


그림 2 시스템 구성도

4.1.1 식별인증 설계

사용자는 보안 객체에 접근하기 위해 자신에게 부여된 보안 레이블 정보를 이용하여 인증과정을 거쳐야 한다(그림 3). EBSM_login 은 사용자가 입력한 보안 등급과 보안 범주 정보를 입력받아 식별인증 접근 집행부(AEF: Access Enforcement Facility)에 인증을 의뢰한다. 식별인증 AEF는 레이블 검사 접근 결정부(ADF: Access Decision Facility)에 인증 허용/불허 유무 결정을 의뢰하고, 결정된 사항을 다시 AEF에 전달한다. 인증 성공 후 사용자의 보안레이블 정보는 주체관리부에 의해 커널의 주체정보테이블에 기록된다. 이 정보는 주체의 객체에 대한 접근 시 접근통제를 하는 기본정보가 된다.

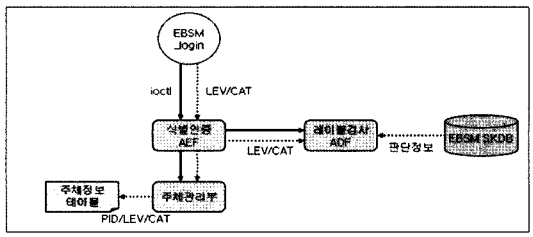


그림 3 EBSM 식별인증

4.1.2 접근통제 설계

인증과정을 거친 주체가 객체에 접근하는 순간 접근

통제 AEF는 E-BLP 특성검사 ADF에 접근 허가/불허 유무 결정을 의뢰한다. E-BLP 특성검사 ADF는 E-BLP의 ss-property, *-property, ds-property 검사를 하고 path-property 검사를 위해 DRC 모듈과 연동한다(그림 4).

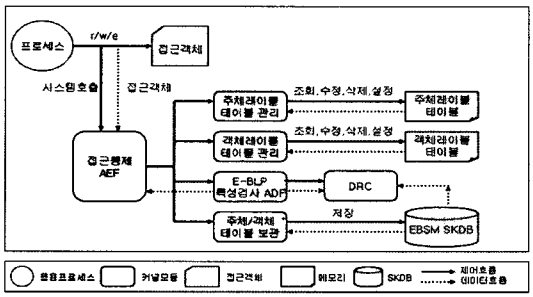


그림 4 EBSM 접근통제

4.2 구현

EBSM은 시스템 콜 후킹을 통해 개발된 리눅스 커널 모듈이다[11,12]. 보안 모듈에 의해서 접근 주체의 중요 객체에 대한 접근시도가 제어될 때 이루어지는 디스크 I/O 작업이 보안 모듈에 의해서 발생하는 가장 큰 오버헤드이다. 시스템 파워가 데스크탑 환경에 비해서 열악한 임베디드 환경에서는 이러한 오버헤드가 시스템 전체에 큰 영향을 미칠 수 있다. 또한, 대부분의 임베디드 장비는 데스크탑에 일반적으로 장착되어 있는 하드 디스크 드라이브 대신 플래쉬롬을 이용하여 파일시스템을 구축하는데, 이 플래쉬롬의 특성상(쓰기위해서 해당 섹터를 지우고 다시 기록해야 되는 특성) 플래쉬롬 파일시스템에서 객체에 대한 정보를 유지하는 작업은 또 다른 오버헤드를 유발시킨다.

본 EBSM 구현은 임베디드 환경을 고려하여 위에서 언급한 문제점들을 최소화 하기위해서 보안 객체에 대한 모든 보안정보를 메모리상에 유지하면서 접근제어를 하도록 구현하였다.

4.2.1 식별인증 구현

표 2와 같이 식별인증부는 사용자의 보안등급, 보안범주 입력정보와 EBSM_SKDB에 저장된 해당 사용자의 보안등급, 보호범주 정보를 비교 판단한다.

보호범주가 같지 않거나, 사용자의 보안등급이 EBSM_SKDB의 보안등급 보다 높으면 로그인은 실패하게 된다.

4.2.2 접근통제 구현

접근통제는 식별인증 성공 후 EBSM_Task에 유지되고 있는 주체 보안 정보와, EBSM_Obj에 저장되어 있는 파일 객체의 보안 정보를 비교함으로써 이루어진다. Read 시도는 주체의 보안 등급이 객체의 보안 등급보다

표 2 식별인증

```
EBSM_AUTH_ADF( ) {
    ...
    U_Level = User Input;
    S_Level = Level in EBSM_SKDB
    U_Cat = User Input;
    S_Cat = Category in EBSM_SKDB

    if (U_Cat != S_cat && U_Level > S_Level)
       Authentication Fail;
    else
       Authentication Success;
    ...
}
```

낮으면 거부되며, Write 시도는 주체, 객체 보안 등급이 같지 않으면 거부된다. 실행 가능한 객체를 실행할 때는 주체 정보가 Public이고 객체 정보가 Common이면 거부된다(표 3).

표 3 접근통제

```
EBSM_AC_ADF( ) {
   ...
   S_Level = EBSM_Task;
   S_Cat = EBSM_Task;
   S_Trust = EBSM_Task;
   O_Level = EBSM_Obj;
   O_Cat = EBSM_Obj;
   O_Trust = EBSM_Obj;

   if (read) {
      if (S_Cat != O_cat || S_Level < O_Level)
         Access Deny;
   } else if (execute) {
     if (S_Cat != O_cat || S_Level < O_Level ||
  ((S_Trust == Public) && (O_Trust == Common)))
         Access Deny;
   } else if (write) {
     if (S_Cat != O_cat || S_Level != O_Level)
        Access Deny;
   }
   ...
}
```

4.2.3 DRC 구현

DRC는 동적으로 프로세스에 대한 신뢰성을 검사한다. 여기서는 루트권한을 취득하기 위한 버퍼오버플로우 공격이 발생했을 때 이를 탐지하기 위한 방법에 대해 소개한다. 표 4에서 보듯이 탐지 방법은 비교적 간단하다. 스택 영역에 실행코드가 삽입되어 실행되는 것을 탐지한다. 프로그램이 수행할 때 호출되는 execve 시스템 호출 함수를 후킹하여, 표 4의 내용과 같이 eip의 범위가 스택 주소 영역인지를 판별함으로써 구현이 가능하다.

표 4 DRC: 버퍼오버플로우 공격탐지

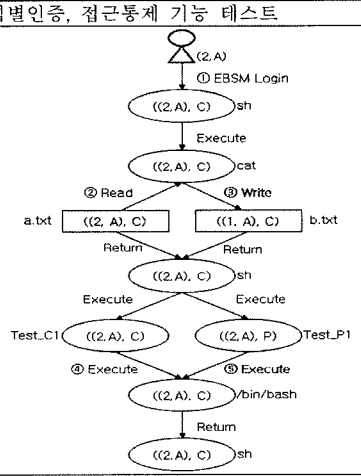
```
EBSM_DRC( ) {
    ...
    Get EIP of current process
    if (EIP in Stack Segment)
    Stack Overflow Attack Occur;
    ...
}
```

4.3 시험

4.3.1 식별인증, 접근통제 시험

표 5는 식별인증과 접근통제 테스팅을 위한 테스트 케이스를 보여준다. 보안 레이블이 (2, A)인 사용자는 ① EBSM 식별인증 절차를 통해 쉘((2, A), C)을 획득하고(그림 5), cat((2, A), C)명령을 통해 a.txt(2, A) 파일 객체에 대한 ② Read와 b.txt(1, A) 파일 객체에 대한 ③ Write를 시도한다. 이때 상위 등급 파일인 b.txt에 대한 Write 시도는 실패한다(그림 7).

표 5 테스트 케이스 1

이 름	EBSM 식별인증, 접근통제
목적	식별인증, 접근통제 기능 테스트
테스트 프로시져	
입력 데이터	사용자 보안레이블 정보 (2, A)
예상결과	① 로그인 식별인증 성공 ② a.txt 내용 읽기 성공 ③ b.txt 쓰기 실패 ④ Test_C1실행 성공, /bin/bash실행 성공 ⑤ Test_P1실행 성공, /bin/bash실행 실패


```
[PXA255-Pro@root]$EBSM_login testuser
password:
Input clearance ?
clearance  category
   2         A

[PXA255-Pro@root]$
```

그림 5 testuser (2,A) 식별인증 시험-①

```
[PXA255-Pro@root]$EBSM_ls
-rwxrwxrwx  2 A       C root     root          19 Jan  1 04:18 a.txt
-rwxrwxrwx  1 A       C root     root          39 Jan  1 04:23 b.txt
-rwxrwxrwx  2 A       C root     root       11505 Jan  1 02:17 test_c1
-rwxrwxrwx  2 A       P root     root       11505 Jan  1 02:17 test_p1
[PXA255-Pro@root]$
```

그림 6 접근객체 보안정보 출력

```
[PXA255-Pro@root]$cat a.txt
This file is (2,A)
[PXA255-Pro@root]$cat a.txt >> b.txt
sh: b.txt: Permission denied
[PXA255-Pro@root]$
```

그림 7 접근통제 시험-②, ③

쉘에서 ④ Test_C1((2, A), C)을 통한 /bin/bash((2, A), C) 실행은 성공하고, ⑤ Test_P1((2, A), P)을 통한 /bin/bash((2, A), C)의 실행은 실패한다(그림 8). ⑤의 경우를 예를 들어 설명하면 Test_P1이 httpd(public 프로세스) 같은 anonymous 사용자를 위해 존재하는 프로그램이고 BOF 취약점이 있는 경우, 이를 악용하여 루트 쉘(common)을 탈취하려는 시도를 차단할 수 있다. 또한 1 등급의 사용자라 할지라도 보안 관리자에 의해서 C(common 프로세스)로 관리되지 않는 프로세스를 실행 할 경우에는 중요한 객체로의 접근이 차단된다.

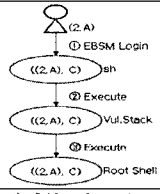
```
[PXA255-Pro@root]$/usr/src/test_c1
test_c1
bash$ exit
exit
[PXA255-Pro@root]$/usr/src/test_p1
test_p1
sh: /bin/bash: Bad file descriptor
[PXA255-Pro@root]$
```

그림 8 실행 가능 객체 실행 시험-④, ⑤

4.3.2 DRC 시험

표 6의 테스트 케이스처럼 내부 사용자가 제작한 악성 프로그램(vul.stack: 스택 오버플로우 익스플로잇)에 의해 ③ 루트 쉘을 탈취하는 공격은 EBSM_DRC에 의해 차단된다(그림 9).

표 6 테스트 케이스 2

이 름	EBSM DRC
목적	버퍼오버플로우공격 탐지기능 테스트
테스트 프로시져	
입력 데이터	사용자 보안레이블 정보 (2, A)
예상결과	① 로그인 식별인증 성공 ② vul.stack 실행 성공 ③ 루트 쉘 실행 실패

```
[PXA255-Pro@root]$id
uid=504(testuser) gid=504 groups=504
[PXA255-Pro@root]$./vul.stack
Segmentation fault
[PXA255-Pro@root]$
```

그림 9 버퍼오버플로우 공격 탐지

4.4 성능 시험

본 절에서는 임베디드 보드에 EBSM을 탑재하여 성능측정을 하고 EBSM 탑재전의 성능과 비교하여 EBSM으로 인한 성능 오버헤드를 계산해본다. 표 7은 EBSM이 탑재 될 임베디드 장비의 하드웨어 정보를 보여주며, 보드에 설치된 OS는 리눅스 커널 2.4.19 버전이다. 성능 측정(마이크로 벤치마킹)을 위한 도구는 파일 시스템 read/write 쓰레드 연산에 대한 결과를 잘 보여주는 tiobench 도구[13]를 사용하였다.

표 7 보드 하드웨어 정보

항 목	설 명
프로세서	Intel Xscale PXA255-400MHz
RAM	128MByte
Flash	32MByte
Ethernet	10BaseT External Ethernet Controller(CS8900A)

4.4.1 Tiobench를 이용한 성능 시험

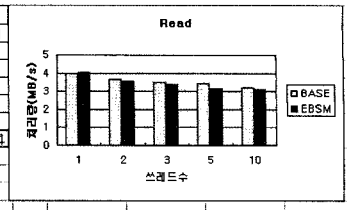
그림 10은 Tiobench 도구를 이용하여 각각 다섯 번의 테스트에 대한 Base와 EBSM의 read/write 처리량을 보여준다. Base(EBSM)에 대한 다섯 번의 read 처리량의 평균은 3.5656(3.4544)이며, Base(EBSM)에 대한 다섯 번의 write 처리량의 평균은 1.3174(1.293)이다. 그림 11은 EBSM을 보드에 탑재했을 때 생기는 오버헤드를 계산하기 위한 공식이다. read에 대한 오버헤드는 약 3%가 발생하였으며, write에 대한 오버헤드는 약 1.5%가 발생하였다. 성능에 대한 요구사항은 수요자(처)의 요구사항 이겠지만 비교적 3%내외의 성능 감소는 감내할 수 있는 것으로 사료된다.

5. 결론 및 향후 연구과제

본 논문에서는 안전한 임베디드 시스템을 위해 E-BLP 보안 모델을 구현한 내용을 다루었다. 비 신뢰적인 프로세스가 신뢰적인 프로세스를 실행함으로써 행해지는 공격에 대한 대응방식을 보였고, 비 신뢰적인 프로세스의 중요 객체에 대한 접근이 차단되는 것을 보였다. 또한 구현된 시스템의 성능 오버헤드를 측정했다. 향후 각종 공격에 대응하기 위해 DRC의 확대 구현이 요구된다.

임베디드 시스템에서의 보안은 아직까지 그 침해사례가 적은 관계로 국내는 물론 외국에서조차 큰 관심을 끌지는 못하고 있다. 그렇기 때문에 국내에서 먼저 임베디드 보안 분야에 많은 투자와 함께 기술 개발을 서두른다면 국제적으로도 충분히 기술적 우위를 점할 수 있을 것이다.

Read		
	BASE	EBSM
1	3.994	4.047
2	3.664	3.558
3	3.502	3.4
5	3.42	3.165
10	3.248	3.102
평균	3.5656	3.4544



Write		
	BASE	EBSM
1	1.359	1.351
2	1.31	1.313
3	1.307	1.293
5	1.314	1.27
10	1.297	1.27
평균	1.3174	1.293

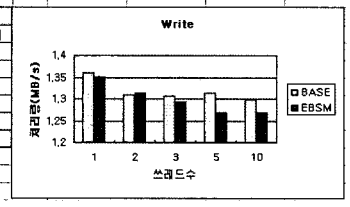


그림 10 Read/Write 처리량

Read 오버헤드 = ((BASE - EBSM) * 100) / BASE
= ((3.57 - 3.46) * 100) / 3.57 = 3.08 %

Write 오버헤드 = ((BASE - EBSM) * 100) / BASE
= ((1.32 - 1.30) * 100) / 1.32 = 1.52 %

그림 11 Read/Write 오버헤드

참 고 문 헌

[1] Anand Raghunathan외, "Securing Mobile Applicances: New Challenges for the System Designer," COMPUTER SOCIETY IEEE, 2003.

[2] Zili shao외, "Defending Embedded Systems Against Buffer Overflow via Hardware/Software," ACSAC, 2003.

[3] Lawrence P. Ricci, Larry B.McGinness, "Embedded System Security Designing Secure Systems with Windows CE," White Paper, 2004.

[4] Edwin Lee "Security in Embedded, Real-time Operating Systems," TOG Real-time and Embedded Systems Forum, 2001.

[5] Wook Shin, Hone Kook Kim, Kouichi Sakurai, "An Implementation of Extended-Role Based Access Control on an Embedded System," Proc. of Computer Security Symposium 2004, pp. 667-671, 2004.

[6] 강정민 외, "프로세스 신뢰도에 기반한 E-BLP 보안 모델과 아키텍처 설계", 한국정보과학회, 2001.

[7] 강정민 외, "다중등급 보안 운영체제에서의 보안 등급 결정 문제", 한국정보처리학회, 2001.

[8] JungMin Kang et al, "Extended BLP Security Model based on Process Reliability for Secure Linux Kernel," 2001 IEEE PRDC, 2001.

[9] 강정민 외, "안전한 리눅스 시스템을 위한 E-BLP 보

안 모델과 구현", 정보처리학회논문지, 2001.
[10] JungMin Kang et al, "E-BLP Security Model," CSS 2003, 2003.
[11] D.P. Bovet and M. Cesati, "Understanding Linux Kernel," O'Reilly, 2001.
[12] A. Rubini and J. Corbet, "Linux Device Drivers," O'Reilly, 2001.
[13] http://sourceforge.net/projects/tiobench/

강 정 민

2002년 2월 광주과학기술원 정보통신공학과 석사 졸업. 2002년 3월~2003년 5월 삼성 SDS 근무. 2003년 6월~현재 국가보안기술연구소 근무

남 택 준

2003년 2월 한국외국어대학교 컴퓨터공학과 석사 졸업. 2003년 11월~현재 국가보안기술연구소 근무

장 인 숙

2001년 2월 경북대학교 컴퓨터과학과 석사 졸업. 2001년 3월~현재 국가보안기술연구소 근무

이 진 석

1990년 2월 한남대학교 수학과 석사 졸업. 2002년 8월 한남대학교 컴퓨터공학과 박사 졸업. 1986년 1월~1999년 12월 한국전자통신연구원 근무. 2000년 1월~현재 국가보안기술연구소 근무