공통평가기준 인증을 위한 SW의 내부 구조 및 복잡도 분석 사례에 관한 연구

최정란*, 서동수**, 배창환***
*한국전자통신연구원 고성능컴퓨팅시스템연구실
**성신여자대학교 IT학부
***호서대학교 정보통신공학과
e-mail:seohyun@etri.re.kr, dseo@sungshin.ac.kr, chbae@hoseo.edu

Case Study on Analysis for Well-Structured Internals and Complexity of Software for Common Criteria

Jeong-Rhan Choi*, Dong-Soo Seo**, Chang-Hwan Bae***
*High Performance Computing System Research Section, ETRI
**School of Information Technology, Sungshin Women's University
***Dept. of Information Communication Engineering, Hoseo University

요 약

본 논문은 EAL6 수준의 공통평가기준 인증을 위해 ADV(개발) 클래스에서 ADV_INT에 대해 기술하였다. 특히, 테스트용 스마트 운영체제 소스코드 분석을 통해 구현된 내부 구조가 잘 구조화되었는지, 지나치게 복잡하지 않았는지 입증하기 위한 시도를 하였다. 다양한 소스코드 분석 도구를 통해 사이클로매틱복잡도(CyC), 정보흐름복잡도(IFC), Weighted IFC, fan-in, fan-out 등의 정보를 추출하였고, 추출된 정보를 기반으로 분석을 수행하였다. 구조화된 정보 분석을 위해 객체지향 분석 도구를 사용한 재구조화 기법을 적용하여 수행하였다. 객체간 결합도, 팬아웃 등의 정보 등을 추출하였다. 추출된 정보를 기반으로 SW의 복잡도 및 구조적 정보를 분석한 결과 응집도 분석에 한계, TOE의 형상관리 정보 등의 부재에 따른 추출된 정보 분석의 한계, 활용된 도구의 분석 정보의 재반영 부재 및 구조적 분석 등의 한계점이 드러났다.

1. 서론

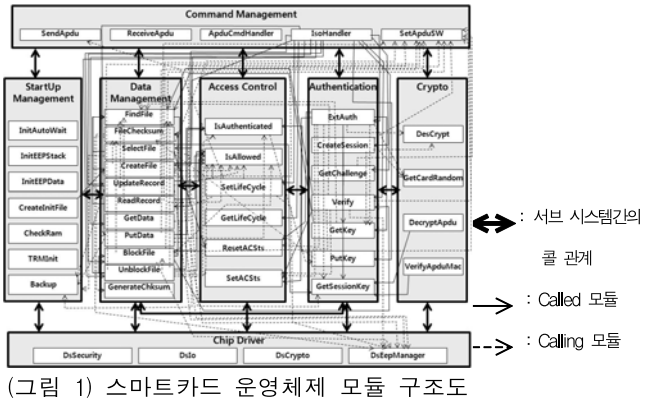
공통평가기준(CC, Common Criteria for Information Technology Security Evaluation)은 보안관련 컴퓨팅 기술의 평가를 위한 국제 표준이다. 국가 및 공공기관에 도입하는 정보보호제품은 CC인증을 받아야한다.

본 논문에서는 EAL6 수준의 ADV(개발) 클래스에서 ADV_INT(TSF 내부)에 대해 기술한다. CC 평가에서 EAL 4이상의 고수준에서는 정형 혹은 준정형 기법을 활용한 보안 평가가 필수로 요구되고 있다. CC에서 기술하고 있는 TSF(TOE[1] Security Function, 평가대상 보안기능) 내부의 목적은 다음과 같다. 첫째, TSF가 "잘 구조화된" 내부를 가지도록 설계하고 구현되었음을 입증(정당화)해야 한다. 둘째, TSF 내부가 "지나치게 복잡하지 않음"을 입증해야 한다. 이를 위해 TSF 전체는 적절한 SW 공학 원리를 이용하여 설계하고 구현되어야 한다[1].

본 논문은 C언어로 개발된 테스트용 스마트카드 운영체제에 대해 보안 기능들을 사전에 정의하고, 정의에 따라 구축된 결과물을 가지고 보안기능 내부 목적에 따른 ADV_INT에 부합되어 개발되었는지 분석하고, 분석된 내용의 한계점 및 향후 개선방안에 대해 기술하였다.

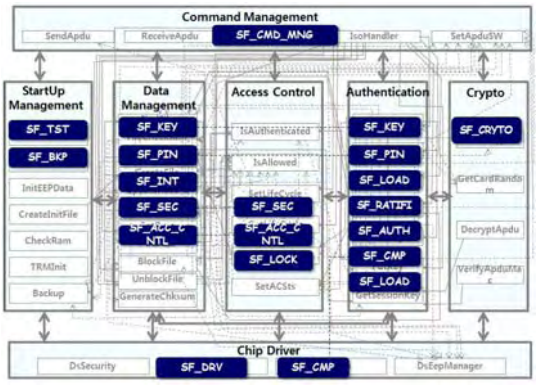
2. 적용 사례

본 논문에서 적용된 TOE는 그림 1과 같이 7개의 서브시스템, 42개의 모듈의 약 4,300라인의 C언어로 개발된 스마트카드 운영체제이다. 그림 2는 스마트카드의 보안 기능에 따라 시스템에 반영한 구조로 보안 기능은 개발 결과물(TOE)에 모두 반영되어야 한다.



(그림 1) 스마트카드 운영체제 모듈 구조도

1) 공통평가기준은 평가 대상 선정에 융통성이 있기 때문에 적용 범위가 IT 제품에만 한정되지는 않는다. 따라서, "IT 제품"이라는 표현 대신 "TOE(평가 대상, Target of Evaluation)"라는 용어를 사용한다.



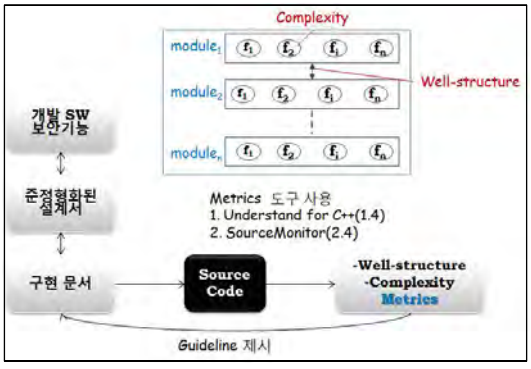
(그림 2) 서브시스템내의 보안 기능 수행도

절차적 소프트웨어 구조

C언어로 작성된 운영체제는 절차적 소프트웨어 구조를 따르고 있다. 일반적으로 절차지향 소프트웨어의 구조는 모듈화에 따라 평가된다. 모듈화 설계로 작성된 소프트웨어는 한 모듈과 다른 모듈간의 의존관계(결합도, coupling)를 명확하게 하고 한 모듈안에 서로 밀접하게 관련된 작업만을 포함시킴(응집력, cohesion)으로써 이해도를 높인다. 모듈화 설계를 사용하면 TSF의 요소간 상호의존성을 감소시켜 한 모듈의 변경사항이나 오류로 인해 TOE 전체에 영향을 미칠 위험을 낮추게 된다. 또한 모듈화 설계의 사용은 설계의 명확성을 증대시키고, 예상하지 않은 결과가 일어나지 않을 것이라는 강화된 보증을 제공한다.

절차적 소프트웨어의 복잡도

복잡도는 코드 실행의 결정 시점과 논리적 경로에 대한 척도이다. 소프트웨어 공학 문헌은 복잡도를 소프트웨어의 부정적 특성으로 언급하고 있다. 이는 복잡도가 코드의 논리와 흐름에 대한 이해를 방해하기 때문이다. 코드 이해를 방해하는 또 다른 요소는 사용되지 않거나 중복되는 불필요한 코드의 존재를 들 수 있다. 복잡도를 줄이는 것은 상호 의존성을 감소시키거나 제거하는 것을 포함한다.



(그림 3) SW 내부 구조화 분석 접근방법

접근방법

SW 내부가 잘 구조화되었는지 여부를 분석하는 접근방법은 그림 3과 같다. 보안기능을 바탕으로 설계 및 구현을 진행하고 구현된 결과물를 소스코드 분석도구를 사용하여 모듈의 내부의 복잡도와 외부의 구조화 분석을 바탕으로 가이드라인을 제시해준다.

일반적으로 모듈의 복잡도는 다음과 같이 2가지 요인에 의존한다: 모듈 코드의 복잡도와 시스템의 환경에서 모듈들의 연결에 대한 복잡도. 모듈 코드 복잡도는 LOC(Line of Code), CyC (Cyclomatic Complexity)[2]를 사용하여 추출할 수 있다. CyC는 모듈내의 에러에 강한 상호관계를 가진다. 구조적 복잡도를 의미하는 시스템 환경과의 복잡도는 IFC(Information Flow Complexity), WIFC (Weighted IFC)로 추출할 수 있는데 모듈간의 상호연관성에 초점을 두었다[3]. 이는 fan-in과 fan-out에 의해 결정된다.

모듈의 응집도(cohesion)와 결합도(coupling)도 TOE의 구조적 정보를 추출하기 위한 중요한 척도다.

본 논문은 개발 SW의 메트릭을 추출하기 위해 4종류[2]의 SW 정적 분석 도구를 사용하였다. 복잡도 정보는 sourceMonitor, Understand 도구를 사용하여 LOC, CyC, fan-in, fan-out, WIFC, IFC 정보를 추출하여 분석하였다. 구조적 정보를 추출하기 위해 Togerther와 SD메트릭스 도구를 사용하여 C코드를 C++구조로 재구조화(refactoring) 기법을 적용하여 추출하였다. 올바른 값을 얻기 위해 재구조화는 원래 코드의 특성을 가장 잘 보존하도록 하는 범위에서 모듈을 구성하는 개별 함수에 대해 클래스로 재구조화하여 측정을 수행하였다.

3. 분석 결과

임베디드 시스템인 스마트카드 OS는 CyC와 LOC 분석 결과 모듈별 코드의 사이즈가 크지 않지만, 코드 사이즈와 내부 실행 복잡도가 강한 연관성을 없었다. 이들 수치는 1~16사이의 값들로써 테스트가능한 정도의 코드 복잡도를 보였다[2].

코드 결합도 측정에 있어서 내부 복잡도(CC)와 외부복잡도(IFC)와의 관계를 분석한 결과 서로 강한 관련성이 없음을 알 수 있었다. 예를 들어 서브시스템 ChipDriver의 DsEepManager 모듈은 CC가 7로 내부 복잡도에 대한 위험도가 상대적으로 낮은 반면 IFC의 수치가 166,464로 모듈과 모듈사이의 상호작용이 많이 발생하는 것을 알 수 있다.

스마트카드 OS의 경우 모듈은 대부분이 함수 단위로 구성된다. IFC의 수치가 대한 편차가 크지 않으나, 유독 ChipDriver 서브시스템내의 DsEepManager와 DataManagement의 CreateFile 모듈이 눈에 띄게 높게 나타났다. DsEepManager의 경우 3개의 함수로 구성되어 있고 fan-in의 수치가 34로 높게 나타나고 있음을 알 수 있다. 이는 모듈의 응집도가 낮을 수 있고, 재설계 혹은 단순화 작업이 필요한지 코드의 검토가 요구된다. 즉, DsEepManager모듈은 EEPROM에 데이터를 쓰고, 읽는

2) SourceMonitor, Understand for C++, Boland Together, SD메트릭

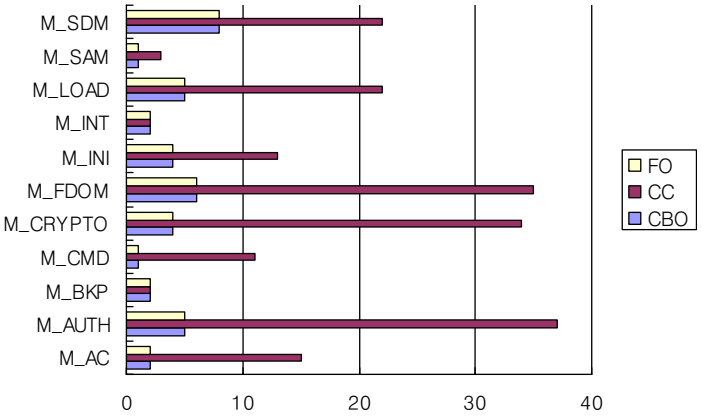
과정을 구현한 모듈로서, 모듈의 기능적 특징으로 인해 데이터에 대한 수정, 즉, 인증, 상태값 등의 변화에 따라 모든 모듈에서 사용되는 유틸리티 기능을 가진 모듈이라 할 수 있다. 따라서 fan-out값은 상대적으로 높아지고, 그 값에 따라 IFC가 높게 측정되고 있다. CreateFile의 경우 두 개의 함수로 구성되어 있고 유독 fan-out의 수치가 29로 다른 모듈에서의 활용도가 높음을 알 수 있다.

응집도와 결합도는 Together 툴의 품질 매트릭 생성 기능을 이용하여 추출하였으며 이 과정에서 각 모듈을 구성하고 있는 C 함수를 C++ 구조로 재구조화 (refactoring)하였다. Together나 SDMetrics와 같은 최근 도구들은 주로 Java나 C++만을 인식하도록 설계되었으며 해당 C 코드에 대해서는 함수를 인식 못하는 문제가 발견되었다. 따라서 올바른 값을 얻기 위해 재구조화는 원래 코드의 특성을 가장 잘 보존하도록 하는 범위에서 모듈을 구성하는 개별 함수에 대해 클래스로 재구조화 하여 측정을 수행하였다.

<표 1> 응집도와 결합도에 관한 파라미터 측정 값

모듈명	CBO	CyC	DAC	FO	NOC	NOO	NORM	RFC	WMPC
M_AC	2	15	0	2	1	0	0	1	15
관련함수	isAllowed()								
M_AUTH	5	37	0	5	3	10	2	11	37
관련함수	isoExtAuth(); isoVerify(); proCreateSession()								
M_BKP	2	2	0	2	2	1	0	2	2
관련함수	trmCommit(); trmStart()								
M_CMD	1	11	0	1	1	2	0	3	11
관련함수	apudCmdHandler()								
M_CRYPTO	4	34	0	4	1	6	3	9	34
관련함수	Descrypt()								
M_FDOM	6	35	0	6	7	7	2	9	35
관련함수	isoGetData(); isoPutData(); isoReadRecord(); isoUpdateRecord(); proBlockFile(); proCreateFile(); proUnblockFile()								
M_INI	4	13	0	4	9	4	3	8	13
관련함수	ReceiveApdu(); createInitFile(); initAutoWait(); initEEPData(); initEEPStack(); ioTxByteArray(); sendApdu(); trmInit(); trmMemSetU4()								
M_INT	2	2	0	2	2	1	0	1	2
관련함수	abend(); generateChecksum()								
M_LOAD	5	22	0	5	1	3	2	6	22
관련함수	proPutKey()								
M_SAM	1	3	0	1	4	0	0	1	3
관련함수	resetGlobalACSts(); resetLocalACSts(); setGlobalACSts(); setLocalACSts()								
M_SDM	8	22	0	8	2	5	5	10	22
관련함수	GetKey(); updateSecFile()								

표 1에서 CBO는 객체 간의 결합도를, DAC는 자료 추상화 결합도를, FO는 팬 아웃을, NOC는 클래스의 개수를, NOO는 오퍼레이션의 개수를, NORM은 리모트 오퍼레이션의 수를, RFC는 클래스에 대한 응답의 수를, 그리고 WMPC는 각 클래스당 가중 메서드의 수를 나타낸다.



(그림 4) 모듈의 FO, CBO 관계

그림 4의 차트는 11개 모듈에 대한 모듈의 복잡도와 결합도 관계를 보여주는 차트이다. 차트에 의하면 사이클로매틱 복잡도 (CyC)가 높은 모듈은 모듈 결합도 (CBO) 사이에는 강한 관련성이 없다는 점이다. 예를 들어 모듈 M_AUTH는 가장 강한 복잡도를 갖는 모듈임에도 불구하고 CBO 결합도는 M_SDM 보다도 적으며 M_LOAD 수준의 결합도를 갖는 것을 알 수 있다. 이러한 현상은 모듈 M_CRYPTO나 M_FDOM에서도 나타나는 것을 알 수 있다.

모듈 M_INI와 같은 모듈은 복잡도는 낮지만 결합도가 높아 다른 모듈에서 많은 호출이 있는 경우이다. 그 이유 중의 하나는 다른 모듈에 비해 M_INI 내부에는 9 개나 되는 작은 단위의 함수들이 많이 존재하여 상대적으로 다른 모듈보다 외부와 인터페이싱할 수 있는 가능성이 많다는 점이다. M_INI와 같이 모듈내의 함수가 너무 많은 수로 정의될 때 다른 모듈에서 호출하는 횟수가 빈번해 질 수 있는 문제가 있으며 따라서 이러한 서버 모듈에 변경이 발생할 경우 이의 영향을 받는 클라이언트 모듈은 많아지게 된다. 따라서 개발자는 이러한 모듈을 더욱 철저하게 관리할 필요가 있다.

4. 한계점

절차적 소프트웨어 구조는 일반적으로 그 모듈화에 따라 평가된다. 모듈화 설계로 작성된 소프트웨어는 한 모듈과 다른 모듈간의 의존관계(결합도)를 명확하게 하고 한 모듈 안에 서로 밀접하게 관련된 작업만을 포함시킴(응집도)으로써 이해도를 높인다[1].

현재 사용하는 결합도와 응집도 관련한 대부분의 도구들이 모듈의 결합도 측정에 맞춰져 있다. 그 이유는 결합도는 모듈의 구문 분석만으로도 쉽게 파악될 수 있어 쉽게 분석/구현될 수 있다. 반면 응집도는 모듈 내부에서 사용되는 변수에 대한 의미 해석을 통해 정확한 판단을 할 수 있기 때문에 구현이 힘들고, 분석자의 주관적 견해가 반영될 가능성이 높다. 예를 들어 if 조건문 블록의 분기를 결정하는 변수가 단순 플래그 역할인지, 의미있는 값을 포함하는 변수인지에 따라 절차적 응집도이거나 통신적

응집도로 구분될 수 있다. 이 경우 자동화 도구를 통하여 판단하기엔 어려움이 따르고 현실적으로 평가자가 개발자의 의도를 파악하지 못하여 옳지 않은 결정을 내릴 수 있는 문제들이 존재한다.

응집도 분석의 경우, 응집도가 높은 모듈이 그렇지 않은 모듈에 비해 문제(에러 등)를 발생시킬 가능성이 낮거나 그 반대로 응집도가 낮은 모듈이 문제를 발생시킬 가능성이 높은지에 대한 상관관계가 명확히 확립되지 못하고 있다. 이 경우 CyC와 같이 잘 정의된 기준을 적용할 수 없다는 것이 한계라 볼 수 있다.

복잡도 최소화 및 증가 원인을 분석하기 위해 형상관리 정보가 있다면, SW 개발과정 동안 버전별 프로그램들을 분석하므로써 복잡도 최소화를 위해 어떠한 과정을 거쳤고, 왜 복잡도가 증가하였는지를 아는데 도움이 될 것이다.

본 논문은 TOE의 구조적 정보 분석을 위해 절차적 SW를 객체지향SW로 재구조화 하여 지표 측정을 수행하였다. 이러한 재구조화된 정보가 실제 코드에 수정에 반영될 수 없어, 단순 분석에만 그친 아쉬움이 있다. 또한 재구조화를 통한 분석도구들이 객체지향 SW에 잘 정의되어 있어, 분석 정보의 효과 측면에서 제한적일 수밖에 없다.

5. 결 론

본 논문은 EAL6 수준의 CC인증을 위해 ADV(개발) 클래스에서 ADV_INT에 대해 기술하였다. 특히, 테스트용 스마트 운영체제 소스코드 분석을 통해 구현된 내부 구조가 잘 구조화되었는지, 지나치게 복잡하지 않았는지 입증하기 위한 시도를 하였다.

이를 위해 복잡도 측정을 위해 SourceMonitor, Understand for C++ 도구를 사용하였다. 이 도구를 통해 CyC, IFC, WIFC, fan-in, fan-out 등의 정보를 추출하였고, 추출된 정보를 기반으로 분석을 수행하였다. 구조화된 정보 분석을 위해 Together, SD메트릭 도구를 사용하여 재구조화 기법을 적용하여 수행하였다. CBO, CyC, FO 등의 정보 등을 추출하였다.

적용한 기법들은 응집도 분석에 한계, TOE의 형상관리 정보 등의 부재에 따른 추출된 정보 분석의 한계, 활용된 도구의 분석 정보의 재반영 부재 및 구조적 분석 등의 한계점이 드러났다.

참고문헌

[1] ISO std. 15408, Common Criteria for Information Technology Security Evaluation Version 3.1 Revision 4, ISO, Sep. 2012.

[2] McCabe, A Compliexity Measure, IEEE Transaction on Software Engineering, Dec. 1976, pp.308-320.

[3] S. Henry, D. Kafura, Software Structure 메트릭 Based on Information Flow, IEEE Transaction on Software Engineering, vol. SE-7, no.5, September 1981, pp. 510-518.