情報保護學會論文誌
第 22 卷 第 5 號, 2012. 10

결합척도를 이용한 복합 공격 패턴 분석 방법

권 예 진,† 박 용 범‡
단국대학교

An analysis method for complex attack pattern using the coupling metrics

Ye-jin Kwon,† Young-bom Park‡
Dankook University

요 약

최근 대부분의 소프트웨어는 인터넷 환경에서 네트워크를 중심으로 데이터를 교환하기 때문에, 소프트웨어 자체의 보안성이 큰 이슈로 다루어지고 있다. 또한 소프트웨어 개발 과정에서 시큐어 코딩 규칙을 적용하여 소프트웨어의 취약점을 최소화하고 안전성이 높은 코드로 개발하려는 노력이 이루어지고 있다. 하지만 소프트웨어 취약점을 이용한 공격 사례들이 한 가지 공격이 아닌 복합 공격의 형태를 띠고 있어 단일 특성 분석으로는 소프트웨어 취약점 분석에 한계가 있다.

본 연구에서는 소프트웨어의 다양한 특성 중에 하나인 결합도를 기반으로 복합 소프트웨어 취약점을 이용한 공격에 대응하고자 하였다. 더불어 여러 공격 방법의 조합을 이용한 복합 공격 패턴을 사전 분석하여 소프트웨어 내의 모듈간의 파급력과 연관관계에 있는 모듈의 공격 가능한 패턴을 예측하고 이에 대한 소프트웨어 취약점을 분석할 수 있는 방법을 제시한다.

ABSTRACT

Recently, since the most software intensive systems are using internet environment for data exchange, the software security is being treated as a big issue. And, to minimize vulnerability of software system, security ensuring steps which are applying secure coding rules, are introduced in the software development process. But, since actual attacks are using a variety of software vulnerabilities, it is hard to analyze software weakness by monotonic analysis.

In this paper, it is tried to against the complex attack on the variety of software vulnerability using the coupling which is one of the important characteristic of software. Furthermore, pre-analysis of the complex attack patterns using a combination of various attack methods, is carried out to predict possible attack patterns in the relationship between software modules. And the complex attack pattern analysis method is proposed based on this result.

Keywords: Secure Coding, Coupling Metrics, Software security

I. 서 론

소프트웨어가 대형화되고, 복잡도가 높아짐에 따라 소프트웨어의 개발 과정에는 수많은 오류들이 발생하였고 이를 해결하기 위해 많은 비용이 소모되고 있다. 또한 최근 대부분의 소프트웨어는 인터넷 환경에서 네트워크를 중심으로 데이터를 교환하기 때문에 소프트웨어의 안정적인 운영과 함께 데이터의 인위적인 조작이나 악의적인 접근을 차단해야 하는 보안성 문제도 크게 다루어지고 있다. 가트너에 의한 발표에 따르면 사이버 공격의 약 75%가 소프트웨어의 취약점을 이용한 공격이라고 하였다[1]. 따라서 지금까지 소프트웨어의 개발 환경에서는 소프트웨어의 안정적인 운영

접수일(2012년 9월 10일), 게재확정일(2012년 10월 11일)
† 주저자, kwon6838@dankook.ac.kr
‡ 교신저자, ybpark@dankook.ac.kr

이나 성능, 메모리 사용 효율, 플랫폼 호환성 등의 각 소프트웨어의 종속적인 환경에 국한되어 있었다면 앞으로 개발하고 운영되어야 하는 소프트웨어에서는 외부의 악의적인 접근으로부터 보호되어야 하며 이를 위해 다양한 연구가 필요하다.

최근의 국내외 소프트웨어 공격 방식을 살펴보면, 한가지의 소프트웨어 취약점을 공격하여 데이터를 유출시키는 형태가 아니라 두 가지 이상의 취약점을 복합적으로 사용하여 공격하는 사례가 늘어나고 있다. 2011년 Sony의 PlayStation의 공격 사례를 보면 DDoS 공격과 SQL Injection을 동시에 사용하여 사용자의 개인 결재정보를 유출시켰고, 독일 지멘스사의 산업 자동화 제어 시스템을 공격 목표로 제작된 악성코드인 스턱스넷(Stuxnet)은 제로데이 취약점 4가지를 복합적으로 사용하여 공격하는 방식으로 시스템을 마비시켰다. 국내에 최근 DDoS를 이용한 공격인 34 DDoS, 77 DDoS의 경우 국내 주요 정부기관을 목표로 DDoS 공격과 HDD 파괴 명령을 동시에 복합적으로 사용하여 많은 혼란을 야기시켰다. 따라서 소프트웨어의 취약점을 분석하고 안전성이 높은 개발을 하기 위해서는 취약점을 분석하는 것뿐만 아니라 이와 관련된 예외 흐름이나, 연관된 모듈간의 관계를 고려해야 한다.

소프트웨어의 내부적인 구조나 데이터 흐름, 예외 흐름 등을 수치적으로 표현하고 분석할 수 있는 방법으로는 소프트웨어의 결합도 메트릭을 이용한 측정 방법이 있다. 소프트웨어 결합도 메트릭은 소프트웨어 결함의 가능성을 예측하는데 사용되고 있으며, 소프트웨어의 결함의 파급도를 추정할 수 있다. 소프트웨어의 취약점을 이용한 실제 공격 사례들을 살펴보면 소프트웨어에 취약점을 대비한 방어코드가 짜여져 있다고 하더라도 한 가지 공격이 아닌 복합적인 공격을 통해 실제 사용자들의 정보를 도청하거나 변조시키는 공격이 가능했다. 이러한 공격은 소프트웨어 내부의 데이터 흐름이나 예외 흐름, 제어 정보 흐름, 또는 외부의 환경과 소프트웨어와의 연결 모듈과 같은 부분을 집중적으로 설계되었다. 따라서 각 모듈간의 정보 흐름과 제어 정보 전달, 예외 처리, 외부 소프트웨어 또는 인터넷 환경과의 연결 부분을 수치화해서 분석할 수 있는 결합도를 이용하여 소프트웨어 공격 패턴을 분석하고 예측할 수 있다[10].

소프트웨어의 보안적인 취약점을 해결하고 이를 예방할 수 있는 가장 대표적인 방법은 현재 많은 회사와 연구기관에서 정의하고 있는 안전한 소프트웨어를 개발하기 위한 코딩 규칙이나 개발 보안에 관한 규약인 시큐어 코딩(Secure coding) 규칙을 준수하고 이를 테스팅 할 수 있는 분석 도구를 개발하는 것이다[2][3]. 시큐어 코딩은 미국 국토안보부에서 관리하는 CWE(Common Weakness Enumeration), 카네기 멜론 대학의 소프트웨어 공학 연구소에서 관리하는 CERT(Computer Emergency Response Team) 등이 있으며, 언어적인 특성을 고려하여 C, C++, Java 프로그래밍 언어에 대한 표준을 제공하고 있다[4]. 그 외에도 Java언어를 개발한 Sun Microsystems에서도 코딩 지침을 제공하고 있으며 국내 기관과 연구 단체에서도 시큐어 코딩에 대한 연구가 활발히 진행 중에 있다.

본 논문에서는 소프트웨어의 다양한 특성 중에 하나인 결합도를 기반으로 소프트웨어의 취약점을 이용한 공격에 대해 방어하고자 한다. 소프트웨어의 결합도는 모듈 또는 클래스들이 상호 의존하는 정도를 나타내는 것으로, 모듈간의 상호 작용의 정도를 측정하는 품질 요소 중에 하나이다. 또한 최근 연구에 따르면 소프트웨어의 결합도가 높아지면, 소프트웨어 내의 모듈간의 데이터의 전송이 잦아지고, 상호 참조하고 호출하는 횟수가 증가하기 때문에 소프트웨어의 오류 발생이나 보안 취약점이 발생하였을 때 다른 모듈로 피해가 전파되는 파급효과(ripple effect)가 일어날 가능성이 높아진다고 하였다[9][10]. 따라서 본 연구에서는 각 모듈간의 연관관계를 기반으로 소프트웨어의 취약점의 파급력을 분석하고 이를 예측하는 프로세스를 제안한다. 또한 최근 소프트웨어 공격 사례의 특징에 따라 소프트웨어의 취약점이 다른 모듈로 전파되는 파급효과에 따른 복합 공격을 예측하고 이에 대한 소프트웨어 취약성 예방을 돕는다.

II. 관련 연구

2.1 시큐어 코딩(Secure Coding)

최근 개발되고 있는 소프트웨어는 네트워크를 통해 중요한 정보를 교환하기 때문에 그 과정에서 생기는 소프트웨어의 취약점을 목적으로 하여 악의적인 공격을 시도할 수 있는 가능성이 커지게 되었다. 소프트웨어 개발자가 소프트웨어의 취약점을 인지하고 실제 소프트웨어 개발 단계에 반영하여 적용하려면 보안적인 전문성을 가지고 접근해야하기 때문에 현실적으로 어려움이 존재한다. 따라서 소프트웨어의 취약점을 최소

화하고, 소프트웨어 개발과정에서 안전한 코드를 작성하는 시큐어 코딩(Secure Coding)에 대한 연구가 활발하게 연구되고 있다.

시큐어 코딩이란 방어적 프로그래밍 개념을 포함한 강건하고 안전한 어플리케이션을 만드는 방법이다[6]. 미국 국토안보부에서 관리하는 CWE는 소프트웨어의 취약점을 사전식으로 분류하여 사용자가 접근하기 쉽고 각 소프트웨어의 취약점을 분류하고 접근하기 쉽게 다양한 뷰를 제공하고 있다. 또한 CERT는 카네기 멜론 대학에서 시큐어 코딩의 표준화 작업을 진행하고 있다. 소프트웨어의 결함으로 인해 치명적인 문제가 발생할 수 있는 항공기, 자동차 등의 산업에서는 이미 JSF(Joint Strike Fighter), MISRA Coding Rule 등의 코딩 규약을 도입하여 소프트웨어의 취약점을 최소화하고 보다 안전한 소프트웨어의 개발을 위한 지속적인 연구가 진행되어 왔다.

프로그래밍시에 시큐어 코딩을 적용하기 위해 프로그래머가 모든 상황을 이해하고 취약점에 대한 다양한 경우를 전부 헤아리는 것은 많은 어려움이 있으므로 도구를 이용하여 시큐어 코딩을 효과적으로 적용할 수 있는데 이러한 도구로 규칙 검사기(Rule Checker)와 취약점 분석기(Weakness Analyzer) 등이 있다[4]. 실제 Java 언어만을 기준으로 소프트웨어 취약성 항목을 보면 CERT에서 138개, Sun에서 20개, CWE에서 171개의 항목이 존재 할 뿐만 아니라 지속적으로 소프트웨어 취약점은 증가하고 있다. 따라서 이러한 소프트웨어 취약점을 프로그래머가 일일이 확인하고 시큐어 코딩에 따른 방어 코드를 실제 소프트웨어 개발시에 추가하여 테스트하는 것은 매우 어려운 일이다.

많은 소프트웨어 취약점이 지속적으로 CWE, CERT, Sun 등의 사이트에 매일 업데이트 되고 있으며, 실제 어떻게 해당하는 취약점을 분석하고 방어해야 하는지를 분석하고 이해하는데는 어려움이 따른다. 따라서 매년 CWE와 SANS(SysAdmin, Audit, Network, Securiy)에서는 가장 위험한 소프트웨어 취약점인 "Top 25 Dangerous Programming Errors"를 매년 발표하여 실제 위험도와 관련 공격들을 체계적으로 정리하고 여러 사례와 방어코드를 효과적으로 정의하고 있다. Top 25에서는 25개의 소프트웨어 취약점을 정의하며, 각 소프트웨어 취약점의 랭킹과 정의, 취약점 사례, 방어코드, 관련 패턴 등을 상세히 서술하고 있다.

2.1.1 TOP 25

Top 25란 SANS와 MITRE가 주관한 프로젝트로, 2009년 1월 12일에 SANS, MITRE, CWE, CERT, 미국국토안보부, Microsoft, Sysmantec 등 30개 이상의 단체가 함께 가장 위험한 프로그래밍 에러 25개를 선정한 것을 의미한다[7]. 2009년부터 매년 발표되고 있으며, 가장 최근에는 2011년 9월 13일에 발표되었다. TOP 25는 크게 "컴포넌트의 불안전한 상호작용(Insecure Interaction Between Components)", "위험한 리소스 관리(Risky Resource Management)", "미흡한 방어(Porous Defenses)"의 세 가지 카테고리로 분류되어 있다[8].

Top25를 보면 각 위험도가 높은 순서대로 순위가 매겨져 있으며 카테고리별로 특징을 나눠 소프트웨어 취약점을 정의하고 있다. 컴포넌트의 불안전한 상호작용(Insecure Interaction Between Components)은 서로 다른 컴포넌트, 모듈, 프로그램, 프로세스, 스레드, 시스템 사이에서 발생하는 컴포넌트들 간의 불안전한 데이터 송수신을 의미한다. 위험한 리소스 관리(Risky Resource Management) 카테고리는 시스템의 리소스 생성, 사용, 전달 등에서 발생하며 소프트웨어 상에서 시스템에 중요한 리소스의 관리나 전달에서 발생하는 취약점을 의미한다. 마지막으로 미흡한 방어(Porous Defenses)는 잘못 이용되고 악용되거나 단순히 무시되는 방어적인 프로그래밍 기술을 정의한다.

본 논문에서는 복합 공격의 패턴을 조합하여 실제 위험성이 높은 모듈을 중심으로 취약점을 분석하는 프로세스를 제안하기 위해 TOP 25의 세 가지 카테고리를 중심으로 패턴을 조합한다. 소프트웨어에 대한 복합 공격의 경우 전혀 다른 공격 패턴의 조합으로 이루어지기 때문에 세가지 카테고리에서 한 가지씩을 추출하여 새로운 공격 패턴의 조합을 통해 실제 모듈간의 관계를 기반으로 분석한다. 복합 공격의 패턴 조합은 4절에서 자세히 서술한다.

2.2 결합도(Coupling)

결합도는 모듈 또는 클래스들이 상호 의존하는 정도를 나타내는 것으로 모듈이나 클래스들 간의 결합도를 최소화시켜야 모듈의 독립성이 증가되며, 시스템 전체의 성능이 좋아지고 복잡도가 줄어들게 된다. 또한 두 개의 모듈이 서로 강하게 연결되면 상호간의 메

시지 전송이 잦아지고, 주고받는 데이터들의 수가 많아지며 이는 곧 오류 발생이나 보안 취약성의 파급효과(ripple effect)가 잦아진다[9][10].

결합도는 아래의 [표 1]과 같이 총 6가지의 구분을 가진다. 먼저 내용 결합도(content couplint)는 모듈 x가 모듈 y의 데이터나 제어 정보를 사용하는 결합도를 의미하며 결합 단계가 가장 높다. 공통 결합도(Common coupling)은 모듈 x와 y가 많은 전역변수를 같이 참조하는 경우를 의미한다. 제어 결합도(Control coupling)는 모듈 x가 모듈 y의 파라미터를 제어하는 경우 측정되는 결합도이다. 스탬프 결합도(Stamp coupling)는 각 모듈간 자료구조가 전달되는 경우를 의미하며, 데이터 결합도(Data coupling)는 단순한 모듈간의 매개변수 전달을 의미한다. 결합도를 정리하면 아래 [표 1]과 같다.

높은 결합도를 가진 시스템에서는 소프트웨어 취약점이 다른 모듈로 전달되어 시스템 전체의 위험도가 높아질 수 있다[13]. 최근 Istehad Chowdhurya, Mohammad Zulkernineb의 연구에 따르면 소프트웨어의 품질 요소 중 응집도(Cohesion), 결합도(Coupling), 복잡도(Complexity)를 이용하여 소프트웨어의 취약점을 분석하고 예측하는 프로세스를 설계하고 구현하였다[10]. 해당 연구에 따르면 결합도는 소프트웨어에 위험이나 악의적인 공격이 가해지면 하나의 모듈 뿐만 아니라 결합되어 있는 모듈로 위험이 전파되어 위험해 질 수 있다고 하였다. 따라서 결합도는 하나의 모듈이 아닌 여러 개의 모듈의 관계를 통해 소프트웨어에 가해진 손상 또는 피해들을 다른 모듈로까지 전달할 수 있는 소프트웨어 특성이 된다.

결합도 메트릭은 개체들 사이의 관계성을 측정한다. 예를 들면 하나의 메소드에서 다른 메소드를 호출하였다면 두 메소드를 결합되어 있다고 한다. 그리고 만약 클래스 C1에서 클래스 C2로 데이터를 넘겨준다면 C1과 C2 또한 결합되어 있다고 말한다. 즉 위의 [표 2]에서 서술한 바와 같이 두 개의 개체가 결합되어 있다는 것은 어떠한 데이터를 넘겨주거나, 같은 데이터를 공유하거나, 다른 메소드 또는 클래스를 호출하는 경우이다. 결합도를 측정하는 방법은 다양하나 보안적인 특성상 본 논문에서는 모듈간 또는 클래스간의 정보의 흐름에 기반을 둔 fan-in, fan-out 메트릭을 중심으로 결합도를 측정하고 가장 큰 정보의 흐름을 가진 모듈을 통해 조합된 소프트웨어 취약점을 테스트 하여 복합적인 공격 패턴을 예측한다.

Henry-Kafura의 fan-in, fan-out 메트릭은 정보의 흐름에 기반한 메트릭으로 변수 전달(parameter passing), 전역 변수 접근(Global variable access), 입력 데이터 또는 출력 데이터와 같은 모듈 내의 정보 흐름을 측정한 것이다. fan-out은 클래스에 의해서 다른 클래스로 참조되는 수로 정의된다. 또한 fan-in은 클래스를 참조하는 다른 클래스의 수로 정의된다. 또한 측정된 fan-in, fan-out 메트릭을 이용하여 복잡도를 측정하였다. 해당 정의는 아래의 식 1과 같다[14].

$$Module's\ Complexity,\ CP = (fan-in \times fan-out)^2 \quad (1)$$

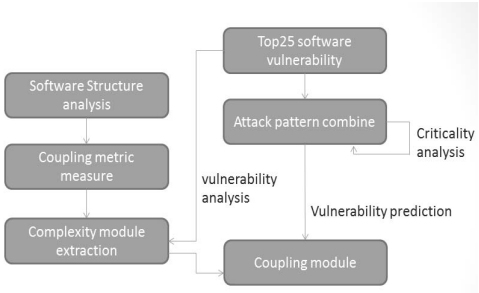
이 외에도 CBO(Coupling Btween Objects), RFC(Response for Class), NOC(Number of Children), MPC(Message Passing Coupling), DIT(Depth of Inheritance Tree) 등의 소프트웨어 결합도 메트릭으로 소프트웨어의 결합도를 측정할 수 있다[15].

[표 1] Fenton과 Melton의 결합도 순위 지정[11][12]

결합도 종류	순위	정의
내용 (Content)	5	모듈 x가 모듈 y의 내부자료나 제어 정보 사용
공통 (Common)	4	모듈 x와 모듈 y가 같은 전역변수를 참조
제어 (Control)	3	모듈 x가 모듈 y를 호출할 때 모듈 y의 제어를 지시하는 데이터를 매개변수로 사용
스탬프 (Stamp)	2	모듈 x가 모듈 y에 자료 구조를 전달
데이터(Data)	1	모듈 x가 모듈 y에 단순한 매개변수 전달
없음 (No Coupling)	0	모듈 x와 모듈 y의 상호작용 없음

III. 소프트웨어 결합도 측정과 파급 효과 측정

소프트웨어의 결합도를 측정하는 것은 모듈간의 상호 연결을 측정하는 것이다. 즉 결합도 메트릭을 이용하여 모듈간의 인터페이스 복잡도 모듈의 진입(Fan-in)과 참조점(Fan-out)과 다른 모듈로 전달되는 데이터의 형태 등을 측정할 수 있다. 본 연구에서는 소프트웨어의 결합도를 기반으로 하여 취약점을 분석하고 복합 공격 패턴의 조합으로 복합적인 공격이 발생하는 중요 모듈간의 위치와 공격 방법을 예측하는 프로세스를 제안한다. 전체적인 프로세스는 아래의 [그림 1]과 같다.

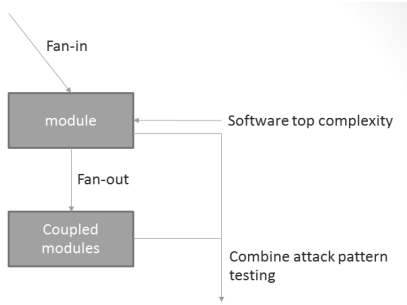


〔그림 1〕 복합 공격 패턴 예측 프로세스

소프트웨어의 취약점을 분석하고 복합적인 공격 패턴을 예측할 수 있는 전체적인 프로세스는 〔그림 1〕과 같이 진행된다. 먼저 소프트웨어의 소스코드를 기반으로 해당 소프트웨어의 전제적인 구조를 분석한다. 소프트웨어의 내부 모듈간의 데이터 흐름이나, 참조 관계, 호출관계들을 분석하고, 결합도를 측정할 수 있는 모듈의 관계를 분석하는 단계인 "Software Structure anlysis" 단계를 거치고 나서 실제 결합도 메트릭을 이용하여 결합도를 측정한다. 결합도 메트릭을 이용하여 소프트웨어를 측정하면, 모듈간의 정보흐름을 수치적으로 표현하여 해석할 수 있으며, 소프트웨어의 보안적인 취약점을 분석하기 위해 모듈 내의 정보 흐름을 예측할 수 있다. 결합도를 측정하고 모듈 내의 복잡도를 계산하는 식 1을 통해 전체 소프트웨어의 많은 모듈 중에서 가장 복잡도가 높은 모듈을 추출할 수 있다. 복잡도가 가장 높은 모듈은 소프트웨어의 취약점을 내포하고 있을 가능성이 높으며, 복잡도가 높은 모듈부터 우선적으로 Top 25를 이용한 소프트웨어 취약점 분석을 한다.

결합도를 분석하여 가장 복잡도가 높은 모듈 순서대로 소프트웨어 취약점을 분석하고, 해당 모듈에서 취약점이 발견되었다면, 해당 모듈과 상호 작용을 하는 즉 결합되어 있는 모듈을 중심으로 복합 공격 패턴을 기반으로 복합 공격이 가능한 공격 패턴을 예측하고 테스트 할 수 있는 단계를 거친다. 기존에 소프트웨어의 취약점을 발견하고 방어 코드를 추가하여 소프트웨어를 악의적인 접근으로부터 차단시키는 형태로 보안적인 취약점을 해결하였다면 본 연구에서는 최근 복합적인 공격 패턴을 분석하고, 복합 공격이 가능한 여러 형태를 추출하여 소프트웨어의 결합도 분석과 함께 복합 공격이 가능한 모듈에 대해 예측하고 테스트할 수 있게 제안하였다.

〔그림 2〕를 보면 모듈 사이의 결합도를 이용하여 복합 공격 패턴을 테스팅하는 단계를 개념도로 표현하



〔그림 2〕 결합 모듈 테스팅

였다. 소프트웨어의 전체 모듈 중에서 가장 복잡도가 높은 모듈을 선정하고 해당 모듈에 대해 소프트웨어 취약점 분석을 수행한 다음, 복잡도가 가장 높은 모듈과 결합되어 있는 다른 모듈에 대해 복합 공격 패턴에 대해 테스트를 수행한다. 〔그림 2〕와 같이 소프트웨어 내부의 모듈간의 결합도를 기반으로 Top 25의 취약점에서 생성된 복합 공격 패턴 테스트를 수행할 수 있는 프로세스를 설계하였다.

IV. 공격 패턴의 조합

본 논문에서는 소프트웨어의 공격 패턴의 조합을 위해 TOP 25를 기반으로 가능한 공격 패턴을 조합하여 정의 하였다. CWE와 SANS의 TOP 25 Most Dangerous Software Errors의 문서를 살펴보면 각 소프트웨어 취약점에 대해 자세한 설명이 포함되어 있고 각 취약점의 마지막에 관련된 공격 패턴들이 서술 되어 있다. 따라서 Top 25에서 서술된 관련된 공격 패턴을 기반으로 하여 세 가지 카테고리에서 서로 관련된 공격 패턴의 가능한 조합을 분석하여 정의하였다.

본 논문에서 사용되는 기호는 아래의 〔표 2〕과 같이 정의 된다.

Top 25의 관련된 공격 패턴은 CAPEC의 ID를 기반으로 정의되어 있으며 이 관련 패턴을 중심으로 복합 공격의 패턴을 추출한다. 복합 공격 패턴은 두 개의 공격 패턴의 조합으로 구성되며, 각기 다른 카테고리에서 연관된 공격 패턴의 조합으로 구성된다. 먼저 관련된 공격의 패턴을 정의하면 다음 식 2와 같다.

$$RP_{C_n} = set\ of\ Relation\ Pattern\ in\ category\ n \quad (2)$$

RP_{C_n}는 Top 25의 카테고리 n의 관련 패턴의 집합을 의미한다. 따라서 Top 25의 세 가지 카테고리의

〔표 2〕 Notation

Notation	definition
RP_{C_n}	Top 25에서 정의된 소프트웨어 취약점 카테고리에서 각 소프트웨어 취약점의 관련 패턴들의 집합
$D-RP_{C_AC_B}$	카테고리 A의 RP_{C_A}와 카테고리 B의 RP_{C_B}의 각각의 ID를 비교하고 일치하는 ID가 있는 소프트웨어 취약점의 집합
$inD-RP_{C_AC_B}$	카테고리 A의 RP_{C_A}의 내부적인 연관 패턴의 ID를 비교하여 일치하면 합집합으로 만든 후 카테고리 B의 RP_{C_B}와 각각의 ID를 비교하여 일치하는 ID가 있는 소프트웨어 취약점의 집합
$Mach(RP_{C_A}, RP_{C_B})$	카테고리 C_A와 카테고리 C_B의 소프트웨어 취약점의 관련 패턴이 일치하는지를 검사하는 함수
$Find(RP_{C_A}\&\&RP_{C_B})$	카테고리 C_A와 카테고리 C_B의 소프트웨어 취약점의 관련 패턴이 일치하는지를 찾는 함수
$loop(inD-RP_{C_AC_A})$	내부 카테고리 취약점의 관련 패턴을 비교할 때, 원하는 수만큼 과정을 반복한다.
$Criticality(D-RP_{C_AC_B})$	$D-RP_{C_AC_B}$의 중요도 측정 결과
$Criticality(inD-RP_{C_AC_B})$	$inD-RP_{C_AC_B}$의 중요도 측정 결과
$Rank\ of\ P_{C_A}$	Top 25에서 카테고리 C_A 정의된 각 소프트웨어 취약점의 랭크 값
$Count\ of\ mach(P_{C_A};P_{C_B})$	카테고리 C_A와 카테고리 C_B의 속해 있는 소프트웨어 취약점에서 두 취약점의 관련 패턴들이 일치하는 수
$Min(Rank\ of\ P_{C_A})$	$inD-RP_{C_AC_B}$에서 카테고리 내부 관련 패턴이 일치하는 경우 두 소프트웨어 취약점 중 더 작은 랭크 값
$S-P_{C_A}$	$inD-RP_{C_AC_B}$에서 카테고리 내부적으로 비교하여 일치하는 경우 두 취약점의 관련 패턴을 합한 합집합

관련 패턴의 집합은 Top25의 Relation Pattern ID의 집합과 동일하다. 복합 공격 패턴은 각기 다른 카테고리에서 두 개의 패턴 조합으로 생성되기 때문에 연관 패턴들이 일치하는 경우 복합 공격 패턴으로 추출한다. 복합 공격 패턴을 생성하는 방법으로는 서로 다른 카테고리에서 직접적으로 연관 공격 패턴의 일치성을 확인하여 복합 공격 패턴을 추출하는 방법과 각 카테고리에서 내부적인 연관 패턴의 일치성을 조합하여 외부 카테고리의 소프트웨어 취약점을 비교하여 추출하는 간접적인 방법이 있다. 아래의 식 3에서 직접적인 연관 패턴의 일치성을 비교하여 복합 공격 패턴을 생성하는 정의하였고, 식 4에서는 간접적인 연관 패턴의 조합 방법을 정의하였다.

$$D-RP_{C_AC_B} = Mach(RP_{C_A}, RP_{C_B}) = Find(RP_{C_A}\&\&RP_{C_B}) \quad (3)$$

$$n \bullet inD-RP_{C_AC_B} = Mach((n-1) \bullet loop(inD-RP_{C_AC_A}), RP_{C_B}), \quad n \geq 2 \quad (4)$$

$D-RP_{C_AC_B}$는 관련 패턴 카테고리 A의 집합과 관련 패턴 카테고리 B의 집합의 각각의 ID를 비교하고 하나라도 일치하는 ID가 있는 경우 직접적인 관련 패턴(Direct Relation Pattern)의 집합으로 정의한다. 실제 예시를 들면 아래의 식 5와 같다.

$$\begin{aligned} &RP_{C_1} = \{\{7, 66, 108, 110\}, \{6, 15, 43, 88, 108\} \ldots \{194\}\} \\ &RP_{C_2} = \{\{8, 9, 10, 14 \ldots\}, \{23, 64, 76, 79, 139\} \ldots \{92\}\} \\ &D-RP_{C_1C_2} = Mach(RP_{C_1}, RP_{C_2}) = Find(RP_{C_1}\&\&RP_{C_2}) \\ &= \{\{CWE-352, CWE-829\}\} \end{aligned} \quad (5)$$

$n \bullet inD-RP_{C_AC_B}$는 $D-RP_{C_AC_B}$에서 추출되는 복합 공격 패턴은 아니지만 같은 카테고리에서 관련 패턴 집합의 일치성을 먼저 검사 한 후 일치하는 패턴이 있다면 해당 소프트웨어 취약점과의 연관 패턴들의 합집합을 만들어 다른 카테고리의 존재하는 소프트웨어 취약점의 연관 패턴과 일치성을 검사해 복합 공격 패턴 집합으로 정의된다. 본 논문에서는 2단계까지의 복합 공격 패턴을 생성하여 테스트 하였지만 실제 하나의 카테고리 안에서 연관된 패턴의 조합은 재귀적으로 반복 될 수 있기 때문에 n번의 루프를 통해 반복될 수 있다.

연관 패턴들과의 일치성을 기준으로 하여 복합 공격의 패턴을 생성한 결과는 아래의 〔표 3〕와 같다.

추출된 복합 공격 패턴은 직접적인 관련성이 있는 $D-RP_{C_AC_B}$이 7개, 간접적인 패턴 일치성을 보이는 복

〔표 3〕 복합 공격 패턴

복합 공격 패턴 추출 방법	생성된 복합 공격 패턴
$D-RP_{C_AC_B}$	CWE-79, CWE-732
	CWE-434, CWE-862
	CWE-434, CWE-863
	CWE-352, CWE-306
	CWE-352, CWE-732
	CWE-352, CWE-829
	CWE-434, CWE-732
$n \bullet indD-RP_{C_AC_B}$	CWE-494, CWE-352
	CWE-79, CWE-862
	CWE-79, CWE-863

합 공격 패턴인 $n \bullet indD-RP_{C_AC_B}$가 3개이다. 이와 같이 추출된 복합 공격 패턴을 이용하여 한가지의 소프트웨어 취약점만을 테스트 하는 것이 아닌, 복합적인 공격이 가능한 테스트 케이스를 추출하여 테스팅을 한다. 즉 소프트웨어의 결합도 메트릭에서 추출된 가장 위험성이 높은 모듈의 소프트웨어 취약점을 분석하고, 해당 모듈과 연관성이 있는 모듈을 중심으로 생성된 복합 공격 패턴의 테스팅을 수행하게 되면 단일 소프트웨어 취약점 테스트가 아닌 복합적인 공격의 테스팅을 하여 분석할 수 있게 된다.

Top 25의 연관 패턴을 이용하여 복합 공격 패턴을 추출하고 이를 테스팅 하기 위해서는 복합 공격 패턴의 중요도 산출이 필요하다. 따라서 각 복합 공격 패턴의 중요도를 산출하기 위해서 $D-RP_{C_AC_B}$를 통해 계산된 복합 공격 패턴인지, 또는 $n \bullet indD-RP_{C_AC_B}$를 통해 계산된 복합 공격 패턴인지를 기준으로 해서 중요도 산출을 하는 중요도 계산식을 아래의 식 6과 같이 정의하였다.

$$Criticality(D-RP_{C_AC_B}) = \left(\frac{1}{Rank\ of\ P_{C_A}} + \frac{1}{Rank\ of\ P_{C_B}}\right) \times Count\ of\ mach(P_{C_A}, P_{C_B}) \quad (6)$$

직접적인 연관이 있는 $D-RP_{C_AC_B}$의 중요도는 최종적으로 도출된 복합 공격 패턴들의 순위인 Rank의 값을 역수를 취해 더하고, 두 공격 패턴의 연관 패턴들의 일치된 수를 곱해 구한다. Rank 값은 결국 〔표 4〕에서 $D-RP_{C_AC_B}$로 도출된 복합 공격 패턴들의 Rank를 의미하게 된다. 복합공격 패턴으로 추출된 두 개의 소프트웨어 취약점의 연관 패턴들이 몇 개가 일치했는가가 추가로 곱해져 중요도로 계산된다.

$$Criticality(indD-RP_{C_AC_B}) = \left(\sum_{m=1}^{n-1} m' Min(Rank\ of\ P_{C_A}) + \frac{1}{Rank\ of\ P_{C_B}}\right) \times Count\ of\ mach(S-P_{C_A}; P_{C_B}) \times 0.1^{n-1} \quad (7)$$

간접적인 연관이 있는 $n \bullet indD-RP_{C_AC_B}$의 경우에는 기존의 직접 연관이 있는 $D-RP_{C_AC_B}$와 비교해 중요도가 낮아지며, 실제 하나의 카테고리 내부에서 반복적으로 내부 패턴끼리 비교하며 관련 패턴끼리 일치하는 공격 패턴 ID가 존재하면 두 개의 관련 패턴 집합을 합하여 다른 카테고리에 존재하는 패턴과 비교한다. 실제 본 논문에서 추출한 복합 공격 패턴들의 중요도를 산출하면 아래의 〔표 4〕과 같다.

〔표 4〕 복합 공격 패턴의 중요도

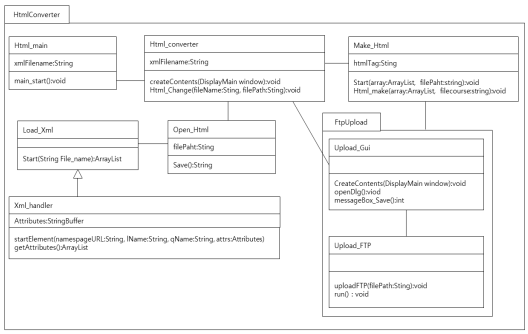
생성된 복합 공격 패턴	연관	중요도	Rank
CWE-79, CWE-732	$D-RP_{C_AC_B}$	0.308	2
CWE-434. CWE-862	$D-RP_{C_AC_B}$	0.277	4
CWE-434, CWE-863	$D-RP_{C_AC_B}$	0.177	5
CWE-352, CWE-306	$D-RP_{C_AC_B}$	0.283	3
CWE-352, CWE-732	$D-RP_{C_AC_B}$	0.142	7
CWE-352, CWE-829	$D-RP_{C_AC_B}$	0.145	6
CWE-434, CWE-732	$D-RP_{C_AC_B}$	0.339	1
CWE-494, CWE-352	$n \bullet indD-RP_{C_AC_B}$	0.041	8
CWE-79, CWE-862	$n \bullet indD-RP_{C_AC_B}$	0.015	10
CWE-79, CWE-863	$n \bullet indD-RP_{C_AC_B}$	0.031	9

V. 사례 연구

지금까지 소프트웨어 결합도를 이용한 측정을 통해 소프트웨어의 구조를 분석하고 각 모듈간의 상호작용을 이용하여 복합 공격 패턴이 가능한 모듈을 추출하여 테스팅 하는 프로세스를 제안였다. 따라서 실제 Java 소스코드 분석을 통한 결합도와 복합 공격 패턴의 연관성을 증명하고 실제 소프트웨어의 취약점을 이용한 테스팅을 수행한다. 본 사례 연구에서는 Java 소스코드를 이용하여 소프트웨어 취약점 분석을 먼저

한 후 연관 있는 복합 공격 패턴을 기반으로 소프트웨어 취약점이 포함된 클래스 파일과 결합되어 있는 클래스들을 분석하여 공격 패턴의 실제 적용 사례를 분석한다. 또한 복합 공격 패턴을 통해 소프트웨어에 포함된 복합 공격 패턴이 가능한 모듈간의 결합도를 분석하여 실제 복합 공격 패턴과 결합도의 상관관계를 증명하는 단계를 거친다.

먼저 Java 소스코드 분석을 할 프로그램은 SWT를 기반으로 만든 애플리케이션으로 Ftp 통신을 이용한 파일 업로드를 하는 프로그램이다. xml 파일에서 필요한 파일을 읽어들여 해당 소스 코드에서 지정하는 위치에 파일을 업로드하며, 또한 애플리케이션에서 만든 GUI 프로그래밍 코드를 html로 변환하는 소스코드가 포함된 프로그램이다. 전체 구조를 살펴보면 아래의 〔그림 3〕과 같다.



〔그림 3〕 Upload 프로그램의 구조

위의 〔그림 3〕의 구조와 같이 전체 프로그램은 크게 Html 파일을 변환하는 HtmlConverter 패키지와 파일을 업로드하는 FtpUpload 패키지로 나눠진다. 실제 프로그램의 결합도를 측정하면 아래의 〔표 5〕와 같이 측정되며, 결합도는 Java의 메트릭 측정을 지원하는 툴인 JHawk Metric 프로그램을 이용하여 측정되었다.

〔표 5〕 Upload 프로그램 결합도 측정 결과

클래스	Fan-in	Fan-out	Complexity
Html_main	1	0	0
Html_converter	1	0	0
Load_Xml	1	1	1
open_Html	1	1	1
Xml_handler	1	0	0
Make_Html	1	0	0
Upload_Gui	2	1	4
Upload_FTP	1	0	0

결합도 측정 결과 가장 높은 복잡도를 가진 클래스는 Upload_Gui 클래스로 측정되었다. 따라서 Upload_Gui 클래스는 실제 다른 클래스보다 결합도가 높고, 실제 소프트웨어에 공격을 받았을 경우 파급효과가 일어날 가능성이 높은 모듈이 된다. Upload_Gui 클래스를 중심으로 먼저 소프트웨어 취약점을 분석한 결과 Upload_Gui에서는 사용자에게 파일을 업로드 할 수 있는 권한을 설정하는 사용자의 아이디와 패스워드를 입력받아 서버와 연결해 파일을 업로드 할 수 있는 기능을 포함하고 있지만, 실제 사용자의 아이디와 패스워드에 대해 인증부분이 포함되어 있지 않았다. 해당 소스 코드 부분은 아래의 〔그림 4〕와 같다.

〔그림 4〕 인증되지 않은 사용자

해당 소프트웨어 취약점은 CWE와 SANS에서 지정한 Top 25의 소프트웨어 취약점 중에 하나인 "CWE-862 Missing Authorization"에 해당한다. 따라서 본 논문의 4절에서 공격 패턴의 조합을 통해 생성된 복합 공격 패턴에 따르면 CWE-862와 복합적으로 공격이 가능한 패턴은 아래의 〔표 6〕과 같이 CWE-434와 CWE-79가 해당한다.

〔표 6〕 "CWE-862"의 복합 공격 패턴

복합 공격 패턴	중요도	Rank
CWE-434. CWE-862	0.277	4

Upload_Gui클래스와 결합되어 있는 클래스는 Html_converter, Make_Html, Uplod_FTP이다. 세 클래스를 중심으로 소스 코드를 분석한 결과 Upload_FTP 클래스에서 파일 업로드를 하는 과정에서 파일의 위험도를 막을 수 있는 방어코드가 포함되어 있지 않았다. 파일을 서버에 업로드하는 소스 코드는 아래의 〔그림 5〕와 같다.

```
System.out.println("test defaultPath : " + defaultPath + Path);
ftpClient.changeWorkingDirectory(defaultPath + Path);

// xml파일 업로드

for (int i = 0; i < fileNameList.length; i++) {
    FileInputStream inputStream = new FileInputStream(filePath
            + "\\" + fileNameList[i]);
    String temp = fileNameList[i];
    System.out.println(temp);
    Boolean result = ftpClient.storeFile(temp, inputStream);
```

〔그림 5〕 파일 업로드 소스코드

파일에 대한 내용에 대해 무제한적으로 업로드를 하게 되므로 파일 업로드를 이용한 소프트웨어와 서버 쪽에 악의적인 접근의 가능성이 높을 수 있다. 파일 업로드를 이용한 소프트웨어를 공격 할 수 있는 공격 취약점은 처음 Top 25에서 정의한 바와 같이 "CWE-434, Unrestricted Upload of File with Dangerous Type"이다. 따라서 처음 결합도 측정을 통해 발견된 Upload_Gui 클래스의 사용자의 인증이 부족한 소프트웨어의 취약점 뿐만 아니라 해당 클래스와 결합되어 있는 클래스인 Upload_FTP클래스의 파일 업로드 부분에 대한 복합적인 소프트웨어 취약점을 이용한 공격에 대해서도 고려하여 전체적인 안전성을 높여야 한다.

위의 분석된 내용과 같이 결합되어 있는 두 개의 클래스를 중심으로 복합 공격 패턴을 분석한 결과 한 가지 공격 방식이 아닌 두 가지 이상의 공격 방식에 대한 방어 코드가 소프트웨어의 내부에 필요한 것으로 증명되었다. 따라서 한 가지의 공격에 대한 소프트웨어 취약점 분석이 아닌 두 가지 이상의 공격 패턴의 조합을 이용하여 실제 모듈 또는 클래스 간의 결합도를 기반으로 관계성을 측정하여 방어 코드를 분석해야 한다. 또한 한 모듈이나 클래스에서 한 가지 공격에 대해 방어 코드가 짜여져 있다고 하더라고 해당 모듈과 결합되어 있는 다른 모듈에 대해 복합 공격 패턴의 분석과 테스트가 필요하다.

VI. 결 론

지금까지 소프트웨어의 취약점을 분석하기 위해 소프트웨어의 구조적인 분석 방법 중에 하나인 결합도 메트릭을 이용하여 모듈간의 상호 작용을 기반으로 하는 복합 공격 패턴의 테스팅 방법론을 제안하고 분석하였다. 결합도 메트릭을 이용하여 결합도를 측정하는 것은 소프트웨어 내부의 모듈간 정보 흐름, 데이터의 변환과 다른 모듈의 컨트롤 등을 수치적으로 표현할 수 있기 때문에 보안적인 중요도가 높은 모듈을 선별하고, 해당 모듈과의 결합도를 기반으로 복합 테스팅을 위한 기반을 제시할 수 있었다. 또한 복합 공격 패턴을 정의하고 CWE와 SANS에서 매년 발표하고 있는 치명적인 소프트웨어 취약점인 TOP 25를 기반으로 복합 공격 패턴을 조합하고 중요도를 측정할 수 있는 방법을 제안하였다. 따라서 기존에 정의되어 있는 소프트웨어의 취약점을 기반으로 복합적인 공격 패턴을 추출하고 생성하여 테스트 할 수 있도록 일련의 프로세스를 제시함으로써, Top 25 뿐만이 아니라 더 많은 소프트웨어의 취약점을 기반으로도 새로운 공격 패턴의 조합을 만들어 낼 수 있는 기초를 다질 수 있었다.

보안 자료는 비공개를 원칙으로 하여 실무 보안 문제를 다루는 실제 적용 사례를 제시하는데는 한계가 있었다. 따라서 실제 더욱 다양한 소프트웨어 취약점을 분석하고, 복합적인 공격 패턴을 조합하여 테스트 할 수 있도록 실제 사례를 분석하는 연구가 필요하다. 또한 현재에는 결합도 메트릭을 Fan-in과 Fan-out을 이용하여 측정하였지만 향후 연구에서는 CBO, RFC, NOC, MPC, DIT 등과 같이 다양한 결합도 메트릭을 사용하여 실제 보안적인 취약점을 가장 정확하게 판단할 수 있도록 비교 분석할 것이다.

참고문헌

〔1〕 Gartner, "Now is the time for security at application level," 2005. 12

〔2〕 Gary McFraw, "Software Security: Building Security In, Addision-Wesley," 2006

〔3〕 John Viega, Gary MaFraw, "Sofrware Security: How to Avoid Security Problems the Right Way, Addsion-Wesley," 2006

〔4〕 손윤식, 오세만, "자바 시큐어 코딩," *한국정보과학회*, 정보과학회지, 제 28권 제 2호, pp55-62, 2010 . 2.

〔5〕 문일룡, 오세만 "모바일 애플리케이션을 위한 취약점 분석기의 설계 및 구현," *Journal of Korea Multimedia Society Vol. 14*, No. 10., pp1335-1347J, October 2011.

〔6〕 Mark G. Graff, Kenneth R. Van Wyk, "Secure Coding: Principles and Practices," *O´Reilly&Associates*, Inc., pp.14,

Sebastopol, CA, 2003.
〔7〕 오준석, 최진영, "시큐어 코딩을 적용한 입력유효성 검사기법제안," *2010 한국컴퓨터종합학술대회 논문집* Vol.37, No.1(B), pp73-76, 2010.
〔8〕 CWE, "2011 CWE/SANS Top 25 Most Dangerous Software Errors".
〔9〕 Roger S. Pressman, "Software Engineerignn A Practitioners Approach, Fourth Edition," *McGrawHill*, 1997.
〔10〕 Istehad Chowdhurya, Mohammad Zulkernineb, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," *Journal of Systems Architecture*, Volume 57, Issue 3, pp. 294 - 313, March 2011
〔11〕 Norman Fenton and Austin Melton, "Deriving Structurally Based Software Measures," *J. System Software*, pp. 177 - 187, 1990.12.
〔12〕 H. Dhama, "Quantitative Models of Cohesion and Coupling in Software," *Journal of System and Software*, pp.65 - 7, 1995
〔13〕 I. Chowdhury, B. Chan, M.Zulkernine, "Security metrics for source code structures, in : Proceedings of the Fourth International Workshop on Software Engineering for Secure Systems," *Leipzig Germany*, pp.57-64, May 2008,
〔14〕 SALLIE HENRY, DENNIS KARUFR, "Sofrtware Structure Metrics Based on Information Flow," *IEEE Transaction on software engineering*, Vol.SE-7, No.5, September, 1981.
〔15〕 Sherif Yacoub, Tom Robinson, H. Ammar, "Dynamic Metrics for Object Oriented Designs," *IEEE*, 1999.

〈著者紹介〉

권 예 진 (Ye-jin Kwon) 학생 회원
2011년 2월: 단국대학교 컴퓨터과학과 졸업
2011년 9월~현재: 단국대학교 전자계산학과 석사과정
〈관심분야〉 소프트웨어 테스팅, 소프트웨어 품질 측정

박 용 범 (Young B. Park) 일반 회원
1991년 6월: N.Y. Polytechnic University Couputer Sci. & Eng. Ph.D.
1993년 3월~현재: 단국대학교 컴퓨터과학 교수
〈관심분야〉 지능형 소프드웨어 공학, 보안 소프트웨어 개발