소프트웨어의 잠재적 오류가능성 및 보안취약점 비교분석 연구

이성민*, 오준석**, 최진영**
*고려대학교 디지털정보미디어공학과
**고려대학교 컴퓨터정보통신공학과
e-mail: idreply@korea.ac.kr, {jsoh, choi}@formal.korea.ac.kr

Comparative analysis on potential error-possibility and security vulnerability in software.

Seoung-Min Lee*, Joon-Seok Oh**, Jin Young Choi**
*Dept. of Digital Information and Media Engineering, Korea University
**Dept. of Computer Information & Communication, Korea University

요 약

컴퓨터의 대중화와 네트워크의 발달로 인해 우리 사회는 컴퓨터와 통신이 없이는 생각조차 할 수 없는 시대에 살고 있으며, 또한 많은 정보시스템들을 일상생활 속에서 접하고 살고 있다. 하지만 소프트웨어들의 보안 취약점으로 인해 개인뿐만 아니라 기업은 물론이고 국가에 이르기까지 그 위험성은 모두 열거할 수 없을 정도이며 그에 따른 정보보호의 중요성이 더욱 강조가 되고 있으며, 어느 시스템도 이러한 정보보호에서 자유로울 수 없다. 이러한 보안적 및 오류의 위험은 현재 개발되고 있는 소프트웨어뿐만 아니라, 정상적으로 운영되고 있는 시스템도 예외가 될 수 없다.

이러한 보안취약점 및 오류의 위험은 소프트웨어 개발시 방어적 프로그램(Defensive Programming)을 포함하는 시큐어 코딩(Secure Coding)기법을 적용하여 보다 안정적인 프로그램을 개발 할 수 있다.

본 논문에서는 소프트웨어의 잠재적인 오류를 발생할 수 있는 요소와 보안 취약점으로 인하여 생길 수 있는 요소들을 살펴보고 실제 java로 개발되어 운영되고 있는 시스템들의 보안 취약점을 분석하였다.

1. 서론

지난 20년간, 현대사회는 컴퓨터사용의 대중화로 인하여 사람들의 라이프스타일이 완전히 바뀌었다. 대부분의 생활이 컴퓨터와 관련되어 있으며, 이러한 컴퓨터는 이미 우리의 생활에서 필수적인 요소로 자리 잡았다. 게다가 오늘날 우리는 인터넷을 통해 전세계의 컴퓨터들이 서로 연결돼 있기 때문에 지구 저편에 있는 사람과 일상적인 대화를 나누거나 수백만 달러의 거래를 아주 빠르고 값싸게 할 수 있다. 개인용 컴퓨터가 널리 쓰이고 인터넷에 연결하기가 쉬워졌고, 새로운 통신 장비에 대한 시장이 커져감에 따라 우리의 여가활용 방법과 비즈니스 방식이 급격히 바뀌고 있다. 이는 지금까지 경험해보지 못한 다양한 사람과 다양한 프로그램들을 접할 기회가 늘어나 새로운 세상을 맛보게 해주는 반면에 그에 따른 위험요소들을 접할 기회가 더 많다는 점을 의미한다. 최근 뉴스에 의하면 국내외 수많은 소프트웨어들이 악의적인 사용자에 의해 침해당하고 있으며, 공격을 받은 소프트웨어는 피싱이나 악성 프로그램의 유출지로 활용되는 등 2차적인 공격 용도로 사용되고 있기 때문에 단순 소프트웨어의 침해의 문제가 아니라고 할 수 있다. 소프트웨어의 침해가 개인의 호기심을 벗어나 범국가적인 사이버 범죄에 활용되는 추세인 반면 소프트웨어에 대한 현재 사회에서의 보안 대책은 그리 체계적이지 못하다. 시간이 갈수록 소프트웨어에 의한 업무의 전산화등으로 인해 정보의 중요한 정보의 양은 늘어가고 있으며 이에 비례하여 정보보호의 중요성도 크게 대두되고 있다.

본 논문에서는 현재 기업에서 1년 이상 실제 사용되고 있는 소프트웨어의 잠재적 오류가능성과 보안취약점을 조사·분석하였고, 이러한 분석을 통하여 여전히 소프트웨어에 남아 있는 오류가능성을 가지고 있는 보안취약점에 대하여 살펴보고자 한다.

2. 주요 이슈사항

2.1 소프트웨어 보안 취약점

소프트웨어 개발에서 코딩은 시스템의 운영, 유지보수

뿐만이 아니라 다양한 분야에서 영향을 미친다. 소프트웨어를 악의적인 위협이나 행위로부터 효과적으로 보호하려면 개발단계의 전 과정에 걸쳐 보안요소를 고려하여 개발하는 것이 중요하다. 단지 개발기간에 쫓겨 보안요소를 간과한다면 취약점이 발생했을 때 기업이 감수해야 하는 피해는 더욱 크다.

대부분의 기업은 개발단계에서 보안을 강구하지 않고, 개발 작업을 끝낸 후, 취약성의 여부에 대한 일종의 모의 해킹이나 점검을 수행하고, 이 단계에서 취약점이 발견되면 이를 제거하기 위한 리엔지니어링을 해야 한다.[1] 따라서 개발초기부터 이러한 취약점을 줄일 수 있는 시큐어 코딩[2]을 적용하면 오히려 개발기간과 비용을 줄일 수 있다.[3] 또한 개발자는 시큐어 코딩의 한 분야인 방어적 프로그램 기법[3, 4]을 코딩에 적용하면 보안취약성으로부터 훨씬 안전한 프로그램을 만들 수 있다. 다음 <표 1>은 방어적 프로그래밍 기법을 간략하게 설명한 것이다.

<표 1> 방어적 프로그래밍 기법[5, 6]

1. 타당하지 않은 입력으로부터 프로그램 보호
- 외부로부터 들어오는 모든 데이터 값을 검사
- 루틴의 모든 매개변수 값을 검사
- 잘못된 입력을 어떻게 처리할 것인지 결정
2. Assertion : 프로그램이 실행될 때 스스로를 검사하는 코드
- 입력(또는 출력)매개변수의 값이 예상된 범위 안에 들어가는지
- 파일이나 스트림이 루틴이 시작될 때(또는 끝날 때) 열려있는지 (또는 닫혀 있는지)
- 파일이나 스트림이 읽기 전용, 쓰기전용, 또는 읽기/쓰기로 열려 있는지
- 입력만 가능한 변수의 값이 루틴에 의해서 변경되지 않는지
- 포인터가 NULL이 아닌지
- 테이블이 실제 값을 포함할 수 있도록 초기화 되었는지
- 최적화되어 있고 복잡한 루틴의 결과가 느리지만 분명하게 작성된 루틴과 일치 하는지
3. 오류 처리 기법
- 중립적인 값을 리턴
- 다음에 오는 타당한 데이터로 대체
- 이전 값과 동일한 값을 리턴
- 가장 가까운 타당한 값으로 대체
- 경고 메시지를 파일에 기록
- 오류코드를 리턴
- 오류처리 루틴이나 객체를 호출
- 지역적으로 가장 잘 작동하는 방법으로 오류 처리
4. 예외
- 무시되어서는 안되는 오류를 프로그램의 다른 부분에 알리기 위하여 예외를 사용
- 정말로 예외적인 조건인 경우에만 예외를 사용
- 책임을 전가하기 위해서 예외사용 금지
- 올바른 추상화 수준에서 오류를 사용
- 예외를 야기한 모든 정보를 예외 메시지에 포함
- 집중된 예외 보고자의 구축을 고려할 것
- 예외의 사용을 규격화 할 것
5. 공격적인 프로그래밍
- Assert가 프로그램을 중단하도록 한다.
- 할당된 모든 메모리를 완벽하게 채워서 메모리 할당 오류를 발견할 수 있도록 한다.
- 파일 형식과 관련된 오류를 발견하기 위해서 할당된 파일만 스트림을 완벽하게 채운다.
- 객체를 삭제하기 전에 쓰레기 데이터로 채운다.
- 모든 오류는 기록하게 할 것
6. 디버깅 보조도구 제거
- 기본적으로 제공되는 전처리기 사용

소프트웨어 공학에서 버그의 발견은 시기와 비용간의 상관관계 대해 이미 많은 분석이 수행되었다. 여기서 얻은 결론은 버그를 빨리 발견할수록 수정하는데 비용이 적게 든다는 것이다.[7] 마이크로소프트가 보안 결함이 있는 소프트웨어를 배포해 놓고, 이를 패치하기 위해 들이는 노력을 생각한다면 같은 버그를 테스트 단계, 혹은 구현 단계에서 발견할 수 있다는 그 경제적 이익은 엄청나다.[7] 예를 들어 자바의 경우는 언어차원에서 멀티쓰레드 프로그래밍을 지원하며, 네트워크를 비롯한 여러 라이브러리에서 쓰레드 사용을 장려하고 있다. 하지만 멀티쓰레드 프로그램의 버그는 프로그램 수행을 예측 할 수 없게 만든다.[6] 항상 잘되다가 가끔씩만 문제가 생기는 버그는 거의 대부분 이 멀티 쓰레드 버그의 속성 때문이다. 하지만 정작 버그 그 자체만 놓고 보면 정말 사소한 개발자의 실수로 밝혀지는 경우가 대부분이다.

```
while (!someCondition) {
    synchronized (lock) {
      lock.wait();
    }
  }
```

위의 예제는 개발자들이 자주 발생시키는 멀티 쓰레드 프로그래밍 관련된 오류이다. 원래는 synchronized 블록에 들어가서 조건 검사를 해야 하는데, 반대로 조건부터 검사하고 락(lock)을 획득한 경우이다. while문의 조건을 검사하는 동안 synchronized 블록 사이에 다른 쓰레드가 조건을 변경할 수 있으므로, 이 예제 프로그램은 레이스 컨디션에 취약하다. 사소한 실수이지만, 이런 종류의 버그를 테스트로 찾아내는 것은 거의 불가능하다. 멀티쓰레드 버그는 타이밍에 민감하기 때문에 보통 소프트웨어가 릴리즈된 후에야 발견된다. 뿐만 아니라 당연하게 버그 파악에 필요한 비용과 시간도 커지게 된다.

2.2 시스템 개발환경

소프트웨어 개발시 한정된 시간과 자원에 의하여 쫓기듯 개발되는 경우가 많다. 그러다 보니 시스템 설계나 개발에 전담으로 프로그램 검사나 해당 전문가가 참여하기가 어렵고, 자꾸 늘어지는 개발 일정 때문에 프로젝트 막바지 무렵이 되기 전에는 전문가에 의한 보안 테스트도 거의 이뤄지지 않는다. 개발 일정에 맞추어서 우선순위를 조정하다보면 구체적으로 당면하지 않은 사소한 코딩에 대해 검사하고 테스트하기 보다는 시스템 오픈일 까지 작동이 되는 시스템을 개발하는 일이 당연히 먼저 고려될 수밖에 없다. 일반적으로 소규모 조직이라면 새로운 시스템을 평가하는데 겨우 며칠의 검토 시간을 투입하는 것이지 세부적이고, 지속적인 테스트를 통한 미묘하고도 중요한 취약점을 찾는 것은 거의 불가능 할 것이다.

개발에 관한 플랫폼과 개발도구가 워낙 편리하게 되어 있어서 초보 개발자라도 아주 강력한 프로그램을 할 수

있어 아주 단시간에 간단한 프로그램을 만들 수 있다. 그러나 작동이 되는 코드를 만드는 것과 안전한 코드를 만들어내는 것은 판이하게 다른 일이다. 소프트웨어의 보안 문제를 미리 예상할 지식이나 경험조차 없는 초보개발자들에 의해 만들어지는 경우가 많다.

2.3 코딩의 오류나 버그로 인해 발생할 수 있는 취약점

잘못된 코딩이나 버그로 발생할 수 있는 위험이나 문제점들은 구지 언급하지 않아도 이미 잘 알고 있는 내용이다.

2.3.1 크로스사이트 스크립팅 : 사용자가 시스템상에서 입력하거나 제어할 수 있는 부분이 다시 사용자에게 보여질 때 발생하는 것으로 초기 취약점을 파악하기 상당히 어려운 경우는 사용자가 제어할 수 있는 입력 값이 즉시 사용자에게 보여지는 것이 아니라 사용자에게 나중에 보여지는 변수에 이용되는 경우가 있다.

2.3.2 SQL인젝션 : 일반적으로 사용자가 입력한 값이 SQ: 쿼리에 포함되고 사용자가 입력한 값이 더해져서 만들어진 SQL 쿼리가 데이터베이스에서 실행될 때 발생하며, “SELECT, INSERT, DELETE, AND, OR, WHERE, ORDER BY”를 포함하는 문자열을 살펴보고 SQL 키워드가 포함된 부분에서 사용자가 입력한 값이 적절하게 처리되는지 확인해야 한다.

2.3.3 경로탐색 : 경로 탐색 취약점에 대한 일반적인 시그니쳐는 사용자가 제어할 수 있는 입력 값에 대해 입력 값 검증 없이 파일 시스템 API에 전달되는 경우다. 대부분의 경우 사용자가 입력한 값은 시스템에 지정된 디렉터리 경로에 더해지고 결과적으로 공격자는 점-점-슬래시 공격 문자열을 입력해서 시스템상에 있는 다른 디렉터리로 접근 가능하며 이 경우 특히 업로드나 다운로드와 같은 기능을 주의해야 한다.

2.3.4 임의의 리다이렉션 : 피싱 공격과 같이 사용자를 임의의 사이트로 리다이렉션시키는 공격

2.3.5 운영체제 명령어 인젝션 : 외부 시스템에 특정 명령어를 실행시키게 하는 코드는 가끔 코드 인젝션 취약점을 가지고 있다.

2.3.6 백도어 비밀번호 : 시스템에 있는 특정 코드가 악의적인 프로그래머에 의해 고의로 숨겨진 코드

2.3.7 네이티브 소프트웨어 버그로는 버퍼 조작에 대해 검사하지 않은 API를 사용할 때 발생하는 버퍼 오버플로우 취약점, 부호의 유무에 따라 발생하는 정수 취약점이 있으며 그 외에도 포맷 스트링 취약점, 소스코드 주석에 의한 취약점 등이 있다.

<표 2> 자바에서의 사용자가 제공한 데이터 확인API

자바 API	내용
getParameter	URL 쿼리 문자열과 POST 요청에서 사용자가 요청한 값을 문자열값으로 저장
getParameterNames	
getParameterValues	
getParameterMap	
getQueryString	요청에 포함된 전체 쿼리 문자열을 반환하고 getParameter 대신 사용
getHeader	문자열 값에 문자열 이름을 매핑
getHeaders	
getHeaderNames	
getRequestURI	쿼리문자열에 포함된 요청 값을 포함한 URL을 반환
getRequestURL	
getCookies	쿠키의이름과 값등 세부사항을 담고 있는 Cookie 객체의 집합을 반환
getRequestedSessionld	요청에서 받은 세션ID를 반환
getinputStream	클라이언트로부터 받은 요청에 대해 다른 표현을 반환하고 다른 모든 API에 의해 획득한 정보에 접근
getReader	
getMethod	HTTP 요청에서 사용된 메소드 반환
getProtocol	HTTP 요청에서 사용된 프로토콜 반환
getServerName	HTTP 호스트 헤더의 값을 반환
getRemoteUser	인증된 사용자에 대해 로그인명을 포함해 사용자에 대한 상세한 정보 반환
getUserPrincipal	

3. 시스템 프로그램 취약 가능성 분석

범용 정적 프로그램 코드 검사도구인 findbugs를 이용한 java 프로그램을 적용하여 검사하였으며, 분석대상 프로그램은 현재 각각 A, B, C, D사의 소프트웨어로 개발되어 현재 1년 이상 운영된 프로그램들을 대상으로 적용해 보았으며, 개발언어는 공통적으로 java로 개발되었으나 각자 다른 개발자가 코딩하였으며, 개발방법 등 서로 아무런 연관이 없으며, 각각의 소속회사에서 보안성 검사를 모두 실시한 프로그램이다.

Findbugs는 java 바이트 코드를 분석하여 잠재적 버그를 찾아주는 도구로 프로그램을 별도의 컴파일 하지 않고서도 분석 할 수 있는 정적분석 도구로써 다음에 나타난 검사결과는 버그나 그 외 취약점이 발생할 수 있는 잠재가능성을 지닌 요소이다.

관계가 없으며, 자체적으로 나름 공통 findbugs에 의하여 수집된 버그가능성 항목들은 잠재가능성을 지닌 요소임을 참고 하여야 한다.

3.1 A, B, C, D사 프로그램 분석

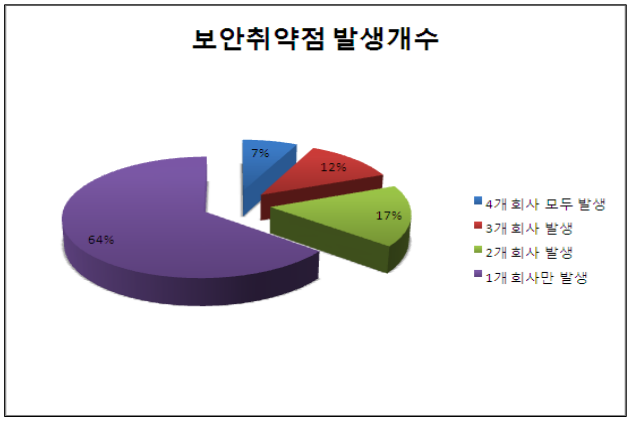
<표 3> A, B, C, D사 프로그램 분석 내용

내 역	A	B	C	D
Call to equals() with null argument	3			
Call to equals() comparing differenct types		1		
Call to equals() comparing unrelated calss and interface			1	
Call to static DateFormat			1	
Class names should start with an upper case letter	20	38		
close() invoked on a value that is always null		3		
Comparison of String parameter using ==or !=	5	1		
Dead store to local variable	26	1	12	2
Exception created and dropped rather than thrown		1		
Exception is caught when Exception is not thrown		3	1	3
Explicit garbage collection; extermely dubious except in benchmarking code			1	
Immediate dereference of the result of readLine()		2		
Inefficient use of keySet iterator instead of entrySet iterator			1	
integral division result cast to double or float		2		
Inconsistent synchronization	1			
Invalid syntax for regular expression	2			
Method call passes null for nonnull parameter		1		
Method concatenates strings using + in a loop	4	20	2	3
Method ignores exceptional return value	4	4		
Method ignores return value		1		
Method might ignore exception	1			
Method invokes inefficient new String(string) constructor		8		
Method invokes inefficient Number constructor; use static valuOfinstead		2		
Method invokes System,exit(...)				3
Method may fail to close stream		1	1	4
Method might ignore exception		135		1
Method names should start with a lower case letter	4	46	1	
Method uses the same code for two branches	1	1		
Non-transient non-serializable instance field in serializable calss		12		
Null pointer dereference		1		
Nullcheck of value previously dereferenced	4	2		
Possible null pointer dereference	3	3		2
Possible null pointer dereference in method on exception path	25			
Primitive value is boxed and then immediately unboxed		1		
Private method is never called				1
Redundant nullcheck of value known to be null	4	3		
Store of non serializable object into HttpSession			1	
Unread field	1	34	1	
Unread field: should this field be static?	3	3	9	16
Unwritten field		1		
Value is null and guaranteed to be dereferenced on exception path	2			
Write to static field from instance method	5			

3.2 공통적으로 발생하는 버그가능성 코드

서로다른 개발자가 각기 코딩한 프로그램이지만 4개의 시스템모두 공통적으로 가지고 있는 버그나 악성코드 발생소지가 있는 항목으로 42개 중 3개로 7%를 차지하고, 적어도 2개 이상 공통적으로 발생할 수 있는 항목은 전체 42개 항목 중 15개로 전체의 36%를 차지한다.

<그림 1> 보안취약점 발생개수



또한 각 시스템의 각 항목당 검출개수 <표4>와 같다.

<표 4> 각 소프트웨어별 오류발생가능 개수

구 분	A사	B사	C사	D사
검출개수	118	331	32	35

이는 개발자의 개발습관이나 개발방법이 프로그램에 있어서 적지 않은 영향을 미치는 것을 의미한다고 볼 수 있으며 그 차이는 10배 이상 차이를 보이고 있다.

4. 결론

본 논문에서는 증가하는 소프트웨어들 중, 현재 1년 이상 정상적으로 운영되고 있는 시스템들의 보안 취약점등을 살펴보았고 이러한 시스템들의 여전히 가지고 있는 문제점들을 짚어 보았다. 시스템의 결함제거는 일종의 전선이라고 해도 과언이 아니다. 개발시 개발자들의 보안의식과 개발습관의 차이가 프로그램의 취약점과 바로 연결될 뿐만 아니라 보안취약점에 대한 시간 및 비용 및 프로그램의 품질에 까지 영향을 미친다. 이러한 프로그램의 품질을 높일 수 있는 요인은 코딩단계 뿐만이 아니라 단위 테스트, 기능 테스트 등의 단계가 있으며 향후 이러한 분야에 대한 추가적인 연구가 필요하다.

참고문헌

[1] Sngju Jang, Study On a Security Software Development Methodology Using Formal Verification Tool, THE RESEARCH INSTITUTE OF INDUSTRIAL TECHNOLOGY DEVELOPMENT, Vol.20, 2006
[2] Robert C Seacord, "The CERT C Secure Coding Standard", Addison-Wesley, Oct. 2008
[3] Jason A Rafail, "CERT Secure Coding Initiative", May. 2007
[4] Jason Lee, "Secure coding - the principles and practices", Oct, 2008.
[5] Steve McConnell, Code Complete se
[6] Frederic P, Miller, Agnes F. Vandome, and John McBrewster, Defensive Programming, lphascript publishing. 2009.
[7] sunil, Kim, A Study on characteristic analysis for software reliability estimation, 2008
[8] Kyungho Son, 소프트웨어 생명주기에서의 설계문서에 대한 보안성 체크리스트, KIISC REVIEW, Vol, 16 NO.4 2006
[9] URL http://skyul.tistory.com
[10] URL http://findbugs.sourceforge.net
[11] 국가사이버안전센터, 「2009년도 사이버 침해사고 사례집」, 2006.