공개 정적 분석도구를 활용한 소프트웨어 보안취약성 개선

장영수*, 정금택**, 최진영**
*고려대학교 컴퓨터정보통신대학원 소프트웨어공학과
**고려대학교 컴퓨터정보통신공학과
e-mail : jyskkh@chol.com, {jkt76,choi}@formal.korea.ac.kr

Software Security Vulnerability Improvement Using Open Static Analysis Tool

Young Su Jang*, Geum Taek Jung**, Jin Young Choi**
*Dept. of Software Engineering, Graduate School of Computer & Information Technology, Korea University
**Dept. of Computer Information & Communication, Korea University

요 약

인터넷의 발전으로 인터넷을 통한 서비스가 증대하고 있다. 반면 응용 소프트웨어의 보안 취약점으로 인해 국가, 기업, 개인 모두에게 정보보호의 중요성이 더욱 강조 되고 있다. 임베디드 소프트웨어인 우주, 항공, 원자력 소프트웨어 등 오류 없이 수행되어야 하는 고안전성 소프트웨어의 개발 기법은 이제 응용 소프트웨어의 보안강화 활동에 활용 되고 있다. 특히 시큐어 코딩 (Secure Coding)은 방어적 프로그램(Defensive Programming)을 포함하는 개념으로 소프트웨어의 안전성과 보안성을 향상 시킬 수 있다. 본 논문에서는 범용 보안 취약가능성 분석 도구를 이용하여 소프트웨어의 취약 가능성을 분석하고 보안 취약점 유발 명령어를 분류한다. 그 다음에 시큐어 코딩 기법을 적용하여 취약한 코드를 개선하였다. 이러한 개선을 통해 보안 취약성 가능한 코드 부분을 손쉽게 수정하여 소프트웨어 보안을 개선할 수 있다.

1. 서론

오늘날 컴퓨터 사용의 보편화, 일상생활과 연계된 인터넷 환경으로의 변화, 전자상거래의 확산 등으로 국가, 기업, 개인 모두에게 정보보호의 중요성은 매우 증대 되고 있다. 과거 오류 없이 수행되어야 하는 시스템은 국방, 항공, 우주 등 일부 정형화된 소프트웨어에 국한되어 있었다. 그러나 응용 소프트웨어의 활용 및 적용 범위가 넓어 지면서 소프트웨어의 안전성과 보안성은 일부 정형화된 소프트웨어의 범위를 넘어 응용 소프트웨어까지 보안취약성 강화를 요구하고 있다.

본 논문에서는 사례를 들어 소프트웨어 보안품질을 분석하였다. 또한 보안 취약성 분석을 위해 범용 정적 분석 도구를 이용 하였으며, 취약성 적용 단계에서 취약점이 발견된 명령어는 시큐어 코딩(Secure Coding)[1]을 적용하여 개선 하였다.

2. 방어적 프로그램

방어적 프로그래밍(Defensive Programming)[5]은 시큐어 코딩의 한 분야로 안전하고 강건한 프로그램을 작성하기 위하여 고안되었다. 방어적 프로그래밍 기법에는 여러 기법들이 있다. 본 논문에서는 응용 소프트웨어에 적용 가능한 방어적 프로그래밍 기법 중 C 언어를 사례로 CERT 시큐어 코딩 표준(Secure Coding Standard)[2]을 적용하였다. 또한 프로그램의 취약성 분석을 위해 flawfinder[3] 도구를 활용 하였다.

2-1. 소프트웨어 보안 취약성

프로그램 코드(Code)의 품질은 프로그램 보안·유지보수 등 다양한 분야에 영향을 미치며, 나아가서는 시스템 전반에 영향을 미치게 된다. 또한 프로그램 에러의 대부분은 예상하지 못한 많은 입력 값 에서 발생하며, 발생한 에러의 디버깅(Debugging)은 상대적으로 많은 인적, 물적 자원을 필요로 한다[2]. 그러므로 프로그램 수행 중 발생한 에러에 대해 발생 위치를 추측, 추적하여 오류를 검출 하고 제어할 수 없거나 예상하지 못했던 오류의 발생 시 그 피해를 최소화 하며, 수정하기 쉽고 제품 코드에 손상을 덜 입히도록 만드는 것이 필요하다[2,3].

2-2. 방어적 프로그래밍

방어적 프로그래밍의 한 기법은 타당하지 않은 입력 값으로부터 프로그램 보호를 최 우선으로 하여 프

로그램 수행 중 발생한 오류를 즉시 노출시키고, 검출된 에러의 수정을 용이하게 한다. 또한 일부 코드에서 오류가 발생 하더라도 다른 코드 및 전체 시스템이 피해를 입지 않도록 한다. 코드 작성 시 작성 스타일과 포맷을 정의한 코딩표준을 사용하도록 하며 프로그램 개발이나 컴파일러 설계 시 발생하는 취약성을 보완하여 코드를 보다 응집력 있고, 읽기 쉬우며, 버그의 가능성을 줄여준다[2]. 이러한 방어적 프로그래밍 기법으로 어셜션(Assertion)과 예외처리(Exception handling) 가 있다[3].

어셜선은 프로그램이 실행될 때 자체적으로 검사할 수 있도록 개발 도중에 사용되는 코드이며 일반적으로 함수나 매크로를 사용하여 프로그램 수행 중 예상치 못한 조건이나 외부에서 들어오는 모든 데이터, 루틴의 매개변수 등 타당하지 않은 입력 값 및 발생해서는 안 되는 조건을 찾아내기 위해서 사용된다.

예외처리는 코드가 오류나 예외적인 이벤트 루틴을 호출한 코드에 전달할 수 있는 특수한 방법이다. 만약에 어떤 루틴에 있는 코드가 어떻게 처리해야 하는지를 모르는 예외적인 상황이 발생할 경우나 무시 되어서는 안 되는 오류를 프로그램의 다른 부분에 알리기 위하여 예외처리를 사용한다. 이때 중요한 것은 예외처리 사용을 규격화해야 한다는 것이다.

<표 1> 소프트웨어 품질에 영향을 미치는 요소들

프로그램 요소	검증방법
입력 값 검증	- 정상 입력 값의 범위 집합을 정의 - 입력 값에 대해 범위집합과 대조하여 검증 - 입력 값이 올바르지 않을 경우 종료/재수행/경고 처리
사용변수의 범위 값 검증	- 운영체제 별 사용 변수의 범위 값 상이 - 하드웨어 플랫폼 별 사용 변수의 범위 값 상이
코딩표준	- 코드를 보다 응집력 있게 함 - 버그의 가능성을 줄여줌 - 도출에러에 대한 수정을 용이하게 함 - Variable name, indentation, position of brackets, function 선언 등
함수	- 입력->처리-> 결과 의 3 단 구조로 구성 - 함수 고유의 정의된 기능만 수행 - 프로그램 코드의 중복성 및 복잡성 감소 - 데이터 구조의 은폐 및 재사용 증진 - 코드의 가독성(Readability) 증가
코드의 재사용	- 안전성이 입증된 기존의 코드를 재사용 - 코드 재사용 기능(동작) 여부 확인 - 재사용 코드의 안내 설명서 존재 여부 확인

2-3. CERT 시큐어 코딩 표준

소프트웨어의 품질을 만족시키기 위한 능력과 관련된 소프트웨어 시큐어 코딩을 위한 표준은 미국의 CERT/CC(Computer Emergency Readiness Team Coordination Center) 가 주도하고 있다. 특히 미국 정부는 소프트웨어에 대한 CVE(정보보안취약점 표준 DB)를 제정하여, 소프트웨어의 품질 특성과 취약점 및 해결방안을 관리하고 있으며, 취약점 발견 시 이에 대한 해결방안을 제시·권고 하고 있다[2].

CERT/CC 에서 권고하는 시큐어 코딩 표준은 프로그래머 임의로 프로그램에 행하는 행동을 제한하고, 프로젝트 혹은 조직의 요구사항에 의해 결정된 지침서와 규칙(Rule)의 집합을 따르도록 하는 것이다[2]. 시큐어 코딩은 시큐어 코딩 표준을 함께 사용 함으로써 소프트웨어 취약성의 근본 원인이 되는 시큐어 코딩 에러를 항목별로 분류하고 오버플로, 포멧 문자열 취약성, 소프트웨어 취약점 등의 치명적인 코딩 에러를 제거할 수 있으며, 코드를 보다 응집력 있고, 읽기 쉬우며, 버그의 가능성을 줄일 수 있어 소프트웨어의 안전성, 신뢰성, 의존성, 견고성, 가용성, 유지보수성 등 품질 지수를 높일 수 있다[2].

<표 2> C 언어 시큐어 코딩 표준

표준항목	세부설명
Preprocessor	- macro function 대신 static 사용
선언 및 초기화	- define 대신 typedef 사용 - 상수 값은 const 를 붙여서 만약의 변형에 대비 하여 보호함
Expression	- 연산자 우선순위를 고려하여 '()' 사용 - sizeof 시 d_array 와 *d_array 의 차이점을 숙지 - 정수크기 할당 시 꼭 sizeof 사용하기 - 함수 호출 시 계산식 넣지 않기 - assert 안에는 할당, 증가, 감소, 함수 호출을 사용하지 않기
Memory	- 동일한 블록 수준에서 동일한 모듈로 메모리 할당 및 해제 하기 - free() 한 다음에는 Null 로 초기화 - pointer validation function 사용하기 - 해제한 메모리에 재 접근 하지 않기
Arrary	- 배열크기를 sizeof 만으로 측정하지 않기 - 배열 초기화 시 명시적으로 크기 표현 - 인자가 배열의 범위 내에 존재하는지 체크 - 배열 복사 시 충분한 공간이 있는지 확인 후 사용
Character & String	- buffer overflow 를 예방 하기 위해 다음 함수의 사용을 권장 · strcpy() 대신 strncpy() 사용 · strcat() 대신 strncat() 사용 · gets() 대신 fgets() 사용 · sprintf() 대신 snprintf() 사용 - 복사할 데이터가 Null 이거나 buffer 사이즈 보다 큰지 확인 - 변하지 않는 공간에 문자 저장하기

2-4. 취약성 분석 프로그램 및 분석 도구

본 논문에서는 A 사의 C 언어로 작성된 프로그램을 선정 하여 보안 취약 가능성을 분석 하였다. A 사의 프로그램 중 1) 사용자 사용빈도가 빈번하며, 2) 보안 취약성 노출 시 개인 정보가 노출될 수 있으며, 3) 프로그램 형상관리(Configuration)가 빈번한 온라인 프로그램 중 50 본을 선정하여 이를 범용 정적 분석 도구인 flawfinder 를 사용하여 분석 하였다.

Flawfinder 는 David A Wheeler 가 작성한 Open Source 프로그램으로 C/C++ 소스코드를 분석하여 위험(Risk) 레벨(Level) 별로 보안 취약성을 표시하는 정적 분석 도구 이다[4].

<표 3> Flawfinder 취약성 분석 Category

분석 Category	취약성 명령어
버퍼 오버플로 위험성	strcpy, strcat, sprintf 등
포멧 문자열 취약성	[v][f]printf, [v]sprintf 등
경쟁조건	access, chown, chgrp, chmod, tmpfile, tmpnam 등
잠재적 메타문자 위험성	system, exec, popen 등

분석된 취약성 레벨은 Level 0 (very little risk) ~ Level 5 (great risk)로 표시 된다.

<표 4> Flawfinder 취약성 레벨 분류

취약성 레벨	레벨 분류기준	관련 취약성 명령어
0	-외부 프로그램에서 함수로 입력되는 값	Recv, recvfrom, readv, recvmsg, fread
	○ 취약성 해결 방안 : 외부 입력 값 사전검증	
1	-저장 버퍼의 마지막에 종료문자('₩0')가 없음 -주의를 요하는 명령어	strncat, lstrcatn, strlen, getchar, fgetc, getc, umaks
	○ 취약성 해결 방안 : 저장 버퍼의 마지막 종료문자('₩0') 확인 및 주의를 요하는 명령어 사용시 사전 검증	
2	-입력 값이 저장 버퍼크기를 초과 (고정길이) -파일이 경쟁 조건을 유도	memcpy, char, vfork, fopen/open, atoi/atol, gsignal/ssignal
	○ 취약성 해결 방안 : 저장 버퍼의 크기 확인 및 경쟁조건 사전 검증	
3	-입력 값이 저장 버퍼크기를 초과(Null byte 포함)	strtrns, realpath, getpass, getwd,
	○ 취약성 해결 방안 : 저장 버퍼의 크기 확인 및 시큐어 코딩 명령어 사용	
4	-입력 값이 저장 버퍼크기를 초과 (가변길이) -경쟁 조건을 유도	strcpy, strcat, scanf, sprintf, mktemp, system, access
	○ 취약성 해결 방안 : 저장 버퍼의 크기 확인 및 시큐어 코딩 명령어 사용	
5	-입력 아규먼트 변경,오류시 경쟁 조건 유도	chown, chgrp, chmod, readlink, gets
	○ 취약성 해결 방안 : 입력 아규먼트(파일명) 및 시큐어 코딩 명령어 사용	

3. 응용 소프트웨어 보안 취약성 분석

범용 정적 취약성 분석 도구인 flawfinder 를 이용한 C 언어 온라인 프로그램 보안 취약성 분석은 보안 취약성 적용 전/후 단계로 구분된다. 보안 취약성 적용 전 단계는 프로그램 내 시큐어 코딩이 적용되지 않은 단계로 응용 프로그램 별로 flawfinder 정적 분석 결과의 위험 레벨과 코드 취약성 수를 집계 하며, 프로그램 내 시큐어 코딩 여부가 검증되지 않은 명령어 별 보안 취약성 권고에 따른 집계이다. 보안 취약성 적용 후 단계는 보안 취약성 적용 전 단계에서 집계된 flawfinder 위험 레벨과 코드에 시큐어 코딩 표준을 적용·검증 후 시큐어 코딩 표준이 미 적용된 명령어 별 보안 취약성을 재 집계 한다.

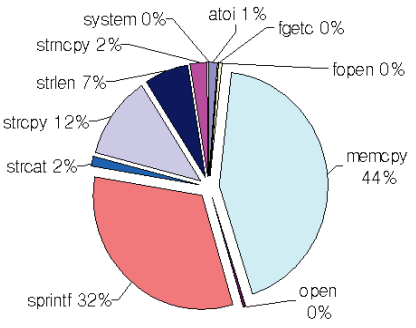
3-1. 보안 취약성 분석 적용 전 단계

온라인 프로그램의 flawfinder 취약성 위험 레벨과 코드 취약성 수를 집계 하며, 프로그램 내 시큐어 코딩 여부가 확인되지 않은 flawfinder 정적 분석 결과 이다.

<표 5> 온라인 프로그램 flawfinder 취약성 집계

취약성 / 프로그램	취약성 레벨 0	취약성 레벨 1	취약성 레벨 2	취약성 레벨 3	취약성 레벨 4	취약성 레벨 5
프로그램 1	[0] 0	[1] 0	[2] 29	[3] 0	[4] 7	[5] 0
프로그램 2	[0] 0	[1] 28	[2]225	[3] 0	[4]172	[5] 0
프로그램 3	[0] 0	[1]122	[2]998	[3] 0	[4]252	[5] 0
프로그램 4	[0] 0	[1] 52	[2] 77	[3] 0	[4] 61	[5] 0
프로그램 5	[0] 0	[1] 37	[2]169	[3] 0	[4] 58	[5] 0
프로그램 6	[0] 0	[1] 71	[2]276	[3] 0	[4]1068	[5] 0
.....						
프로그램 48	[0] 0	[1] 17	[2] 27	[3] 0	[4] 95	[5] 0
프로그램 49	[0] 0	[1] 19	[2]128	[3] 0	[4]158	[5] 0
프로그램 50	[0] 0	[1] 0	[2] 33	[3] 0	[4] 0	[5] 0
합계	**[0] 0**	**[1] 1,279**	**[2] 8,777**	**[3] 0**	**[4] 1,815**	**[5] 0**

분석 결과 취약성 레벨 0, 레벨 3, 레벨 5 는 분석 대상 프로그램 내에 해당 레벨 별 취약성 명령어 미 존재로 취약성 집계 건수가 0 이다.



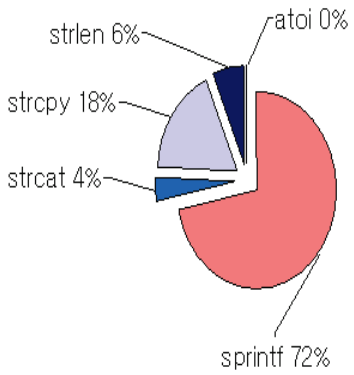
명령어	건수
atoi	141
fgetc	7
fopen	44
memcpy	4433
open	14
sprintf	3303
strcat	156
Strcpy	1219
Strlen	696
Strncpy	243
합계	10256

(그림 1) 보안 취약성 분석 전 취약성 명령어 집계

보안 취약성 분석 적용 전 단계에서 집계된 명령어는 시큐어 코딩 표준을 적용·검증 후 보안 취약성 분석 적용 후 단계에서 사용된다.

3-2. 보안 취약성 분석 적용 후 단계

보안 취약성 분석 적용 전 단계에서 집계된 명령어에 시큐어 코딩 표준을 적용하여 보안 취약점 유발 명령어를 재 집계 하였다. 특히 보안 취약성 분석 전 단계에서 취약성을 유발할 수 있는 명령어에 대해 버퍼 오버플로, 포맷 문자열 취약성, 소프트웨어 취약점 등을 조사 하였다.



명령어	건수
atoi	1
sprintf	956
strcat	75
strcpy	350
strlen	125
합계	1,507

(그림 2) 보안 취약성 분석 후 취약성 명령어 집계

3-3. 보안 취약성 개선

보안 취약성 분석 적용 후 단계에서 집계된 명령어는 잠재적 보안 취약 가능성을 야기 할 수 있으므로 시큐어 코딩 표준을 적용하여 보안 취약성을 유발하지 않도록 수정·보완 되어야 한다.

<표 6> 시큐어 코딩 표준 이용 보안 취약점 개선

취약성 명령어	보안 취약성 유형	시큐어 코딩 적용사항
atoi	입력 값이 정수표현 범위(4byte) 초과시	정수로 변환될 값의 MIN, MAX 값 사전검증
	입력 값이 정수표현 범위(4byte) 미 초과시	
sprintf	복사할 문자열 크기 > 저장 버퍼 크기	snprintf 시큐어 코딩 명령어 사용
	복사할 문자열 크기 <= 저장 버퍼 크기	.기존 sprintf 명령어 사용 가능 .snprintf 시큐어 코딩 명령어 사용
strcat	복사할 문자열 크기 > 저장 버퍼 크기	strncat 시큐어 코딩 명령어 사용
	복사할 문자열 크기 <= 저장 버퍼 크기	.기존 strcat 명령어 사용 가능 .strncat 시큐어 코딩 명령어 사용
strcpy	복사할 문자열 크기 > 저장 버퍼 크기	strncpy 시큐어 코딩 명령어 사용
	복사할 문자열 크기 <= 저장 버퍼 크기	.기존 strcpy 명령어 사용 가능 .strncpy 시큐어 코딩 명령어 사용
strlen	사용 배열의 마지막에 '₩0' 값 입력 여부	사용 배열의 마지막에 '₩0' 값 입력 및 검증

4. 결론

본 논문에서는 소프트웨어 보안 취약성 감소를 위해 시큐어 코딩 과 공개 소프트웨어도구를 활용하는 방안을 제시 하였다. 넓은 의미의 시큐어 코딩은 개발자와 개발 조직간 유기적인 품질활동을 할 수 있는 구성 과 방법론이며, 이에 대해서는 추가적인 연구가 필요하다.

참고문헌

[1] Robert C Seacord,"The CERT C Secure Coding Standard", Addison-Wesley, Oct. 2008.
[2] Jason A Rafail,"CERT ecure Coding Initiative", May. 2007.
[3] Jason Lee,"Secure coding - the principles and practices", Oct. 2008.
[4] URL http://www.dwheeler.com/flawfinder
[5] Frederic P. Miller, Agnes F. Vandome, and John McBrewster,"Defensive Programming", Lphascript Publishing, 2009