소프트웨어 보안품질 향상을 위한 시큐어 코딩표준 활용

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

Software Security Quality Improvement Using Secure Coding Standard

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

요 약

과거 인터넷을 사용하지 않는 시스템의 경우 소프트웨어의 안전성과 강건성은 철도, 국방, 우주, 항공, 원자력 등 오류 없이 수행되어야 하는 임베디드 소프트웨어에 국한되어 있었다. 그러나 인터넷의 발전으로 인터넷을 통한 정보의 교류 및 서비스가 증대하면서 소프트웨어의 보안품질은 개인, 사회, 국가 모두에게 정보보호의 중요성을 더욱 강조하고 있다. 특히 오류 없이 수행되어야 하는 고안전성 소프트웨어의 개발 기법은 이제 응용 소프트웨어의 보안강화 활동에 활용 되고 있다. 시큐어 코딩 (Secure Coding)은 방어적 프로그램(Defensive Programming)을 포함하는 개념으로 소프트웨어의 안전성과 보안성을 향상 시킬 수 있다. 본 논문에서는 C 언어의 취약가능성 유발 명령어를 예를 들고 시큐어 코딩 기법을 적용하여 취약한 코드를 개선하였다. 이러한 개선을 통해 보안 취약성 유발 가능한 코드 부분을 손쉽게 수정하여 소프트웨어 보안품질을 개선할 수 있다.

1. 서론

인터넷을 이용하는 시스템의 경우 기존 백신과 방화벽으로 일관하던 보안정책은 네트워크 복잡성 증가, 다양한 외부공격 등으로 새로운 보안 대책과 기술의 도입이 필요하게 되었다. 그리고 오늘날 인터넷을 활용한 컴퓨터 사용의 보편화, 일상생활과 연계된 인터넷 유비쿼터스 환경으로의 변화, 전자상거래의 확산 등으로 정보보호의 중요성은 매우 증대 되고 있으며, 응용 소프트웨어는 활용 및 적용 범위가 넓어 지면서 소프트웨어의 안전성과 보안성 강화를 요구하고 있다.

본 논문에서는 사례를 들어 소프트웨어 보안품질을 분석하였다. 보안취약성 유발 명령어를 예를 들고 취약한 코드의 개선을 위한 시큐어 코딩(Secure Coding)[1]을 적용하여 소프트웨어 보안품질을 개선하였다.

2. 방어적 프로그램

방어적 프로그래밍(Defensive Programming)[2]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.

2-1. CWE (Common Weakness Enumeration)

2010 년 4 월에 발표된 “2010 CWE/SANS Top 25 Most Dangerous Programming Errors”[3] 에는 프로그램 보안 취약성을 유발할 수 있는 주요 25 가지 취약성이 있으며, 이중 “Cross-site Scripting”, “SQL Injection”, “Classic Buffer Overflow” 는 보안 취약성을 유발할 수 있는 위험한 취약성 오류의 대표적인 예들이다. 이중 “Classic Buffer Overflow” 는 고전적인 보안 취약성 유발 유형이며, 여전히 취약성 유발 위험 요소로 분류되고 있다.

2-2. CERT 시큐어 코딩 표준

소프트웨어의 보안품질 향상 및 만족을 위한 활동인 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 데이터베이스)를 제정하여, 소프트웨어의 품질 특성과 취약점 및 해결방안을 체계적으로 관리하고 있으며, 취약점 발견 시 이에 대한 해결방안을 제시·권고 하고 있다[2].

<표 1> C 언어 시큐어 코딩 표준

표준항목	세부설명
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

3. 응용 소프트웨어 보안 취약성 개선

보안 취약성 유발 명령어 중 버퍼 오버플로(Buffer overflow)는 고전적인 취약성 유발 유형으로 저장 공간보다 저장할 자료가 클 경우 발생 하므로 시큐어 코딩표준을 사용하여 취약성을 유발하지 않도록 수정, 보안 되어야 한다.

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 있는 프로그램 중 50 본을 선정하여 버퍼 오버플로(Buffer overflow)를 유발할 수 있는 명령어를 선정 후 분석 하였다.

3-1. sprintf 명령어

snprintf 또는 vsnprintf 사용(복사할 문자열의 크기가 버퍼의 크기보다 작을 경우 sprintf 사용가능)

취약성 확인사항		시큐어 코딩 적용사항
복사할 문자열의 크기가 버퍼의 크기보다 클 경우	시큐어 코딩 적용 전	#define S_BUFF_SIZE 20 #define L_BUFF_SIZE 30 char useno[L_BUFF_SIZE]; char tmp_log[S_BUFF_SIZE]; ... sprintf (tmp_log, "%s", useno); ...
	시큐어 코딩 적용 후	#define S_BUFF_SIZE 20 #define L_BUFF_SIZE 30 char useno[L_BUFF_SIZE]; char tmp_log[S_BUFF_SIZE]; ... *1. snprintf (tmp_log, sizeof(tmp_log),"%s",useno);* ...
복사할 문자열의 크기가 버퍼의 크기보다 작거나 같은 경우	시큐어 코딩 적용 전	#define S_BUFF_SIZE 20 char useno[S_BUFF_SIZE]; char tmp_log[S_BUFF_SIZE]; ... sprintf (tmp_log, "%s", useno); ...
	시큐어 코딩 적용 후	#define S_BUFF_SIZE 20 char useno[S_BUFF_SIZE]; char tmp_log[S_BUFF_SIZE]; ... *1. sprintf (tmp_log, "%s", useno);* *또는* *2. snprintf (tmp_log, sizeof(tmp_log),"%s",useno);* ...

3-2. strcat 명령어

strncat 또는 strlcat 사용 (복사할 문자열의 크기가 버퍼의 크기보다 작을 경우 strcat 사용가능)

취약성 확인사항		시큐어 코딩 적용사항
복사할 문자열의 크기가 버퍼의 크기보다 클 경우	시큐어 코딩 적용 전	#define S_BUFF_SIZE 20 #define L_BUFF_SIZE 30 char useno[L_BUFF_SIZE]; char tmp_log[S_BUFF_SIZE]; char log_buf[S_BUFF_SIZE]; ... int i,j; strcat (tmp_log, useno); ...

취약성 확인사항		시큐어 코딩 적용사항
복사할 문자열의 크기가 버퍼의 크기보다 클 경우	시큐어 코딩 적용 후	#define S_BUFF_SIZE 20 #define L_BUFF_SIZE 30 char useno[L_BUFF_SIZE]; char tmp_log[S_BUFF_SIZE]; ... *1. strncat(tmp_log, useno, S_BUFF_SIZE-1);* *2.tmp_log[S_BUFF_SIZE-1]='₩0';*
복사할 문자열의 크기가 버퍼의 크기보다 작거나 같은 경우	시큐어 코딩 적용 전	#define S_BUFF_SIZE 20 char useno[S_BUFF_SIZE]; char tmp_log[S_BUFF_SIZE]; ... strcat (tmp_log, useno);
	시큐어 코딩 적용 후	#define S_BUFF_SIZE 20 char useno[S_BUFF_SIZE]; char tmp_log[S_BUFF_SIZE]; ... *1. strcat (tmp_log, useno);* *또는* *2. strncat(tmp_log, useno, S_BUFF_SIZE-1);* *3.tmp_log[S_BUFF_SIZE-1]='₩0';*

3-3. strcpy 명령어 개선

strncpy 또는 strlcpy 사용(복사할 문자열의 크기가 버퍼의 크기보다 작거나 같은 경우 strcpy 가능)

취약성 확인사항		시큐어 코딩 적용사항
복사할 문자열의 크기가 버퍼의 크기보다 클 경우	시큐어 코딩 적용 전	#define S_BUFF_SIZE 20 #define L_BUFF_SIZE 30 char useno[L_BUFF_SIZE]; char tmp_log[S_BUFF_SIZE]; ... strcpy (tmp_log, useno);
	시큐어 코딩 적용 후	#define S_BUFF_SIZE 20 #define L_BUFF_SIZE 30 char useno[L_BUFF_SIZE]; char tmp_log[S_BUFF_SIZE]; *1. strncpy(tmp_log, useno, S_BUFF_SIZE-1);* *2. tmp_log[S_BUFF_SIZE-1] = '₩0';*
복사할 문자열이 버퍼의 크기보다 작거나 같은 경우	시큐어 코딩 적용 전	#define S_BUFF_SIZE 20 char useno[S_BUFF_SIZE]; char tmp_log[S_BUFF_SIZE]; ... strcpy (tmp_log, useno);
	시큐어 코딩 적용 후	#define S_BUFF_SIZE 20 char useno[S_BUFF_SIZE]; char tmp_log[S_BUFF_SIZE]; ... *1. strcpy (tmp_log, useno);* *또는* *2. strncpy(tmp_log, useno, S_BUFF_SIZE-1);* *3. tmp_log[S_BUFF_SIZE-1] = '₩0';*

3-4. strlen 명령어 개선

사용 배열의 맨 뒤에 '\0' 입력

취약성 확인사항	시큐어 코딩 적용사항
시큐어 코딩 적용 전	#define BUFF_SIZE 5 char nptyk[BUFF_SIZE]; ... If (strlen(nptyk) < 5)
시큐어 코딩 적용 후	#define BUFF_SIZE 5 char nptyk[BUFF_SIZE]; ... *1. strncat(nptyk,"DOOD", BUFF_SIZE-1);* *2. nptyk[BUFF_SIZE-1] = '₩0';* *또는* *3. strncpy(nptyk, "DOOD", BUFF_SIZE-1);* *4. nptyk[BUFF_SIZE-1] = '₩0';* if (strlen(napyk) < 5)

4. 결론

본 논문에서는 소프트웨어 보안 품질 향상을 위해 시큐어 코딩표준 활용 방안을 제시 하였다. 특히 소프트웨어 보안 취약성 유발 유형 중 고전적인 버퍼 오버플로(Buffer overflow)를 사례로 시큐어 코딩 적용 전 과 적용 후를 비교함으로써 보안 취약성을 개선 하고자 하였다.

참고문헌

[1] Robert C Seacord,"The CERT C Secure Coding Standard", Addison-Wesley, Oct. 2008.

[2] Frederic P. Miller, Agnes F. Vandome, and John McBrewster,"Defensive Programming", Lphascript Publishing, 2009

[3] http://cwe.mitre.org/top25/, "2010 CWE/SANS Top 25 Most Dangerout Programming Errors"

[4] Jason A Rafail,"CERT ecure Coding Initiative", May. 2007.

[5] Jason Lee,"Secure coding - the principles and practices", Oct. 2008.