6

안전한 C 프로그램을 위한 코딩 표준

홍익대학교 ▌표 창 우*
한국산업기술대학교 ▌한 경 숙**

1. 서 론

보안 정책과 도구들은 네트워크, 시스템, 사용자, 프로그램 4 계층으로 구분해 볼 수 있다[1]. 앞의 3개 분야는 접근 권한과 제한, 정보 보호에 대하여 고려하는 반면, 프로그램 보안은 공격자들이 악용할 수 있는 취약성을 제거하는데 초점을 맞추고 있다. 프로그램 보안의 시작은 안전한 코딩(secure coding)으로서, 구문적으로 이상 없는 프로그램이라도 해킹에 악용될 소지가 있는 프로그래밍 방식은 사전에 배제하거나 작성 전후로 분석하여 제거하는 방식이다. 구문적으로는 이상이 없더라도, 악용될 소지가 있는 코드는 "프로그래밍 오류"라고 분류한다.

단순한 형태의 오류라도 보안에 치명적일 수 있다. 1988년 모리스 웜 사건[2]은 프로그램의 취약성을 이용하기는 했지만, 복제에 의한 빠른 확산은 의도하지 않았다. 그러나 실제 벌어진 일은 의도와는 무관하게 요즘의 DDOS 공격이 되어 버리고 말았다. 2003년 8월 인터넷을 마비 시켰던 블래스터 웜 사건[3]은 프로그램의 오류를 파고든 공격으로부터 시작되었다. 피해액수는 5억 2,500만 달러 규모로 추산되며, 시간당 100,000대의 컴퓨터가 공격당하여 3일 동안 1백만 대의 컴퓨터가 피해를 입었다. 지정문 하나로 구성된 while 문장이 갖고 있는 버퍼 오버플로우[4] 취약성을 악용한 공격이었다.

네트워크 수준의 보안 강화 노력에도 불구하고, 컴퓨터 보안 사고는 수그러들지 않는다. 보다 근본적인 문제점은 사례가 알려지지 않은 제로데이 공격에는 속수무책인 점이며, 사후 대응으로 추가 피해를 막는 것 밖에 할 수 없다. 안전한 코딩은 악용당할 수 있는 취약성이 없는 충분히 건강한 프로그램을 만들자는 예방적인 접근이며, 시스템 아키텍처의 하부 소프트웨어 층을 강화시키려는 노력이다. 행정안전부의 전자정부 보안 강화체계 구축사업[5]에서는 안전한 코딩을 보안 강화의 핵심으로 다루고 있다.

안전한 코딩에 대한 접근 방법은 2가지로 압축할 수 있다. 하나는 코딩 표준을 강화시켜 문법이 요구하는 이상으로 코딩 방식을 규제하는 것이다. 다른 하나는 프로그램 분석 도구를 사용하여 프로그램이 갖고 있는 약점을 찾아내어 이를 제거하는 것이다. 상업용인 Fortify[6], 공공에게 무상 제공되는 Compass/ROSE[7]와 같은 우수한 분석 도구들이 발표되고 있다. 이 두 가지 접근 방법은 상호 보완적이어서 함께 적용되어야한다. 코딩 표준을 적용하여 오탐(false alarm)의 원인이 되는 코딩 방식을 배제시키면, 자동 분석 도구의 결과의 정확도를 높이는데 도움이 된다.

이 글은 안전한 C 프로그램 작성을 위한 코딩 표준에 대하여 살펴본다. 2절에서는 취약성과 연결되는 C 언어 특성과 CERT의 C에 대한 안전한 코딩 표준을 비롯하여 몇 가지 코딩 표준에 대하여 소개한다. 3절과 4절은 SANS와 MITRE가 중심이 되어 발굴한 상위 25 프로그래밍 오류 관점에서 CERT의 코딩 표준과 분석 도구들의 효능을 연결하여 살펴본다.

2. C 언어를 위한 코딩 표준

2.1 C의 언어적 취약성

C 언어는 1969년에 벨 연구소에서 BCPL이라는 무타입 언어로부터 파생되어 나왔다. 1970년대에 UNIX 운영체제 구현 언어로 사용되기 시작하여 현재 광범위한 시스템 프로그래밍과 응용 프로그램 개발에 사용되고 있다. C89가 1989년 ANSI 표준으로 정해졌고, 그 후 C90이 ISO/IEC 국제 표준으로 정해졌고, 현재는 C99가 표준이다.

C 언어의 약한 타입 시스템과 통제되지 않는 타입 변환, 포인터의 무제한적인 사용, 배열 타입의 느슨

* 종신회원
** 정회원
† 행정안전부 "전자정부 보안 강화체계 구축사업"의 지원으로 연구되었음.

한 처리가 C 프로그램 취약성의 원인이 되고 있다. 예를 들어 정수형에는 여러 가지 타입이 있으며, 자동적으로 타입 변환이 수행되는데, 값의 범위가 타입마다 다르기 때문에 예상치 못한 오류가 발생할 수 있다. 포인터는 참조하는 데이터 타입의 호환성과 무관하게 원하는 타입의 데이터에 대한 포인터 타입으로 변환이 가능하다. 또한 정수형으로도 변환이 되며, 산술 연산에도 쓰이다 다시 포인터로 변환될 수도 있다. 배열은 온전한 타입으로서 취급되지 않고, 주소 표시의 한 방편으로 사용된다. 배열 이름은 포인터 상수로, 인덱스 수식은 시작 주소로 부터의 거리를 나타내기만 한다. 접근 원소가 배열 원소 수 범위 내에 있는지, 피호출 함수에 전달해야 하는 배열이 1차원 배열인지 다차원 배열인지 검사하는 일은 없다. 즉 시작 주소와 원소 타입만 일치하면 매개 변수로서 전달하는데 아무 문제가 없다.

유연한 타입 시스템과 포인터 사용이 통제되지 않고 사용되기 시작하면 여러 가지 문제점이 발생하게 된다. 많은 종류의 공격이 포인터 변수에 원하는 값을 기록하는 일로 시작된다. 포인터 값을 변질시켜 놓으면 그 포인터를 사용하여 간접 분기(indirect branch)할 때에 공격자가 원하는 주소로 이동할 수 있게 된다. 이런 공격 유형의 대표적인 예가 버퍼 오버플로우 공격이다. 버퍼 오버플로우 취약성은 잘 알려진 문제이나 아직도 많은 프로그램에서 계속해서 발견되는 취약성이다. CERT의 웹 사이트를 방문하여 최근의 취약성 리스트를 살펴보면, 버퍼 오버플로우 취약성이 쉽게 발견된다.

C 언어가 만들어졌을 때의 상황은 보안 문제가 지금처럼 심각하지 않았고, 모든 프로그램이 실행되는 환경이나 사용자가 우호적이라는 가정이 함축되어 있다. 그러나 더 이상 이런 가정은 옳지 않다. 무시하고 넘어 가는 오류도 공격자들에게 노출되면 침투 방편으로 악용된다. 프로그램의 보안과는 관계가 없어 보이는 오류도 예외는 아니다. 프로그램 코딩 실수는 프로그램을 취약하게 만들며 결과적으로 공격 루트를 제공한다. 보안의 기초는 취약성 없는 프로그램에서 시작된다.

2.2 CERT C 코딩 표준

안전한 코딩은 개발이 완료되기 전에 코딩 오류를 줄임으로써 결과물의 취약점을 감소시키고 결과적으로 개발 후 취약점 제거에 드는 비용을 감소시킨다. 안전한 코딩을 위해서는 소프트웨어 취약성을 만들어내는 공통적인 프로그래밍 오류를 찾아내고, 이를 기반으로 안전한 코딩을 위한 표준을 만든 후, 개발자를 교육시켜 안전한 코딩을 하도록 해야 한다.

CERT C 코딩 표준은 수많은 취약성을 분석하여 공통의 프로그래밍 오류를 찾아내고 안전하지 않은 코딩에 대응되는 안전한 코딩 방법을 구축하였다. 이 표준은 89개의 규칙(rule)과 132개의 권고(recommendation)로 구성되어 있다[8](웹에 발표되어 있는 표준은 출판된 형태[8]와 차이가 있을 수 있는데, 계속 진화되고 있기 때문이다). 규칙은 반드시 준수해야 하는 것으로, 준수 여부를 정적 분석, 정형 기법, 또는 수작업으로라도 검증할 수 있으며, 위반하는 경우 안전을 위협하는 취약성을 가지게 된다. 권고는 준수하는 경우 시스템의 안전성을 개선할 수 있는 지침이나 제안이라 볼 수 있다.

CERT C 코딩 표준은 전처리기와 선언, 초기화, 수식, 정수형과 실수형, 배열, 문자열, 메모리 관리, 입출력, 환경, 시그널, 오류 처리, API, POSIX와 기타 15가지로 분류한 규칙과 권고를 포함한다. CERT C 코딩 표준에서는 각 분류별로 00~29번은 권고를, 30번 이후는 규칙을 나타낸다. 각 표준은 오류의 심각성과 발생 가능성, 수정 비용의 세 가지 요소를 세 단계로 분류하여 1~27의 우선순위를 정하고, 이를 기반으로 L1~L3로 중요도를 표시해준다.

규칙의 예를 들면 다음과 같다.

- STR31-C: 문자열을 위한 공간은 문자 데이터와 널 문자를 포함할 수 있는 충분한 크기를 확보할 것. 심각성: 높음(3), 발생가능성: 높음(3), 수정비용: 중간(2), 우선순위: P18, 수준: L1

그림 1의 취약한 코드 예에서 환경변수를 저장할 문자 배열 buff를 충분하다고 생각되는 256으로 선언해서 사용했으나, getenv("EDITOR")함수 결과 값의 길이를 예측할 수 없으므로 buff의 크기가 충분한지 알 수 없다. 만일 환경변수를 나타내는 문자열이 255 이

취약한 코드 예

```
/* ... */
char buff[256];
char *editor = getenv("EDITOR");
if (editor == NULL) {
  /* EDITOR environment
     variable not set */
} else {
  strcpy(buff, editor);
}
```

안전한 코드 예

```
char *buff;
char *editor = getenv("EDITOR");
if (editor == NULL) {
  /* EDITOR environment
     variable not set */
} else {
  size_t len = strlen(editor)+1;
  buff = (char *)malloc(len);
  if (buff == NULL) {
    /* Handle malloc() Error */
  }
  memcpy(buff, editor, len);
}
```

그림 1 CERT C 규칙의 예 (STR31-C)

상의 크기를 가지게 되면 strcpy() 함수를 수행하는 과정에서 버퍼 오버플로우 오류가 발생할 수 있다. 안전한 코드에서는 이런 문제를 해결하기 위하여 함수의 결과 값에 대한 길이를 계산하여 필요한 메모리를 할당 받아 사용하는 것을 볼 수 있다. 또한 memcpy() 함수를 사용함으로써 복사하는 데이터의 크기를 제한한다.

2.3 전자정부 시스템을 위한 코딩 규칙

행정안전부에서는 정보시스템(e-Government) 보안강화체계 구축 사업[5]을 통해 소스코드 취약점 데이터베이스를 구축하고 이를 기반으로 소스코드 자동진단도구를 개발하며, 국내외 개발모델을 분석하여 정보화사업용 보안개발 모델을 구축해 나가고 있다. 사용 빈도가 높은 자바와 C 언어에 대하여 우선적으로 취약점 데이터베이스 구축과 자동진단도구 개발을 진행하였다. 취약점 데이터베이스 구축을 위해 언어별 소스코드의 취약점을 조사하고 분류하였으며, 이에 대응하기 위한 방법 중 하나로 안전한 코딩에 대한 규칙과 패턴을 정의하였다. 자동진단도구 구축을 위해서는 이러한 데이터를 기반으로 하여 언어별로 소스코드를 검사하는 엔진을 개발하였다.

C 언어를 위한 안전한 코딩 규칙은 언어 독립적인 부분과 의존적인 부분으로 구분하여 정의하였으며, 언어 독립적인 규칙은 명칭(identification)이나 형식에 대한 규칙, 주석의 구성과 변수 선언, 상수와 수식, 문장, 자료형, 라이브러리, 환경에 대한 규칙 42개를 포함한다. 언어 의존적인 규칙은 이러한 분류 외에 함수 선언이나 초기화, 동적 메모리 관리, 포인터와 배열, 구조체와 공용체, 문자열과 파일 입출력, 오류 처리에 대한 규칙으로 85개의 규칙을 포함하고 있다.

C 언어를 위한 안전한 코딩 규칙은 프로그램의 가독성을 높여 프로그램에 대한 이해를 돕기 위한 규칙과 C 언어가 가지고 있는 자료형이나 포인터, 배열의 경계검사 부재 등에 의한 취약성에 대응하기 위한 규칙들을 포함한다. 언어 독립적인 규칙은 주로 프로그램의 이식성이나 가독성에 대한 규칙을 포함하고 있으며, C 언어의 취약성과 관련된 규칙은 언어 의존적인 규칙에 포함되어 있다. 개발자들은 프로그램을 구현하는 과정에서 이러한 규칙을 준수하여야 하며, 자동 안전 분석/검사 도구는 안전한 코딩 규칙을 준수하는지 여부를 검사하게 된다.

2.4 MISRA-C 표준

MISRA-C[9]는 자동차 산업과 관련된 소프트웨어 개발 과정에 사용하기 위한 것으로, 사용하는 언어의 구성요소나 컴파일러에 제한을 둠으로써 C 언어의 부분집합을 사용하는 새로운 언어로 볼 수 있다. 본래의 목적은 자동차 업계 뿐 아니라 C 언어를 사용하는 모든 개발자에게 가능한 한 안전한 소프트웨어를 만들어 내게 하는데 있다. C 언어의 정의의 닮은꼴로 21개의 영역에 걸쳐 121개의 필수 규칙(required rules)과 20개의 권장 규칙(advisory rules)으로 구성되어 있다. 4절에서 소개되는 정적 도구들 대부분이 MISRA-C 표준 검사에 대하여 준비되어 있다.

2.5 국내 산업계 현황

국내에서도 일부 산업체에서 안전한 코딩을 위한 표준을 사용하거나 적용 예정이며, 소프트웨어 보안 도구를 활용하여 소프트웨어 품질을 높이려는 시도가 계속되고 있다. L 사의 경우 협력회사에 코딩 표준을 적용하고 있는데, C 코드에 대해 11개 그룹의 33개 규칙을 준수하도록 하고 있으며 C++에 대해 12개 그룹의 44개 규칙을 지정하고 있다. 또한 자동화 도구를 활용하여 주기적으로 검증하고 결함을 측정, 분석하며 추적관리를 함으로써 약 22%의 결함을 조기에 수정하여 소프트웨어 품질 향상에 기여하고 있다[10]. 금융업계나 포털 관련 업계 등 소프트웨어 보안이 중요하게 대두되고 있는 업종에서도 보안팀과 같은 기구를 두고 도구를 이용하여 소스코드의 취약 부분을 검사하거나 코딩 가이드라인을 사용하고 있는 업체들이 늘어나고 있는 추세이다.

3. 가장 위험한 25개의 프로그래밍 오류

MITRE가 주관하여 구축한 CWE(Common Weakness Enumeration)는 소프트웨어의 구성, 설계 단계와 관련된 소프트웨어 취약성의 관리에 대한 이해를 높일 뿐 아니라 소스코드나 운영체제에서의 소프트웨어 취약성을 찾아내는 소프트웨어 보안 도구나 서비스에서 사용하고 논의하거나 기술하는데 유용한 소프트웨어 취약성을 모아놓은 것이다[11]. CWE는 CVE 리스트의 다양한 실제 취약성을 기반으로 CWE 커뮤니티의 많은 소스와 예제를 통해 취약성 리스트를 만들어 나간다.

전체 799개의 CWE 항목 중에는 659개의 취약성이 포함되어 있고 나머지 항목은 분류에 의해 단계적으로 묶어놓은 항목들이다. CWE는 다양한 분야의 소프트웨어 보안 문제를 트리 구조로 구성하며, 가장 상위 분류로는 개발자, 연구자, 사업가 등의 전략적 분류로 나뉜다. 중간 단계의 분류는 소프트웨어 보안이

나 개발사에 유용한 분류로, 하위 분류는 도구 개발 업체나 상세한 연구에 사용될 수 있는 전체 리스트로 구분되어 있다.

CWE 리스트는 코드, 설계, 구성에 관련된 소프트웨어 결함을 명백하게 표현하고 논의할 수 있도록 공통의 언어를 제공하며, 보안 도구 개발 업체나 서비스 업체에는 보안의 취약성을 명확하게 나타낼 수 있게 해 준다. 또한 소프트웨어 보안 문제에 대해 도구나 서비스가 얼마나 많은 부분에 대해 대처할 수 있는지 비교, 평가할 수 있으며 소프트웨어 보안의 수준을 명시할 수 있도록 해 준다.

2009년 1월 MITRE와 SANS가 주도하여 30개 이상의 미국과 유럽의 기업, 연구소, 정부 부처가 공동으로 "상위 25개의 가장 위험한 프로그래밍 오류(Top 25 Most Dangerous Programming Errors)"를 발표했다. 이 리스트는 컴포넌트 사이의 안전하지 못한 상호작용, 위험성이 높은 자원 관리, 허점 많은 방어 3부분으로 구성되어 있다. 각 부분은 CWE 개별 항목으로 구성되어 있다. CERT가 상위 25 오류를 구성하는 CWE 항목에 대하여 대응하는 CERT C 코딩 표준을 각 CWE 항목 별로 해당하는 CERT 코딩 표준을 연결해 놓았는데, 그 내용은 부록에 있다. CWE 항목과 CERT 표준 항목 사상 관계를 살펴보면, 40개의 CERT 표준 규칙이나 권고 항목이 사용되고 있다. 이중 17개가 규칙에 해당한다. CERT C 코딩 표준의 항목 수와 비교할 때 많은 차이가 있어 코딩 표준 항목이 불필요한 것이 있다고 생각될 수도 있고, CERT의 코딩 표준은 상위 25 오류를 막는데 충분하다고 판단할 수도 있지만, CERT와 상위 25 오류의 분류 관점이 달라 사상 관계를 부여하기 힘들기 때문이기도 하다.

앞서 살펴본 C 언어의 특성에서 비롯되는 버퍼 오버플로우 취약성 CWE-119에 대응하는 코딩 표준이 제일 많이 발견된다. 메모리 할당과 관계된 것, 데이터 타입과 관계된 것이 그 다음으로 많이 발견된다.

4. 도 구

안전한 코딩을 위한 도구들은 원시 프로그램 분석을 위한 것과 코딩 표준 준수 여부를 검사하는 도구로 나누어 생각할 수 있는데, 사용에 있어 이 두 구분의 경계는 명확하지 않다. 분석 도구가 검사해 내는 많은 취약성은 이에 대응하는 코딩 표준에 대한 검사 결과로 볼 수 있고, 반대로 코딩 표준 준수 여부를 확인하는 것이 취약성 탐지라고 볼 수도 있다. 다음 표 1은 몇 가지 도구들의 CERT C 코딩 표준에 대한 검사 능력을 평가한 결과를 요약한 것이다. CERT 사이트에서는 CERT C 코딩 표준에 대하여 자동으로 오류를 검출할 수 있는 도구에 대해 기술하고 있으며, 부분적으로 검출하는 경우 어떤 오류를 검출하는지에 대한 정보를 제공한다. 이 정보를 사용하여 Top 25 오류에 대응하는 CERT C 표준에 대한 오류 검출 능력을 정리하였다.

상위 25 오류 분류에 따라 CERT C 표준을 나열하였고, 이에 대해 5 개 규칙 검사 도구의 오류 검출 기능 유/무를 표시하였다. △는 부분적으로 자동 검출되는 것을 의미하는 것으로, 일부 함수나 특정 상황에 대해서만 오류 검출이 가능하다.

Fortify[6]는 소스코드 정적 분석과 테스트 단계의 동적 분석을 통해 취약점 분석 및 차단 기능을 수행하는 도구로, 자바를 비롯해 C 계열의 언어, ASP, JSP, VB.NET 등 17개의 언어를 대상으로 하고 윈도우즈 시스템을 비롯하여 유닉스 계열, 리눅스, MAC OS 등의 운영체제 상에서 동작한다. Compass/ROSE[7]는 ROSE 컴파일러 기반의 소스코드 검사 도구로, 제어흐름 분석이나 데이터 흐름 분석, 의존도 분석 등을 통하여 패턴 기반의 검사를 수행하며 C, C++, 포트란 등의 언어를 대상으로 하고, 리눅스와 MAC OS 환경에서 사용할 수 있다. Klocwork[12]은 보안 취약성과 위험 질적 분석, 복구 및 측정을 위한 도구로 C, C++, 자바 언어를 대상으로 하며 윈도우즈와 유닉스, 리눅스 환경을 대상으로 한다. LDRA[13]는 Testbed에서 채택한 표준에 의거하여 소스코드를 검사하며, MISRA-C 2004 기반으로 개발된 검사도구이다. 자바를 비롯하여 C 계열, 포트란, Algol, PL/I 등 대부분의 고급언어와 어셈블리어를 대상으로 하며, 윈도우즈 시스템을 비롯하여 유닉스 계열, 리눅스 등의 환경에서 동작한다. Splint[14]는 C 언어를 대상으로 동적 메모리 할당 오류나 버퍼 오버플로우와 같은 오류를 탐지하는 정적 분석기로, 유닉스 환경에서 동작한다.

상위 25 오류의 분류 기준으로 볼 때, 구성원 사이의 상호작용에 해당하는 오류에 대해서는 Fortify와 Klocwork, LDRA가 유사한 정도로 적용됨을 볼 수 있다. Fortify와 Klocwork은 OS 명령 관련 오류에 대해서, LDRA는 적합하지 않은 인코딩이나 출력에 대한 오류에 대해 주로 적용된다. 이 분류에서 Splint는 오류 검출에 대해 언급되지 않은 것을 볼 수 있다.

위험한 자원관리 오류와 허점 많은 방어 오류에 대해서는 Compass/ROSE가 많은 부분을 검출해 내는 것을 볼 수 있다. Compass/ROSE는 특정한 함수를 사용

표 1 분석, 표준 검사 도구의 Top 25 오류 검출 능력

Top25 분류	CERT 표준	Fortify V 5.0	Compass/ ROSE	Klocwork V. 8.0.4.16	LDRA V. 7.6.0	Splint V 3.1.1
구성원 사이의 안전하지 않은 상호작용	INT06-C	O	△			
	ERR07-C					
	APP00-C				O	
	MSC09-C				O	
	MSC10-C				O	
	MEM10-C					
	ENV03-C		O	O		
	ENV04-C	O		O		
	STR02-C	O		O		
	FIO31-C					
위험한 자원 관리	ARR00-C					
	ARR33-C	O	O	O		O
	ARR34-C		△			
	ARR35-C		△	O		
	ENV01-C		△			
	FIO37-C	O	△			
	MEM09-C				O	
	STR31-C	O	△	O	O	O
	STR32-C		O	O		
	STR33-C		△			O
	FIO01-C		△	O	O	
	FIO02-C		△	O		
	ENV03-C					
	FIO42-C	O	O	O	O	
	ARR02-C		O			
	FLP32-C	O				
	FLP33-C		O		O	O
	INT07-C	O	O		O	O
	INT13-C	O	O		O	O
허점 많은 방어	MSC30-C	O	O		O	
	MSC32-C		O			
	POS02-C					
	POS36-C		△	O		
	POS37-C			O		

하거나 특정 환경에서 검출이 명시되어 있는 경우가 많이 있으며, 주로 많이 사용되는 함수나 환경에 대한 오류를 검출하는 것을 볼 수 있다. Splint는 배열 범위를 벗어나는 연산이나 잘못된 계산 오류에 대한 검출만 가능하며, 나머지 세 개의 도구는 서로 검출하는 오류에 차이는 있으나 비슷한 정도의 검출 비율을 가지고 있다.

5. 결 론

C 언어의 특성인 약한 타입 체계와 포인터에서 여러 가지 취약성이 비롯된다. 타입 유연성과 포인터 문제를 동시에 드러내는 언어 구조물이 각종 배열이며, 오래 전부터 C 프로그램이 버퍼 오버플로우와 같은 공격에 취약한 이유이다. C 언어는 시스템 프로그래밍을 지배하는 언어로서 현재도 미래에도 계속 사용될 것으로 전망된다. 취약성 문제의 근본적인 해결책은 C 언어의 정의를 수정하는 것인데, 이미 큰 군집을 이룬 코드 기반 때문에 거의 불가능한 일이 될 것이다. 또 다른 해결책은 C 보다 안전한 언어를 사용하는 것인데, 적용 분야에 제약이 있을 수 있으며, 특히 하드웨어와의 경계에서 진행되는 일에는 쉽지는 않을 것이다. 한편 C99 표준에 충실하다고 해도 취약성을 유발하는 프로그래밍은 지속될 것이다.

안전한 코딩을 위한 표준 적용과 이를 지원하는 도구들의 사용은 시스템의 안전을 위해 경제적이며 효과적인 방법이라고 생각된다. 코딩 표준을 정할 때 중요하게 고려해야 하는 사항은 프로그래머가 준수할 수 있는 수준을 인지하는 것이다. 프로그래머 개인 능력에도 차이가 많을 수 있기 때문에 이 수준을 정하는 것은 쉽지 않으며, 이 문제에는 코딩 표준 검사 도구가 유용할 것이다. 조금 복잡하고 지켜야 할 것이 많더라도 도구를 사용하여 표준 준수 여부를 검사할 수 있다면 규칙 수가 많아도 문제되지 않을 것이다. 공공에게 열려 있는 프로그램 분석 연구용 기반 도구인 Compass/ROSE에 아직 추가할 부분이 많이 남아 있다. 궁극적으로는 특정 표준을 선택 사항으로 컴파일러에게 주면, 컴파일러가 주어진 표준을 위반하는 코드에 대한 경고 메시지를 내보내게 하는 것이 바람직하다.

상위 25 프로그래밍 오류를 발표할 때 중요한 의도 중의 하나는 프로그래밍 교육에 있어 안전한 코딩 개념을 도입하게 하는 변화를 촉진시키는 것이라고 밝히고 있다. 아직 체계적인 교과 과정이 국내에는 도입되지 않고 있는데, 이 시기를 앞당길 필요가 있다.

참고문헌

[1] Matt Bishop, Computer Security: Art and Science, Addison-Wesley, 2002

[2] T. Eisenberg, D. Gries, etal., "The Cornell commission: on Morris and the worm," CACM vol 32, no 6, pp. 706-709, 1989

[3] CERT Advisory CA-2003-2- W32/Blaster worm, http://www.cert.org/advisories/CA-2003-20.html, 2003

[4] Aleph One, Smashing The Stack For Fun And Pro-

fit, Phrack Magazine vol 7, no 49, 1996
[5] 한근희, "시큐어코딩 표준과 전자정부 시스템", Secure Coding Standard 단기강좌, 동국대학교, 2009. 12
[6] Fortify, http://www.fortify.com
[7] ROSE, http://www.rosecompiler.org
[8] R. Seacord, The CERT C Secure Coding Standard, Addison-Wesley, 2009
[9] MISRA-C 2004, Guidelines for the use of the C language in critical systems, MISRA Limited, 2004
[10] 김명호, "고품질 SW를 위한 Coding Rule과 정적도구 활용 및 사례", Secure Coding Standard 단기강좌, 동국대학교, 2009. 12
[11] CWE, http://cwe.mitre.org
[12] Klocwork Insight, http://www.klocwork.com/products/insight
[13] LDRA Software Technology, http://www.ldra.com/test-bed.asp
[14] Splint - Secure Programming Lint, http://www.splint.org

부 록

가장 위험한 25 개 프로그래밍 오류와 CERT C 코딩 표준

대응되는 CERT C 코딩 표준이 명시되지 않은 CWE 항목은 C 언어와 관련이 없거나, CERT C 코딩 표준에 해당하는 규칙이나 권고가 없는 경우이다.

컴포넌트 사이의 안전한지 못한 상호작용

- CWE-20 적절하지 못한 입력 검증
  - INT06-C 문자열을 정수로 변환할 때에는 strtol() 또는 유사 함수를 사용한다.
  - ERR07-C 같은 일을 하는 함수이면 오류 검사를 수행하는 함수를 사용한다.
  - API00-C 함수 인자는 반드시 검증되어야 한다.
  - MEM10-C 포인터 검증 함수를 정의해서 사용한다.
- CWE-116 적절하지 못한 출력 구성(encoding and escaping)
  - MSC09-C 문자 코딩 - 안전을 위해서는 ASCII의 부분 집합을 사용한다.
  - MSC10-C 문자 코딩 - UTF8 관련 사항, 유효한 값의 범위 내에 있는지 검증한다.
- CWE-89 SQL 쿼리 보존 실패(SQL 주입). C 언어 관련 없음
- CWE-79 웹 페이즈 구조 보존 실패. C 언어 관련 없음
- CWE-78 OS 명령에 사용되는 특수 원소의 적절하지 못한 위생처리(OS 명령어 주입)
  - STR02-C 복잡한 서브시스템에 전달된 데이터는 위생처리 한다.
  - ENV03-C 외부 프로그램을 기동시킬 때에는 환경을 위생처리 한다.
  - ENV04-C 명령어 처리기가 필요하지 않으면, system() 함수를 호출하지 않는다.
- CWE-319 민감한 정보의 단순 텍스트 전송. 설계 단계의 취약성.
- CWE-352 교차-사이트 요청 위조. 설계 단계의 취약성. C언어 관련 없음
- CWE-362 경쟁 조건
  - FIO31-C 같은 파일을 동시에 두 개 이상 열지 않는다.
- CWE-209 오류 메시지 정보 유출
  - MEM03-C 재사용 가능한 자원에 저장된 민감한 정보는 반드시 지운다.

위험성이 높은 자원 관리

- CWE-119: 메모리 버퍼의 경계 안으로 연산 제한 실패
  - MEM09-C 메모리 할당 루틴이 초기화까지 할 것이라 가정하지 않는다.
  - FIO37-C 문자 데이터만 읽었을 것이라고 가정하지 않는다.
  - ENV01-C 환경 변수의 크기에 대하여 어떤 가정도 하지 않는다.
  - ARR00-C 배열 관련 연산에 대하여 이해하라.
  - ARR33-C 충분한 크기의 저장소로 복사되었음을 보장해야 한다.
  - ARR34-C 표현식의 배열 타입이 호환성이 있음을 확인한다.
  - ARR35-C 배열 한계를 넘는 루프 회전은 허락하지 않는다.
  - STR31-C 문자열 저장소는 문자 데이터를 NULL 마감 문자를 위한 충분한 공간을 갖고 있음을 보장한다.
  - STR32-C NULL 마감 바이트를 갖는 문자열이 되게 한다.
  - STR33-C 두 바이트 문자(wide character)로 구성되는 문자열의 크기를 정확히 한다.
  - MSC34-C 사용하지 말라고 권고된 함수나 오래되어 사용하지 않는 함수는 사용하지 않는다.
- CWE-642 치명적인 상태 데이터의 외부로부터의

제어

- CWE-73: 파일 이름이나 경로의 외부로부터의 제어
  - FIO01-C 파일 이름을 사용하여 신원 확인하는 함수에 대하여 주의한다.
  - FIO02-C 신뢰할 수 없는 출처로부터의 경로 이름은 정규화 한다.
- CWE-426 신뢰하지 못하는 탐색 경로
  - ENV02-C 같은 유효 이름을 갖는 여러 환경 변수에 주의한다.
- CWE-94 코드 발생 제어 실패 (코드 주입). C언어 관련 없음
- CWE-494 무결성 검사 없는 코드 내려받기. C 언어 관련 없음
- CWE-404 적절하지 못한 자원 폐쇄 또는 방출
  - FIO42-C 파일이 더 이상 필요하지 않을 때에는 반드시 적절한 절차로 닫는다.
  - MEM00-C 메모리 할당과 해제는 같은 모듈의 같은 추상화 수준에서 수행한다.
  - MEM11-C 힙 공간이 무한하다고 가정하지 않는다.
  - MEM31-C 동적 할당된 메모리는 정확히 한번만 해제하다.
- CWE-665 적절하지 못한 초기화
  - MEM09-C 메모리 할당 루틴이 초기화까지 할 것이라 가정하지 않는다.
  - ARR02-C 초기값 지정이 함축적으로 크기를 나타내더라도 배열 크기는 명시적으로 표시한다.
- CWE-682 부정확한 계산
  - INT10-C % 연산자를 사용할 때 나머지가 양수일 것이라 가정하지 않는다.
  - INT07-C 수치값에 대해서는 명시적으로 signed 또는 unsinged 지정된 문자 타입만을 사용한다.
  - INT13-C 비트 연산자는 무부호(unsigned) 피연산자에 대해서만 적용한다.
  - FLP32-C 수리 함수에서는 정의역과 치역 오류를 탐지하거나 방지하도록 한다.
  - FLP33-C 부동소수점 연산을 위해서는 정수를 부동소수점으로 변환한다.

허점 많은 방어

- CWE-285 부적절한 접근 제어(인증). 언어 독립적 취약성. 웹 서버/DB 관련
- CWE-327 깨지거나 위험 앙호학적 알고리즘 사용
  - MSC30-C 의사난수 발생을 위해서 rand() 함수를 사용하지 않는다.
  - MSC32-C 난수 발생기에 적절한 시드값이 주어지는 확인한다.
- CWE-259 엮여 들어간 패스워드 (프로그램 내에 패스워드사 문자열 상수로 나타남)
- CWE-732 치명적인 자원에 대한 안전하지 못한 허용 지정
- CWE-330 충분히 임의성을 확보하지 못한 값 사용
  - MSC30-C 의사난수 발생을 위해서 rand() 함수를 사용하지 않는다.
  - MSC32-C 난수 발생기에 적절한 시드값이 주어지는 확인한다.
- CWE-250 불필요한 권한으로 실행
  - POS02-C 최소 권한 원칙을 따른다.
  - POS36-C 권한을 내놓을 때에는 올바른 철회 순서를 준수한다.
  - POS37-C 권한을 내놓는 것이 제대로 수행되었음을 확인한다.
- CWE-602 클라이언트 측에서 강제할 수 있는 서버 측 보안. 설계 단계의 취약성

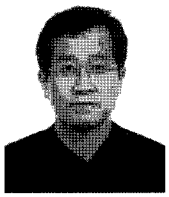

표 창 우

1980 서울대학교 공과대학 전자공학과 학사
1982 서울대학교 대학원 컴퓨터공학과 석사
1989 University of Illinois at Urbana-Champaign, Computer Science 박사
1991~현재 홍익대학교 컴퓨터공학과 교수
관심분야 : 프로그래밍 언어, 프로그램 최적화, 신뢰성 컴퓨팅
E-mail : pyo@hongik.ac.kr

한 경 숙

1993 홍익대학교 컴퓨터공학과 학사 졸업
1995 홍익대학교 대학원 컴퓨터공학과 석사
2002 홍익대학교 대학원 컴퓨터공학과 박사
1995~1997 삼성전자 반도체사업부 주임연구원
2000~2003 (주)참좋은인터넷 부사장
2003~현재 한국산업기술대학교 부교수
관심분야 : 컴파일러, 안전한 코딩, 임베디드 소프트웨어
E-mail : khan@kpu.ac.kr