정 보 보 호 학 회 지
제25권 제1호, 2015. 2

전자정부 응용 개발을 위한 시큐어 코딩 가이드

한 경 숙*, 표 창 우**

요 약

컴퓨터와 네트워크가 보안 상 안전하려면 무엇보다 사용되는 시스템 및 응용 프로그램에 보안약점이 없어야 한다. 보안약점은 공격자가 이용하여 제어 흐름을 탈취하거나 원하는 정보를 유출할 수 있게 하는 프로그램 상의 불완전한 부분을 뜻한다. 보안약점이 없는 프로그램을 만들기 위한 방법으로 시큐어 코딩 규칙을 정의하고, 프로그램 개발 단계에서 이를 적용하게 할 수 있다. 코딩 과정에서 시큐어 코딩 규칙을 준수하여 보안약점 발생을 억제하는 방법은 예방적 조치이다. 아무런 규칙 없이 코딩을 진행한 후 보안약점을 분석, 제거하는 방법보다 프로그래머들에게 부담이 적고, 정적 분석을 사용하여 보안약점을 분석하는 도구들의 치명적인 약점인 오탐 비율을 낮춘다. 이 글은 2014년까지 소프트웨어 개발 보안 센터를 중심으로 진행된 행정자치부의 C/C++, Java, PHP 프로그래밍 언어를 위한 시큐어 코딩 가이드에 대하여 설명한다.

Ⅰ. 서 론

컴퓨터와 네트워크가 보안 상 안전하려면 무엇보다 사용되는 시스템 및 응용 프로그램에 보안약점(weakness)이 없어야 한다. 보안약점은 공격자가 이용하여 제어 흐름을 탈취하거나 원하는 정보를 유출할 수 있게 하는 프로그램 상의 불완전한 부분을 뜻한다. 보안약점이 실제로 이용당해 침해 사례가 발생하게 되면 보안약점은 취약점(vulnerability)으로 분류된다.

실제로는 취약점이 수집되면 분석과정을 거쳐 보안약점으로 분류된다. MITRE의 CVE(Common Vulnerabilities and Exposures)[1]가 수집되어 분석된 취약점 리스트의 데이터베이스이며, CWE (Common Weakness Enumerations)[2]는 분류된 보안약점 데이터베이스이다.

안전한 프로그램을 만들기 위해서는 설계 단계에서부터 보안약점이 발생하지 않도록 해야 한다. 프로그램을 설계할 때에 주어지는 요구 사항들은 대부분 정상적인 입력을 가정하고 작성된다. 물론 악의적이거나 비정상적인 입력에 대하여 어떻게 대응할 것인지 요구 사항에 포함시킬 수도 있지만, 예상치 못한 상황에 대하여 체계적으로 다룰 수 있는 방법은 없다. 비정상적인 상호작용에 대하여 예상할 수 있는 경우에는 요구 사항을 마련할 수 있겠지만 모든 비정상의 경우를 예상한다는 것 자체가 방법론적으로 불가능하다.

보안약점이 없는 프로그램을 만들기 위한 방법으로 시큐어 코딩 규칙을 정의하고, 프로그램 개발 단계에서 이를 적용하게 할 수 있다. 시큐어 코딩은 보안약점이 없는 프로그램을 만드는 코딩 규칙과 이를 적용하기 위한 다양한 자원과 도구들로 구성되어 있다. 넓은 뜻의 시큐어 코딩은 프로그램 설계 단계의 활동도 포함하기도 한다[3,4].

2009년부터 대학과 학회 중심으로 논의되기 시작한 국내 시큐어 코딩이 2012년 소프트웨어 개발 보안 센터의 발족으로 구체화 되었다. 그 성과로서 2012년 6월 당시 행정안전부가 고시한 “정보시스템 구축, 운영 지침”이 입법화 되었고, 반드시 고려되어야 할 보안약점 43개가 확정되었다. 전자정부에 사용되는 프로그램의 개발자들은 명시된 43개의 보안약점이 없는 프로그램을 만들어야 한다. 보안약점은 현재 47개로 확대되었다. 이 글은 2014년까지 소프트웨어 개발 보안 센터를 중심으로 진행된 행정자치부의 C/C++, Java, PHP 프로그래

본 연구는 2014년 한국인터넷진흥원 “시큐어코딩 기반 SW 개발보안 기반 기술 연구” 위탁과제의 연구결과로 수행되었음(No. Q1203813)

* 한국산업기술대학교 컴퓨터공학부(khan@kpu.ac.kr)
** 홍익대학교 컴퓨터공학과(pyo@hongik.ac.kr)

밍 언어를 위한 시큐어 코딩 가이드에 대하여 설명한다.

Ⅱ. 보안약점에 대한 접근

보안약점은 많은 경우 코딩 오류로 볼 수 있으나, 보안 관점이 아니면 오류로 보기 힘든 경우도 있다. 일상적인 프로그램 실행에서는 아무런 문제를 발생하지 않지만, 프로그램의 한계를 넘는 크기의 데이터를 입력받는 것과 같은 상황이 되면 프로그래머가 의도하지 않은 상태에 빠지게 하는 프로그램 상의 결함도 포함된다. 예를 들어 루프 제어 변수 i가 정수형이면 32비트 마이크로프로세서는 2^{31}+1번째 회전에서는 루프 제어 변수가 음수가 된다. 변수 i가 배열 원소 접근에 사용된다면 프로그래머가 예상하는 지역을 벗어난 지역을 접근하게 된다. 변수 i가 가질 수 있는 값의 범위를 묵시적으로 0 ~ 2^{31}-1로 가정하는 것을 프로그래밍 오류로 보아야 하는지는 논란의 여지가 있지만, 공격자는 i의 값이 -1인 경우 어떤 메모리에 저장된 값을 공격자가 원하는 값으로 지정할 수 있다는 것을 알 경우, 루프 회전을 한계까지 밀어 붙일 수 있는 방법을 알아내어 시도할 것이다. 변수 i가 가지는 값의 범위를 검사하지 않는 것이 프로그램 오류라고 무조건 단정 짓는 것은 일률적으로 결정할 수 있는 것은 아니다.

이 문제를 해결하는 방법은 두 가지로 생각해 볼 수 있다. 한 가지는 프로그램을 분석하여 루프 제어 변수 i의 값이 한계를 넘길 가능성이 있는지를 분석하고, 가능성이 있는 경우 i 값을 검사하는 코드를 삽입하여 오류 상황을 사전에 방지하는 것이다. 분석은 프로그래머가 코드를 읽는 과정을 통할 수도 있고, 프로그램 분석 도구를 사용하는 방법도 있다. 분석 도구는 정적 분석을 사용할 수도 있고, 동적 분석, 또는 혼합적 방법을 사용할 수 있다. 간단히 말해 보안약점을 탐색하여 제거하는 방식이다.

이 방법은 보안약점이 발생하지 않는 한 프로그래머가 원하는 방식으로 프로그램을 작성할 수 있게 한다. 이 같은 장점은 동시에 약점으로 작동한다. 프로그래머가 보안약점의 발생에 대하여 전적으로 책임져야 한다. 별도의 보안약점 진단 과정을 거친다고 하더라도 프로그램에서 보안약점이 발생하지 않게 해야 한다. 분석 도구의 도움을 받더라도 전문적인 검토에 의해 보안약점을 찾아내야 한다. 분석 도구의 사용은 보안약점 탐색 시간을 줄여줄 수도 있지만, 오탐 결과를 대량으로 발생시키는 경우가 빈번하기 때문에 오탐과 정탐을 판정하는 작업이 새로운 일이 되기도 한다. 또한 오탐이 있더라도 모든 보안약점을 전부 찾아 준다는 것을 보장하는 것 또한 쉽지 않다.

이와 대조적으로 보안약점이 프로그램에 발생하는 것을 간접적으로 방지할 수 있는 방법이 있다. 보안약점 발생을 억제하는 코딩 규칙들을 정하고, 프로그래머가 프로그램을 설계할 때부터 코딩 과정에서 이 규칙들을 따르게 하는 것이다. 앞의 예에서 루프 제어 변수는 반드시 무부호 정수(unsigned int)로 해야 함을 규칙으로 정할 수 있다. 루프 제어 변수 i가 이 규칙에 따라 정의되었다면 변수 i는 음수 값을 가질 수 없게 되어 정수 오버플로우[5] 보안약점이 초래할 수 있는 문제 중 일부를 피할 수 있게 된다. 또 다른 예로서 버퍼로 사용되는 char 타입의 배열을 순차적으로 접근하는 루프의 회전 한계는 버퍼의 크기 선언과 같거나 작아야 한다는 규칙을 적용하게 되면, 버퍼 오버플로우[6]를 방지할 수 있다. 프로그래머는 규칙이 어떻게 해당 보안약점 발생을 억제하는지 이해할 필요가 없고, 단순히 규칙을 따르기만 하면 보안약점 발생은 간접적으로 억제된다. 어떤 코드가 보안약점으로 작동하는지 알지 못하더라도 규칙만 따르면 프로그램을 작성할 수 있다.

예를 들어, 정수 오버플로우 보안약점을 억제하기 위하여 “비교하거나 저장하기 전에 더 큰 자료형에서 값을 평가한다.”라는 규칙을 적용하는 경우, [그림 1]의 함수가 [그림 2]와 같이 수정될 수 있다. [그림 1]의 6, 7번 줄에서 DataSize+addSpace의 값을 평가하지 않고

```
1:  int CopyData(char **Data, size_t DataSize,
                                    size_t addSize)
2:  {
3:    char *ExpandData;
4:    if (addSize <= 0)
5:      return 1;
6:    ExpandData=(char *)malloc(DataSize+addSpace);
7:    strcpy(ExpandData, *Data);
8:    free(*Data);
9:    *Data = ExpandData;
10:   return 0;
11: }
```

〔그림 1〕 시큐어 코딩 규칙 적용 전

```
1:  int CopyData(char **Data, size_t DataSize,
                                    size_t addSize)
2:  {
3:     char *ExpandData;
4:     unsigned long long TotalSize
           = (unsigned long long)(DataSize+addSize);
5:
6:     if(addSize <= 0 || TotalSize > UINT_MAX){
7:      printf("ERROR\n");
8:      return 1;
9:     }
10:    ExpandData = (char *)malloc(DataSize+addSize);
11:    strcpy(ExpandData, *Data);
12:    free(*Data);
13:    *Data = ExpandData;
14:    return 0;
15: }
```

〔그림 2〕 시큐어 코딩 규칙 적용 후

메모리 할당을 하는 것을 볼 수 있다. 계산 결과가 size_t 범위를 초과하는 결과가 나올 경우 동적 메모리 할당의 크기가 의도한 것보다 작아질 수 있고, 이는 버퍼 오버플로우를 유발할 수 있다.

[그림 2]의 4~9 행에서와 같이 더 큰 자료형 변수를 사용하여 그 값을 평가하게 되면, 정수 오버플로우 발생 여부를 판단할 수 있게 되어 이에 대응할 수 있게 된다. 이를 통해 정수 오버플로우 보안약점을 일부 억제하는 효과를 볼 수 있다.

Ⅲ. 전자정부 개발을 위한 코딩 가이드

국내의 시큐어 코딩 발전은 정부 주도로 시작되었다. 2012년 6월 당시 행정안전부가 고시한 "정보시스템 구축, 운영 지침"에 따라 정보화 사업 수행 중에 안전한 소프트웨어를 개발하는데 사용하기 위한 소프트웨어 개발보안 가이드가 개발되었다. 개발보안 가이드는 소프트웨어 개발 과정에서 나타날 수 있는 선별된 43 개의 보안약점을 중심으로 기술되어 있으며, 분석 도구를 사용하거나 코드 리뷰를 통해 지정된 보안약점을 검사하여 제거하도록 하고 있다[7,8].

보안약점 중심의 개발 보안 가이드는 보안약점 제거를 위한 활동에는 도움이 되지만, 프로그래머들에게는 불편한 점이 있다. 프로그래머가 개발 과정 중에 프로그램의 정확성과 기능, 성능을 고려하여 작성하는 것은 본연의 임무이지만, 발생할 수 있는 기능의 오용이나 악용 사례까지 고려하는 것은 별도의 노력이 필요하다. 설계 및 코딩 단계에서 개발 명세에 포함되지 않은 발생 가능한 모든 보안 위험 요소를 분석하여 사전에 대비하는 것은 명세에 따른 프로그램을 작성하는 활동보다 더 많은 노력과 시간을 요구하기 때문이다. 프로그램이 개발 명세를 만족하는지를 확인하는 것에 비하여 아직 밝혀지지 않은 보안약점을 포함하여 가능성이 있는 모든 보안약점을 찾아내는 것은 프로그램이 명세를 만족하게 하는 일과 비교하여 난이도의 규모가 다르다. 또한, 프로그램을 분석하여 존재하는 보안약점을 제거하는 방식은 많은 개발 경험과 소프트웨어 보안 분야의 지식이 필요하다.

보안약점 제거에 초점을 둔 개발 보안 가이드와 대조적으로 프로그래머들이 코딩 시점에서 참조할 수 있는 시큐어 코딩 가이드의 필요성이 제기되었고, 2012년부터 시작된 소프트웨어 개발 보안 센터[9]를 통해 주요 프로그래밍 언어에 대한 시큐어 코딩 가이드 개선과 신규 개발이 3개년에 걸쳐 진행되었다. 2013년 8월에는 보안약점 리스트가 갱신되어 47개로 증가하였고, 개발 보안 가이드도 이에 맞추어 개선되었다. C/C++와 Java에 대한 가이드는 사용 가능한 형태로 완성되었고, PHP에 대한 가이드가 새로 개발되었다.

이 가이드들은 보안약점을 억제하는 효과가 있는 코딩 규칙들을 제시하고 이를 적용한 예를 보여주는 형태로 구성되어 있다. 프로그래밍 언어의 구문과 문맥에 따라 기술된 코딩 규칙을 준수하는 방식은 많은 기업체에서 현재 적용하고 있는 방법이다. 다수의 기업체나 산업분야에서 코딩 스타일이나 신뢰성을 고려한 코딩 규칙을 규정하여 사용하고 있으며, 프로그래머는 사용 중인 언어의 구문과 문맥에 따라 대응되는 규칙을 적용하여 프로그램을 개발한다. 따라서 제거해야 할 보안약점을 검출하는 보안약점 제거 방식보다 프로그래밍 언어의 구문과 문맥에 따라 보안약점을 억제할 수 있는 규칙을 준수하며 개발하는 보안약점 억제 방식이 프로그래머들에게는 친숙한 개발 관행이라 할 수 있다. 이 방법은 초기 개발 시간이 증가할 수 있으나 개발자에게 익숙하고 좀 더 쉽게 적용할 수 있는 방법이다.

Ⅳ. 시큐어 코딩 가이드 개발 방법

C/C++와 Java 가이드에 대해서는 이미 주어진 보안 약점이 있었지만, 보안약점이 주어지지 않은 경우에 시큐어 코딩 가이드를 개발할 때 적용 가능한 방법을 소개한다. 목표 프로그래밍 언어가 정해지면, 그 언어로 개발된 프로그램에서 발생한 보안 사고에 대한 기록을 수집하는 일로 시작한다. 이미 노출되어 공격당한 보안약점은 취약점으로 분류되어 CVE 리스트에 목록화 되어 있다. CVE에 나타나는 빈도와 최근 발생 시기를 고려하여 탐색 범위를 정하고, 이를 기반으로 주요 취약점을 걸러낸다.

걸러진 취약점을 분류하여 CWE와 매핑 한다. 경우에 따라 CWE에 바로 사상되지 않는 CVE가 있을 수 있다. 이런 경우 CVE를 포함할 수 있는 상위 CWE에 사상한다. CWE 리스트가 확정되면 이를 억제할 수 있는 코딩 규칙을 만든다. 규칙을 만들 때에는 보안약점이 있는 실제 프로그램 사례가 도움이 된다. 프로그램의 구체적인 사례는 레드햇의 버그질라[10]와 같은 CVE에 연결된 사이트들에서 발견할 수 있다. 사례 프로그램의 보안약점이 드러나지 않도록 안전한 프로그램으로 만들기 위한 방법을 개발하고, 이를 그러한 사례에 대한 코딩 규칙으로 만든다.

Ⅴ. 개발 가이드 현황: C/C++, Java, PHP

현재 개발된 C/C++ 가이드와 자바 가이드는 3 개의 계층으로 구성되어 있으며, 3 계층은 소프트웨어 보안에 대한 방어선에 대응한다. 외부 공격에 대하여 네트워크, 호스트, 어플리케이션 계층 순으로 방어선이 구축되었다고 보는 견해이다.

각 계층에서는 프로그램 구문과 보안 요소에 따라 규칙을 분류하였으며, 각 분류에 해당하는 코딩 규칙은 관련된 보안약점 순서에 따라 기술하였다. 행정자치부의 47개 보안약점에 대하여 관련된 코딩 규칙을 개발하였고, 각 규칙은 구체적인 세부 규칙을 포함하도록 하였다.

프로그램 구문과 보안 요소에 따른 단원 구성은 프로그램 개발 과정에서 개발하고 있는 부분에 따라 쉽게 참고할 수 있도록 하기 위한 것이다. 예를 들어 자바 가이드의 변수 부분에는 '지역변수 초기화', 'public static 변수 사용 제한', '공유된 기본형 변수', '권한이 있는 블록에서 오염된 변수 사용 금지'와 같은 규칙이 포함

〔표 1〕 C/C++ 가이드 규칙과 보안약점

계층	분류(구문, 보안요소)	규칙 수	세부규칙 수	관련 CWE
어플리케이션	1. 정수	3	4	120, 190, 676, 681
	2. 변수	1	1	457
	3. 배열과 포인터	2	2	119, 476
	4. 문자열	3	3	119, 134, 676
	5. 수식	3	5	390, 674, 754
	6. 동적 메모리	7	10	120, 131, 190, 404, 415, 416, 476, 754
	7. 함수	8	13	367, 390, 476, 495, 674, 676, 754
	8. 입출력	5	10	22, 73, 80, 90, 99, 134, 367, 404, 676, 770
	9. 질의와 API 사용	6	11	20, 73, 78, 80, 88, 89, 192, 197, 350, 367, 464, 676, 754
	10. 자원관리	3	5	22, 73, 404, 415, 770
	11. 정보유출	4	13	259, 311, 321, 326, 476, 489, 754, 798
	12. 인증과 인가	1	3	676
	13. 암호화	5	8	22, 73, 80, 90, 99, 134, 495, 676
	소 계	51	88	
호스트	14. 입출력	1	2	367, 676
	15. 실행환경	1	1	404, 770
	16. 정보유출	1	2	89
	17. 인증과 인가	3	5	20, 73, 78, 80, 88, 192, 197, 367, 464, 676
	18. 암호화	3	8	22, 73. 350, 754
	소 계	9	18	
네트워크	19. 입출력	1	3	404, 770
	20. 암호화	1	3	404, 415
	소 계	2	6	
합 계		62	112	

〔표 2〕 Java 가이드 규칙과 보안약점

계층	분류(구문, 보안요소)	규칙 수	세부규칙 수	관련 CWE
어플리케이션	1. 변수	4	5	266, 272, 413, 457, 493, 500, 567, 667, 732
	2. 수식	7	10	190, 197, 252, 362, 366, 413, 476, 567, 662, 667, 674, 681
	3. 객체	4	7	302, 362, 470, 476, 495
	4. 메소드	3	10	252, 581, 674, 697
	5. 입출력	5	11	80, 90, 99, 116, 134, 171, 180, 289, 647
	6. 예외처리	4	4	209, 390, 584, 754
	7. 스레드	1	2	367
	8. 공유와 동기화	5	6	362, 366, 367, 413, 567, 609, 662, 667
	9. 직렬화	1	2	499, 502
	10. 질의와 API 사용	6	14	78, 89, 300, 319, 327, 347, 494, 652, 643
	11. 자원관리	1	3	404, 405, 459, 770
	12. 정보유출	2	3	259, 489, 798
	13. 인증과 인가	2	4	307, 494
	14. 암호화	2	5	327, 330
	소 계	47	86	
호스트	15. 입출력	1	3	404, 405, 459, 770
	16. 자원관리	2	5	276, 277, 278, 279, 281, 285, 404, 405, 459, 732, 770
	17. 정보유출	5	6	144, 150, 276, 279, 359, 377, 459, 532, 533, 542, 615, 732
	18. 인증과 인가	4	10	276, 277, 278, 279, 281, 285, 521, 732
	19. 암호화	3	5	310, 311, 312, 326, 759
	소 계	15	29	
네트워크	20. 변수	1	3	488
	21. 입출력	2	6	434, 601, 807
	22. 질의와 API 사용	2	3	311, 350
	23. 자원관리	1	3	404, 405, 459, 770
	24. 인증과 인가	2	5	287, 302, 306, 307, 352, 602
	25. 암호화	1	2	311, 319
	26. 웹 서비스	7	15	113, 287, 302, 306, 307, 352, 488, 539, 602, 614, 807
	소 계	16	37	
합 계		78	152	

되는데, 변수에 관련된 코딩 규칙을 참고하고자 할 때 이 단원을 참고할 수 있다.

각 코딩 규칙은 구체적인 세부 규칙으로 세분화 하였고, 이를 기술하고 설명하였다. 각 규칙에 대해 비준법 코드 예와 그 문제점을 기술하고 이에 대응하여 세부 규칙을 적용한 준법 코드를 예시하였다. 비준법 코드에서는 보안약점을 내포하고 있는 코드의 위치를 표시하고 그 부분에서 위배되는 규칙과 발생할 수 있는 문제에 대하여 설명하였다. 준법 코드에서는 어떤 세부 규칙을 적용하여 코드를 수정하였는지 설명하고 해당 코드 부분을 표시하였다. 또한 관련 보안약점에 대하여 간략하게 설명하였다. 각 코딩 규칙과 관련된 참고자료는 코

〔표 3〕 PHP 가이드 규칙과 보안약점

분류(구문, 보안요소)	규칙 수	세부규칙 수	관련 CWE
1. 변수	3	6	119, 120, 453, 457, 621
2. 입출력	3	6	20, 22, 79, 94, 95, 184, 297, 346, 434, 611, 616
3. 질의와 API 사용	1	2	89
4. 실행환경	1	1	16
5. 정보유출	2	4	201, 209, 215, 359
6. 인증과 인가	1	2	285
7. 암호화	2	6	259, 261, 311, 327
8. 직렬화	1	1	502
합 계	14	28	

딩 규칙과 CERT 규칙, CWE ID의 매핑 테이블을 이용하여 참조할 수 있다.

C/C++ 가이드는 현재 [표 1]과 같이 3부, 20 분류, 62 규칙, 112 세부규칙을 포함하고 있으며, 자바 가이드는 [표 2]와 같이 3부, 26분류, 78 규칙, 152 세부규칙을, PHP 가이드는 [표 3]이 보여 주는 것과 같이 8분류, 14 규칙, 28 세부규칙을 포함하고 있다.

Ⅵ. 정적 분석도구의 사용

정적 분석 도구를 적절히 사용하면 프로그램에서 보안약점을 효율적으로 분석하여 제거할 수 있다. 정적 분석도구는 실행 파일을 대상으로 하기도 하지만, 일반적으로 소스코드를 분석한다. 정적 분석기가 사용하는 분석 기법은 어휘 분석, 구문 분석, 타입 추론, 데이터 흐름 분석, 기호적 실행(symbolic execution), 추상 해석(abstract interpretation), 모델 검사, 제약 조건 풀기(constraint solving), 정리 증명(theorem proving)과 같은 분석 기법을 사용한다. 각 도구들은 모든 보안약점을 분석 대상으로 하기도 하지만, 버퍼 오버플로우와 같은 특정 보안약점 분석에 집중되어 있기도 하다. 실제 입력값이나 실행 상황을 배제하고 프로그램을 분석하는 방법이므로 오탐과 미탐의 소지가 있고, 도구의 분석 결과가 정탐인지 여부를 파악하기 위한 전문가의 코드 리뷰가 필요한 경우가 빈번하게 발생한다. 이러한 문제는 개발 단계에서 정적 분석도구 사용에 의한 보안약점 탐지를 기피하게 되는 원인이 되기도 한다.

많이 알려진 상용 도구로는 Coverity[11], Fortify[12], Klocwork[13], LDRA[14], CodeSonar[15] 와 같은 외산 도구와 국내 개발 제품으로 공통인증(CC 인증)을 통과한 Sparrow[16]와 SecurityPrism[17]이 있다. 공개 도구로는 Compass/ROSE[18]와 Splint[19], CppCheck[20], Clang[21]과 같은 C 언어 관련 도구와 PMD[22], Findbugs[23] 같은 자바 관련 도구들을 예로 들 수 있다. 상용 도구는 대부분 C/C++, 자바 이외의 다양한 언어를 지원한다. 공개 도구는 Compass/ROSE를 제외하고는 분석 기능이 제한적이지만, 대부분 사용자가 분석 규칙을 추가하여 확장할 수 있도록 구성되어 있다. 국내에서 개발한 정적 분석도구들은 행정자치부에서 지정한 47개 보안약점에 대한 분석 기능을 일부 제공하고 있다.

이러한 도구들은 CWE에서 분류한 보안약점에 대해서, 또는 CERT의 코딩 규칙에 대해서 검사 여부를 나타내는 경우가 있다. 그러나 진단 도구가 어떤 코딩 규약에 대하여 검출한다는 것은 그 규약을 준수하지 않는 모든 보안 문제점을 검출한다는 것을 의미하는 것은 아니다. 일부 특정한 경우에 대해서만 진단 가능하기도 하고, 특정 함수에 대해서만 진단하기도 한다. 이러한 진단도구에 대한 정보는 개발자나 감리자가 명확하게 범위를 이해하고 사용할 수 있도록 해야 하며, 진단의 보조 도구로 사용해야 한다.

이러한 정적 분석도구 사용 이외에도 실행시점의 정보를 사용하는 퍼징[24]과 같은 동적 분석도구의 사용이나 실행 환경 강화, 코드 리뷰 등의 다양한 방법을 고려할 수 있다. 또한 모든 보안활동은 개발자에게 있어서 추가의 작업임을 인식할 필요가 있다. 기능성 위주의 사고를 하는 개발자에게 있어서 보안을 고려한 개발은 필요성은 인정하지만 반드시 해야 하는지 의구심이 드는 추가의 작업 정도로 인식될 수 있다. 개발자들이 이러한 보안활동을 수행할 수 있도록 하기 위해서는 교육을 통한 인식 변화와 함께 그에 합당한 보상이 주어져야 할 것이다.

2012년 KISA는 보안약점 분석 도구 평가를 위한 테스트 코드 모음[25]을 개발하였다. NIST의 SAMATE 테스트 모음[26]과 유사하다. 한편 C/C++ 프로그램 분석기 평가에 대해서는 ISO 명세 17961[27]에서 예시로 사용된 프로그램으로 구성된 검증 패키지를 CERT가 제작한 것이 있다. 국내 시큐어 코딩에 대한 투자는 국제 표준화 움직임과 긴밀히 연계하여 중복되거나 흐름에서 이탈하는 일이 없게 해야 한다.

Ⅶ. 결 론

사회 전반적인 정보 보안 수준을 높이기 위해서는 공개 원칙 하에서 다루어져야 한다. 실제로 사용되는 프로그램에서 인지된 보안약점은 공공 목적의 중요 자산으로 취급되어야 한다. 물론 공개 범위와 접근을 허용할 대상들에 대해서는 논의 과정을 거쳐 적절한 정책이 마련되어야 하지만, 연구자, 개발자, 그리고 무엇 보다 정보 분야 학생들이 교육을 목적으로 할 때에 사용 가능해야 한다. 이 분야에서 선두를 달리고 있는 미국의 경우를 보면, 상당 부분 공개가 이루어지고 있다. 시간차를 두고 공개하는 것도 현실적인 방법으로 보인다. 또한 무분별하게 공개하기보다 비영리 기관이나 관련 정부기관 또는 연구소, 대학에 위임하여 관리하게 하는 것도

바람직하다고 판단된다.

시큐어 코딩 규칙에 대한 연구는 현재 행정자치부 주관으로 C/C++, Java에 대해서는 상당 부분 완성되어 진화 과정을 거칠 것으로 예상되며, PHP 언어에 대해서는 이제 시작되었다고 볼 수 있다. 다른 언어들, 특히 스마트 폰에 사용되는 앱 개발에 사용되는 언어에 대한 코딩 가이드가 개발되어야 한다. 코딩 규칙 준수 여부를 검증할 수 있는 정적 분석 도구가 개발되면 시큐어 코딩 확산과 효용성 증진에 도움이 될 것으로 예상된다. 또한 전문 인력 양성이 이루어져야 한다. 전문 인력은 감리 기능인들이 담당할 수 있으나 보안이나 시큐어 코딩에 대하여 더 체계적인 훈련을 거친 인력의 필요성이 높다.

시큐어 코딩은 보안약점 제거 방식과 억제 방식을 함께 적절히 사용해야 그 목적을 달성할 수 있다. 이 두 방식은 상호 보완적이어서 억제 방식이 강하게 적용되면 제거 되어야 할 보안약점 수가 감소하게 되며, 동시에 분석 도구를 사용할 때 오탐을 줄여 준다. 프로그래머들에게는 코딩 규칙 적용을 강력히 요구해야 하며, 프로그램의 일부가 완성되면, 코드 리뷰를 통해 보안약점 존재 여부를 면밀히 살펴야 한다. 보안약점 검색이나 규칙 준수 여부를 검증하는데 정적 분석 도구를 적절히 활용하면 도움이 될 것이다.

참 고 문 헌

[1] “Common Vulnerabilities and Exposures,” http://cve.mitre.org/

[2] “Common Weakness Enumeration,” http://cwe.mitre.org/

[3] Gray McGraw, “Software Security: Building Security in,” Addison-Wesley, 2006

[4] “CERT,” http://www.cert.org/

[5] Iván Arce, Elias Levy, “The rising threat of vulnerabilities due to integer errors,” Security & Privacy, IEEE, 2003. 8

[6] Alef One, “Smashing The Stack For Fun And Profit,” Phrack Magazine, Vol. 7, No. 49. 1996

[7] 행정안전부, “전자정부 소프트웨어 개발·운영자를 위한 소프트웨어 개발보안 가이드,” 행정안전부, 2012. 5

[8] 행정안전부, “정보시스템 구축 운영 지침(행정안전부고시 제2011-36호),” 행정안전부, 2012. 9

[9] “고려대, '소프트웨어 개발보안 연구센터,' 선정,” http://www.newswire.co.kr/newsRead.php?no=624730

[10] “Red Hat Bugzilla,”, https://bugzilla.redhat.com/

[11] “Coverity Prevent,” http://www.coverity.com/

[12] “HP Fortify Static Code Analyzer,” http://www8.hp.com/us/en/software-solutions/static-code-analysis-sast/

[13] “Klockwork,” http://www.klockwork.com/

[14] “LDRA Software Technology,“ http://www.ldra.com/

[15] “CodeSonar,“ http://www.grammatech.com/codesonar

[16] “Sparrow,“ http://www.fasoo.com/site/fasoo/sourcecodeanalysis/sparrow.do

[17] “SecurityPrism,“ http://www.gtone.co.kr/main/ag/sp.php

[18] “ROSE compiler infrastructure,“ http://rosecompiler.org/

[19] “Splint-Secure Programming Lint,“ http://www.splint.org/

[20] “CppCheck,“ http://cppcheck.sourceforge.net/

[21] “Clang Static Analyzer,“ http://clang-analyzer.llvm.org/

[22] “PMD,“ http://pmd.sourceforge,net/

[23] “Findbugs,“ http://findbugs.sourceforge,net/

[24] Godefroid, Patrice, Michael Y. Levin, and David A. Molnar. “Automated Whitebox Fuzz Testing.” PLDI’08, Tucson, USA, July 2008

[25] 방지호, 하란, “소프트웨어 보안약점 기반의 오픈소스 보안약점 진단도구 분석,“ 한국정보과학회 2013 한국컴퓨터종합학술대회, 2013. 6

[26] “NIST SAMATE,“ http://samate.nist.gov/

[27] “ISO/IEC TS 17961:2013 Information technology -- Programming languages, their environments and system software interfaces -- C secure coding rules,“ http://www.iso.org/iso/

〈저 자 소 개〉

한 경 숙 (Kyungsook Han)
정회원

1993년 2월 : 홍익대학교 컴퓨터공학과 졸업
1995년 2월 : 홍익대학교 컴퓨터공학과 석사
2002년 2월 : 홍익대학교 컴퓨터공학과 박사
2003년 3월~현재 : 한국산업기술대학교 컴퓨터공학과 부교수
관심분야 : 시큐어코딩, 컴파일러, 임베디드 소프트웨어

표 창 우 (Changwoo Pyo)
정회원

1980년 2월 : 서울대학교 전자공학과 졸업
1982년 2월 : 서울대학교 컴퓨터공학과 석사
1989년 12월 : U. of Illinois at Urbana-Champaign, 전산학 박사
관심분야 : 프로그램 분석, 최적화, 병렬화, 프로그램 보안