안드로이드 앱 시큐어 코딩 가이드 연구

오준석*, 최진영**
*고려대학교 컴퓨터・전파통신공학과
e-mail: {jsoh, choi}@formal.korea.ac.kr

Research on Android App Secure Coding Guide

Joon-Seok Oh*, Jin-Young Choi**
*Dept of Computer and Radio Communications Engineering, Korea University

요 약

소프트웨어가 대형화되고 복잡해짐에 따라 소프트웨어에 내재하고 있는 소프트웨어 허점(weakness)의 발생률이 높다. 이런 허점은 컴파일러에 의해 탐지되지 않고, 공격자에 의해 발견되기 쉽다는 특징이 있기 때문에 소프트웨어 취약성을 야기한다. 스마트폰의 확산으로 인해 다양한 종류의 스마트폰 앱이 개발되고 있다. 이에 따라 스마트폰 앱이 대형화되고 복잡해지고 있으므로, 여기에 내재하는 소프트웨어 허점을 사전에 예방하는 것은 중요하다. 본 논문에서는 안드로이드 앱을 개발할 때, 소프트웨어 취약점을 야기하며, 개발자가 간과하기 쉬운 소프트웨어 허점을 사전에 제거하고자 안드로이드에 특화된 시큐어 코딩 가이드를 제시한다.

1. 서론

소프트웨어가 대형화되고 복잡해지고 있으며, 이에 따라 소프트웨어 허점(weakness)의 발생률도 높아지고 있다. 이런 허점은 컴파일러에 의해 발견되지 않기 때문에 소프트웨어에 오랫동안 잠재하고 있으며, 공격자에 의해 노출되기 쉽다는 특징이 있다. 이런 소프트웨어 허점이 공격자에 의해 노출되면 소프트웨어 취약점을 야기한다. 예를 들면, 전자정부서비스에서 발생한 사고가 있다[1]. 이 사고로 인해 수많은 개인정보들이 유출되었으며 경제적으로 손실이 컸다. 이를 예방하기 위해서는 공격을 받을 수 있는 소프트웨어 허점을 사전에 제거해야 한다.

개발 후, 테스팅을 통해 소프트웨어 허점들을 탐지하기 위해서는 침투테스팅을 해야 한다. 이 단계에서 발견된 허점을 수정하기 위해서는 많은 시간과 비용이 소비되며, 테스팅을 통해 모든 허점을 탐지할 수 없다는 문제점이 있다. 또한 이와 같은 소프트웨어 허점은 개발초기에 발견할수록 소프트웨어 개발 전체에 소비되는 시간과 비용측면에서 유리하다. 따라서 이런 허점들은 개발단계에서 사전에 제거되어야 한다.

소프트웨어 허점을 개발단계에서 제거하기 위한 다양한 기법들이 존재한다. 그 중에 하나는 시큐어 코딩(Secure Coding)이다. 시큐어 코딩은 악용 가능한 소프트웨어 취약점들의 주요 원인인 소프트웨어 허점을 쉽게 피하기 위해 고안된 기법이다[2]. 시큐어 코딩은 소프트웨어 허점을 제거하기 위한 다양한 규칙과 가이드를 포함한다. 이런 규칙 및 가이드는 개발자가 간과하기 쉽고, 공격자가 발견하기 쉬운 소프트웨어 허점들로 구성되어 있다.

스마트폰의 확산으로 인해 다양한 종류의 스마트폰 앱이 개발되고 있으며, 앱이 대형화되고 복잡해지고 있다. 애플 앱스토어와는 달리 구글 안드로이드 마켓의 경우, 개발된 앱은 적절한 검수과정 없이 마켓에 업로드 될 수 있다[3]. 따라서 개발된 앱은 개발자가 간과하기 쉽고, 소프트웨어 취약점을 야기하는 소프트웨어 허점이 존재할 가능성이 크다. 본 논문에서는 안드로이드에 특화된 시큐어 코딩 가이드를 연구하여 안드로이드 앱을 개발할 때, 개발자가 간과하기 쉬운 소프트웨어 허점을 초기에 제거하여 소프트웨어의 의도하지 않은 동작 및 소프트웨어 취약점을 예방하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 가장 위험한 프로그래밍 오류인 Top25, 각 프로그래밍 언어별로 시큐어 코딩 표준을 제공하는 CERT, 그리고 안드로이드 코드 스타일 가이드라인을 간략히 소개하고, 3장에서는 본 논문에서 제시하는 안드로이드 앱 시큐어 코딩 가이드 구성 및 이 가이드를 정당화 방법을 소개하며, 그리고 4장에서는 안드로이드 앱 시큐어 코딩 가이드를 제시한다. 마지막으로 5장에서는 결론 및 향후연구로 맺는다.

본 연구는 교육과학기술부/한국과학재단 우수연구센터 육성사업(R11-2008-007-03002-0)과 정보통신산업진흥원의 SW공학 요소기술 연구개발사업의 결과물임을 밝힙니다

2. 관련연구

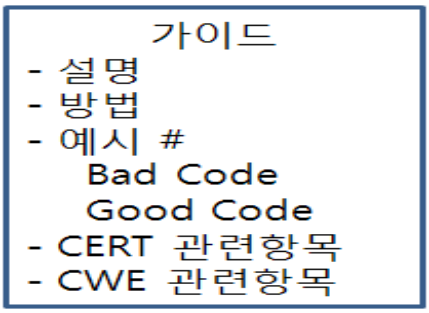
CWE/SANS는 "가장 위험한 프로그래밍 오류 Top25"[4]라는 타이틀로 2009년에 프로젝트를 시작하였으며, 현재에도 지속적으로 연구가 진행되고 있다. 여기에 포함된 25개의 오류는 가장 빈번히 발생하고, 반드시 예방되어야 하는 오류들로 구성되어 있다. 2009년 버전에는 Top25에 대해 순위가 부여되지 않았지만 2010년 버전에는 1위부터 25위까지 순위가 부여되었다. 이 순위는 보안 전문가 및 28개의 전문기관의 투표를 통해 생성되었으며, 소프트웨어 오류를 사전에 제거하기 위한 목적으로 연구가 진행되고 있다.

CERT는 프로그래밍 언어별로 시큐어 코딩 표준을 제공한다. 여기에 포함된 언어는 C, C++, 그리고 Java가 있다. 각 언어에 맞는 시큐어 코딩 표준을 적용하여 코드를 개발하면 다양한 소프트웨어 오류를 예방할 수 있다. 이는 소프트웨어 보안을 향상시키고, 국제적인 표준을 개발하기 위한 목적으로 지속적인 연구가 진행되고 있다.

안드로이드 코드스타일 가이드라인[5]은 자바 프로그래밍 언어를 기본으로 하여, 자바 언어규칙, 자바 라이브러리규칙, 자바 스타일규칙, 자바 테스트규칙으로 구성되어 있다. 이 가이드라인은 앞으로 개발되는 모든 안드로이드 코드가 여기에 포함된 가이드라인을 준수하여 개발되기 위한 목적으로 설립되었다.

3. 안드로이드 앱 시큐어 코딩 가이드 구성

안드로이드 앱 시큐어 코딩 가이드의 각 항목은 다음 (그림 1)과 같이 구성되어 있다.



(그림 1) 항목별 안드로이드 앱 시큐어 코딩 가이드 구성

가이드에는 개발자가 준수해야 하는 가이드에 대한 간략한 내용이 들어가며, 설명에는 개발자의 이해를 한층 더 돕기 위해 해당 가이드에 대한 구체적인 설명이 들어간다. 방법에는 해당 가이드를 준수하기 위한 수행해야 하는 방법이 들어가며, 예시에는 본 가이드를 지키지 않아 소프트웨어 허점을 유발할 수 있는 나쁜 코드(Bad Code), 그리고 이를 올바르게 고친 좋은 코드(Good Code)로 구성된다. 마지막으로 CERT 관련항목 및 CWE 관련항목에는 해당 연구기관이 제시하는 항목과 본 논문에서 제시하는 항목과 연관성이 있는 항목이 들어간다.

3.1 가이드 정당화 방법

본 논문에서 제시하는 시큐어 코딩 가이드가 정말 유용하고, 개발자로 하여금 반드시 지켜야하도록 정당화 시킬 수 있는 근거자료가 있어야 한다. 이런 근거자료가 없다면 본 논문에서 제시하는 가이드는 많은 개발자들에 의해 무시당할 것이고, 이를 준수하지 않는 대다수의 개발자가 발생하기 때문이다.

본 논문에서 제시한 안드로이드 앱 시큐어 코딩 가이드 항목에 포함된 “CERT 관련항목”과 “CWE 관련항목”은 가이드가 유용하고 반드시 지켜야하도록 정당화시키기 위한 목적으로 설립된 항목이다. 이 항목들에는 본 논문에서 제시한 가이드와 각 기관에서 제시한 항목들 사이의 연관성을 근거로 하여 가이드를 정당화시켰다.

4. 안드로이드 앱 시큐어 코딩 가이드

본 논문에서 제시하는 안드로이드 앱 시큐어 코딩 가이드는 안드로이드 앱을 개발할 때, 소프트웨어 취약점을 야기하는 소프트웨어 허점(weakness)을 개발단계에서 제거하기 위한 목적으로 설립되었다.

본 논문에서 제시하는 가이드에는 타입 캐스팅, 문자열 비교, 스레드, 그리고 배열첨자에 관련된 내용의 가이드로 구성되어 있다.

4.1 [가이드] 정수형 또는 실수형에서 범위가 작은 타입으로 명시적인 캐스팅을 할 때, 반드시 범위 검사를 하라.

- 설명

정수형 또는 실수형을 갖는 타입에서 범위가 작은 타입으로 명시적인 캐스팅을 할 때, 캐스팅할 값의 범위검사를 하지 않고, 이 값이 캐스팅될 타입보다 큰 범위를 가지면 의도하지 않은 값을 가지게 되며 이로 인해 허점을 야기한다.

- 방법

정수형 또는 실수형에서 범위가 낮은 타입으로 명시적인 캐스팅을 할 경우, 범위검사를 통해 캐스팅될 값이 캐스팅될 타입의 범위 내에 존재한다면 캐스팅을 하라.

- 예시 #1

다음은 정수형과 실수형 타입으로 캐스팅시 반드시 범

위검사를 해야 하는 경우이다. 캐스팅시, 다음에 제시된 경우에 해당한다면 반드시 범위검사를 해야 한다.

1. short에서 byte 또는 char로 명시적인 캐스팅을 할 경우
2. char에서 byte 또는 short로 명시적인 캐스팅을 할 경우
3. int에서 byte, short, 또는 char로 명시적인 캐스팅을 할 경우
4. long에서 byte, short, char, 또는 int로 명시적인 캐스팅을 할 경우
5. float에서 byte, short, char, int 또는 long으로 명시적인 캐스팅을 할 경우
6. double에서 byte, short, char, int, long 또는 float로 명시적인 캐스팅을 할 경우

- 예시 #2

다음은 int형을 byte형으로 명시적인 캐스팅을 하는 코드이다. 이 코드에서 범위검사를 하지 않기 때문에 numByte변수는 의도하지 않은 값을 할당 받게 된다. 이 문제는 적절한 범위검사를 통해 해결할 수 있다.

// Bad Code

```
int numInt = 128;
byte numByte = (cast)numInt;
```

// Good Code

```
int numInt = 128;
byte numByte;
if((Byte.MIN_VALUE <= numInt) && (numInt <= Byte.MAX_VALUE)) {
  numByte = (cast)numInt;
}
else {
  // The value is out of range
  numByte = 1; // default setting..
}
```

- CERT 관련항목

INT01-J. Check ranges before casting integers to narrower types

INT31-C. Ensure that integer conversions do not result in lost or misinterpreted data

INT31-CPP. Ensure that integer conversions do not result in lost or misinterpreted data

FLP03-J. Range check before casting floating point numbers to narrower types

FLP34-C. Ensure that floating point conversions are within range of the new type

FLP34-CPP. Ensure that floating point conversions are within range of the new type

- CWE 관련항목

CWE-197: Numeric Truncation Error

4.2 [가이드] '=='과 '!='을 문자열 비교에 사용하지 말라.

- 설명

'==' 연산자와 '!=' 연산자를 문자열 비교에 사용할 경우, 올바른 결과를 얻지 못한다. 이 연산자들로 인해 얻어지는 결과는 문자열 객체가 저장된 주소를 비교한 결과가 된다. 만약 이를 인지하지 못하고 이를 문자열 비교에 사용하면 예상치 못한 결과로 인해 프로그램의 허점을 야기한다.

- 방법

문자열 비교를 위한 연산을 수행할 때, 문자열 비교를 위해 제공하는 메서드를 사용하라.

- 예시 #1

다음은 '==' 연산자를 사용하여 문자열을 비교하는 코드이다. 이 코드에 사용된 문자열 변수들은 "formal"이라는 동일한 문자열을 저장하고 있기 때문에 if(..)문이 "true"가 될 것이라고 생각할 것이다. 하지만 "==" 연산은 실제 문자열에 저장된 값을 비교하는 것이 아니라 이를 저장하고 있는 주소 값을 비교한다. 따라서 if(..)문은 "false"가 되어 올바른 결과를 얻지 못한다.

// Bad Code

```
String mString1 = new String("formal");
String mString2 = new String("formal");
if(mString1 == mString2) {
        // do something
}
```

// Good Code

```
String mString1 = new String("formal");
String mString2 = new String("formal");
if(mString1.equals(mString2)) {
        // do something
}
```

- CERT 관련항목

EXP01-J. Avoid comparing objects using reference equality operators

- CWE 관련항목

CWE-597: Use of Wrong Operator in String Comparison

4.3 [가이드] 스레드를 실행할 때, Thread.run()메서드를 호출하지 말라.

- 설명

스레드를 구현하기 위해서는 Thread 클래스를 확장하거나 Runnable 인터페이스를 구현해야하며, run()메서드를 재정의 함으로써 구현한다. 구현한 스레드를 실행하기 위해 Thread.run()메서드를 직접적으로 호출하는 것은 올바른 방법이 아니다. 이는 단지 스레드를 정의한 클래스에 있는 run()메서드를 호출하는 것이기 때문에 새로 생성한 스레드가 실행되지 않는다. 반면에 Thread.start()메서드를 호출하여 스레드를 실행하면, 실제 스레드가 생성되고, 생성된 스레드의 run()메서드가 실행되어 올바르게 스레드를 실행할 수 있다.

- 방법

스레드의 Thread.start()메서드를 호출하여 스레드를 시작하라.

- 예시 #1

다음은 Activity클래스의 onCreate()메서드에서 Thread.run()을 호출하여 스레드를 실행하는 코드이다. Thread.run()은 스레드를 실행하는데 올바른 방법이 아니며, 올바른 스레드를 실행을 위해 Thread.start()를 호출하라.

// Bad Code

```
class MyThread implements Runnable {
  ...
  @Override
  public void run() {
    // do something
  }
}
...
private MyThread mThread = null;
@Override
public void onCreate(Bundle savedInstanceState) {
  ...
  private MyThread mThread = new MyThread();
  new Thread(mThread).run();
  ...
}
```

// Good Code

```
@Override
public void onCreate(Bundle savedInstanceState) {
  ...
  private MyThread mThread = new MyThread();
  new Thread(mThread).start();
  ...
}
```

- CERT 관련항목

THI02-J. Do not invoke Thread.run()

- CWE 관련항목

CWE-572: Call to Thread run() instead of start()

4.4 [가이드] 할당되지 않은 배열첨자를 사용하지 말라.

- 설명

프로그램에서 배열을 사용할 때, 첨자는 0부터 시작한다. 하지만 1부터 시작될 거라 생각하는 개발자는 배열첨자를 1부터 시작하여, 마지막 배열첨자로 할당되지 않은 메모리를 참조하는 문제를 야기한다.

- 방법

배열 사용 시, 배열첨자는 0부터 사용하라.

- 예시 #1

다음은 크기 10을 가진 배열에 1~10까지 값을 할당하는 예이다. 실제로 배열첨자를 0~9까지 사용해야 하지만 이를 인지하지 못해 배열첨자를 1~10까지 사용한다면 배열첨자 10은 메모리 할당이 되지 않았기 때문에 문제를 야기한다.

// Bad Code

```
int [] array = new array[10];
array[1] = 1;
...
array[10] = 10;
```

// Good Code

```
int [] array = new array[10];
array[0] = 1;
...
array[9] = 10;
```

- CWE 관련항목

CWE-129: Improper Validation of Array Index

5. 결론 및 향후연구

본 논문에서는 안드로이드 앱 시큐어 코딩 가이드의 일부를 제시하였다. 각 가이드는 항목별로 가이드, 설명, 방법, 그리고 예시를 포함하고 있으며, CERT 관련항목과 CWE 관련항목을 통해 가이드를 정당화시켰다.

본 논문에서 제시한 안드로이드 앱 시큐어 코딩 가이드를 적용하여 안드로이드 앱을 개발하면, 타입 캐스팅, 문자열 비교, 스레드, 그리고 배열첨자에 관련되어 발생할 수 있는 소프트웨어 허점(weakness)을 예방할 수 있다. 하지만 새로운 소프트웨어 허점이 발생할 수 있기 때문에 이를 예방하기 위한 지속적인 연구가 되어야 한다.

향후연구로는 본 논문에서 제시한 가이드를 개발자로 하여금 가이드를 반드시 준수하도록, 이를 검수할 수 있는 도구를 개발하고자 한다.

참고문헌

[1] 전자신문, “행안부, 보안강화에 대응비용 75% 절감”, http://www.etnews.co.kr/news/detail.html?id=201001280037, 2010-02-01

[2] CERT, “Secure Coding”,http://www.cert.org/secure-coding/. Last updated February 18, 2010

[3] Android Market, http://www.android.com/market/#app=com.com2us.HG

[4] CWE/SANS, “2010 CWE/SANS Top 25 Most Dangerous Programming Errors”, http://cwe.mitre.org/top25/index.html

[5] Android open source project, "Code Style Guidelines for Contributors", http://source.android.com/source/code-style.html