임베디드 소프트웨어에서 발생하는 정수 확장 (Integer Promotion) 버그 검출을 위한 정적 분석 기법

김태진 O1 , 김문주 1, 이호정 2, 장훈 2, 박민규 2

1 한국과학기술원, 2LG 전자

taejin@kaist.ac.kr, moonzoo@cs.kaist.ac.kr, sophiahj.lee@lge.com, bill.jang@lge.com, mingyu88.park@lge.com

Detecting Integer Promotion Bugs with Embedded Software using Static Analysis Technique

Taejin KimO1, Moonzoo Kim1, Ho-Jung Lee2, Hoon Jang2, Mingyu Park2

1Korea Advanced Institute of Science and Technology, 2LG Electronics

요 약

양산형 가전제품용 소프트웨어에서의 정수 값 변환은 표준 ANSI C를 사용하는 소프트웨어에서의 변환 방식과 달라서 개발자들이 쉽게 인지하지 못하고 디버깅도 어렵다. 본 논문에서는 표준 ANSI C의 변환 방식과 달라 8-bit MCU에 사용되는 소프트웨어에서 발생하는 버그인 정수 확장 버그를 소개하고, 정수 확장 버그가 발생하는 구문에 대해 5가지 버그 패턴을 정의하였다. 또한 이 패턴들을 검출하는 자동화 도구를 사용하여 거짓 경보 수를 감소시키기 위해 정적 범위 분석 기법을 적용하였다. 정수 확장 버그 패턴을 이용해 LG전자 세탁기 소프트웨어에서 버그를 검출한 결과 컴파일러 옵션을 잘못 선택한 경우 발생하는 27개의 정수 확장 버그를 발견하였고, 정적 범위 분석 기법으로 미탐지 버그 없이 73개의 거짓 경보 중 15개 (20.5%)를 제거함을 확인하였다.

1. 서 론

우리는 임베디드 시스템을 사용하는 제품을 일상생활의 곳곳에서 마주하고 있다. 이러한 임베디드 시스템의 세계 시장은 약 200조원에 육박하고 매년 9% 이상의 성장률을 보인다 [1]. 이러한 급격한 성장에 따라, 임베디드 소프트웨어의 품질을 향상시키는 것 또한 중요해지고 있다.

이러한 임베디드 시스템 시장에서 8-bit MCU (Micro Controller Unit)의 시장 규모는 전체 MCU 시장의 약 40%를 차지할 정도로 거대하다 [2]. 이러한 8-bit MCU에 특화된 소프트웨어에서만 발생하는 버그는 특정 시스템 내에서만 발생하므로 개발자들이 인지하기 어려움에도 불구하고, 이 버그를 검출하는 방법에 대한 연구는 부족한 실정이다.

8-bit MCU에 특화된 소프트웨어에서 발생하는 버그 중에는 대표적으로 정수 확장 방식이 표준 ANSI C와 달라 발생하는 정수 확장 버그가 있다. 정수 확장 (integer promotion)이란 피연산자 타입의 크기가 정수형 (32-bit integer)보다 작은 경우 중 8-bit인 피연산자의 타입을 암묵적으로 정수형으로 변환하는 것이다. 정수 확장을 기본적으로 수행하는 표준 ANSI C 컴파일러와는 달리 일부 8-bit MCU 컴파일러는 소프트웨어의 성능 향상을 위해 기본적으로 정수 확장을 수행하지 않는다. 정수 확장을 수행하지 않는 경우, 개발자들이 의도하지 않았던 오버플로우 등의 문제가 발생하게 된다. 개발자들은 일반적으로 표준 ANSI C를 따르는 PC에서 개발하기 때문에 이러한 정수 확장 규칙이 있다는 사실을 잘 알지 못하는데, 이에 따른 버그 발생 시 디버깅하는데도 많은 시간이 필요하다. LG전자의 한 개발부서 내부에서 자체 조사한 결과, 8-bit MCU와 관련된 개발자들 중 약 66%가 정수 확장 규칙을 잘 모르거나 전혀 모른다고 답하였다.

본 논문에서는 코드의 구조 조건 (syntactic condition)과 피연산자의 값 조건 (semantic condition)으로 5가지 버그 패턴을 정의하고, 정수 확장 버그를 검출하고자 한다. 정수 확장 버그는 코드의 각 구문이 우리가 정의한 5가지 버그 패턴의 조건들과 일치하는지 여부를 정적으

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학 ICT연구센터육성 지원사업(IITP-2015-H8501-15-1012)과 LG전자 H&A 어플라이언스 연구소의 지원으로 수행되었음.

```
1  void pattern3(){
2      unsigned char op1 = 200;
3      unsigned char op2 = 200;
4      int result = op1 + op2; }
5  void pattern4(){
6      signed char op1 = -1;
7      unsigned char op2 = 1;
8      if (op1 < op2) { /* ANSI C SW */ }
9      else { /* 8-bit MCU SW */ } }
```

그림 1 정수 확장 버그의 코드 예시

로 확인하여 검출 할 수 있다.

하지만 정적 패턴만으로는 정보가 부족하여 피연산자가 변수인 경우 검출 과정에서 값 조건을 알 수 없으므로 거짓 경보가 발생한다. 검출된 경보들 중 거짓 경보들을 제외하는 작업을 개발자들이 직접 하면 많은 시간이 소요되므로 거짓 경보를 줄이는 것이 중요하다. 따라서 변수들이 각 코드의 위치마다 가질 수 있는 값의 범위를 알려주는 정적 범위 분석 (static range analysis) [3] 기법을 추가 적용하여 정수 확장 버그 패턴으로 버그를 검출하면서 발생하는 거짓 경보를 줄인다.

본 연구에서는 Clang과 LLVM [4]을 이용하여 정수 확장 버그 패턴 검출 기법과 정적 범위 분석 기법을 자동화한 정수 확장 버그 검출기를 만들어 개발자가 거짓 경보를 분석하는 시간을 줄이며 정수 확장 버그를 찾도록 하였다. 사례 연구에서는 이 검출기를 LG전자의 세탁기 소프트웨어 (약 41KLOC)에 적용하였다. 그 결과 컴파일러 옵션을 잘못 선택 한 경우 발생하는 27개의 정수 확장 버그를 발견하였고, 정적 범위 분석 기법으로 미탐지 버그 없이 73개의 거짓 경보 중 15개 (20.5%)의 거짓 경보를 제거하였다.

2. 정수 확장 (Integer Promotion) 버그 패턴 정의

이 장에서는 8-bit MCU에서 발생할 수 있는 정수 확장 버그를 설명하고 정수 확장 버그를 5가지 패턴으로 정의한다. 정수 확장 버그는 표준 ANSI C와 정수 확장 방식이 다른 소프트웨어에서만 발생하는 버그를 말한다. 표준 ANSI C를 따르는 소프트웨어에서는 피연산자가 문자형 (8-bit character)인 경우, 정수형으로 타입을 확장 (promotion) 한 뒤 연산을 수행한다. 반면 8-bit MCU에 사용되는 소프트웨어에서는 정수 확장 없이 피연산자의 타입 그대로 연산을 수행한다. 또한 이항연산자와 사용되는 피연산자들의 타입이 각각 부호없는 (unsigned) 타입과 부호있는 (signed) 타입인 경우, 피연산자의 타입 일치를 위해 부호있는 타입의 피연산자는 강제로 부호없는 타입의 피연산자로 타입 변환 (coercion)된다.

예를 들어, 그림 1 `pattern3()` 함수 4번째 줄의 `result` 값은 표준 ANSI C를 사용하는 소프트웨어와 8-bit MCU의 소프트웨어에서 다르게 나타난다. 표준 ANSI C 소프트웨어는 부호없는 문자형인 `op1`, `op2`의 값을 덧셈하기 전에 `op1`, `op2`를 정수형으로 타입 확장하여 계산하므로 `op1 + op2`가 정수형으로 계산되고 `result`에 400이 들어간다. 반면, 8-bit MCU의 소프트웨어는 `op1`, `op2`의 타입인 부호없는 문자형 그대로 덧셈 연산을 하고, `op1 + op2`가 부호없는 문자형의 최댓값인 255를 초과하면서 오버플로우가 발생한다. 따라서 `result`에는 144가 들어간다.

표 1 정수 확장 버그 패턴 정의

	코드 구조 조건		피연산자 값 조건
	연산자 형태	피연산자 타입	
1	~p	uchar이고 연산 결과를 정수형에 할당	모든 값에 대해 발생
2	-p	schar 또는 uchar	schar이고 p==-128 이거나, uchar이고 p!=0
3	p+q, p*q, p-q, p<<q	p와 q는 schar 또는 uchar	연산 결과가 schar/uchar의 범위를 벗어나는 경우
4	p/q, p%q, p|q, p^q, p==q, p!=q, p<q, p<=q, p>q, p>=q	p가 schar이고 q가 uchar이거나, p가 uchar이고 q가 schar	p가 schar이고 p<0 이거나, q가 schar이고 q<0
5	x ? p : q		

또한, 그림 1의 7번째 줄 조건문의 참 여부도 정수 확장 여부에 따라 달라진다. ANSI C 소프트웨어에서는 참인 분기문을 수행하지만, 8-bit MCU 소프트웨어는 부호있는 문자형인 `op1`을 조건문 확인 시 부호없는 문자형으로 타입 변환하고, 그 값은 255가 되어 거짓인 분기문을 수행하게 된다.

표 1은 위 예시에서 설명한 정수 확장 버그가 발생하는 구문을 코드 구조 조건과 피연산자의 값 조건에 따라 5가지 패턴으로 나누어 정의한 것이다. 1, 2번 패턴은 단항연산자, 3, 4번 패턴은 이항연산자, 5번 패턴은 삼항연산자이다. 표1에서 uchar는 unsigned char 타입을 의미하고, schar는 signed char 타입을 의미한다. 코드의 구문이 연산자 형태, 피연산자 타입 그리고 피연산자의 값 조건이 모두 일치하면 해당 구문은 정수 확장 버그가 발생한 것이다.

표 1의 정의를 토대로 코드 구조 조건과 피연산자 값 조건을 만족하는 소프트웨어 코드의 각 구문을 구문의 정보와 함께 경보로 띄우는 자동 검출기를 구현하였다.

```
1  int main(){
2      unsigned char x = rand();
3      unsigned char y = rand();
4      if(x<10 && y<20) { foo(x, y);} }
5  void foo(unsigned char x,
             unsigned char y){
6      int a = x + y; }
```

그림 2 정적 범위 분석 기법 적용을 위한 코드 예시

3. 거짓 경보 감소를 위한 정적 범위 분석 기법

앞서 정의한 정수 확장 버그 패턴을 이용해 경보 검출 시 피연산자가 변수인 경우 피연산자의 값 조건을 알 수 없어 해당하는 모든 구문을 검출하고, 따라서 많은 거짓 경보가 발생하는 문제가 있다. 이 문제를 해결하기 위해 정적 범위 분석 (static range analysis) [3] 기법을 적용한다.

정적 범위 분석 기법은 프로그램 코드의 데이터 흐름을 이용해 변수들이 각 코드의 위치에서 가질 수 있는 값의 범위를 정적으로 분석한 것으로, 건전성 (soundness)이 증명되어 있다. 변수들 값의 범위를 정적으로 알 수 있으므로 정수 확장 버그 패턴이 검출된 위치에서 피연산자 값 조건의 만족 여부를 알 수 있고, 피연산자 값의 범위로 연산한 결과가 값 조건을 만족하지 않으면 거짓 경보로 판단하여 제거한다. 또한, 함수 간 (inter procedural) 범위 분석 기법을 사용하기 때문에, 각 함수에서 사용하는 매개변수 값의 범위도 알 수 있다.

그림 2의 코드에 정적 범위 분석 기법을 적용하면 `x`와 `y` 변수가 사용되는 위치에서 그 값의 범위를 알 수 있다. 4번째 줄의 조건문에 의해 `x` 값의 범위는 [0, 9]가 되고, `y` 값의 범위는 [0, 19]임을 알 수 있다. 또한 `x`, `y` 값이 `foo` 함수의 매개 변수로 사용되면서, 6번째 줄의 `x + y` 의 연산 결과 값 범위는 [0, 28]이 된다. 이 연산 결과 값은 255를 초과하지 않기 때문에 해당 버그 패턴은 정수 확장 버그가 아니라고 판단하고, 이 거짓 경보를 검출하지 않는다.

4. 사례 연구

사례 연구에서 사용한 대상은 LG 전자에서 8-bit MCU를 사용하는 세탁기 프로그램이다. 이 모델은 약 41KLOC의 크기에 약 1.4K 개의 함수를 가진 C 프로그램이다. 실험은 Intel Core i7-2600K 3.4GHz 프로세서와 32GB RAM이 장착되어 있고, Debian 8.2 64 bit OS가 설치된 워크 스테이션에서 수행하였다.

표 2는 정수 확장 버그 패턴과 정적 범위 분석 기법을 자동화한 검출기를 이 세탁기 소프트웨어에 적용한 결과이다. 정수 확장 버그 패턴을 소프트웨어에 적용하였을 때 1.55초의 수행 시간으로 100개의 경보가 검출되는 것을 확인 할 수 있었고, 여기에 정적 범위 분석 기법을 추가 적용하여 24.58초의 수행 시간으로 미탐지 버그 없이 거짓 경보 73개 중 15개 (20.5%)의 거짓 경보를 제거하였다. 남은 85개의 경보 중 27개의 경보는 컴파일러 옵션에 따라 실제 버그임을 LG 전자 개발팀과 대상 코드를 분석하여 확인하였다.

표 2 정수 확장 버그 검출기를 통해 세탁기 소프트웨어에서 검출된 경보 분석

	정수 확장 패턴 적용	정적 범위 분석 기법 적용
버그인 경보의 수	27	27 (-00.0%)
거짓 경보의 수	73	58 (-20.5%)
전체 경보의 수	100	85 (-15.0%)

5. 결론 및 향후 연구

본 논문에서는 8-bit 양산형 임베디드 소프트웨어에서 발생하는 정수 확장 버그를 5가지 패턴으로 나누어 정의하고, 정적 범위 분석 기법을 활용하는 방법을 소개하고 자동화된 정수 확장 버그 검출기를 개발하였다. LG전자 세탁기 소프트웨어에 이 검출기를 적용하면서 정적 범위 분석 모듈로 거짓 경보 수를 20.5% 감소시켰고, 옵션을 잘못 설정하는 경우 발생하는 27개의 정수 확장 버그를 발견하였다.

이 결과를 통해 정적 범위 분석 기법으로 정수 확장 버그 검출 시 거짓 경보를 효과적으로 줄일 수 있음을 확인하였다. 그러나 동시에 배열 값 참조나 포인터의 사용 등으로 인해 정적 분석 기법으로 제거하지 못하는 거짓 경보들이 여전히 존재 함을 알 수 있었다. 따라서 향후 Concolic 테스팅 기법 [5] 등의 동적 분석 기법을 추가 적용하여 정적 분석 기법의 한계점들을 극복하고, 더 효과적으로 거짓 경보를 제거하는 연구를 진행 할 예정이다.

참고 문헌

[1] Ebert et al. "Embedded software: Facts, figures, and future", Computer, 4: 42-52, 2009.

[2] Fredriksen et al. “Choosing a MCU for your next design; 8 bit or 32 bit?”, Atmel Corp., 2014.

[3] Rodrigues et al. "A fast and low-overhead technique to secure programs against integer overflows", CGO, 2013.

[4] Clang, LLVM. http://llvm.org

[5] Sen et al. “CUTE: A Concolic Unit Testing Engine for C”, ESEC/FSE, 2005.