중간언어 생성을 위한 바이너리 코드 자료형 및 변수 추론 기술 조사 분석

민예슬*, 정현오*, 손윤식*, 정준호**, 고광만***, 오세만*
*동국대학교 컴퓨터공학과
**동국대학교 경주캠퍼스 전자성거래 연구소
***상지대학교 컴퓨터정보공학부
e-mail: smoh@dongguk.edu

A Survey of the Scheme of Data Type and Variables Inference for Intermediate Language Generation from Binary Code

Ye Sul Min*, Hyunoh Jung*, Yunsik Son*, Junho Jeong**, Kangman Ko***, Seman On*
*Dept of Computer Engineering, Dongguk University
**Electronic Commerce Institute, Dongguk University Gyeongju Campus
***Dept. of Computer and Information Engineering, SangJi University

요 약

소프트웨어 내제되어 있는 보안약점과 보안취약점으로 인해 사회적으로 많이 비용이 발생함에 따라 안전한 소프트웨어를 개발하고자하는 요구가 증가하고 있다. 최근 바이너리 코드에 내제된 보안약점을 분석하기 위해서 중간코드를 이용하여 정적분석을 수행하는 다양한 연구가 진행되고 있다. 중간언어를 사용함으로 실행환경에 따라 달라지는 바이너리 코드가 중간언어로만 변환이 된다면 동일한 형태의 보안약점 분석기술을 통해 효과적인 수행이 가능하다. 이 기술의 핵심은 바이너리 코드로부터 얼마나 코드내의 자료형 및 변수를 재구성하여 중간언어로 변환하는 것이다. 본 논문에서는 이와 같은 바이너리 코드로부터 보안약점 분석을 위한 중간언어 변환시 효과적으로 자료형 및 변수 등에 관한 정보를 재구성하는 기법들에 대해서 조사 분석하였다.

1. 서론

소프트웨어 내제되어 있는 보안약점과 보안취약점으로 인해 사회적으로 많이 비용이 발생함에 따라 안전한 소프트웨어를 개발하고자하는 요구가 증가하고 있다. 그래서 개발 단계에서부터 보안약점이 발생시킬 수 있는 요소들은 제거를 하는 시큐어 코딩 기법부터 소스코드를 분석하여 존재하는 보안약점 및 취약점을 분석하는 방법들을 적용하여 보다 안전한 소프트웨어를 개발하기 위한 연구가 다양하게 진행되었다.[1-3]

그러나 최근 들어 개발되는 소프트웨어의 대부분은 요구되는 기능과 기술이 다양해지면서 소프트웨어의 생산성 향상을 위해서 기 개발된 다양한 라이브러리들이 사용된다. 이 같은 라이브러리 중에는 라이브러리의 소스가 제공되는 경우도 있고 그렇지 않은 경우도 존재한다. 소스가 제공이 되는 경우는 소스코드를 분석하여 보안약점 등을 제거하여 사용하면 되나 소스코드가 없는 바이너리 코드 형태로 제공되는 라이브러리의 경우는 일반적인 보안약점 분석으로 이 문제를 쉽게 해결하기 어렵다.

그러므로 이와 같은 바이너리 코드로부터 보안약점을 분석하고 해결하는 것은 매우 중요하다. 일반적으로 바이너리 코드에서 보안약점 및 취약점을 분석할때는 동적분석을 통해 진행된다. 그러나 이러한 동적분석은 분석자의 역량에 따라 그 결과 좌우되는 단점이 있다.

또한 컴파일이 모두 완료된 바이너리 코드에서는 보안약점이 될 수 있는 여러 가지 자료구조와 객체에 대한 정보가 손실된다. 이러한 이유로 바이너리 코드만으로는 보안약점을 파악하기 힘들다. 따라서 바이너리 코드보다 많은 정보를 가지고 있는 중간언어로 복원하여 보안약점 분석이 가능하다.

최근 바이너리 코드에 내제된 보안약점을 분석하기 위해서 중간코드를 이용하여 정적분석을 수행하는 다양한 연구가 진행되고 있다. 중간언어를 사용함으로 실행환경에 따라 달라지는 바이너리 코드가 중간언어로만 변환이 된다면 동일한 형태의 보안약점 분석기술을 통해 효과적인 수행이 가능하다. 이 기술의 핵심은 바이너리 코드로부터 얼마나 코드내의 자료형 및 변수를 재구성하여 중간언어로 변환하는 것이다.

본 논문에서는 이와 같은 바이너리 코드로부터 보안약점 분석을 위한 중간언어로 변환시 자료형 및 변수를 재구성하는 기법들에 대해서 조사 분석하였다. 2 장에서 이를 위한 바이너리 코드 역공학 기술에 대해서 간략히 소개하며

"본 연구는 방위사업청과 국방과학연구소 (계약번호 UD160035ED)의 연구비 지원에 의한 연구 결과임"

3 장에서 중간언어 위한 자료형 및 변수 추론 대해서 살펴본다. 마지막으로 4 장에서 분석에 대한 결론을 맺는다.

2. 바이너리 코드 역공학

바이너리 코드로부터 자료형, 함수원형, 객체 등의 정보를 파악하기 위한 일반적인 방법은 실행코드를 디스어셈블하여 어셈블리를 추출한다. 그리고 이로부터 컴파일 과정에서 손실된 자료형, 변수, 함수원형, 객체 등과 같은 정보를 추론하여 복원하므로 실행코드로부터 올바른 추론을 수행하기 위해서는 프로그래밍 언어, 컴파일러, 운영체제, 타겟 아키텍처를 고려해야하는 특징이 있다.

프로그래밍 언어의 경우 구성되는 언어에 대한 정의와 프로그래머가 정의한 자료형이 존재하며 컴파일러는 자료형 표현, 자료구조, 호출관계, class간의 상속, 파일형태 객체 등을 다루기 위해 적합한 ABI(Application Binary Interface)를 선택하고 운영체제의 경우 표현에 영향을 준다. 예를 들어 윈도우의 경우 문자열을 UTF-16으로 변환하는 것이나 타겟 아키텍처에 따라 빅 엔디안과 리틀 엔디안 또는 정수형과 포인터의 크기와 같은 부분들이 데이터 표현에 영향을 미치는 것이다.

3. 바이너리 코드 자료형 및 변수 추론 기술

가. Mycroft

Mycroft는 타입 추론 기술을 통해 타겟 머신 코드로 부터 C프로그램을 디컴파일하는 시스템 제안하며 특정 아키텍처로부터 독립된 형태로 만들기 위해서 RTL (Register Transfer Language) 형태이며 같은 타겟 코드는 SSA (Single Static Assignment) 형태로 변형하는 방법이다.[4]

저급언어(RTL)와 고급언어(C)를 대상으로 하며 RTL은 다양한 머신 코드 표현을 아키텍처에 독립적인 요소로 만들어 주는 장점이 있으나 단지 벡터와 포인터만으로 표현되는 것에서 적절한 자료구조나 배열 타입을 발견하기 어려운 문제를 가지고 있다. 이 문제를 해결하기 위해서 해당 연구에서는 데이터로부터 코드를 합리적으로 식별할 수 있고 현재의 시스템은 프로시저 경계를 사용 가능하다고 가정하였다.

그럼에도 불구하고 해당 연구에ㅅ해당 연구에서 존재하는 세 가지 문제점이 있는데 첫 번째는 타입 충돌을 해결할 수 없는 경우가 존재한다. 타입 통일은 두 가지 이유로 실패할 수 있는데 첫 번째는 타입 변수가 재귀적인 데이터 타입의 종합일 경우, 두 번째는 타입 구성 시 충돌이 발생하는 경우이고 이 경우는 아래 그림을 예로 설명할 수 있다.

```
h:    ld.w  4[r0], r1
      xor   r1, r0, r0
      ret
```

(그림 1) 타입 구성 시 충돌 발생 코드

이 경우 r0는 xor명령어에 의해 피연산자로 int로써 간주되고, 또한 ld.w 명령어에 의해 offset이 4인 곳에 int를 포함한 포인터로 간주되기에 int와 ptr(mem(4 : int))는 통일될 수 없는 것을 알 수 있다.

두 번째는 배열 재구성 시 발생할 수 있는 문제점으로 C 프로그래머들은 배열의 크기를 지정하는데 크게 신경을 쓰지 않기에, 특히 배열 매개변수를 넘기는 경우 포인터로 맵핑되어 크기 정보를 잃게 될 수 있는 것이다.

세 번째는 배열과 구조체 재구성 시 선택해야 하는 경우가 존재하는데 배열과 구조체가 다른 배열이나 구조체를 포함하는 경우는 일반적으로 유일하게 디코딩되지 않기에 다음과 같이 각각의 객체로 정의되어 구분해야 한다.

```
struct S1 { int a; int b[4]; int c; int d[4]; } x1;
struct S2 { int a; int b[4]; } x2[2];
struct S3 { struct S2 c,d; } x3;
```

(그림 2) 구조체 내에 다른 구조체를 포함한 예제 코드

세 제약조건을 해결하기 위해 다양한 선택사항을 고려하다 보면 배열의 인덱싱과 구조체 중에 선택해야 하는 경우가 발생하므로 이 경우 추가적인 정보가 주어진다면 명령어는 완전한 정보를 가지고 하나의 결론으로 결정할 수 있으나 그러지 않을 경우 하나의 결론을 내리기 어려운 문제가 있다.

나. ObjDigger

해당 연구는 객체 인스턴스, 데이터 멤버, 일반적인 클래스의 메서드를 복원하는 것을 목적으로 하고 이를 복원하기 위해 심볼 실행과 프로시저간의 데이터 흐름 분석을 사용하여 정적 분석을 수행하여 클래스가 배치되는 방법을 이해하고 자신의 메소드을 식별하는 ObjDigger라는 도구로 구현하였다.[5]

C++ 언어에 대해서 this 포인터를 찾은 후 이를 이용하여 객체 인스턴스와 메서드 식별하며 가상 함수 테이블의 객체들은 메모리의 this 포인터의 주소로 초기화 되는 점을 이용하여 생성자 영역안에서 전형적인 mov [reg], vtableAddr과 같은 형태를 발견하면 잠재적인 가상 함수 테이블 주소로 구분하였다. 또한 이 테이블 내에 진입하게 만드는 호출을 식별하기 위해서 mov 명령어에 의존하여 실행함으로써 파악하였다.

그러나 이 연구에서는 상속에 대해서 문제 해결을 위한 분석 방법은 제안하였으나 완벽히 구현하진 못하였다.

다. 함수관련 정보 추론 기술

Balakrishnan 등은 x86환경에서 메모리 접근을 분석하는 방법을 서술하였다.[6] 이를 위해서 사용하는 기술은 데이터 개체가 가지는 포인터 값과 정수 값을 포함하는 요약 도메인을 설정한 후 정적 분석 알고리즘인 VSA를 통해

실제 값을 추적하여 프로그램의 정보를 찾는 특징이 있다. 또한 함수의 정보를 분석하기 위해 VSA를 두 가지 방법으로 나누었는데 첫 번째 방법은 함수 안에서의 분석 방법으로 함수의 제어 흐름 그래프에서 수행되며 제어 흐름 그래프는 x86의 명령어 당 하나의 노드로 이루어지고 간선 또한 명령어로 표기하는 것이다. 예를 들어, 간선이 조건 명령어인 경우, CFG가 형성되면 요약 저장 공간은 각 실행 포인트에서 요약 해석으로 얻을 수 있고 함수의 요약 표현은 재귀적으로 호출되기 때문에 각 변화기는 요약 저장 공간을 새로운 요약 저장 공간으로 변환하는 것이다.

진입 노드의 요약 저장 공간은 초기화된 전역 변수와 스택 포인터(esp)의 초기 값에 대한 정보로 구성되고, 요약 도메인은 자식노드를 만들 수 있는 공간이 유한하기 때문에 VSA가 종료되기 위해서는 확장이 필요하다.

두 번째 방법은 함수 사이에서의 분석방법이다. 함수 사이의 분석 방법은 함수의 형식과 호출의 실제에 대해 분석하는 것이다. 이 분석 방법은 실제 매개변수와 레지스터 값의 저장, 형식 매개변수, 함수의 호출과 반환으로 나눠진다.

Yoo 등은 C++ 바이너리 중에서 심볼이나 재배치 정보가 없는 바이너리에 대한 객체지향의 특징을 복원하는 방법으로 RTTI(Run-Time Type Information), 클래스 계층구조, 가상 멤버 함수와 비가상 멤버 함수, 클래스의 멤버 변수를 재구성하였다.[7]

가상 메소드는 어떤 컴파일 모델을 사용하는가에 따라 컴파일 타임에 의존성이 약간씩 다르기에 일반적으로 가상 메소드는 다음의 단계를 따르는데 첫 번째 단계로 객체 포인터 형 변환을 통해 메소드를 호출하는 경우, 변경된 객체의 관점으로 보며, 두 번째 단계로 객체의 변화된 관점이 주어진 경우, 해당 객체로부터 가상테이블의 주소를 가져오고, 세 번째 단계로 가상 테이블로부터 가상함수의 주소를 얻어 호출된 객체의 관점에서 정확한 변화 값을 획득한다.

RTTI는 모든 특징을 복원하기 위해 필요한 정보들을 모두 가지고 있으며, 바이너리에 존재하며 각 클래스를 위한 구역을 가지고 있고 각 구역은 가상 함수 테이블 포인터, 기본 클래스 포인터, 모든 하위클래스의 포인터, 일치하는 클래스를 위해 임의로 변경된 이름을 포함하는데 이는 표준 ABI에 정의된다.

해당 연구는 C++프로그램의 바이너리를 입력으로 하여 객체 지향적 특징을 찾아내는데 첫 번째 단계로 가상 함수 테이블을 복원함으로써 가상 메소드 호출 코드와 RTTI 레이아웃을 복원하고, 두 번째 단계로 컴파일러에 독립적인 직관적인 패턴으로 매칭을 통해 생성자 디스패처를 복원하며, 세 번째 단계로 멤버 함수와 멤버 변수를 복원하고, 클래스와 연결한다.

다른 플랫폼에서 작동하는 다른 컴파일러에 의한 결과일지라도 결과를 분석하기 위해 세 집합의 정규 형태를 유도했으며, 첫 번째 집합은 객체 포인터로부터 옳은 가상 테이블을 찾기 위해서이며 두 번째 집합은 가상 테이블로부터 가상 메소드의 주소를 획득하기 위한 정규 형태이고 세 번째 집합은 어떤 함수의 호출로 전달되는 this 포인터와 같은 첫 매개변수의 값을 계산하기 위한 집합이다.

해당 연구에서 제안한 방법을 통해 클래스는 100%, 멤버 함수는 78%, 멤버 변수는 55% 식별하였고 멤버 함수와 멤버 변수에 대한 검출이 100%가 되지 못한 이유는 크게 세 가지로 분석되어진다.

첫 번째 가상함수의 호출을 100% 찾아낼 수 없기에 모든 가상 함수 테이블에 배치하는 것이 어렵기 때문이며, 두 번째 C++의 템플릿내의 멤버 함수와 멤버 변수로부터 클래스를 검출하는 것을 구현하지 못했고 마지막으로 인라인 함수의 호출 지점을 발견하지 못했기 때문이다.

라. 타입 추론 기법에 따른 추론 결과 요약

주요 타입 추론 기법에 따른 최종적인 추론타입의 결과는 아래의 <표 1>과 같다. ●은 지원됨을 의미하며 ◐은 부분적인 지원, 그리고 ○은 지원되지 않음을 의미한다. 결과에서 알 수 있듯이 모든 타입을 지원하는 기법은 없는 것을 확인할 수 있으며 각 분석에 따라 지원되는 추론 타입이 상이 함을 알 수 있다.

<표 1> 타입 추론 기법에 따른 추론 타입 비교

추론 타입	Mycroft[4]	ObjDigger[5]	Yb[7]
정수	●	○	◐
실수	○	○	◐
포인터	●	◐	●
클래스 메서드	○	●	◐
클래스 계층	○	◐	◐
클래스 가상테이블	○	●	●
레코드	●	◐	●
배열	◐	○	●
공용체	◐	○	○
함수 타입	◐	○	●

4. 결론 및 향후 연구

본 논문에서는 바이너리 코드로부터 보안약점 분석을 위해서 바이너리 코드를 중간언어 재구성을 위해 자료형과 변수 등을 추론하는 기법들에 대해서 분석하였다.

분석을 통해 바이너리 코드로부터 자료형과 변수 등을 완벽히 원래의 소스코드 복원하는 것은 불가능한 일이지만 보안약점 분석을 위한 중간언어를 위해 재구성하는 것은 불가능한 일은 아닌 것을 확인 할 수 있었다.

향후 연구로써 보안약점 분석에 효과적일 수 있는 중간언어를 설정하고 바이너리 코드를 중간언어로 재구성하고 평가하고자 한다.

참고문헌

[1] 김정숙. (2013). 소프트웨어 보안을 위한 시큐어 코딩. 한국콘텐츠학회지, 11(4), 56-60.

[2] Prinz, A., Scheidgen, M., & Tveit, M. S. (2007, September). A model-based standard for SDL. In International SDL Forum (pp. 1-18). Springer Berlin Heidelberg.

[3] Bang, J., & Ha, R. (2013). Research on major weakness rules for secure software development. The Journal of Korean Institute of Communications and Information Sciences, 38(10), 831-840.

[4] Mycroft, A. (1999, March). Type-based decompilation (or program reconstruction via type reconstruction). In European Symposium on Programming (pp. 208-223). Springer Berlin Heidelberg.

[5] Jin, W., Cohen, C., Gennari, J., Hines, C., Chaki, S., Gurfinkel, A., ... & Narasimhan, P. (2014, January). Recovering c++ objects from binaries using inter-procedural data-flow analysis. In Proceedings of ACM SIGPLAN on Program Protection and Reverse Engineering Workshop 2014 (p. 1). ACM.

[6] Balakrishnan, G., & Reps, T. (2004, March). Analyzing memory accesses in x86 executables. In International conference on compiler construction (pp. 5-23). Springer Berlin Heidelberg.

[7] Yoo, K., & Barua, R. (2014, December). Recovery of object oriented features from C++ binaries. In Software Engineering Conference (APSEC), 2014 21st Asia-Pacific (Vol. 1, pp. 231-238). IEEE.