情報保護學會論文誌
第 22 卷 第 3 號, 2012. 6

단편화된 실행파일을 위한 데이터 구조 역공학 기법*

이 종 협†‡
한국교통대학교

Reverse engineering of data abstractions on fragmented binary code*

JongHyup Lee†‡
Korea National University of Transportation

요 약

실행파일에 대한 정적인 분석을 통한 역공학 과정은 소프트웨어 보안에서 필수적인 단계이다. 역공학은 크게 소프트웨어가 사용하는 데이터 구조와 제어 구조에 대하여 수행된다. 특히 데이터 역공학은 소프트웨어를 이해하기 위해 필수적이지만 기존의 VSA 기법은 단편화된 실행파일에 대한 한계를 가지고 있다. 본 논문에서는 동적인 영역할당을 통해 이러한 문제점을 해결하고 데이터 구조 역공학의 성능을 향상시킨다.

ABSTRACT

Reverse engineering via static analysis is an essential step in software security and it focuses on reconstructing code structures and data abstractions. In particular, reverse engineering of data abstractions is critical to understand software but the previous scheme, VSA, is not suitable for applying to fragmented binaries. This paper proposes an enhanced method through dynamic region assignment.

Keywords: Software security, Reverse Engineering

I. 서 론

소프트웨어 보안은 소프트웨어에 대한 분석을 통하여 해당 프로그램을 이해하는 과정으로 시작한다 〔1,6〕. 특히 개발과정이 아닌 보안이라는 측면에서의 소프트웨어 분석에서는 프로그램의 소스코드가 없는 실행파일(binary code)에 대한 분석에 초점을 둔다. 따라서 실행파일만을 가지고 주어진 실행파일이 어떠한 데이터 구조와 값을 사용하며 어떠한 작업을 수행하는지를 알아내는 역공학(reverse engineering) 과정이 필수적이다. 이러한 역공학 과정은 해당 프로그램을 주어진 환경에서 실행시켜서 그 실행과정을 분석하는 동적인 분석(dynamic analysis)과 프로그램의 실행 없이 프로그램 코드를 분석하는 정적인 분석(static analysis) 방법으로 구분된다〔1〕. 동적인 분석 방법은 수행 과정에서 실질적으로 이용되는 내부 데이터의 구조와 변화를 파악할 수 있다는 점에서 장점을 가지고 있지만, 전체 프로그램 코드 중에서 실행이 된 일부분만을 파악할 수 있고 악성코드와 같이 악의적인 행동을 하는 프로그램의 경우 잘 보호된 공간 내에서 프로그램을 실행(예, sandboxing)해야 하는 수행과정 자체의 어려움들을 가지고 있다. 이러한 이유로 정적 분석방법이 선호되지만, 정적인 분석은 프로그램의 코드만을 가지고 해당 프로그램이 사용하는 데이터 구조와 그 변화를 유추해야 하기 때문에 분석의 난이도가 높다.

접수일(2012년 4월 16일), 수정일(2012년 5월 14일), 게재확정일(2012년 5월 15일)
* 본 논문은 2009년 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임. 〔NRF-2009-352-D00282〕
† 주저자, jhlee@ut.ac.kr
‡ 교신저자, jhlee@ut.ac.kr

정적인 분석을 통한 역공학 과정은 프로그램의 데이터 구조에 대한 분석에서 시작한다. 주어진 프로그램에서 사용하는 변수들의 위치나 종류를 찾아내고 그 구조를 분석하여 프로그램의 각 단계에서 어떠한 값이 실행 스택(stack)이나 전역(global) 변수를 채우고 있는지를 아는 것이 프로그램의 행동을 이해하는 데에 필수적이다. 이러한 과정을 위해 alias analysis를 기반으로 하는 value analysis 기법이 필요하다. 이러한 연구 중 가장 대표적인 연구는 Balakrishnan의 VSA(Value-Set Analysis)이다[2]. VSA는 SI(Stride Interval)이라는 변수 위치 표현에 효과적인 표현방법을 기반으로 프로그램을 구성하는 각 영역(region)마다 SI로 표현되는 값을 하나씩 가지는 value set을 구성하여 프로그램에서 사용하는 변수들을 찾아내는 작업을 수행하지만 단편화된 프로그램 조각에서 많이 발생하는 '정의되지 않은(undefined)' 변수를 단순히 전범위 값([-∞, ∞])으로 놓고 이후 분석을 수행하게 되어 그 정확도가 크게 떨어지는 한계점을 가지고 있다.

따라서 본 논문에서는 기존 VSA의 한계점을 극복하여, 다양한 환경에서의 역공학에 적합한 분석 방법을 제안한다. 특히 단편화된 프로그램 조각에서 자주 등장하는 정의되지 않은 값들의 변화를 유추하는 방법을 제공하여 프로그램의 데이터 구조 역공학 방법을 향상시킨다.

II. 소프트웨어 역공학과 기존 기법의 문제점

2.1 소프트웨어 역공학의 변수 탐색 및 복원

역공학을 통해 직접 분석해야하는 실행파일 코드는 개발자가 작성한 소스코드가 컴파일 과정을 통해 주어진 플랫폼에서만 수행될 수 있는 assembly 언어로 변환되고 이를 encoding한 기계어 코드가 되는 과정으로 구성된다. 그래서 역공학을 위한 분석 단계에서는 이러한 기계어 코드를 이해하기 쉬운 형태로 변환하기 위한 처리과정을 거친 후에 데이터 역공학을 수행한다.

소스코드가 주어진 경우에는 소스를 분석하여 내부에서 사용하는 변수와 구조를 손쉽게 확인 할 수 있지만, 실행파일에서는 소스코드에 있던 변수 정보는 사라지고 저수준의 메모리 접근 연산으로 단순화되어 실제 데이터 구조를 찾을 수 없게 된다. 따라서 역공학 과정에서는 이러한 메모리 접근 연산을 변수 존재의 단서로 활용하여 변수의 위치와 구조를 유추한다.

하지만 대부분의 메모리 접근 연산의 메모리 주소는 절대적인 값을 가지는 것이 아니라 스택관련 포인터 레지스터들(ESP or EBP)을 기준으로 하는 상대적인 주소(예, EBP + 8)를 가지게 된다. 따라서 각 변수에게 할당된 메모리 주소를 정확하게 파악하기 위해서는 상대주소가 가지게 될 값을 분석하여야 해당 위치에 있는 변수를 찾아낼 수 있다. 또한 주소 연산을 위해 주로 EBP와 ESP의 두 가지 레지스터를 혼용하여 사용하기 때문에 동일한 주소를 찾는 것이 더 어렵게 된다. 예를 들어, 두 변수가 ESP+4의 주소와 EBP-16의 주소에 있는 경우, ESP와 EBP의 값에 따라서 두 변수는 동일 변수일 수도 있고 별개의 변수일 수도 있다.

2.2 Value-Set Analysis

이러한 목적을 위하여 VSA가 제안되었다[2]. VSA에서 기본 값은 SI로 표현되는데, SI는 숫자의 범위와 범위 내부의 간격값으로 구성되어, s[lb,ub]의 형태로 표현한다. 여기서 lb와 ub는 숫자의 범위의 최소값과 최대값 그리고 s는 간격값을 의미한다. 예를 들어, 0, 4, 8, 12의 네 숫자는 0부터 12의 범위에 속하고 그 사이 간격값이 4가 된다. 즉 SI를 이용하여 이러한 값을 4[0,12]라고 표현할 수 있다. 특히 이러한 표현 방법은 배열(array)와 같이 정해진 범위 내에서 일정한 간격마다 변수가 존재하는 경우가 많은 메모리 주소를 표현하는데 효과적이다.

VSA는 SI라는 형태를 기반으로 복수의 SI를 포함하는 value set이라는 구조를 사용한다. VSA에서는 각 함수가 자신만의 지역 스택 프레임 (local stack frame)을 가지고 이를 이용한 상대주소(예, ESP+4) 를 사용한다는 점에서 기인하여 각 함수의 local stack frame을 하나의 독립적인 영역(region)으로 가정한다. 따라서 각 함수의 local stack마다 주소를 가리키는 (SI로 표현되는) 독립된 값이 있고, VSA에서 value set은 이러한 독립적인 값들의 집합으로써 일반 상수나 절대 주소를 가리키는 global 영역의 SI를 기본으로 하여 영역 수(함수 수) 만큼의 SI를 가지게 된다. 이때 값이 정의되지 않은 영역에 대해서 SI는 정의역(domain)의 모든 값 1[-∞, ∞](전체영역에 간격값은 1)을 의미하는 ⊤ (lattice의 top, [5] 참조)으로 초기화 된다. 예를 들어, 3개의 함수로 구성된 프로그램에서 일반 상수 3은

```
int foo(int *intArr) {
  int vv = 10;              /* (1) */
  intArr[2] = 11;           /* (2) */
  intArr[3] = vv;           /* (3) */
  return intArr[3];
}
```

(a) C

```
push   ebp
mov    ebp,esp
sub    esp,0x10
mov    DWORD PTR [ebp-0x4],0xa    ; (1)
mov    eax,DWORD PTR [ebp+0x8]    ; (2)
add    eax,0x8                    ;  .
mov    DWORD PTR [eax],0xb        ;  .
mov    eax,DWORD PTR [ebp+0x8]    ; (3)
lea    edx,[eax+0xc]              ;  .
mov    eax,DWORD PTR [ebp-0x4]    ;  .
mov    DWORD PTR [edx],eax        ;  .
mov    eax,DWORD PTR [ebp+0x8]
add    eax,0xc
mov    eax,DWORD PTR [eax]
leave
ret
```

(b) Assembly

〔그림 1〕 C 코드에 해당하는 Assembly 코드 예제. 변수의 사용이 저수준의 메모리 입출력 명령어로 표현된다.

global 영역에서 정의되고 다른 함수 영역에 속하지 않기 때문에 (0[3,3], ⊤, ⊤, ⊤)과 같은 value set으로 나타나고, 첫 번째 함수의 스택의 기준점(EBP)에서 8만큼 떨어진 주소를 가리키는 경우는 (⊤, 0[8,8], ⊤, ⊤)과 같이 표현될 수 있다.

2.3 VSA의 문제점

정적인 분석을 통한 역공학을 하는 경우에, 완전한 프로그램 코드를 전체적으로 분석하는 경우도 있지만, 그 복잡성 때문에 많은 경우 프로그램을 나누어 분석하게 되는 방법이 효과적이다. 함수 단위로 나누는 경우에 정의되지 않은 값들이 발생하는 데, 함수의 인자(argument)나 context sensitive 분석이 적용되지 않을 때 malloc과 같은 함수의 return 값들이 이에 속한다. 하지만, 기존의 VSA는 이러한 값을 ⊤, 즉 1[-∞, ∞]로 초기화해버리기 때문에 이후의 변화를 확인할 수 없어 정의되지 않은 값을 기반으로 한 메모리 접근에서 새로운 변수를 발견할 수 없다. 이러한 문제는 포인터를 인자로 사용하는 경우 빈번하게 발생하는데, 예를 들어, 〔그림 1〕 (a)의 C 코드에서 integer 포인터인 intArr을 함수 인자로 넘겨받는 경우, 각 C 코드에 해당하는 assembly 코드에서 확인할 수 있듯이 %ebp+8에 있는 값(첫번째 인자, intArr의 내용)이 %eax에 저장되고 이 값을 기반으로 %eax+8에 위차 한 변수에 11이 저장되고 (3)번 라인에서 %eax+12에 위치한 변수에 %ebp-4(변수 vv)에 저장된 값이 할당되는 것을 알 수 있지만, 기존 VSA 경우 %ebp+8에 저장되어 있는 값이 정의되어 있지 않아 1[-∞, ∞]로 초기화하게 되고 이후의 +8, +12과 같은 연산을 수행할 수 없기 때문에 intArr이 내포하는 변수들을 찾아낼 수 없게 된다.

III. 제안 내용

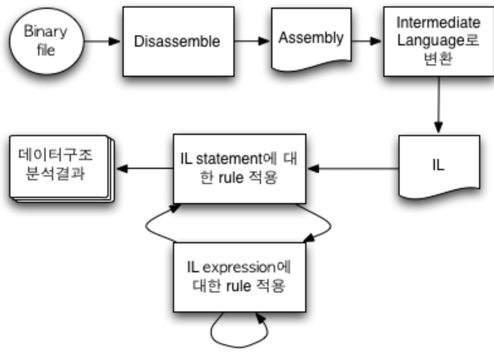
본 논문에서는 미리 정해진 영역에 대한 값들만을 분석할 수 있다는 기존의 VSA의 한계점을 극복하기 위해 개선된 기법을 제안한다. 제안하는 기법에서는 기존의 global 영역과 각 함수의 local stack frame 당 하나씩 영역을 가지는 고정된 구조에서 벗어나서 정의되지 않은 값을 만날 때, 그 값을 위한 새로운 영역을 동적으로 할당하여 해당 값의 변화를 추적한다. 또한, 이러한 동적인 영역 생성을 지원하고 더 효율적인 역공학 과정을 위해서 value set이 아닌 새로운 값을 표현 방법을 제안한다. 특히 변수를 복원한다는 데이터 구조 역공학의 입장에서는 동시에 여러 영역의 값을 독립적으로 유지하는 기존의 value set 구조가 오히려 모호성을 가중시킨다는 측면에서 하나의 값이 자신이 정의된 최근의 영역만을 유지하여 효율성을 높인다.

3.1 value의 정의

제안하는 기법에서 하나의 값은 다음과 같은 문법(syntax)을 가진다(문법 구조는 [5] 참조).

```
value       ::= global_val | region_val
global_val  ::= Global * si
region_val  ::= Region string * si
si          ::= int * int * int
```

SI를 이용한다는 것은 VSA와 동일하지만, 각 값은 하나의 영역에 대해서만 정의된다. 영역은 일반적인 상수나 절대주소를 통하여 접근하는 전역변수의 주소나 상수를 표현하기 위해 Global 영역(global_val)과 유일한 문자열(string) 이름을 통해 구



〔그림 2〕 제안하는 기법의 전체 진행 과정

별되는 일반 영역(region_val)로 구성된다. 이러한 일반 영역은 동적으로 할당된다.

3.2 제안하는 기법의 동작 원리

제안하는 방법은 〔그림 2〕와 같은 과정을 통해서 동작한다. 우선 binary 상태의 대상 실행파일을 disassemble하여 assembly 상태로 변환하고 이를 효율적인 분석을 위해 간단한 형태인 IL(Intermediate Language)상태로 다시 변환한다. 이러한 IL의 각 statement와 expression에 대해서 정의된 규칙을 통해서 내부에서 사용하는 변수 값의 변화를 추적한다.

대상 프로그램의 모든 statement를 분석하고 난 뒤에 최종적으로 얻게 된 메모리 상태를 바탕으로 프로그램이 사용하는 변수와 데이터 구조를 복원한다.

실행파일을 IL로 단순화한 결과를 바탕으로 하고 있기 때문에, assembly에서 사용하는 register들과 임시적인 값으로 쓰는 값들을 '내부변수'에 저장하여 사용한다. 메모리 자체 또한 큰 배열을 가진 하나의 내부변수로 다루어지게 된다. 따라서 분석이 끝난 후에는 최종적으로 메모리에 해당하는 내부변수의 인덱스 값들이 소스코드의 실제 변수가 있는 메모리 주소가 된다. 이러한 IL을 통한 분석 방법 기법은 정적분석 기법 연구〔1,3〕, IL의 구조는 BIL에 대한 연구〔4〕에 설명되어 있다.

IL에 대한 expression의 SI연산은 VSA에 정의된 규칙과 동일하다. 동적인 영역 설정과 할당을 위해 추가적으로 필요한 기능은 〔표 1〕과 같다. (기본적인 expression과 statement에 대한 규칙은 〔2〕의 data flow analysis를 위한 transfer function에 정의되어 있다.)

〔표 1〕 동적인 영역할당을 위해서 추가적으로 필요한 규칙

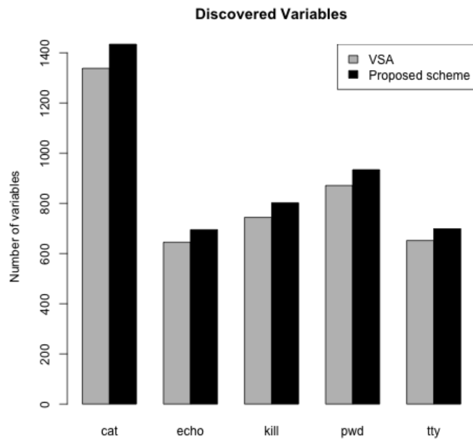
해당 statement 혹은 expression	수행 작업
함수 foo를 호출 (call foo)	내부변수 ESP를 새로운 영역 값, (Region "foo" 0〔0,0〕) 으로 초기화 한다.
상수 값을 내부변수 v에 할당 (v = c, c는 상수)	내부변수 v를 global 영역의 값으로 설정한다. (Global, 0〔c,c〕)
정의되지 않은 내부변수 u를 이용 (v = u, u는 undefined)	내부변수 u를 새로운 영역의 값 (Region "u", 0〔0,0〕) 으로 초기화하여 이용한다.

제안하는 기법에서는 다른 함수를 호출하거나 정의되지 값을 사용하고자할 때, 새로운 이름의 영역을 생성한다. 함수를 호출할 때는 호출하는 함수의 local stack frame에 해당하는 새로운 영역을 생성하고 호출하는 함수 이름을 Region의 이름으로 사용한다. 정의되지 않은 값이 사용되는 경우는 내부변수의 값이 정의되지 않은 경우와 메모리의 특정 주소에 있는 값이 정의되지 않은 경우가 있다. 두 가지 모두에 대해서 새로운 영역을 할당하여 사용하지만, 정의되지 않는 값이 1) 내부변수에 있는 경우, 해당 변수의 이름을 새로운 영역의 Region 이름으로 사용하고, 2) 메모리 내에 있는 경우, 주소(인덱스)를 변수 이름 뒤에 붙여서 Region 이름으로 사용한다. 예를 들어, 정의되지 않은 내부변수 u의 값은 (Region "u", 0〔0,0〕)이 되고 메모리 주소 (Region "mem1", 0〔4,4〕)에 있는 정의되지 않은 값은 (Region "mem1_0〔4,4〕", 0〔0,0〕)으로 초기화하여 구분한다.

IV. 구현 및 성능 평가

제안하는 시스템은 CMU에서 개발된 BAP (Binary Analysis Platform)〔3,4〕을 기반으로 구현되었다. BAP은 실행파일을 disassemble하여 BIL를 제공하고 이를 대상으로 제안된 기법을 구현함으로써 데이터 구조 역공학을 수행하였다.

〔그림 3〕에서 기존의 VSA 모듈과 제안된 기법을 구현한 모듈을 이용하여 실험 결과를 분석하였다. 실험은 Linux의 GNU Coreutil 8.4에 포함된 5개의 실행파일(cat, echo, kill, pwd, tty)을 대상으로 수행되었다. 각 실행파일은 적법한 함수의 단위로 분리하여, 총 375개의 함수에 대해 실험하였다. 결과 그래프에서 확인할 수 있듯이 제안하는 기법이 각 프로



〔그림 3〕 Coreutil의 실행파일에 대해 기존 VSA와 제안기법을 이용한 역공학을 통해 발견된 변수 수 비교

그램들에 대하여 기존의 VSA 비해 평균적으로 60.5개의 변수를 더 찾아낼 수 있는 것을 확인할 수 있다. 추가적으로 찾아낸 변수들은 2절에서 지적된 것과 같이 정의되지 않은 값들을 1차 혹은 2차적으로 사용하였기 때문에 기존의 기법으로는 찾을 수 없던 변수들이다. 특히 역공학으로 발견되는 변수의 50% 이상이 임시적으로 함수 call등에 사용되는 변수라는 것을 고려할 때, 제안하는 기법을 통해 추가적으로 발견되는 변수들은 소스코드에서 존재하는 변수들이라는 점에서 제안하는 기법이 실행파일의 동작과정 이해라는 역공학 목적에서 더 의미를 가진다고 할 수 있다.

V. 결 론

소프트웨어의 보안성 강화를 위한 실행파일에 대한 역공학 기법 중 필수적인 요소인 데이터 구조 역공학을 위한 기존의 기법은 프로그램 조각에 대해서는 적법한 함수 구조라 할지라도 분석율이 떨어지는 문제점을 가지고 있다. 따라서 본 논문에서는 정의되지 않은 값들에 대한 독립적인 영역을 할당하는 기법으로 데이터 구조 역공학의 성능을 향상시킨다.

참고문헌

〔1〕 Flemming Nielson, Hanne R. Nielson, and Chris Hankin. Principles of Program Analysis. Springer, Dec. 2004.

〔2〕 G. Balakrishnan and T. Reps, "Analyzing memory access in x86 executables," Proceedings of International Conference on Compiler Construction (CC), pp. 5-23, Apr. 2004.

〔3〕 J. Lee, T. Avgerinos, and D. Brumley, "TIE: Principled Reverse Engineering of Types in Binary Programs," Proceedings of Network and Distributed System Security (NDSS), Feb. 2011.

〔4〕 D. Brumley, I. Jager, T. Avgerinos, and E. J. Schwartz, "BAP: A Binary Analysis Platform," Proceedings of Computer Aided Verification (CAV), pp 463-469, Jul. 2011.

〔5〕 A. Aho, M. Lam, R. Sethi, and J. Ullman, Compilers: Principles, Techniques, and Tools, Pearson Education Inc, Sep. 2006.

〔6〕 김민성, 정덕영, "클라이언트 애플리케이션에서의 해킹," 한국정보보호학회논문지, 18(3), pp. 48-52, 2008년 6월.

〈著者紹介〉

이 종 협 (JongHyup Lee) 종신회원
2002년 2월: 연세대학교 기계전자공학부 졸업
2004년 2월: 연세대학교 컴퓨터과학과 석사
2009년 8월: 연세대학교 컴퓨터과학과 박사
2009년~2012년: Carnegie Mellon University, CyLab, Postdoc. 연구원
2012년 3월~현재: 한국교통대학교 소프트웨어학과 전임강사
〈관심분야〉 소프트웨어 보안, 네트워크 보안