Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.27, NO.4, Aug. 2017 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2017.27.4.803

동적 기호 실행을 이용한 윈도우 시스템 콜 Use-After-Free 취약점 자동 탐지 방법

강 상 용,† 이 권 왕, 노 봉 남‡
전남대학교 정보보안협동과정

Automated Method for Detecting Use-After-Free Vulnerability of Windows System Calls Using Dynamic Symbolic Execution

Sangyong Kang,† Gwonwang Lee, Bongnam Noh‡
Interdisciplinary Program of Information Security, Chonnam National University

요 약

최근 소프트웨어 산업의 발달에 따른 사회적 보안 문제가 지속적으로 발생하고 있으며, 소프트웨어 안정성 검증을 위해 다양한 자동화 기법들이 사용되고 있다. 본 논문에서는 소프트웨어 테스팅 기법 중 하나인 동적 기호 실행을 이용해 윈도우 시스템 콜 함수를 대상으로 Use-After-Free 취약점을 자동으로 탐지하는 방법을 제안한다. 먼저, 목표 지점을 선정하기 위한 정적 분석 기반 패턴 탐색을 수행한다. 탐지된 패턴 지점을 바탕으로 관심 밖의 영역으로의 분기를 차단하는 유도된 경로 탐색 기법을 적용한다. 이를 통해 기존 동적 기호 실행 기술의 한계점을 극복하고, 실제 목표 지점에서의 취약점 발생 여부를 검증한다. 제안한 방법을 실험한 결과 기존에 수동으로 분석해야 했던 Use-After-Free 취약점을 제안한 자동화 기법으로 탐지할 수 있음을 확인하였다.

ABSTRACT

Recently, social security problems have been caused by the development of the software industry, and a variety of automation techniques have been used to verify software stability. In this paper, we propose a method of automatically detecting a use-after-free vulnerability on Windows system calls using dynamic symbolic execution, one of the software testing methods. First, a static analysis based pattern search is performed to select a target point. Based on the detected pattern points, we apply an induced path search technique that blocks branching to areas outside of interest. Through this, we overcome limitations of existing dynamic symbolic performance technology and verify whether vulnerability exists at actual target point. As a result of applying the proposed method to the Windows system call, it is confirmed that the use-after-free vulnerability, which had previously to be manually analyzed, can be detected by the proposed automation technique.

Keywords: Dynamic Symbolic Execution, Windows Kernel, Use-After-Free, Software Vulnerability

I. 서 론

최근 소프트웨어 산업의 발달에 따른 사회적 보안 문제가 지속적으로 발생하고 있다[1]. 이 중 Use-After-Free(이하, UAF) 취약점은 매년 지속적으로 발생하고 있으며, 2015년과 2016년에도 각각 4.38%, 4.11%로 소프트웨어 취약점의 큰 비중을 차지하고 있다[2]. 이와 관련해 소프트웨어 신뢰도를 향상시키기 위한 테스팅의 필요성은 날이 갈수록 증가하고 있으며, 그 기술 역시 계속해서 발전을 거듭하고 있다. 하지만 소프트웨어 취약점을 탐지하

Received(04. 20. 2017), Accepted(06. 06. 2017)
† 주저자, scytalezz@gmail.com
‡ 교신저자, bbong@jnu.ac.kr(Corresponding author)

기 위해서는 전문 지식이 요구될 뿐만 아니라 많은 시간과 비용이 소모된다. 이를 극복하기 위해 많은 자동 테스팅 기법에 대한 연구가 진행되고 있으며, 이러한 연구들은 정적 탐지 방법과 동적 탐지 방법으로 나눌 수 있다. GUEB[3]와 같은 정적 탐지 도구는 높은 코드 커버리지를 가지지만 정확성은 떨어진다. 반면에 Undangle[4], Dangnull[5]과 같은 동적 탐지 도구는 높은 정확성을 가지지만 낮은 코드 커버리지를 보인다.

동적 기호 실행(Dynamic symbolic execution, 이하 DSE)은 소프트웨어의 입력 데이터를 심볼릭 변수로 정의함으로써 실행 가능한 모든 경로를 빠르게 탐색하는 분석 방법이다[6]. 높은 코드 커버리지와 높은 정확성을 모두 가지는 DSE는 소프트웨어 취약점 탐지에 활용될 수 있으며, 지난 몇 년 동안 EXE[7], KLEE[8], FuzzBALL[9], SAGE[10], S2E[11]와 같은 자동화 도구들이 연구되었다. 하지만 기존의 DSE 기술은 상용 소프트웨어에 대한 적용을 방해하는 경로 폭발 문제(path explosion)에 직면해있다.

본 논문에서는 DSE 기술의 장점을 이용한 새로운 윈도우 시스템 콜 UAF 취약점(이하, UAF) 자동 탐지 방법을 제안한다. 먼저 목표 지점을 선정하기 위해 정적 분석 기반 패턴 탐색을 수행한다. 이후 추출된 후보 지점을 바탕으로 관심 밖의 영역으로의 분기를 차단하는 유도된 경로 탐색 기법을 통해 기존 DSE 기술의 한계점을 극복하고, 해당 취약점을 탐지한다.

본 논문에서 제안한 기법의 우수성을 검증하기 위해 윈도우 커널에서 발생하는 기존 취약점을 대상으로 실험을 수행하였다. 그 결과 제안한 기법을 활용하여 기존에 수동으로 수행했던 윈도우 시스템 콜 UAF 취약점 색출의 자동화가 가능함을 입증하였다.

II. 관련 연구

2.1 동적 기호 실행 관련 도구

지난 몇 년 동안 EXE[7], KLEE[8], FuzzBALL[9], SAGE[10], S2E[11]와 같은 DSE 기술 활용을 위한 자동화 도구들이 연구되었다. 그 중 높은 평가를 받고 있는 S2E(Selective Symbolic Execution)[11]는 KLEE[8] 엔진을 기반으로 한 오픈 소스 DSE 수행 도구이다. S2E는 최근까지 업데이트가 진행되고 있으며[12], 개발자 포럼을 통해 활발한 의사소통 또한 이루어지고 있다[13]. 뿐만 아니라 S2E 도구를 활용한 DSE 기술 활용에 대한 연구도 계속되고 있다[14-17]. S2E는 QEMU 에뮬레이터의 모니터링을 통해 KLEE에서 불가능했던 바이너리 대상의 DSE 기술 적용을 지원한다. 이를 통해 응용 프로그램이나 라이브러리, 커널 드라이버 등을 실제 스택 내에서 분석할 수 있으며, 바이너리에 대한 직접 접근이 가능하기 때문에 소스코드가 공개되지 않은 소프트웨어를 대상으로 DSE 기술의 적용이 가능하다.

2.2 유저모드 콜백에 의한 윈도우 커널 공격

win32k.sys에서 유저모드 콜백에 의한 UAF 취약점이 다수 발견되고 있다. MS11-034, MS11-054 등에서 유저모드 콜백과 관련된 취약점을 보여주고 있다[18-21].

콜백이 실행될 때마다 커널은 win32k의 유저 크리티컬 섹션을 떠난다. 그리고 win32k는 유저모드 코드가 실행되는 동안 다른 일을 수행한다. 콜백에서 돌아온 직후, win32k는 참조된 객체가 해제되었는지를 검증해야 한다. 이 과정에서 불충분한 검사가 이루어지면서 취약점이 발생하게 된다. 객체가 이전에 락킹이 수행 되었다면 유저 크리티컬 섹션을 떠난 이후에도 타당한 객체라고 가정할 수 있다. 하지만 락킹이 이루어지지 않고 유저모드 콜백이 이루어졌다면, 이후 재참조 패턴 발생시 UAF 취약점이 발생한다. 이러한 형태의 취약점은 커널과 유저모드에서 나타나는 복잡한 과정에 의해 계속해서 발생할 가능성이 농후하며, 실제로 꾸준히 발생하고 있다[22].

III. DSE 기반 윈도우 시스템 콜 UAF 취약점 자동 탐지 방법

제안하는 DSE 기반 윈도우 시스템 콜 UAF 취약점 자동 탐지 방법은 Fig. 1과 같이 총 3단계로 이루어진다. Step1에서는 윈도우 시스템 콜의 취약점 탐색을 위해 목표 지점을 탐색한다. Step2에서는 심볼릭 도메인을 정의한다. 유도된 경로 탐색 기법을 활용해 관심없는 블록의 방문을 차단함으로써 목표 지점에 도달하는 경로의 효과적인 탐색을 도모한다. Step3에서는 DSE를 수행하고, 취약점 발생 위치 및 유효한 테스트 케이스 집합을 추출한다.

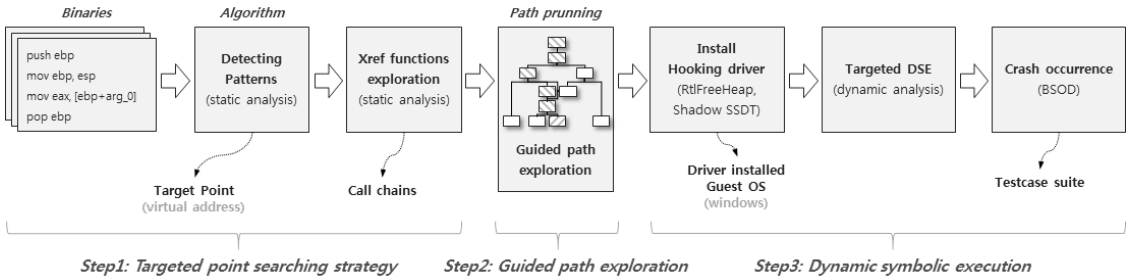

Fig. 1. An overview of proposed method

3.1 Step1 : 목표 지점 탐색 전략

Step1에서는 DSE 수행하기 앞서 목표 지점을 선정하는 작업을 수행한다. 본 논문에서는 유저모드 콜백에 의해 발생하는 윈도우 커널 UAF 취약점을 탐지하기 위한 패턴을 정의한다. 정의된 패턴을 바탕으로 정적 분석을 통해 크래시가 발생할 가능성이 있는 후보 지점을 추출한다. 추출된 주소는 목표 지점 후보로 등록되며, 추후 동적 실행을 통해 해당 지점에서 크래시가 실제로 발생하는지 여부를 검증하기 위해 사용된다.

잠재적 UAF 취약점 패턴 추출은 세 단계로 나뉘어 진행된다. 먼저, 선택된 객체에 대한 유저모드 콜백 발생 여부를 확인한다. 그리고 유저모드 콜백 발생 이전까지의 코드에서 해당 객체에 대한 락킹 수행 여부를 확인한다. 해당 객체에 대한 락킹이 수행되지 않았다면, 이후 재참조 패턴을 탐색하여 UAF 취약점이 발생할 가능성을 확인한다.

3.1.1 목표 지점 후보 추출

목표 지점 후보를 추출하기 위한 과정은 Table 1에 제시한 알고리즘을 따른다. 먼저, 추적할 객체가 저장되어 있는 변수(레지스터 혹은 스택)를 선정하고, 추적한다. 해당 변수가 'mov' 혹은 'lea' 명령어에 의해 이동, 복사될 경우 또한 계속해서 추적을 유지한다. 레지스터 뿐만 아니라 스택을 통해 호출 파라미터로 전달될 때도 마찬가지로 추적을 수행한다.

유저모드 콜백 발생을 탐지하기 위해 KeUserModeCallback 함수의 크로스 레퍼런스(Xref to KeUserModeCallback)를 추출한다. 만약 추출된 함수가 추적 중인 객체를 파라미터로 사용한다면, 유저모드 콜백이 발생했다고 가정한다.

유저모드 콜백을 통해 발생하는 예기치 못한 객체의 해제는 해당 객체에 대한 락킹이 수행되지 않았기 때문에 일어난다. 따라서 유저모드 콜백 발생 이전의 코드에서 해당 객체에 대한 락킹 수행 여부의 확인이 필요하다. 이를 위해 추적중인 객체의 특정 오프셋에 대한 접근 여부를 모니터링한다. 이 값은 락킹 카운트를 의미하며, 이 값을 증가시키는 코드가 존재하지 않다면 락킹이 정상적으로 이루어지지 않았다고 판단한다.

조건을 모두 만족하고, 해당 객체를 저장하는 메모리에 읽기 혹은 쓰기가 수행되었다면 재참조가 되었다고 판단한다. 재참조가 발생하는 코드의 주소를 추출하고, 목표 지점 후보로 선정한다.

3.1.2 함수 호출 체인 추출

목표 지점 후보까지의 함수 호출 체인을 추출한다. 목표 지점에서 백 트레이싱을 통해 해당 모듈의 시작 지점부터 목표 지점까지의 생성 가능한 경로를 추출한다. 이 때, 시작지점은 유저영역에서 임의로 트리거가 가능한 Nt 함수로 선정한다. 정해진 깊이(depth) 이내의 크로스 레퍼런스 함수를 확인하고, 만약 Nt 함수가 존재한다면 목표지점까지의 호출 체인을 추출한다.

추출된 호출 체인을 통해 취약점 발생 흐름을 유추할 수 있고, 유추된 내용을 바탕으로 유도된 경로 탐색 기법을 적용시키는 것이 가능하다.

3.2 Step2 : 유도된 경로 탐색

기존 DSE 기술의 경우 목표 지점에 도달하는 모든 경로를 탐색하기 때문에 많은 시간과 비용이 소요된다. 뿐만 아니라 경로 폭발 문제(path explosion)가 발생할 가능성이 높으며, 이 경우 목표 지점에 영원히 도달하지 못할 수 있다. Step2에

Table 1. UAF pattern detecting algorithm

Algorithm. UAF pattern detecting

```
INPUT     l0: initial location
          depth: maximum searching depth
OUTPUT    T: targeted location list ➤ T = {lt(0), lt(1), lt(2), ..., lt(n)}
          C: possible call chain list ➤ C = {C(0), C(1), C(2), ..., C(n)}
          Pc(n): possible call chain(function list) ➤ Pc(n) = {f(0), f(1),..., f(depth)}
DECLARE   lf: locking flag
          df: UAF detection flag
          ca: user-mode callback address
          R: register name list ➤ target object in register
          S: stack location name list ➤ target object in stack
BEGIN
  L ← function instruction list of l0 ➤ L = {l0, l1, l2, ..., ln}
  WHILE lcurrent in L THEN
    LineAnalyzer(lcurrent)
    IF mnem is 'mov' or 'lea' THEN
      IF op2 in (R or S) THEN
        TraceObject(op1, op2)
    IF mnem is 'call' THEN
      ca ← FindCallback(lcurrent, op1)
    IF ca is not NULL THEN
      lf ← CheckLockingPattern(ca)
    IF lf is false THEN
      df ← DetectReuseObject(ca)
      IF df is true THEN
        return ca, T, C, Pc ➤ success finding the pattern
  return NULL ➤ fail to find the pattern
END

FUNCTION LineAnalyzer(lcurrent)
  mnem ← mnemonic of lcurrent
  op1 ← operand1 of lcurrent
  op2 ← operand2 of lcurrent
END FUNCTION

FUNCTION XrefToHere(l, depth) ➤ call chain from NT* to here
  WHILE i to depth
    XrefTo(Function(l)) ➤ use import library function
END FUNCTION

FUNCTION TraceObject(op1, op2)
  IF op2 is (register or stack) THEN
    R or S ← op1
  ELSE IF op1 is (register or stack) THEN
    R or S ↚ op1
END FUNCTION

FUNCTION FindUserCallback(lcurrent, op1)
DECLARE X: function list triggering user-mode callback
  l ← KeUserModeCallback function starting address
  X ← XrefToHere(l, 2)
  IF op1 in X THEN
    return lcurrent
  ELSE THEN
    DetectUseafterfreePattern(lcurrent, depth) ➤ recursive call
    return NULL
END FUNCTION

FUNCTION CheckLockingPattern(ca)
  WHILE l0 to ca THEN
    IF mnem is 'inc' and op1 in (R or S) THEN
      return true
  return false
END FUNCTION

FUNCTION DetectReuseObject()
  IF mnem is (read or write) THEN ➤ memory read, write pattern
    IF (op1 or op2) in (R or S) THEN
      return true
  return false
END FUNCTION
```

서는 유도된 경로 탐색 기법을 적용하여 불필요 경로의 탐색을 최소화한다. 이전 단계에서 얻은 호출 체인을 바탕으로 불필요 코드 블록을 추출하고, 이 영역에 도달할 경우 탐색을 중단시킨다. 이러한 가지치기(path pruning)를 통해 유도된 경로를 탐색함으로써 원하는 결과를 도출할 수 있을 뿐만 아니라 경로 폭발 문제(path explosion) 또한 완화시킬 수 있다.

3.2.1 전체 코드블록 리스트 추출

하나의 단일 함수는 Fig. 2와 같은 코드블록 단위로 표현할 수 있다. 가지치기 대상 코드블록을 식별하기 위해 먼저, 단일함수 내에 존재하는 모든 코드블록을 추출한다. Fig. 2에서 총 11개의 코드블록이 존재하고, 마킹된 블록(CB_t)이 목표 지점이라 가정한다.

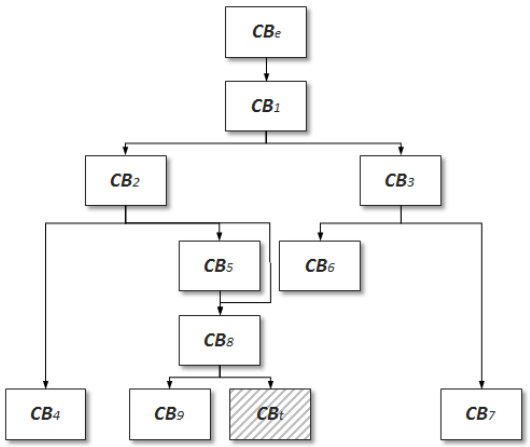

Fig. 2. Code block list of single function contains the target code block

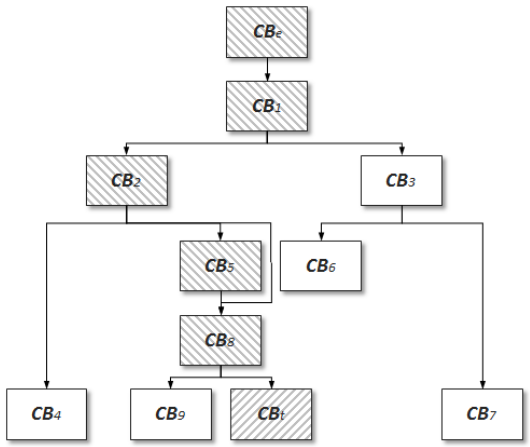

Fig. 3. Identify the code block list related to the target code block

3.2.2 관련 코드블록 리스트 식별

가지치기 대상 코드블록을 식별하기 위해 실제로 관심 있는 코드블록을 추출한다. 목표 블록인 CB_t의 상호참조 블록은 CB_8이며, CB_8의 상호참조 블록은 CB_2와 CB_5이다. 이와 같은 과정으로 실제 타깃 코드블록과 관련 있는 모든 블록 리스트를 식별한다.

3.2.3 가지치기 대상 코드블록 리스트 추출

전체 코드블록 리스트에서 타깃 코드블록 관련 리스트를 제외한 나머지 블록이 가지치기 대상 코드블록이다. 이 중 상호참조 블록이 관련 코드블록 리스트에 포함되지 않은 경우도 제외한다. 최종적으로 CB_7을 제외한 $\{CB_3, CB_4, CB_6, CB_9\}$가 가지치기 대상 코드블록 리스트가 된다.

이와 같이 최종적으로 추출된 리스트를 탐색 제외 대상으로 등록하면 유도된 경로 탐색이 가능하다. 결국 실제 DSE 수행 과정에서 가지치기 대상 블록들이 탐색 과정에서 제외되기 때문에 빠른 속도의 분석이 가능하고, 경로 폭발 문제 또한 완화된다.

3.3 Step3 : 동적 기호 실행 적용

3.3.1 RtlFreeHeap 후킹 드라이버

UAF 취약점이 발생했다 할지라도 해당 메모리에 쓰기 작업이 이루어지지 않았다면 윈도우는 정상적으로 동작한다. 크래시 발생 여부를 탐지하기 위해 할당 해제된 영역을 비정상적인 값으로 채워넣는 과정을 수행한다. DSE를 수행할 게스트 시스템에 후킹 드라이버를 설치하여 ntoskrnl!RtlFreeHeap 함수를 패치한다. 이와 같은 과정을 추가함으로써 윈도우 시스템 콜이 할당 해제된 메모리를 재사용할 경우 '접근 위반(access violation)'에 의한 BSOD가 발생하게 되고, UAF 취약점 발생 여부의 식별이 가능해졌다.

3.3.2 동적 기호 실행

Step1에서 추출한 함수 호출 체인의 첫 함수는 Nt 함수이고, 이는 유저 애플리케이션에서 직접 호출이 가능하다. 따라서 소스코드 레벨에서 원하는 변수에 심볼릭 마킹을 수행한 후 해당 함수를 호출할 수 있다. 파라미터를 심볼릭 마킹 후 Nt 함수를 호출하면, Step2의 과정으로 인해 유도된 경로를 동적으로 탐색하게 되고, 이는 DSE에서 발생하는 경로 폭발을 억제한다. 또한 동적 실행을 통해 UAF 취약점의 발생 여부를 확인할 수 있다.

IV. 실험 및 평가

본 논문에서 제안한 DSE 기반 윈도우 시스템 콜 UAF 취약점 자동 탐지 방법의 우수성을 입증하기 위한 실험 및 평가를 수행한다. 평가의 용이성을 위해 2011년부터 2015까지 발생한 UAF 취약점 중 분석이 완료된 취약점을 대상으로 실험을 수행한다.

Table 2. Experimental results

CVE IDs	Detecting pattern type		The number of call chain (depth)	BSOD Generation	Valid test suite
	User callback	UAF			
CVE-2011-1241	O	O	2(6)	O	O
CVE-2011-1242	O	O	2(6)	O	O
CVE-2011-1878	O	O	16(6)	O	O
CVE-2015-2360	O	O	22(7)	O	O
CVE-2015-2546	O	O	7(6)	O	O

제안한 방법을 이용해 추출된 결과와 기존에 알려진 분석 결과 및 PoC 코드와의 비교를 통해 우수성을 검증한다. 실험에 사용된 호스트 시스템은 Intel Core i7-3770 CPU 3.0GHz와 8GB RAM 사양의 Ubuntu 12.04 운영체제이며, 게스트 시스템은 S2E DSE 도구에 가장 최적화되어 있는 Windows XP SP3 운영체제 환경을 대상으로 실험을 수행했다.

Table 2는 제안한 방법을 이용해 취약점 탐지를 수행한 결과다. 먼저 Step1에서 제안한 알고리즘을 이용해 취약점 패턴을 탐지한 결과, 5개의 기존 UAF 취약점 발생 위치를 정확하게 탐지했다. 이후 Nt 함수에서 선정된 목표 지점까지 도달하는 경로인 각각 2개(depth 6), 2개(depth 6), 16개(depth 6), 22개((depth 7), 7개(depth 6)의 호출 체인을 추출했다. 이를 바탕으로 S2E에서 제공하는 Annotation 플러그인을 이용해 Step2의 유도된 경로 탐색을 적용하고, DSE를 수행했다.

제안한 알고리즘에 의해 탐지된 5개의 목표 지점을 대상으로 DSE 기술을 적용한 결과, 수분 내에 모두 BSOD가 발생하는 것을 확인할 수 있었다. 실험 결과 로그를 통해 BSOD를 발생시키는 테스트 케이스 즉, 호출 체인의 최상단에 위치한 Nt 함수의 파라미터를 얻을 수 있었으며, 이는 실제 공개된 PoC 코드에서 사용하는 값과 일치함을 확인했다. 또한 추출된 로그의 파싱을 통해 BSOD가 발생하는 호출 체인도 얻을 수 있었다.

V. 결론 및 향후 연구

본 논문에서 윈도우 시스템 콜의 유저모드 콜백에 의한 UAF 취약점을 자동으로 탐지하는 방법을 제안하였다. 취약점 탐지의 자동화를 위해 정적 분석 기반 UAF 취약점 탐지 모델을 개발하였으며, 이를 활용해 잠재적 크래시 유발 지점을 추출하였다. 추출된 지점을 바탕으로 관심 밖의 영역으로의 분기를 차단하는 유도된 경로 탐색 기법을 적용하였으며, DSE 기술의 경로 폭발 문제를 완화시켰다. 또한 RtlFreeHeap 후킹 드라이버를 설치하여 목표 지점에서의 취약점 발생 여부를 식별할 수 있도록 하였다. 결론적으로, DSE 기술을 활용해 윈도우 시스템 콜 안정성 검증의 자동화가 가능함을 증명할 수 있었다. 제안한 방법은 운영체제 커널 안정성 검증의 자동화에 활용될 수 있을 것으로 기대한다. 향후 추가적인 버그 체크 모델을 개발할 것이며, 추가적인 실험을 통해 신규 커널 취약점을 탐지하기 위한 지속적인 연구를 수행할 계획이다.

References

[1] SPri, "Software Industry Annual Report", http://spri.kr/post/11313, 2015.

[2] NIST, "National Vulnerability Database: Statistics Results Page", https://web.nvd.nist.gov/view/vuln/statistics-results?adv_search=true&cves=on&query=use+after+free&cvss_version=3, 2017

[3] J Feist, L Mounier and ML Potet, "Statically detecting use after free on binary code", Springer, Vol. 10, Aug. 2014.

[4] J. Caballero, G. Grieco et al, "Undangle:Early Detection of Dangling Pointers in Use-after-free and Double-free Vulnerabilities," ISSTA, 2012.

[5] B Lee, C Song, Y Jang, T Wang, T Kim, L Lu, W Lee, "Preventing Use-after-free with Dangling Pointers Nullification",

NDSS, 2015
〔6〕 James C. King, "Symbolic Execution and Program Testing", Communications of the ACM, Volume 19 Issue 7, July. 1976.
〔7〕 Cadar C, Ganesh V, Pawlowski P M, et al. "EXE:Automatically generating inputs of death", Computer Andcommunications Security, 2006, 12(2):3296-3306.
〔8〕 Cadar C, Dunbar D, Engler D R. "KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs", USENIX, 2008.
〔9〕 D Babic, L Martignoni, S McCamant, and D Song, "Statically-Directed Dynamic Automated Test Generation", In Proceedings of the ACM/SIGSOFT (ISSTA), July. 2011.
〔10〕 E. Bounimova, P. Godefroid, and D. Molnar, "Billions and Billions of Constraints: Whitebox Fuzz Testing in Production", IEEE, 2013.
〔11〕 Vitaly Chipounov, Volodymyr Kuznetsov, George Candea. "S2E: A Platform for In Vivo Multi-Path Analysis of Software Systems", 16th Intl. ASPLOS, Newport Beach, CA, March. 2011.
〔12〕 GitHub, "dslab-epfl/s2e", https://github.com/dslab-epfl/s2e
〔13〕 Google Groups, "S2E Developer Forum", https://groups.google.com/d/forum/s2e-dev
〔14〕 VT Pham, WB Ng, K Rubinov et al, "Hercules : Reproducing Crashes in Real-World Application Binaries", ACM, 2015.
〔15〕 C Yeh, H Lu, C Chen et al, "CRAXDroid: Automatic android system testing by selective symbolic execution", 8th International Conference on Software Security and Reliability, 2014.
〔16〕 N Stephens, J Grosen, C Salls et al, "Driller : Augmenting Fuzzing Through Selective Symbolic Execution", Internetsociety, 2016.
〔17〕 B Zhang, C Feng, B Wu et al, "Detecting integer overflow in Windows binary executables based on symbolic execution", IEEE/ACIS. 2016.
〔18〕 Yu Wang, "A New CVE-2015-0057 Exploit Technology", BlackHat, Sep. 2015.
〔19〕 Aaron Adams, "Exploiting the win32k!xxxEnableWndSBArrows use-after-free (CVE-2015-0057) bug on both 32-bit and 64-bit", An NCC Group Publication, 2015.
〔20〕 Dominic Wang, "Exploiting MS15-061 Microsoft Windows Kernel Use-After-Free(win32!xxxSetClassLong), nccgroup, 2015.
〔21〕 Udi Yavo, "Class Dismissed: 4 Use-After-Free Vulnerabilities in Windows", http://breakingmalware.com/vulnerabilities/class-dismissed-4-use-after-free-vulnerabilities-in-windows, July. 2015.
〔22〕 Tarjei Mandt, "Kernel Attacks through User-Mode Callbacks", BlackHat USA, Aug. 2011.
〔23〕 Godefroid, Patrice, Nils Klarlund, and Koushik Sen, "DART: directed automated random testing", ACM Sigplan Notices, Vol. 40, No. 6, ACM, 2005.
〔24〕 Sen, Koushik, Darko Marinov, and Gul Agha, "CUTE: a concolic unit testing engine for C", Vol. 30, No. 5, ACM, 2005.
〔25〕 Kunal Taneja, "Guided Path Exploration for Regression Test Generation", ICSE Companion, 2009.

〈저 자 소 개〉

강 상 용 (Sangyong Kang) 학생회원
2014년 8월: 전남대학교 컴퓨터공학과 공학사
2014년 9월~2016년 8월: 전남대학교 정보보안협동과정 이학석사
2016년 9월~현재: 전남대학교 정보보안협동과정 박사과정
〈관심분야〉 소프트웨어 취약점 분석 및 탐지, 시스템 보안

이 권 왕 (Gwonwang Lee) 학생회원
2016년 2월: 조선대학교 컴퓨터공학과 공학사
2016년 3월~현재: 전남대학교 정보보안협동과정 석사과정
〈관심분야〉 네트워크 보안, 소프트웨어 취약점 분석 및 탐지, 시스템 보안

노 봉 남 (Bongnam Noh) 종신회원
1978년: 전남대학교 수학교육과 학사
1982년: KAIST 대학원 전산학과 석사
1994년: 전북대학교 대학원 전산과 박사
1983년~현재: 전남대학교 전자컴퓨터공학부 교수
2000년~현재: 전남대학교 시스템보안연구센터 소장
〈관심분야〉 정보보안, 시스템 및 네트워크 보안