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.2017.27.4.811

소프트웨어 안전성 검증을 위한 입력 파일 기반 동적 기호 실행 방법

박 성 현,† 강 상 용, 김 휘 성, 노 봉 남‡
전남대학교 정보보안협동과정

Input File Based Dynamic Symbolic Execution Method for Software Safety Verification

Sunghyun Park,† Sangyong Kang, Hwisung Kim, Bongnam Noh‡
Interdisciplinary Program of Information Security, Chonnam National University

요 약

최근 소프트웨어 자동화 기술 연구는 단일 경로의 테스트 케이스 생성뿐만 아니라, 다양한 테스트 케이스를 통해 취약점으로 도달할 수 있는 최적화된 경로를 파악하는 것에 중점을 두고 있다. 이러한 자동화 기술 중 Dynamic Symbolic Execution(이하, DSE) 기술이 각광 받고 있지만, 현재 대부분의 DSE 기술 적용 연구는 리눅스 바이너리 혹은 특정 모듈 자체만을 대상으로 적용하는 데 그치고 있는 실정이다. 하지만 대부분의 소프트웨어의 경우 입력 파일을 기반으로 작업이 수행되고, 또 이러한 과정에서 취약점이 다수 발생하고 있다. 따라서 본 논문은 소프트웨어 안전성 검증을 위한 입력 파일 기반 동적 기호 실행 방법을 제안한다. 실제 바이너리 소프트웨어 3종에 적용한 결과 제안하는 방법을 통해 효과적으로 해당 지점에 도달하는 테스트 케이스를 생성할 수 있었다. 이는 DSE 기술이 실제 소프트웨어 분석의 자동화에 활용될 수 있음을 보여준다.

ABSTRACT

Software automatic technology research recently focuses not only on generating a single path test-case, but also on finding an optimized path to reach the vulnerability through various test-cases. Although Dynamic Symbolic Execution (DSE) technology is popular among these automatic technologies, most DSE technology researches apply only to Linux binaries or specific modules themselves. However, most software are vulnerable based on input files. Therefore, this paper proposes an input file based dynamic symbolic execution method for software vulnerability verification. As a result of applying it to three kinds of actual binary software, it was possible to create a test-case effectively reaching the corresponding point through the proposed method. This demonstrates that DSE technology can be used to automate the analysis of actual software.

Keywords: Dynamic Symbolic Execution, Vulnerability, Automatic Exploit Generation

I. 서 론

최근 소프트웨어 산업은 다양한 IT 기기의 발달과 함께 소프트웨어의 활용 영역이 확장되어 왔다. 또한 국내외 유무선 네트워크 인프라를 통해 온오프라인 소프트웨어가 활성화됨에 따라 다양한 분야에서 소프트웨어를 이용한 융복합화가 빠르게 진행되고 있다. 이처럼 사회의 여러 분야에서 소프트웨어를 활용함으로써 많은 이득을 얻게 되었지만, 동시에 소프트웨어 취약성으로 인한 많은 사회적 이슈들이 발생하고 있고, 이를 악용한 소프트웨어 공격이 증가하고 있다

Received(04. 26. 2017), Accepted(06. 13. 2017)
† 주저자, everpall@gmail.com
‡ 교신저자, bbong@jnu.ac.kr(Corresponding author)

[1][2].

최근 소프트웨어 분석의 자동화 기술 연구는 단일 경로의 테스트 케이스 생성뿐만 아니라, 다양한 테스트 케이스 중에서 취약점으로 도달 할 수 있는 최적화된 경로를 파악하는 것에 중점을 두고 있다[3, 4,5,6]. 이러한 자동화 기술 중 Dynamic Symbolic Execution(DSE)[7] 기술이 각광 받고 있다. 동적 기호 실행은 바이너리의 자동 테스팅을 위한 분석 플랫폼으로써, 대상 소프트웨어의 특징이나 행위를 분석하는 연구가 가능하다. 더 정확히 바이너리 코드의 타겟 지점까지의 거의 모든 경로를 탐색할 수 있는 장점을 갖고 있다. 하지만 실제 바이너리 소프트웨어에 적용하는데 있어서 경로 폭발 문제가 발생하곤 한다.

동적 기호 실행은 소프트웨어 취약점을 찾는데 유용하게 사용될 수 있고 KLEE[8], CUTE[9], DART[10], CREST[11], S2E[12]와 같은 자동화 도구들이 존재한다. 현재까지 이러한 도구들은 특정 모듈 및 바이너리 자체만을 대상으로 적용하는데 초점을 맞춘 연구가 진행되어 왔다. 하지만 대부분의 소프트웨어는 입력 파일을 기반으로 작업이 수행된다. 따라서 이와 같은 소프트웨어를 기존의 기호 실행 도구를 통해서 분석을 진행하는 과정에 몇 가지 제약사항이 따른다. 이에 효율적인 소프트웨어 분석을 위한 입력 파일 기반 동적 기호 실행 방법이 필요하다.

본 논문에서는 소프트웨어 안전성 검증을 위한 입력 파일 기반 DSE 기술을 활용하는 방법을 제안한다. 제안하는 방법은 기본적으로 심볼릭 파일(symbolic file)생성 메커니즘(mechanism)을 이용한다. 심볼릭 파일 생성을 통해 모듈 내부 변수 또는 레지스터가 아닌 파일 오프셋 기반 심볼릭 적용이 가능하다. 또 동적 기호 실행이 진행되는 과정에서 탐색 경로 지점 선정 및 탐색 종료 시점을 선정하는 방법을 제안하여 보다 복잡한 파일 에 대해서도 점검이 가능함을 보여준다. 실험을 통해 실제 바이너리 소프트웨어에 적용한 결과 효과적으로 해당 지점에 도달하는 테스트 케이스를 생성할 수 있었다.

논문의 구성은 다음과 같다. 2장에서는 관련연구를 통해 소프트웨어 분석에 따른 기존 동적 기호 실행 방법의 한계점을, 3장에서는 제안하는 분석 메커니즘을, 4장에서는 제안하는 방법에 따른 특정 소프트웨어에 대한 실험 및 평가를, 5장에서는 결론을 소개한다.

II. 관련 연구

2.1 동적 기호 실행 도구

2.1.1 KLEE

KLEE[8]는 현재 수행하는 스테이트(state)가 분기 지점에 도달하면, 해당 분기 조건이 참인지 거짓인지 여부를 판단하기 위해 STP solver에 심볼릭 경로 수식과 함께 분기 조건을 쿼리 형태로 전달한다. 참과 거짓 모두 실행 가능한 경로 조건일 경우에는 자식 프로세스를 생성하여, 실행가능한 모든 경로에 대한 분석을 수행한다.

Table 1에서 제시한 간단한 예제 코드와 기호 실행 경로 트리를 통하여 KLEE의 구조를 쉽게 이해할 수 있다. Fig.1 은 예제 코드를 이용하여 생성된 기호 실행 경로 트리를 나타낸다.

위의 기호 실행 경로 트리의 노드들은 예제 코드에서 위치하는 각각의 분기 지점을 나타낸다. 처음 main 함수에 해당하는 a1 스테이트의 생성 이후, 심볼릭 변수 i에 대한 초기화가 이루어지지 않았기 때문에, 조건식 i == 0 을 만족하지 않게 된다. 각각의 분기 식에서 거짓일 경우의 실행 경로는 자식 프로세스에서 처리되어야 하므로, b1 스테이트가 새롭게 생성되어 이를 처리하게 된다. b1 스테이트를 진행하는 과정에서는 새로운 조건식 j == 0을 만나게 되는데 j값 역시 초기화가 이루어지지 않았기 때

Table 1. Sample source code for KLEE

```
 1:  int main() {
 2:          int i, j, ret;
 3:          klee_make_symbolic(&i);
 4:          klee_make_symbolic(&j);
 5:
 6:          if(i==0) {
 7:                  if(j==0)
 8:                          ret=1;
 9:                  else
10:                          ret=2;
11:          }
12:          else {
13:                  if(j==1)
14:                          ret=3;
15:                  else
16:                          ret=4;
17:          }
18:          return ret;
19:  }
```

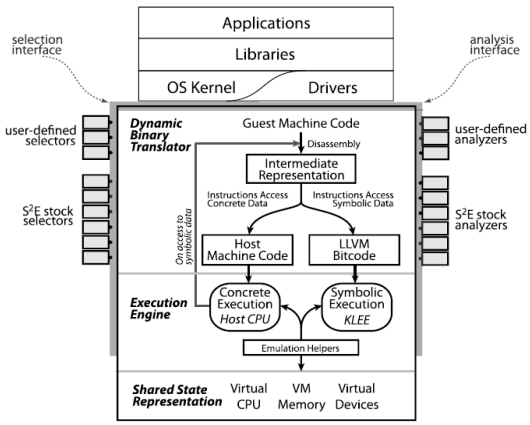


Fig. 1. The entire structure of the S2E

문에, c 스테이트를 생성하여 거짓일 경우의 실행 경로를 처리하도록 한다. 심볼릭 변수와 분기 조건에 따른 실행 경로의 탐색 과정은 조건에 따른 분기가 발생하지 않을 때까지 반복되며, 이렇게 분류된 각각의 실행 경로들은 분석의 목적에 따른 작업을 진행하게 된다.

2.1.2 S2E

S2E(Selective Symbolic Execution)[12]는 성능 프로파일링(profiling), 상용 소프트웨어에 대한 리버스 엔지니어링(reverse engineering), 커널 모드 및 사용자 모드 바이너리의 자동 테스팅 등을 위한 개발 플랫폼으로서 대상 소프트웨어의 특징이나 행위를 분석할 수 있다. S2E는 개념적으로 모듈의 경로를 자동으로 탐색하는 탐색기(explorer)와 해당 경로에 대한 분석을 수행하는 분석기(analyzer)로 구분할 수 있다. 탐색기는 분석 대상 시스템의 하위에 존재하는 관심 있는 모든 실행 경로를 탐색하기 위해 기호 실행 엔진(symbolic execution engine)을 사용하고, 분석기는 탐색기에서 발견한 모든 실행 경로에서 분석기에 명시된 분석 행위를 수행한다. 사용자는 S2E에서 제공하는 분석기를 사용할 수 있고, S2E에서 제공하는 API를 이용하여 기존 분석기를 변형한 사용자 정의 분석 도구를 자체적으로 제작하여 사용할 수 있다.

Fig. 2는 S2E의 전체 구조를 보여준다. S2E에서는 QEMU[13] 가상머신과 KLEE[8] 심볼릭 실행 엔진, LLVM[14] 툴 체인을 이용한다. S2E는 두 가지 핵심 인터페이스를 제공한다. 첫 번째는 경

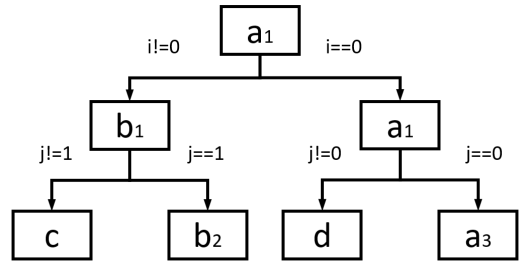


Fig. 2. Symbolic execution path tree generated by KLEE example code

로 선택이고, 두 번째는 분석이다. S2E는 가상 머신에서 대상 시스템을 실행하여 경로를 탐색한다. 그리고 선택적 실행은 심볼릭 하게 실행될 부분을 선택한다. 선택된 부분의 요구에 따라 분석될 시스템의 기계어 명령의 일부는 나머지 부분이 호스트 명령어 셋으로 변환되는 동안 동적 기호 실행에 적합한 중간 표현(intermediate representation)으로 VM 내에서 동적으로 변환된다. S2E는 VM 내에서 심볼릭 영역과 콘크리트(concrete) 영역 사이를 번갈아 가며 실행함으로써 데이터를 앞뒤로 변환한다. 그래서 마치 전체 시스템(운영체제, 라이브러리, 어플리케이션 등)이 다중 경로 모드(multipath mode)로 실행되는 것처럼 수행된다.

2.2 기존 동적 기호 실행 방법

실제로 동적 기호 실행은 심볼릭 실행 엔진이 조건 분기에서 두 가지 결과를 모두 따를 수 있도록 하는 프로그램 입력에 심볼릭 값을 삽입하는데 있다. 이는 도달하기 어려울 수 있는 프로그램 영역을 커버할 수 있다[7].

동적 기호 실행 도구는 Fig.3 과 같이 두 가지 방법으로 심볼릭 변수를 지정하는 것이 가능하다. 첫 번째로 소스코드 레벨의 도구에서 제공하는 API를 이용하여 심볼릭 변수를 지정할 수 있다. 이는 컴파

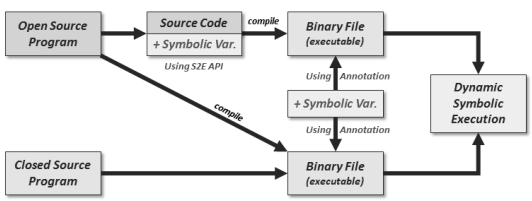


Fig. 3. Symbolic variable assignment method of S2E

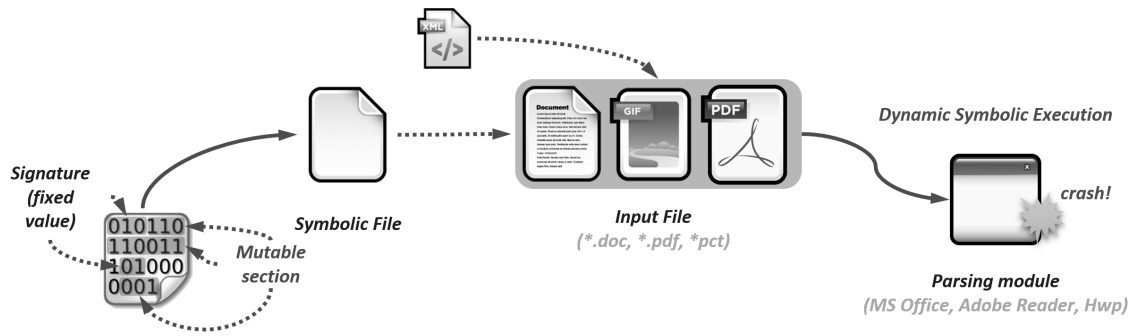


Fig. 4. Input file based dynamic symbolic excution method

일 후에 동적으로 심볼릭 실행을 수행할 수 있다. 두 번째는 바이너리 레벨의 Annotation 을 통해 동적으로 심볼릭 변수를 지정하는 것이 가능하다. 또한 Annotation을 이용하면 런 타임에서 동작하는 프로세스의 메모리, 레지스터, 콜 스택 등 여러 가지 정보들을 모니터링하고 계측하는 것이 가능하다.

2.3 기존 도구의 문제점 및 한계점

대부분의 소프트웨어는 디스크에 기록된 파일 형태의 입력을 통해 동작이 수행된다. 하지만 기존의 DSE 도구를 이용해서 이런 파일 입력 특징을 갖는 소프트웨어를 분석하는데 몇 가지 한계점과 문제점이 존재한다.

현재 대부분의 DSE 기술 적용 연구는 리눅스 또는 윈도우 바이너리 혹은 특정 모듈 자체만을 대상으로 적용하는데 그치고 있는 실정이다. KLEE는 오래전에 나온 동적 기호 실행 도구로써 다양한 제약조건 해결 최적화(constraint solving optimi-zation) 기능을 지원하며, 다수의 state 탐색 방법을 제공한다. 하지만 단순히 소스코드기반 탐색만을 지원하기 때문에 바이너리 및 모듈 분석을 진행함에 있어서는 한계점을 갖는다. S2E는 사실상 실제 환경에서 실행되는 대규모 소프트웨어에 대해서 동적 기호 실행을 기반으로 분석하는 데 도움이 된다. 그리고 이러한 S2E는 KLEE 엔진과 QEMU 가상 머신 에뮬레이터 위에 구축되기 때문에 소스코드기반 탐색뿐만 아니라 바이너리기반 탐색 또한 가능하다. 하지만 파일을 기반으로 한 동적 기호 실행을 진행하는 과정에서 몇 가지 문제점 및 한계점이 발생한다.

이에 본 논문에서는 DSE 도구인 S2E를 이용하여 입력 파일 기반소프트웨어를 효과적으로 분석하는 방법을 제안한다. 분석하는 과정에서 발생하는 기존의 DSE 도구의 문제점 및 한계점을 해결하고, 또 최근에 발생하는 취약점 탐색에 대해서도 활용 될 수 있는 자동화된 테스팅 방법을 제안한다.

III. 제안하는 파일 기반 동적 기호 실행 방법

본 논문에서는 입력 파일 기반 동적 기호 실행 방법을 제안한다. DSE를 통해서 소프트웨어를 분석하는데 있어서 가장 중요한 것은 파일 내의 데이터가 심볼릭 변수로 정의되는 것이다. S2E는 바이너리 분석을 진행할 경우 특정 메모리 영역의 변수를 S2E의 플러그인을 통해 동적으로 심볼릭 변수를 지정하는 것이 가능하다.

보통 디스크에 기록된 파일 형태의 입력은 타겟 소프트웨어의 특정 모듈에 의해 메모리에 적재되고 파싱된다. Fig 4는 DSE를 통한 입력 파일 기반 소프트웨어 분석의 전체적인 과정을 보여준다. 먼저 해당 파일의 포맷을 파악한 후, 대상 파일에서 분석을 수행할 영역을 추출해 낸다. 이후 각각의 파일은 심볼릭 파일로 저장 된 후 입력 파일로 전달된다. 최종적으로 해당 소프트웨어의 파싱 모듈은 메모리에 적재된 입력 파일을 DSE를 통해서 분석을 진행할 수 있도록 한다.

3.1 심볼릭 파일을 통한 심볼릭 변수 정의

소프트웨어는 대부분 디스크에 기록된 파일 형태의 입력을 통해 동작이 수행된다. 이와 같이 콘크리트 값으로 디스크에 기록된 파일의 경우, 심볼릭 데이터는 인자로 넘겨주는 버퍼에 존재하는 파일의 주소 값에 의존할 수밖에 없다. 하지만 분석 대상 프로

그램 내부에서 어떠한 방식으로 입력 파일을 사용하는 것을 파악하는 것은 쉽지 않기 때문에 직접적인 심볼릭 정의가 불가능한 실정이다. 또 분석 대상 프로그램의 분석을 통해 입력 파일의 파싱되는 내용을 파악할 수 있지만, 이러한 경우 추가적인 시간과 비용이 소모되기 때문에 자동화된 기술 활용의 의미가 축소된다.

실제로 프로그램이 동작하기 위해서는 메모리에 파일이 로딩 되어야 한다. Hwp.exe 파일을 예시로 살펴보면 2MB 미만의 파일의 경우 파일 오브젝트(FILE_OBJECT)를 통해 전체 파일이 로딩 된 메모리 주소를 획득하는 것이 가능하다. 하지만 2MB 이상의 입력 파일의 경우 불규칙하게 메모리에 분할되어 로딩 되기 때문에 실제 주소를 획득하는 과정이 까다롭다. 즉, 원하는 영역에 심볼릭 마킹을 수행하는 것이 쉽지 않다.

파일 오브젝트를 통한 추적 외에 파일 입출력 함수(CreateFile, ReadFile)의 후킹을 통해 해당 API 바운더리에서 심볼릭 마킹을 수행하는 방법도 가능하다. 하지만 이러한 방법 또한 다양한 소프트웨어의 동작 방식을 고려해볼 때 일반적으로 적용하는 것이 불가능하다. 결국 프로그램에 대해 DSE 기술을 효과적으로 적용하기 위해서는 올바른 심볼릭 파일을 정의하는 기술이 필요하다.

본 논문에서는 램 디스크를 활용한 심볼릭 파일 생성 기법을 제안한다. 램 디스크 드라이브는 주기억 장치인 램의 일부를 보조기억 장치인 디스크처럼 활용하는 기법이다. 램 디스크 드라이브로 디스크를 생성한 뒤, 해당 디스크에 파일을 기록한다면 곧 메모리에 파일을 직접 저장하는 형태가 된다. 그렇기 때문에 파일 입출력 API를 모니터링 하여 메모리에 로드된 파일을 추적하지 않고, 디스크에 파일을 기록한 뒤 기록된 파일을 사용하는 것만으로 파일에 심볼릭 기술을 적용할 수 있게 된다.

Fig. 5은 램 디스크를 통해서 심볼릭 파일을 생성하는 과정이다. 이 과정에서 원하는 영역을 심볼릭 변수로 정의하고 파일을 램 디스크에 기록하면 심볼릭 파일을 생성하는 것이 가능하다. 심볼릭 파일은 파일 입출력 함수를 통해서 생성이 가능하며, 생성하는 과정에서 심볼릭 변수를 지정 가능하다.

심볼릭 파일을 생성하는 과정에서 가장 중요한 것은 해당 파일의 파일 포맷을 확인하고 심볼릭 영역을 지정하는 것이다. 각각의 파일별로 고정된 값이 있고 변하는 값이 존재한다. 즉, 원하는 영역을 심볼릭 적

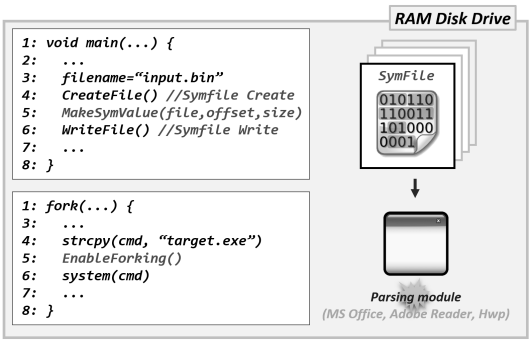


Fig. 5. Symbolic file creation with RAM Disk

용을 하기 위해서 파일의 포맷을 파악하는 것은 매우 중요하다. 따라서 해당 파일의 포맷을 먼저 확인하고, 분기에 영향을 미치는 영역을 심볼릭 변수로 지정한 후 심볼릭 파일을 생성을 진행해야 한다.

생성된 심볼릭 파일은 분석 대상 프로그램을 통해 로드되고 전체적인 프로그램에 수행에 있어서 내부 경로를 탐색하는 DSE 기술을 적용시킬 수 있다.

3.2 탐색 경로 지점 선정

소프트웨어를 분석하는 과정에서 탐색 경로를 지정해주는 것은 분석 대상 프로그램에서 실행 가능한 경로를 미리 배정해주는 것과 같다. 이는 프로그램에서 실행 가능한 경로의 수가 기하급수적으로 늘어남으로써 경로 탐색에 소모되는 시간 및 자원이 고갈되어 성능이 급격하게 저하되는 문제를 일시적으로 차단할 수 있다. 이러한 경로 폭발 문제는 주로 DSE 수행 과정에서 대부분 발생하는 문제로 동일한 경로의 코드를 반복 또는 중복해서 탐색하기 때문에 발생한다.

예를 들어, 루프(loop)는 코드를 반복해서 실행하는데 이 루프에 조건문이 존재하고, 조건문에 심볼릭 변수가 포함되어 있다면, 해당 루프의 반복자 값만큼 경로를 생성할 것이다. 특히 반복자(iterator)가 심볼릭 변수로 정의되어 있는 경우, 무한한 state를 생성하게 될 것이다. 이러한 경우를 제외 하더라도 분석 대상 프로그램 자체가 복잡하고 규모가 크다면, 프로그램 전체를 분석하기 위해 굉장히 많은 경로가 생성되고 탐색될 것이다. 대부분의 소프트웨어에 DSE 기술을 적용시킬 경우 경로 폭발 문제가 발생할 여지는 상당히 크다고 볼 수 있다. Fig. 6과 같이 복잡한 대상의 소프트웨어를 분석하기 위해서는

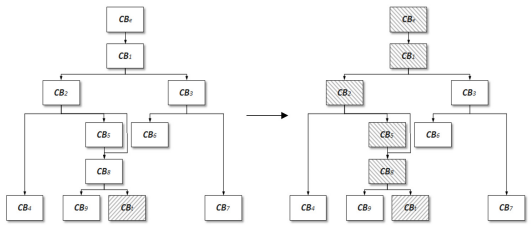


Fig. 6. Navigation Path Selection

사전에 탐색 경로 지점을 선정해주는 것이 중요하다.

3.3 탐색 종료 시점 선정

DSE 도구는 자체적으로 버그를 탐지하는 것이 불가능하다. 따라서 분석 대상 프로그램에 DSE 기술을 적용한다 할지라도 종료 시점을 따로 선정해주지 않을 경우, state가 종료되지 않고 계속해서 생성되는 상황이 발생한다. 결국 이로 인해 메모리 부족 현상이 발생하고 프로그램은 동작을 중단하게 된다. 뿐만 아니라 state가 종료되는 시점의 생성되는 테스트 케이스 또한 추출하지 못하게 된다. 원하는 결과를 얻기 위해서는 적절한 종료 시점 선정을 통해 경로 탐색을 중단하고, 테스트 케이스를 추출하는 방법이 요구된다.

DSE를 이용한 경로 탐색 종료 시점을 정의하기 위해 프로세스 상태를 모니터링 한다. 프로세스는 주어진 목적을 수행하기 위해 일련의 절차를 거친다. 프로세스는 CPU의 자원을 할당 받아 명령을 수행하고, 이벤트 발생을 대기하고, 이벤트 처리를 하는 과정에서 상태가 계속 변화한다. 이러한 변화의 모니터링을 통해 더 이상 연산이 진행되지 않을 때 즉, CPU 사용률이 0에 가까워지는 경우 경로 탐색을 중단한다. 또한 이때 프로세스의 디스크 I/O 또한 0이 된다.

프로세스 유휴 상태 탐지 알고리즘은 Table 2에서 제시한 3 단계로 진행된다. 먼저, 유휴 상태 탐지 목표 프로세스가 프로세스 리스트에 등록되는지를 확인하고, 프로세스 ID를 획득하기 위해 스냅 샷을 획득한다. 이를 통해 대상 프로세스의 존재 여부를 탐색한다.

두 번째로 이전에 획득한 프로세스 ID를 이용해 프로세스 핸들을 얻고 접근 권한을 획득한다. 이후 기준 시간을 중심으로 단위 시간당 프로세스의 CPU와 디스크 사용률을 계산한다. CPU의 시간정보를 획득하고 획득한 정보를 바탕으로 프로세스의 커널 및 유저모드에서의 실행시간을 측정한다. 그리고 대상 프로세스의 CPU 점유율을 측정한다.

Table 2. Detect Process Idle Algorithm

```
INPUT          s : Process Name
DECLARE        p: Process ID
               h : Process Handle
               d1, d2 : DISK State
               c1, c2 : CPU State
               C : CPU Usage
               D : DISK Usage
BEGIN
 DO p ← NameToPID(s)
 WHILE p is Zero
 h ← openprocess(p)
  WHILE p
    c1 ← Get CPU State
    d1 ← Get Process I/O Counter
         sleep unit Time
    c2 ← Get CPU State
    d2 ← Get Process I/O Counter
    C ← Compare c1 with c2
    D ← Compare d1 with d2
    IF C is Under 0.1% AND D is Zero
       Terminated state
       return
    p ← NameToPID(s)
END
```

마지막으로 측정한 단위 시간당 CPU 사용률이 0.1% 이하, 디스크 사용률이 0일 경우 해당 프로세스에서 더 이상 명령어의 수행과 저장장치로의 자료 이동이 없다고 판단하고 해당 프로세스를 유휴 상태로 마킹한다. 유휴 상태가 아닌 것으로 판단될 경우 이전 단계로 돌아가 CPU 및 디스크 사용률을 재측정하는 과정을 반복하여 프로세스가 유휴 상태로 변환될 때까지 프로세스를 모니터링 한다. 프로세스가 유휴 상태로 판단되면 state 종료 함수를 호출하여 프로그램 분석을 중단한다. state가 종료되면 도달한 경로까지의 테스트 케이스를 획득한다.

Fig. 7 은 입력 파일 기반 소프트웨어 분석을 위한 전체적인 동적 기호 실행 프레임워크이다. 입력 파일을 기반으로 생성된 심볼릭 파일은 경로 지점 선정 및 종료시점 선정의 분석 메커니즘을 통해 효율적인 입력 파일 기반 동적 기호 실행이 가능하다.

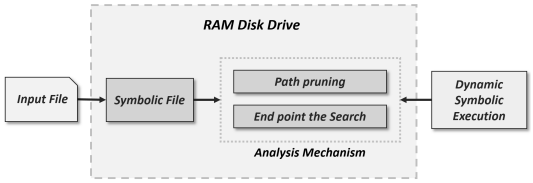


Fig. 7. The Framework of our testing method

IV. 실험 및 평가

본 장에서는 제안한 입력 파일 기반 동적 기호 실행 과정에 따른 내용을 실험하고 그 결과에 대한 내용을 분석한다.

4.1 동적 기호 실행 과정

우리는 동적 기호 실행 도구 중 하나인 S2E 에 기반하여 설계를 진행하였다. S2E에서는 QEMU 가상머신과 KLEE 기호 실행 엔진, LLVM 툴 체인을 이용하여 바이너리 분석을 효과적으로 진행할 수 있다. 입력 파일 기반 동적 기호 실행의 실험 방법으로 먼저 분석 환경을 구축하고 대상 소프트웨어의 선정 및 평가 계획에 따라 분석을 수행하였다. 분석 환경의 전체 구성은 Fig. 8과 같다. 전체적인 실험은 각 소프트웨어에 따른 DSE를 수행하고, 수행 결과로 발생하는 각각의 로그 파일 정보를 저장한다.

Sample은 입력 파일 기반 동적 기호 실행 분석과 기존의 함수 파라미터 및 로드된 메모리 주소를 이용한 수행 방법과 비교하여 올바른 결과 값을 생성해내는 지를 판단하기 위해 선정하였다. 그 외의 분석 대상 소프트웨어는 현재 대중적으로 많이 사용되고 있는 한글과 컴퓨터, 윈도우 드라이버, MS Word를 대상으로 선정하였다. 한글과 컴퓨터의 경우 실제로 취약점이 존재하는 모듈을 선정하여 동적 기호 실행을 수행하였다. 또 윈도우 드라이버 파일 및 MS Word 분석의 경우 정적 분석을 통해 사전에 확인된 영역을 실제로 기호 실행을 통해 해당 영역 도달 및 테스트 케이스(test case) 생성을 확인하였다.

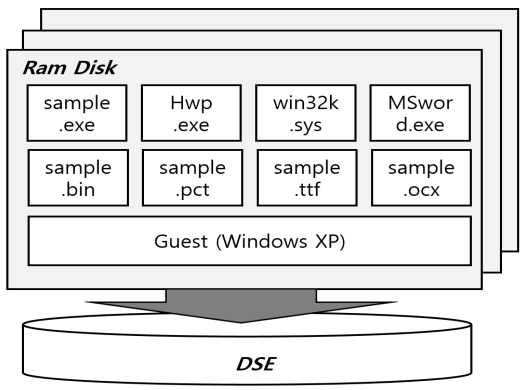


Fig. 8. Whole Components of Experimental Environments

대상 소프트웨어 별로 단 시간의 분석을 수행하였고, 이 때 발생하는 state 개수 및 종료된 state 개수를 평가의 척도로 이용하였다. 더 자세하게 분석에 소요된 시간, 모듈의 크기, 종료시점 선정 유무를 통해 탐색 여부를 확인하고 이를 비교하였다.

4.2 동적 기호 실행 결과

실험의 전체적인 환경은 Host Intel Core i7, 32GB Memory, Ubuntu OS(64bit)와 Guest Intel Core i7, 1GB Memory, Windows XP(32bit) 이다. 실제로 분석을 수행한 Guest 운영체제는 윈도우 XP 환경으로, 실험 당시 동적 기호 실행 도구인 S2E는 XP 환경에 최적화 되어 있다. 상위 버전의 운영체제에 대한 적용 문제는 아직 연구되고 있는 실정이다. 따라서 본 연구에서는 XP 환경에서 분석을 수행하였다.

소프트웨어별 실험 결과는 Table 3과 같다. Sample 실험의 경우 기존의 함수 파라미터 및 로드된 메모리 주소의 심볼릭 적용을 한 결과와 동일한 테스트 케이스를 생성하였다. 이를 통해 정상적으로 파일 기반 동적 기호 실행이 진행되었음을 확인하였다.

한글과 컴퓨터 실험의 경우 기본적인 이미지 파일(*.pct)을 심볼릭 파일로 분석 한 결과 평균 12시간에 걸쳐서 1,936개의 state를 생성하였고, 이 중에서 27 개의 종료된 state를 생성하였다. 또한 실제 취약한 지점에 도달할 수 있는 테스트 케이스를 추출한 후 크래시를 발생시키는 파일을 획득할 수 있었다.

윈도우 드라이버는 win32k.sys 내에 존재하는 영역을 탐색하는 실험 이였다. 먼저, 목표 지점이 명시되었을 경우 단 시간에 해당 지점까지의 21개의 모든 state를 생성하였다. 그 외 목표 지점이 명확하지 않은 기준에서도 해당 지점까지의 20개의 모든 state를 생성하였다. 하지만 이외에 목표지점이 명

Table 3. Test case of the Experimental Results Using DSE with File based of Software

Target Software	search	Target point	Module Size	Symbolic Byte	Time	Tatal state	Finished State
Sample.exe (*.bin)	O	X	59KB	2byte	5min	103	21 (All State)
Hwp.exe (*.pct)	O	impct9!ImsReadChar	157KB	30byte	12hour	1,936	27 (1 crash state)
win32k.sys (*.ttf)	O	win32k!FindBlocStrike	1,802KB	12bytes	2min	75	21 (All State)
win32k.sys (*.ttf)	O	X	1,802KB	8bytes	8min	145	20 (All State)
win32k.sys (*.ttf)	X	X	1,802KB	12bytes	17min	560	0 (Path explosion)
MS Word.exe (*.ocx)	O	wwwlib!GetAllocConters	18.4MB	12byte	20min	90	25 (All State)

확하지 않을 경우 상태폭발이 일어나는 경우도 있었다. win32k.sys는 다른 모듈들에 비해 크기가 상대적으로 크고 복잡하였다. 따라서 복잡한 모듈 분석의 경우 목표 지점을 명확하게 지정해주지 않고, 또 심볼릭 바이트를 상대적으로 크게 할 경우, 경로 폭발이 일어날 수 있다는 단점이 존재한다. 이는 어느 정도 사전 분석을 통해서 분석 모듈의 경로를 지정해주는 경로 가지치기 기법을 통해서 일시적으로 해결이 가능하다.

MS Word 의 경우 또한 상대적으로 모듈이 크고 복잡했기 때문에 종료시점을 지정하여 실험을 진행하였다. 실험 결과 단 시간의 해당 지점까지의 25개의 모든 state를 생성을 완료하였다.

V. 결 론

본 논문에서는 소프트웨어의 분석을 위한 입력 파일기반 동적 기호 실행 방법을 제안하였다. 관련 연구로는 대표적인 동적 기호 실행 도구인 KLEE 와 S2E에 대한 개요와 기존에 도구들에 의해 연구되었던 동적 기호 실행 방법을 분석하였다. 그 결과 기존 도구의 소프트웨어 분석의 문제점과 한계점을 발견할 수 있었다.

본 논문에서는 입력 파일 기반의 소프트웨어를 분석하기 위해 심볼릭 변수 정의, 탐색 경로 지점 선정, 탐색 종료 시점 선정 문제를 해결하였다. 심볼릭 변수 정의는 램 디스크를 활용한 심볼릭 파일 생성 방법을 이용하였다. 탐색 경로 지점 선정은 유도된 경로 탐색 기법을 통한 경로 재선정 기법을 적용하였다. 탐색 종료 시점 선정은 CPU 및 디스크 사용률을 재 측정하여 프로세스의 유휴 상태 탐지기법을 적용하였다. 이렇게 분석을 진행한 결과 기존에 함수 파라미터 및 메모리 주소를 통한 동적 기호 실행 기법뿐만 아니라 입력 파일 기반 분석도 가능했다. 이를 통해 복잡한 소프트웨어의 입력 파일 기반 동적 기호 실행을 효율적으로 할 수 있음을 보였다. 제안하는 프로토타입 시스템을 적용하기 위해 선택적기호 실행이 가능한 S2E 도구를 활용했다. 실제 바이너리 소프트웨어에 적용한 결과 효과적으로 해당 지점에 도달하는 테스트 케이스를 생성할 수 있었고, 이는 입력파일 기반 DSE 자동화 분석에 활용될 수 있음을 보였다.

References

[1] Symantec. "2016 Internet Security Threat Report", 2016.

[2] IBM "IBM X-Force Threat Intelligence Report", 2016.

[3] Heelan, Sean. Automatic generation of control flow hijacking exploits for software vulnerabilities. 2009.

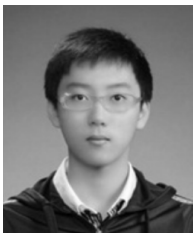
[4] Avgerinos, Thanassis, et al. "Automatic exploit generation." Communications of the ACM 57.2, pp. 74-84, 2014.

[5] Cha, Sang Kil, et al. "Unleashing mayhem on binary code." Security and Privacy (SP), 2012 IEEE Symposium on. IEEE. pp. 380-394, 2012.

[6] Huang, Shih-Kun, et al. "Software crash

analysis for automatic exploit generation on binary programs." IEEE Transactions on Reliability 63.1, pp. 270-289, 2014.
〔7〕 Cadar, Cristian, et al. "Symbolic execution for software testing in practice: preliminary assessment." Proceedings of the 33rd International Conference on Software Engineering, pp. 1066-1071, 2011.
〔8〕 Cadar, Cristian, Daniel Dunbar, and Dawson R. Engler. "KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs." OSDI, Vol. 8. pp. 209-224, 2008.
〔9〕 Sen, Koushik, Darko Marinov, and Gul Agha. "CUTE: a concolic unit testing engine for C." ACM SIGSOFT Software Engineering Notes. Vol. 30. No. 5, pp. 263-272, 2005.
〔10〕 Godefroid, Patrice, Nils Klarlund, and Koushik Sen. "DART: directed automated random testing." ACM Sigplan Notices. Vol. 40. No. 6, pp. 213-223, 2005.
〔11〕 Burnim, Jacob, and Koushik Sen. "Heuristics for scalable dynamic test generation." Automated Software Engineering, 2008. ASE 2008. 23rd IEEE/ACM International Conference on, pp. 443-446, 2008.
〔12〕 Chipounov, Vitaly, Volodymyr Kuznetsov, and George Candea. "S2E: a platform for in-vivo multi-path analysis of software systems." ACM SIGPLAN Notices 46.3, pp. 265-278. 2011.
〔13〕 Bellard, Fabrice. "QEMU, a fast and portable dynamic translator." USENIX Annual Technical Conference. FREENIX Track, pp. 41-46, 2005.
〔14〕 Lattner, Chris, and Vikram Adve. "LLVM: A compilation framework for lifelong program analysis & transformation." Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization. IEEE Computer Society, pp. 75, 2004.

〈저 자 소 개〉



박 성 현 (Sunghyun Park) 학생회원
2016년 8월: 전남대학교 컴퓨터정보통신공학 공학사
2016년 9월~현재: 전남대학교 정보보안협동과정 석사과정
〈관심분야〉 취약점 분석, 악성코드 탐지, 시스템 보안

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

김 휘 성 (Hwisung Kim) 학생회원
2015년 2월: 전남대학교 컴퓨터정보통신공학 공학사
2015년 3월~현재: 전남대학교 정보보안협동과정 석사과정
〈관심분야〉 취약점 분석, 악성코드 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