JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 25, No. 7, pp. 1873-1881, Jul. 2024

Check for updates

WinEco: Windows Time Travel Debugging 기반 Harness 반자동 생성

김 동 준[1] · 손 태 식[2*]
[1]아주대학교 사이버보안학과 학사과정
[2]아주대학교 사이버보안학과 교수

WinEco: Semi-automatic Harness Generation based on Windows Time Travel Debugging

Dongjun Kim[1] · Taeshik Shon[2*]

[1]Bachelor's Course, Department of Cyber Security Ajou University, Suwon, Gyeonggi-do 16499, Korea
[2]Professor, Department of Cyber Security, Ajou University, Suwon, Gyeonggi-do 16499, Korea

[요 약]

퍼징은 현재 많은 취약점을 찾는 데에 활용되고 있는 기술이다. 그동안의 퍼징 커뮤니티는 AFL (American Fuzzy Lop)을 기반으로 하여 Unix 기반 시스템에 적합한 방향으로 많이 발전되었다. 하지만 그동안의 연구를 윈도우 애플리케이션에 적용하는 데는 많은 어려움이 존재한다. 두 시스템의 가장 큰 차이는 harness 프로그램의 작성 여부로, 이는 분석가의 큰 노력이 필요하며 필연적으로 False Positive나 False Negative가 존재할 수밖에 없다. 본 연구에서는 윈도우의 TTD (Time Travel Debugging) 기술을 기반으로 harness를 반자동 생성할 수 있는 프레임워크인 WinEco를 제안한다. WinEco는 프로그램의 코드 흐름, 데이터 흐름, 의존성 등을 분석하여 퍼징에 최적화된 harness를 생성한다. WinEco의 정당성을 검증하기 위해 총 3개의 윈도우 상용 소프트웨어를 대상으로 harness를 생성하였고, WinAFL에 적용해 본 결과 총 10개의 Unique 버그를 찾았다.

[Abstract]

Fuzzing has emerged as a common technique for identifying vulnerabilities. Historically, fuzzing has evolved significantly based on AFL predominantly for Unix-based systems. However, applying these research advancements to Windows applications presents substantial challenges. The primary difference between the two systems lies in the creation of harness programs, which demands significant effort from analysts and inevitably leads to false positives or false negatives. In this study, we propose WinEco, a framework that automates the generation of harnesses based on time travel debugging in Windows. WinEco analyzes program code flow, data flow, and dependencies to create harnesses optimized for fuzzing. To validate WinEco's effectiveness, harnesses were generated for three Windows applications, and subsequent testing with WinAFL revealed a total of 10 unique bugs.

색인어 : 퍼징, 윈도우, 하네스, 취약점, 자동화
Keyword : Fuzzing, Windows, Harness, Vulnerability, Automation

http://dx.doi.org/10.9728/dcs.2024.25.7.1873

Received 01 June 2024; **Revised** 24 June 2024
Accepted 18 July 2024

***Corresponding Author; Taeshik Shon**

Tel: +82-31-219-3321
E-mail: tsshon@ajou.ac.kr

Ⅰ. 서 론

오늘날 퍼징은 소프트웨어 취약점을 찾는 핵심 기술로 활용되고 있으며 여러 연구들을 기반으로 발전되고 있다. 하지만, 지금까지의 퍼징 커뮤니티는 Unix 기반의 애플리케이션이나 커널을 타겟으로 한 방향으로 많이 발전하였고, 이로 인해 윈도우 애플리케이션에 곧바로 적용하기에는 여러 가지 제약이 존재한다. 그래픽 인터페이스의 존재나 윈도우 운영체제의 구조적인 특성 등이 그 주된 이유이다. 이를 개선하기 위해 WinAFL[1], Jackalope[2], Winnie[3], kAFL[4] 등의 프로젝트들이 나왔으며, 이러한 프로젝트들은 윈도우 환경에서도 효과적으로 퍼징을 수행할 수 있도록 도와주는 프레임워크를 제공한다. 해당하는 모든 기술들은 반드시 harness라고 불리는 프로그램을 작성해야 하는데, 이 과정은 정적 분석 및 동적 분석을 요구하기 때문에 상당히 많은 시간과 노력이 필요하다. 또한, 실제 프로그램과 완전히 같은 환경을 구성할 수 없기 때문에 오탐이 발생할 수밖에 없다는 문제를 가진다.

본 논문에서는 기존 기술들의 문제점을 개선하여 TTD (Windows의 Time Travel Debugging) 기술을 기반으로 한 자동 harness 생성 프레임워크인 WinEco를 제안한다. TTD 기술은 프로그램의 실행을 시간을 거슬러 가며 디버깅할 수 있는 환경을 제공한다. 이를 통해 윈도우 애플리케이션의 모든 코드 흐름과 메모리 및 레지스터 상태를 기록할 수 있으며, 이를 통해 프로그램의 실행을 정확하게 분석하고 재현할 수 있게 한다. WinEco는 TTD를 이용하여 프로그램의 코드 흐름, 데이터 흐름과 의존성을 분석하여 퍼징에 가장 적합한 harness를 생성해 준다. WinEco의 정당성을 검증하기 위해 총 3개의 윈도우 애플리케이션에 WinEco를 적용하여 harness를 생성하였다. 생성된 harness는 WinAFL을 이용하여 24시간 동안 퍼징을 수행하였고, 총 10개의 취약점을 찾아냈다. 이를 통해 WinEco가 윈도우 애플리케이션에 대해 효과적으로 harness를 생성하고 퍼징을 수행할 수 있음을 입증하였다.

본 연구의 주요 내용은 다음과 같다.

- 윈도우 애플리케이션의 특성을 분석하여 harness 생성을 어렵게 하는 요인들과 이를 극복하는 방법을 분석한다.
- TTD의 핵심 기술을 분석하고, 이를 이용하여 자동 harness 생성 프레임워크인 WinEco를 제안한다.
- WinEco를 이용하여 실제 윈도우 애플리케이션에 대해 harness를 생성하고, 이를 기반으로 퍼징을 수행하여 WinEco의 정당성을 검증한다.

Ⅱ. 배경지식과 관련 연구

2-1 퍼징의 개념

퍼징은 프로그램 기능을 자동으로 유효성을 검사하고 보안 취약점을 찾는 떠오르는 소프트웨어 테스트 기술이다. 프로그램 입력을 무작위로 변형하여 큰 데이터 집합을 생성하고 각 입력을 프로그램에 전달한다. 프로그램 실행을 비정상적인 동작으로 감지하는데, 이는 크래시, 행과 같은 프로그램 상태를 포함한다. 최근의 퍼징은 오픈 소스 프로젝트에서 수천 개의 취약점을 발견해 냈다. 실제 구글의 OSS-Fuzz 프로젝트에서 총 850개의 오픈 소스 프로젝트로부터 8,800개의 취약점과 28,000개의 버그를 찾아내었다[5].

대부분의 퍼저들은 GreyBox, feedback-guided 퍼징 방식을 채택하고 있다. 프로그램에 대한 코드 계측을 사용해서 유용한 피드백(e.g., 코드 커버리지)을 수집한다. 피드백은 하나의 입력이 프로그램의 내부상태를 얼마나 통과했는지 알려준다. 따라서 더 많은 양의 코드를 탐색할 수 있게 되고 이에 따라 버그를 발견할 확률이 높아진다(그림 1 참고).

Unix 시스템을 기반으로 하는 최신 퍼저들은(AFL-기반 퍼저) Cloning Syscall(e,g., fork())를 사용해서 더 빠르고 더 안정적으로 퍼저를 만들어 냈다[6].

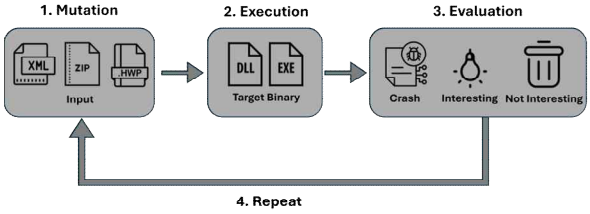

그림 1. 퍼징 개요
Fig. 1. 퍼징 Overview

2-2 Harness의 개념

일반적으로 소프트웨어의 특정 기능을 퍼징이 가능한 형태로 만든 것을 harness라고 부른다. 일반적인 harness는 다음과 같은 조건을 만족해야 한다. 먼저 GUI를 완전히 CLI로 대체할 수 있어야 한다. 두 번째로 목표함수를 부르기 전 함수가 필요로 하는 메모리 세팅이나 함수의 인자를 세팅해야 한다. 세 번째로 목표함수 내부에서 사용하는 콜백, 가상 함수 테이블, 의존성 있는 라이브러리들을 세팅해 주어야 한다. 그림 2에서 앞서 설명한 특징들을 만족하는 harness 예제를 확인할 수 있다.

```
// 1) Declare structures and callbacks
int callback1(void* a1, int a2) { ... }
int callback2(void* a1) { ... }
// 2) Prepare file handle
FILE *fp = fopen("filename", "rb");
// 3) Initialize objects, internally invoking ReadFile()
int *f0_a0 = (int*) calloc(4096, sizeof(int));
int f0_ret = JPM_Document_Start(f0_a0, &callback1, &fp);
if (f0_ret){ exit(0); }

// 4) Get property of the image
int f1_a2 = 0, int f4_a2 = 0;
JPM_Document_Get_Page_Property((void *)f0_a0[0], 0xA, &f1_a2);
...
JPM_Document_Get_Page_Property((void *)f0_a0[0], 0xD, &f4_a2);

// 5) Decode the image
JPM_Document_Decompress_Page((void *)f0_a0[0], &callback2);

// 6) Finish the harness
JPM_Document_End((void *)f0_a0[0]);
```

그림 2. Harness 예제
Fig. 2. Harness example

그림 2를 보면 먼저 목표함수에서 사용되는 콜백 함수들을 선언해 준다. 그리고 입력으로 주입될 파일 핸들을 연다. 목표 함수가 사용하는 객체를 만족시키기 위해 초기화 루틴을 수행한다. 다음으로 실제 목표함수를 실행한다. 마지막으로 사용된 객체를 해제하는 과정을 거친다.

이렇게 복잡한 API의 순서, 콜백과 함수 인자 설정과 같은 복잡성으로 인해 하나의 함수를 호출하는 것만으로는 성공적인 퍼징이 될 수 없다. 따라서 퍼징에 적합한 harness를 만들기 위해 본 연구에서 정의하는 harness는 다음과 같은 조건을 가진다.

1. Harness가 하나의 입력을 받고 정상 종료되어야 한다.
2. Harness는 사용자의 상호작용 없이 동작할 수 있어야 한다.
3. 퍼징 과정 중에 사용되는 객체의 해제나 핸들의 해제를 보장하여야 한다.
4. 목표함수의 인자들과 callback과 같은 의존성은 정확하게 분석되어야 한다.

2-3 Windows Time Travel Debugging

TTD(Time Travel Debugging) [7] 마이크로소프트에서 개발한 디버깅 기술이다. 프로그램의 실행 과정을 모두 기록하여 프로그램의 앞뒤로 디버거를 제어하여 정확한 분석이나 Triage를 가능하게 해주는 기술이다.

특정 함수가 어느 시점에 어떤 인자로 호출되었는지 확인할 수 있으며 특정 heap 메모리가 어떤 시점에서 할당되었고 어떤 시점에서 해제되었는지도 확인할 수 있다. 또한 각각의 스레드 별로 추적할 수 있으므로 윈도우 애플리케이션의 멀티스레드 환경에도 적합하다.

그림 3은 실제 TTD를 통해 Win32 API인 Kernel32!CreateFileW에 대한 추적 결과를 보여준다.

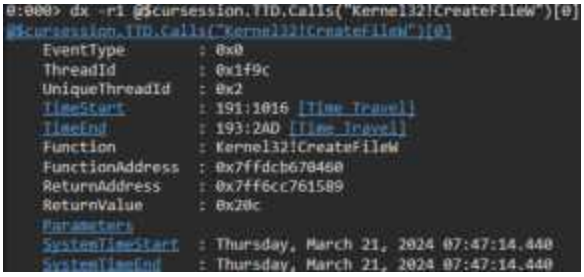

그림 3. TTD기능 예시
Fig. 3. TTD feature example

2-4 관련 연구

관련 연구를 살펴보면 오픈 소스에서 퍼징을 위해 자동화된 harness를 생성하는 연구는 Fudge[8], FuzzGen[9]과 같은 연구들이 있다. 그리고 윈도우 애플리케이션을 대상으로 반자동화된 harness 생성기를 제시한 WINNIE가 있다.

Fudge와 FuzzGen의 경우 오픈 소스를 대상으로 라이브러리를 목표로 하여 퍼징을 할 수 있는 harness를 만드는 것을 목표로 한다. FuzzGen은 LLVM 기반으로 A²DG(API Dependency Graph)이라는 그래프를 만들어 harness를 생성해 준다. Fudge도 LLVM을 이용해서 harness를 생성하며 오픈 소스 프로젝트의 더 많은 코드 커버리지와 자동화된 harness 생성으로 지속적인 퍼징 서비스(OSS-Fuzz)에 통합하는 것을 목표로 한다. 윈도우 애플리케이션은 대부분 Closed Source이기 때문에 소스 코드에 의존성이 있는 Fudge와 FuzzGen은 한계가 있다.

윈도우 애플리케이션을 대상으로 연구된 WINNIE의 경우 Intel-PT[10] 기반의 추적을 기반으로 harness를 생성해 주며 기존 윈도우에 존재하지 않는 기능인 Fast-Cloning의 구현을 통해 퍼징 속도와 안정성으로 높였다. 하지만 하드웨어에 대한 의존성이 존재하는 점, Intel-PT 드라이버에 대한 안정성 문제가 존재한다는 점 등 한계점이 존재했다.

Ⅲ. 문제점과 해결책

3-1 Windows Application Properties

윈도우 운영체제에서 배포되는 대부분 프로그램들을 Closed Source로 운영된다. 즉 오픈 소스와 달리 소스 코드가 제공되지 않는다. 이러한 특성은 특정 부분을 따로 분리하여 컴파일할 수 없다. 코드 커버리지와 같은 피드백을 수집하기 위해서는 기본 블록 사이에 계측을 위한 코드를 삽입해야 하는데 이미 컴파일된 바이너리를 대상으로 쉬운 일이 아니다. 결국 동적 계측(Dyinamo-RIO)이나 하드웨어를 이용하는 방식(Intel-PT)으로 코드 커버리지를 수집하여야 한다. 이는 퍼징 속도를 급격하게 감소시키는 요인 중 하나이며 사용하는 라이브러리에 따라 오류가 발생할 가능성이 있다. 이는 False Positive나 False Negative에 영향을 줄 수 있으며 전체 퍼징 품질에 영향을 줄 수밖에 없다.

두 번째로 윈도우 애플리케이션은 대부분이 GUI 기반으로 사용된다. 퍼징은 CLI를 바탕으로 지속적이고 자동으로 임의의 값을 프로그램에 주입해야 한다. 하지만 GUI는 이러한 과정을 불가능하게 한다. 또한 GUI 프로그램 자체가 화면을 구성하기 위해 많은 오버헤드가 발생하므로 퍼징 속도를 급격하게 떨어지는 원인이 된다.

3-2 Difficulty for Generate Harness

윈도우 애플리케이션 퍼징에서 harness의 생성은 필수적인 요소이다. 앞서 언급한 대로 좋은 harness를 만들어야 퍼징의 전체적인 질을 높일 수 있다. 하지만 harness의 생성은 어려운 작업이다. 만약 특정 함수를 퍼징을 위해 harness로

만들어서 퍼징을 수행해 유의미한 크래시를 발견했더라도 실제 애플리케이션에 목표 함수 이전에 입력에 대한 검증 로직이 존재하면 실제 취약점으로 연결될 수 없다. 따라서 목표함수 선정은 높은 품질의 퍼징을 수행하기 위해서 가장 중요한 단계이다.

Harness의 안정성 역시 퍼징에서 중요한 요소이다. 퍼징은 24시간 혹은 그보다 더 오랜 시간 동안 지속적이고 안정적으로 수행되어야 한다. 하지만 여러 가지 변수로 인해 퍼저의 안정성이 떨어지고 퍼저가 종료되어 버리면 이는 전체 퍼징 결과에 영향을 미칠 수밖에 없다.

3-3 해결책

우리는 이러한 문제를 해결하기 위해 TTD (Time Travel Debugging) 기술을 기반으로 반자동화된 harness 생성기인 WinEco를 제시한다. WinEco는 TTD 기술을 이용해 목표 프로그램의 TTD 녹화 데이터를 기반으로 ① 제어 흐름 분석 ② 데이터 흐름 분석 ③ 목표함수 선택 ④ 인자 분석 및 복구 ⑤ 의존성 분석 및 복구 ⑥ 최종 Harness 생성 총 6단계를 통해 최종 Harness 코드를 생성한다.

TTD 기능을 따로 라이브러리화시켜서 사용하기 위해서 TTDReplay.dll, TTDReplayCPU.dll에 존재하는 Exported Function 들과 Com 객체를 이용하여 구현하였고 MSDN에 있는 공식 참고 자료[11]를 사용하였다.

WinEco는 Fudge, FuzzGen 연구에서 Closed Source 프로그램을 지원하지 못하던 점을 해결하고 Winnie가 사용하던 Intel-PT 기반 추적 및 생성 기술을 TTD로 대체함으로써 하드웨어에 대한 의존성을 제거하고 더 정확한 harness를 생성할 수 있다.

Ⅳ. 구현 및 동작 방식

4-1 전체 동작 개요

WinEco는 크게 4가지 종류의 입력을 통해 최종 결과물인 harness 코드를 만들어 낸다. 각각의 입력은 다음과 같다.

- TTD 추적 파일 : TTD을 이용하여 기록된 파일이며 크게 **.run, .idx, .out** 파일로 구성되어 있다. 프로그램의 특성과 기록하는 시간에 따라 그 크기가 매우 커질 수 있으므로 적절한 크기로 조절이 필요하다.
- 목표 바이너리 : EXE 혹은 DLL과 같은 실행 파일로 우리가 목표로 하는 기능(파일 처리, 이미지 처리)이 존재 하는 바이너리이다. 이 바이너리는 이후 프로세스에서 함수들의 주소를 계산해서 흐름 분석들에 사용된다.
- 입력 파일 : 우리의 목표는 특정 파일에 대한 처리를 퍼징하는 것이기 때문에 TTD 기록 때에 사용된 입력 파일이 필요하다. 이 파일은 이후 4-3에서 언급되는 데이터 흐름 분석에 필수적이다.
- IDB 파일 : IDB 파일은 IDA를 통해 분석된 파일로 이 정보는 이후 4-3에서의 데이터 흐름 분석과 4-5에서 언급되는 함수 인자 복구에 사용된다.

WinEco의 구현체는 크게 4단계의 스테이지를 통해 최종 harness를 생성해 준다. 모든 단계를 자동화하기에는 어려운 부분이 많이 존재하므로 사용자가 적절히 조절해야 하는 부분들은 기준으로 스테이지를 분리하였다.

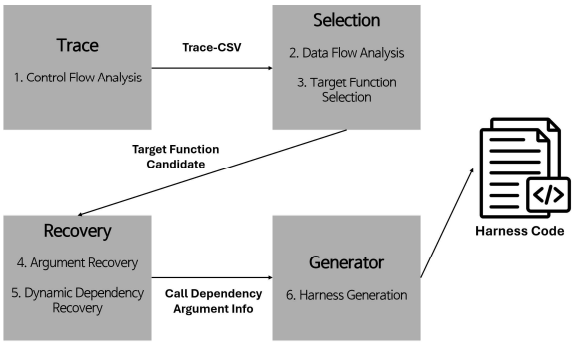

그림 4. WinEco 동작 개요
Fig. 4. Overview of wineco

각각의 단계는 다음과 같은 총 4단계로 처리되며 하나의 단계에서 다음 단계로 넘어갈 때 필요한 정보는 다음과 같다.

1. Trace : TTD 추적 파일을 바탕으로 전처리를 수행하는 단계이다. 이 단계에서 모든 함수 모든 함수 호출이 처리할 수 있는 형태로 변환되며 향후 단계에서의 오버헤드를 줄이기 위해 중요한 단계이다. 또한 가장 시간이 많이 드는 단계이다.
2. Selection : 앞서 전처리 된 그래프 데이터를 바탕으로 퍼징에 적합한 함수를 찾는다. 이때 목표함수를 선정하고 추적하기 위해 데이터 흐름 분석을 수행해 우리의 입력에 대한 흐름을 추적한다. 추적된 정보를 바탕으로 두 가지 종류의 Target Function Candidate를 찾아낸다.
3. Recovery : 앞서 선정된 목표함수를 성공적으로 퍼징하기 위해 함수의 인자들을 분석한다. 또한 애플리케이션 내부에서 사용되는 여러 의존성을 분석하고 기록한다.
4. Generator : 최종적으로 앞선 단계에서 분석된 정보들을 바탕으로 필요한 라이브러리들을 로드하고 목표함수 선언과 적절한 인자들을 설정해 줌으로써 최종적인 harness 코드를 WinAFL 스타일로 생성해 준다.

4-2 Control Flow Analysis

먼저 TTD를 통해 기록된 run 파일을 로드 하여 모든 call, Jmp, ret 명령을 추출한다. 앞서 언급한 윈도우 애플리케이션

의 멀티스레드 특성으로 인해 각각의 스레드에 따라 다른 trace_threadId.csv 파일을 생성한다. 이때 csv 파일에 기록되는 정보들은 그림 5와 같다.

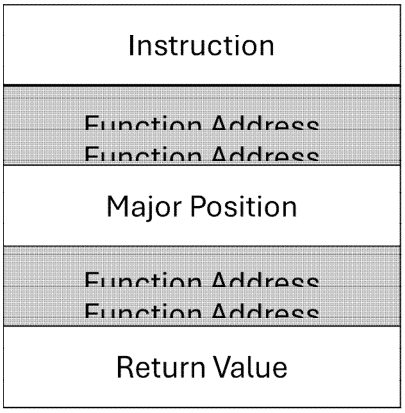

그림 5. Trace_xx.csv 파일 정보
Fig. 5. Trace_xx.csv file information

- Instruction : 해당 정보가 Call인지 Ret 인지 기록한다. 의미상으로 Call과 Jmp는 거의 같으므로 따로 구분하지 않는다.
- Function Address : Call 혹은 Ret 명령어가 호출된 주소를 기록한다. 이 정보는 이후 심볼링 단계에서 어떤 실행 파일의 어떤 함수에서 호출되었는지 변환된다.
- Major Position & Minor Position : TTD에는 현재 위치를 표현하는 포지션이 존재한다. 기본적으로 <XX:XX>와 같은 형태를 가지고 있다. 해당 값을 앞뒤로 나누어 기록한다.
- Return Value : 만약 호출되는 명령어가 Ret 이라면 함수가 종료되고 반환된다는 의미이므로 반환 값을 기록한다.

이렇게 기록된 정보는 가장 먼저 심볼링 단계에 들어간다. 함수 주소를 실행 파일들의 베이스 주소를 토대로 어떤 실행 파일의 어떤 함수에서 호출되었는지 기록한다. 만약 윈도우 내부 DLL들이라면 심볼이 존재하기 때문에 정확한 함수의 이름까지 심볼링 해준다. 심볼이 없는 함수라면 베이스 + 오프셋과 같은 형태로 심볼링을 수행한다.

이렇게 만들어진 CSV 파일을 토대로 함수 호출에 대한 그래프를 생성한다. 그래프 생성에는 오픈 소스 라이브러리인 networkx 패키지를 사용하였다. 각각의 스레드별로 graphml 형식의 그래프가 생성되고 그래프들은 이후 프로세스에서 가장 적합한 목표함수를 찾는 데 이용된다.

4-3 Data Flow Analysis

기본적으로 퍼징은 파일 포맷의 입력을 받는 소프트웨어에 적합하다. 일반적인 퍼저들 모두 파일 포맷의 입력을 퍼징 하도록 설계되어 있다. 따라서 WinEco 역시 목표 소프트웨어의 파일 처리 루틴을 목표로 삼는다.

기본적으로 윈도우의 경우 Kernel32 DLL에 Export되어 있는 CreateFileA, CreateFileW, OpenFile, ReadFile과 같은 파일 입출력과 관련된 API를 이용해서 파일을 열고 읽는다. 따라서 이러한 입력과 관련된 API들로부터 TTD에서 사용된 입력을 추적한다. 파일을 여는 동작의 경우 입력 파일의 경로 문자열을 추적하고 파일을 읽는 동작의 경우 입력 파일의 데이터를 추적하면 성공적으로 데이터 흐름을 추적할 수 있다.

윈도우 애플리케이션은 보통 멀티스레드로 동작한다. 이러한 스레드 간의 의존성을 분석하는 것은 매우 큰 시간과 노력을 필요로 하므로 자동화가 매우 어렵다. 따라서 WinEco는 데이터 흐름 분석 단계에서 파일 관련 API가 없는 Trace_CSV 파일은 무시하고 파일 관련 API가 있는 Trace_CSV 파일 만을 분석한다.

먼저 Trace_CSV 파일을 기반으로 python NetworkX 라이브러리를 통해 graphml 파일로 변환하여 그래프 형태로 API 흐름을 표현한다. 그리고 앞서 언급한 파일 관련 API가 호출되는 노드를 그래프에서 찾는다. 이렇게 찾아진 노드를 기반으로 우리의 입력 파일이 존재하는 적절한 함수들을 저장한다.

이렇게 찾아지는 함수들은 최소 1개 이상이며 이러한 함수들을 TFC(Target Function Candidate)로 정의하였다. TFC는 크게 두가지 종류로 나누어진다.

- Name Type TFC

이 경우는 함수의 인자가 우리가 입력으로 제공한 파일의 경로 혹은 파일명인 경우이다. 해당 함수의 Call Graph에서 하위 노드를 탐색하였을 때 파일 핸들을 여는 Win32 API(eg,.Kernel32!CreateFile, Kernel32!OpenFile)가 반드시 존재한다. Win32 API에서 리턴 되는 핸들 값을 바탕으로 핸들을 닫는 Win32 API(eg,. Kernel32!CloseFile, Nt!NtClose) 추적을 통해 해당 핸들이 언제 닫히는지 알 수 있다.

- Value Type TFC

이 경우 우리가 입력으로 제공한 파일의 값이 실제로 처리되는 함수이다. 즉 함수에 포인터 형태의 값이 인자로 전달되며 상위 노드를 탐색 하였을 때 반드시 파일 핸들로부터 값을 읽어오는 Win32 API(eg,. Kernel32!ReadFile)가 존재한다.

Name Type TFC의 경우 Call Graph에서 가장 먼저 파일 핸들을 여는 Win32 API의 노드를 모두 찾는다. 그리고 찾아진 노드를 기준으로 상위 노드를 탐색하며 함수의 인자로 파일 경로 혹은 파일 명인 노드를 찾아서 반환한다. 또한 해당 핸들이 닫히는 타이밍을 알아야 하므로 반환된 핸들을 추적하여 파일의 핸들이 닫히는 노드도 같이 반환한다. 즉 Name Type TFC의 경우 파일 핸들이 열리는 노드, 파일 핸들이 닫히는 노드 총 두 개의 함수를 반환해 준다.

두 번째로 Value Type TFC의 경우 실제로 파일을 읽어서 처리하는 함수를 의미한다. 해당 함수에서의 인자가 우리가 제공한 입력 파일의 값을 담고 있는 포인터이다. 이러한 노드를 탐색하기 위해 파일을 읽는 Win32 API가 반환한 포인터를 TTD의 하드웨어 브레이크 포인트를 통해 추적한다. 또한 해당 노드 내부에서 해당 포인터에 대한 다양한 연산을 수행해야 퍼징에 적합하므로 포인터에 대한 접근이 50회 이상인 노드를 Value Type TFC로 간주하고 반환해 준다.

4-4 Target Function Selection

앞선 Data-Flow 단계에서 선정된 목표함수 후보군을 바탕으로 가장 최적화된 목표함수의 위치를 선정한다. 앞서 Data Flow 분석에서 찾아진 Target Function Candidate 중 사용자가 선택한 값을, 함수를 기준으로 실제 harness에 사용하기에 가장 적절한 함수를 선택하는 단계이다. 이러한 단계는 굉장히 도전적인 과제이다.

앞서 배경지식의 harness의 조건에 대해서 언급했듯 다양한 조건이 맞아야 높은 품질의 퍼징을 수행할 수 있다. 따라서 이 단계에서 WinEco는 앞서 Data Flow 분석에서 찾아진 TFC에 대해서 lowest common ancestor (LCA)를 수행한다. 즉 Name Type TFC와 Value Type TFC를 공통 조상으로 갖는 가장 가까운 노드를 탐색하고 찾아진 노드를 최종 Target Function으로 선정한다. 그래프 탐색의 경우 앞서 만들어진 graphml 기반의 Call Graph를 통해 수행한다.

4-5 Argument Recovery

함수의 인자를 정확하게 복구하는 것은 매우 도전적인 과제 중 하나이다. 윈도우 생태계의 특성상 대부분의 바이너리가 C/C++로 작성되기 때문에 컴파일 타임에 많은 정보가 사라진다. 즉 우리는 원시 어셈블리로부터 원본 C코드를 유추해 내서 인자들의 자료형 혹은 구조체 형태를 알아내야 한다. 특히 구조체 혹은 공용체의 경우 원본 형태를 유추하는 것이 매우 쉽지 않다. 현재 IDA Pro, Ghidra, Binary Ninja와 같은 상용 분석 도구들이 컴파일된 바이너리의 어셈블리어를 바탕으로 C와 매우 유사한 형태로 분석해 준다. 이번 연구에서는 함수들의 인자를 정확하게 복구해내는 것이 목표가 아니기 때문에 본 연구에서는 IDA Pro를 사용하여 함수에 대한 인자들을 복구하였다.

IDA python 스크립트를 사용해서 모든 함수에 대한 호출 규약 분석 및 대략적인 자료형을 먼저 추출한다. 그리고 각각의 함수 인자에 대한 레지스터 매칭을 통해 TTD를 앞, 뒤로 재생하여 인자의 필요한 값을 찾고 기록한다.

4-6 Dynamic Dependency Recovery

대부분의 상용 소프트웨어는 C가 아니라 C++로 구현되어 있으므로 다양한 가상 테이블이나 동적 객체들이 존재한다. 이러한 객체들은 프로그램이 실행되고 난 이후 동적으로 생성되는 값들이기 때문에 실제 퍼징 프로세스 이전에 동적으로 설정해 주어야 한다. 여러 가지 종류의 동적 의존성이 존재하지만, WinEco는 다음과 같은 의존성만 고려한다.

- Dependency DLL

우리가 목표로 하는 DLL에서 사용하는 DLL들을 미리 메모리에 로드시켜 이후 로직에 문제가 없어야 한다. 대표적인 예로 한컴 오피스의 이미지 처리 코드에서 윈도우 내장 DLL인 gdi.dll을 사용한다.

- Dependency object

함수의 인자로 객체를 초기화하고 반환되는 포인터를 사용해야 하는 경우이다. 일반적으로 XX_init과 같은 형태의 함수들이며 그림 6에서 실제 harness 코드를 통해 확인 할 수 있다.

```
extern "C" __declspec(dllexport) __declspec(noinline) int fuzzme(wchar_t* input, wchar_t* output) {
    CArkManOpen _CArkManOpen = (CArkManOpen)((__int64)ark_x64 + 0x6300);
    DWORD ret = _CArkManOpen((CArkMan*)CArk, input, 0);

    CArkManExtractAllTo _CArkManExtractAllTo = (CArkManExtractAllTo)((__int64)ark_x64 + 0x7130);
    _CArkManExtractAllTo((CArkMan*)CArk, output, 0);

    CArkManClose _CArkManClose = (CArkManClose)((__int64)ark_x64 + 0x64F0);
    _CArkManClose((CArkMan*)CArk);


    return 0;
}
```

그림 6. 반디집 harness 예시 코드
Fig. 6. Example harness code for Bandizip

4-7 Generate Harness Code

현재 윈도우 퍼징에 사용하는 최신 퍼저는 크게 WinAFL과 Jackalope가 있다. 모두 오픈 소스로 공개되어 있으며 그동안 많은 취약점을 찾아왔다. WinEco는 WinAFL에 사용하기 적합한 형태의 코드를 만들어 주도록 설계되었다. WinAFL을 사용한 이유는 좀 다양한 동적 계측을 지원하며 최근 Jackalpoe에서 적용하고 있는 다양한 퍼징 기술을 추가하며 개발되고 있다. 따라서 WinAFL 형태의 harness 코드가 WinEco에 더 적합하다고 판단하였다.

WinEco는 기존에 선언된 서식 코드를 바탕으로 최종 harness를 생성한다. 전체적인 동작 과정은 다음과 같다.

1. 라이브러리 로드 : 먼저 목표 바이너리인 EXE 혹은 DLL을 Kernel32!LoadLibrary 함수를 통해 로드한다. 이때 로드되는 목표 바이너리에 의존성이 있는 DLL이 있다면 추가로 모두 로드한다.
2. 함수 선언 : 앞선 단계인 Control Flow Recovery 단계의 정보와 Argument Recovery 단계의 정보를 바탕으로 필요한 함수들을 선언한다. 함수들은 크게 아래 두 종류가 존재한다.
   - Exported Function : Exported Function의 경우 Kernel32!GetProcAddress 함수를 통해 로드한다. 정확한 오프셋 계산 없이 단순히 함수 이름을 통해 로드 할 수 있다.
   - Non Exported Function : Non Exported Function

의 경우 정확한 오프셋을 통해 직접 선언해 줘야 한다. 템플릿을 바탕으로 함수의 인자와 오프셋을 직접 선언해 준다.

3. Fuzzme 함수 구현 : 이 단계에서는 2단계에서 선언된 함수들을 바탕으로 실제 harness 코드를 작성한다. 필요한 메모리 세팅이나 함수 인자들을 적절하게 수정하고 가장 중요한 입력에 대한 처리를 수행한다.

V. 평 가

5-1 Harness 생성 및 퍼징 결과

우리는 WinEco를 통해 많은 상용 소프트웨어에 대한 harness 코드를 작성하였다. 다양한 소프트웨어에서 이미지 처리 로직, 파일 변환, 압축 해제와 같은 다양한 기능에 대해서 WinEco를 사용하여 harness 코드를 만들어 내었다.

바이너리 내부의 여러 의존성에 대한 완벽한 분석은 불가능하므로 퍼징을 수행하기 이전에 사용자의 적절한 수준의 분석을 추가하여 최종 harness 코드를 만들어 내었다.

우리는 다양한 상용 소프트웨어에 대해서 harness를 생성하여 퍼징을 수행하였다. 표 1에서 목표로 한 소프트웨어들, 입력값의 종류, 발견된 크래시를 확인 할 수 있다.

각각의 harness에 대해서 24시간 동안 I9-14900, 32-Core, 64GB 환경에서 퍼징을 진행하였다. 동적 계측의 경우 Dynamo-Rio를 사용하여 퍼징을 수행하였다. 이외에 Intel-PT, Syzygy와 같은 방식이 존재하지만, Dynamo-Rio에 비해 퍼저의 안정성이 떨어져서 해당 방식을 선택하였다. 아래 표 1을 통해 퍼징 수행 결과 얻은 크래시들과 목표 소프트웨어의 종류를 확인할 수 있다.

표 1. WinEco를 통해 만들어진 harness의 퍼징 결과
Table 1. 퍼징 results of harness created with WinEco

Program	Input	Target Binary	Crash
HWP-bmp	.bmp	HncBmp10.flt	1000+
HWP-tiff	.tiff	HncTiff10.flt	100+
HWP-wmf	.wmf	HncWmf10.flt	1000+
HWP-png	.png	HncPng10.flt	1000+
Bandizip	.zip	ark.x64.dll	4
Ezpdf-html	.pdf	PDF2HTML.dll	200+
Ezpdf-office	.pdf	PDF2OfficeDll.dll	300+

5-2 선행 연구와 비교

퍼징을 위한 자동화된 harness 생성은 선행 연구가 많지 않은 편에 속한다. 특히 윈도우 바이너리를 대상으로 하는 연구는 더욱더 드물다. 하지만 FuzzGen, Fudge 역시 Unix 시스템에서 비슷한 목표의 연구를 수행했으므로 FuzzGen, Fudge 역시 비교 대상에 포함하였다.

표 2. 선행 연구와의 비교
Table 2. Comparison with previous research

	FuzzGen	Fudge	Winnie	WinEco
Target OS	Linux/ Android	Linux	Windows	Windows
Binary Support	×	×	✓	✓
Control Flow	✓	✓	✓	✓
Data Flow	✓	✓	✓	✓
Hardware Dependency	–	–	×	✓

Fudge와 FuzzGen은 오픈 소스 소프트웨어에 대해 harness를 자동으로 생성하는 것을 목표로 한다. Fudge의 경우 라이브러리를 사용하는 기존 소스 코드에서 API 호출 순서를 추출하여 harness를 생성한다. FuzzGen은 소스 코드의 정적 분석을 통해 라이브러리의 API를 유추하고, 이를 이용하여 harness를 생성한다. 표 2에서 기존의 연구와 WinEco의 차이점을 확인할 수 있다. 가장 중요한 점은, Fudge와 FuzzGen은 일반적으로 리눅스 생태계에 속한 오픈 소스 프로젝트를 대상으로 하지만, 우리의 연구는 윈도우 환경의 소프트웨어들을 목표로 하므로 앞 장에서 서술한 여러 가지 문제를 해결해야 했고 이를 성공적으로 해결하였다.

Winnie의 경우 Intel-PT를 이용한 추적을 사용하여 여러 종류의 분석을 수행하여 최종 harness 코드를 생성하였다. 하지만 Intel-PT의 경우 IPT를 지원하는 CPU에 대한 하드웨어 의존성이 존재하고 프로그램의 이전 상태로 돌아갈 수 없으므로 여러 개의 추적 파일을 생성해야 하였다. 이는 분석 시간을 늘리고 각각의 추적 파일에 존재하는 힙, 스택 주소들이 다르므로 harness를 생성하는데 더 많은 시간과 노력이 필요하다.

WinEco는 Winnie와 같이 윈도우 소프트웨어에 대한 harness 반자동 생성을 목표로 하지만 기반 기술을 TTD로 바꾸어서 하드웨어 의존성을 없애고 프로그램의 상태를 앞, 뒤로 자유롭게 움직일 수 있다는 점을 이용해 단일 추적만으로 더 정확한 harness를 반자동 생성할 수 있다.

5-3 Founded Bugs

발견된 크래시 중에서는 오탐도 존재하고 중복된 크래시도 존재한다. 결과적으로 발견된 취약점은 총 10개로 표 3에서 자세한 취약점에 대한 정보를 확인 할 수 있다.

발견된 취약점들은 기본적으로 모두 서비스 거부 공격이 발생한다. 이뿐만 아니라 Use-after-free, integer-overflow, Stack-buffer-overflow, heap-buffer-overflow와 같은 버그 클래스는 익스플로잇을 통해 원격 코드 실행까지 도달할 수 있다. 특히 아직 국산 소프트웨어들은 최신 보호 기법이 적용되지 않은 사례가 많이 존재한다. 가장 기본이 되는 ASLR, DEP 와 같은 보호 기법이 특정 DLL에 적용되지 않은 것을 확

인할 수 있었다. 이는 공격자 관점에서 취약점을 악용하는 익스플로잇을 쉽게 해준다.

표 3. WinEco를 통해 찾은 취약점
Table 3. Founded crash with wineco

Program	Buggy File	Bug Type
BandiZip	ark_x64.dll	heap-buffer-overflow
EzPdfViewer	PDF2HTML.dll	Null-pointer-dereference
EzPdfViewer	PDF2HTML.dll	Stack-buffer -overflow
EzPdfViewer	Pdf2OfficeDll.dll	Use-after-free
EzPdfViewer	Pdf2OfficeDll.dll	Integer-overflow
EzPdfViewer	Pdf2OfficeDll.dll	Null-pointer-dereference
Hancom Office	HncWmf10.flt	Out-of-bound Read
Hancom Office	HncWmf10.flt	divide-by-zero
Hancom Office	HncBmp10.flt	heap-buffer-overflow
Hancom Office	HncPng10.flt	Denial-of-Service

5-4 한계점

완전히 자동화된 harness 생성은 어려운 과제이다. 많은 종류의 바이너리 분석 기술과 다양한 동적 분석 기술이 결합되어야 완전히 자동화된 harness 생성기를 만들어 낼 수 있다. 또한 WinEco가 harness를 생성하지 못하는 3가지 경우가 존재 하였다.

- High Complexity Binary : 마이크로소프트사의 MS-WORD의 경우 WinEco를 이용한 harness 생성에 실패하였다. 이론적으로는 WinEco를 통해 만들어 낼 수 있으나 너무 많은 파일 입출력, 매우 복잡한 파일 처리 로직, 매우 큰 크기의 구조체와 공용체들과 같이 여러 복합적인 문제로 적절한 TFC를 탐색하지 못하였고 이에 따라 이후 과정인 Target Function Selection 단계 역시 정상적으로 동작하지 못하였다.

- Multi-Process Architecture : 최근 Chrome, Adobe와 같은 소프트웨어들은 단순히 하나의 프로세스에서 모든 행위를 수행하지 않는다. 샌드박스 메커니즘을 구현하기 위해 멀티 프로세스 아키텍처를 통해 프로그램을 구현하고 있다. 따라서 현재 WinEco는 단일 스레드를 기준으로 harness를 생성하기 때문에 이러한 구조를 성공적으로 반영할 수 없다.

- Protected Process : 백신이나 EDR 클라이언트 역시 파일을 처리하는 핵심 엔진이 존재하는 퍼징에 적합한 소프트웨어이다. 하지만 이러한 소프트웨어들은 모두 운영체제에 의해 보호되고 있다. TTD는 DLL 인젝션을 필수로 하는 기술이기 때문에 성공적으로 WinEco를 적용할 수 없었다. TTD는 user land 프로세스만을 적용할 수 있으므로 커널 프로그램에 대해서는 분석 자체가 불가능하다.

5-5 개선 방안

WinEco는 TTD 기반의 반자동화된 harness 생성기를 증명하였다. 바이너리 내부에 존재하는 여러 가지 종류의 의존성 복구와 구조체에 대한 정확한 복원이 harness 생성 연구에서 가장 중요한 문제라고 판단된다. 앞서 언급한 한계점과 이러한 연구의 최종 목적인 완전히 자동화된 윈도우 퍼징을 수행하기 위해 다음과 같은 개선 방안이 존재한다.

- Snap Shot 기반의 퍼징

Snap Shot은 최근 많은 퍼저에서 채택하고 있는 방식이다. 대표적인 예시로 kAFL, WTF와 같은 퍼저들이 존재한다. 이러한 퍼저들은 프로그램의 특정 상태를 Snap Shot 형태로 캡처하고 조작할 수 있는 입력을 여러 가지 기술을 통해 바꾸어 준다. 또한 일반적으로 실행 속도가 WinAFL과 같은 일반적인 User land 퍼저보다 빠르다. 또한 함수의 의존성, 변수의 의존성과 같은 다양한 의존성을 거의 고려하지 않고 입력에 대한 정확한 흐름 분석만을 통해 성공적인 퍼징을 할 수 있다.

- Support Non C binary

최근 윈도우에서는 Rust, Go와 같은 다양한 새로운 언어들을 사용하는 소프트웨어들이 개발되고 있다. 이러한 언어로 개발된 소프트웨어는 C/C++ 바이너리에 비해 매우 복잡하다. 이번 연구에서는 C/C++을 사용하는 바이너리만을 목표로 수행했다. 변화하는 흐름에 맞추어 C/C++로 개발되지 않은 바이너리에 대한 harness 생성 연구를 통해 더 다양한 소프트웨어들에 대한 퍼징을 수행할 수 있을 것이다.

Ⅵ. 결 론

본 연구에서는 윈도우 소프트웨어 퍼징에 필요한 harness를 생성하기 위해 TTD 기술을 적용해 반자동화에 성공하였다. TTD에서 제공하는 강력한 추적 기술을 기반으로 소프트웨어의 특정 기능에 대한 분석을 수행하고 퍼징에 필요한 정보들을 복구해 낼 수 있었다. 아직 완전히 자동화된 harness 생성은 불가능하지만, 더 많은 연구를 통해 가능해 보인다.

아직 TTD를 실제 연구에 적용한 사례가 극히 드물다. harness 생성뿐만 아니라 RCA(Root Cause Analysis)에도 적용하거나 악성코드 분석과 같은 분야에도 사용할 수 있어 보인다.

WinEco를 통해 생성된 harness 코드를 기반으로 퍼징을 수행하여 총 10개의 취약점을 발견할 수 있었다. 이를 통해 WinEco가 충분히 성공적인 harness를 생성해 내었다고 생각된다. 발견된 취약점은 모두 제조사 혹은 한국인터넷진흥원에 제보되었고 더 안전한 소프트웨어 생태계에 이바지할 수 있었다.

참고문헌

[1] Ivan Fratric. WinAFL [Internet]. Available: https://github.co

m/ivanfratric/winafl.

[2] Google Project Zero. Jackalope [Internet]. Available: https://github.com/googleprojectzero/Jackalope.

[3] J. Jung, S. Tong, H. Hu, J. Lim, Y. Jin, and T. Kim, "WINNIE: Fuzzing Windows Applications with Harness Synthesis and Fast Cloning," in *Proceedings of Network and Distributed SystemS Security (NDSS) Symposium 2021*, Online, February 2021. https://doi.org/10.14722/ndss.2021.24334

[4] S. Schumilo, C. Aschermann, and R. Gawlik, "kAFL: Hardware-Assisted Feedback Fuzzing for OS Kernels," in *Proceedings of the 26th USENIX Security Symposium*, Vancouver, Canada, pp. 167-182, August 2017.

[5] Google Security Blog. Taking the Next Step: OSS-Fuzz in 2023 [Internet]. Available: https://security.googleblog.com/2023/02/taking-next-step-oss-fuzz-in-2023.html.

[6] M. Zalewski. American Fuzzy Lop [Internet]. 2015. Available: https://github.com/google/AFL

[7] Microsoft. Time Travel Debugging - Overview [Internet]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/debuggercmds/time-travel-debugging-overview.

[8] D. Babić, S. Bucur, Y. Chen, F. Ivančić, T. King, M. Kusano, ... and W. Wang, "FUDGE: Fuzz Driver Generation at Scale," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019)*, Tallinn, Estonia, pp. 975-985, August 2019. https://doi.org/10.1145/3338906.3340456

[9] K. K. Ispoglou, D. Austin, V. Mohan, and M. Payer, "FuzzGen: Automatic Fuzzer Generation," in *Proceedings of the 29th USENIX Security Symposium*, Boston: MA, pp. 2271-2287, August 2020.

[10] Google Project Zero. WinAFL Intel PT Mode [Internet]. Available: https://github.com/googleprojectzero/winafl/blob/master/readme_pt.md.

[11] Microsoft. Introduction to Time Travel Debugging Objects [Internet]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/debuggercmds/time-travel-debugging-object-model.

김동준(Dongjun Kim)

2020년～현　재: 아주대학교 사이버보안학과 학사과정
2024년～현　재: Enki Whitehat 연구원
※관심분야 : 사이버보안, 취약점 분석, 퍼징

손태식(Taeshik Shon)

2000년 : 아주대학교 정보및컴퓨터공학부 졸업(학사)
2002년 : 아주대학교 정보통신전문대학원 졸업(석사)
2005년 : 고려대학교 정보보호대학원 졸업(박사)
2004년～2005년: University of Minnesota 방문연구원
2005년～2011년: 삼성전자 통신·DMC 연구소 책임연구원
2017년～2018년: Illinois Insitute of Technology 방문교수
2011년～현　재: 아주대학교 정보통신대학 사이버보안학과 교수
※관심분야 : Digital Forensics, ICS/Automotive Security