WINNIE:

Fuzzing Windows Applications with Harness Synthesis and Fast Cloning

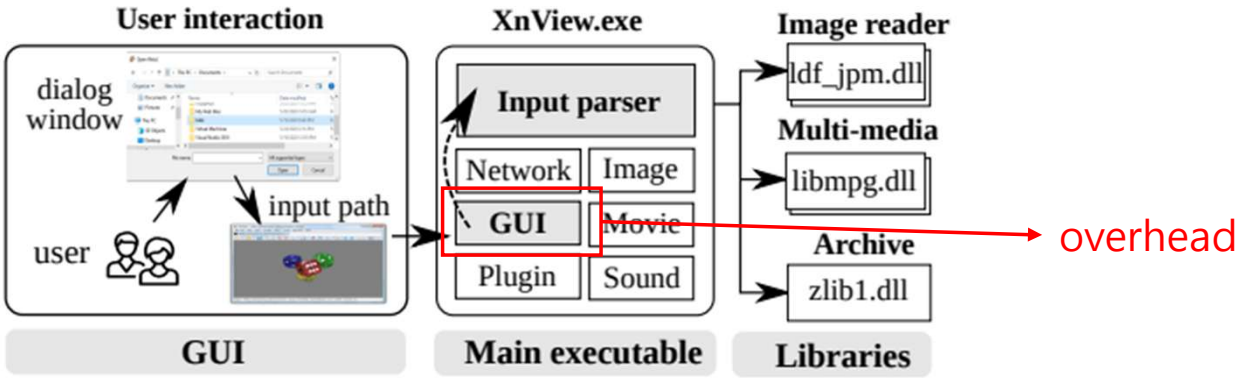
2025/01/24 이동하

background

- Fuzzing은 버그를 찾는데에 유용하지만, Unix 중심으로 발전
- Windows도 많은 memory corruption bug가 발견되고 있음

background

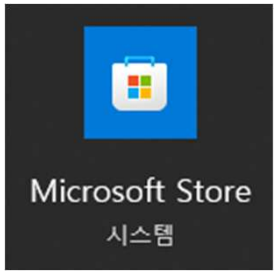
- Windows 환경에서의 fuzzing에는 추가 제약사항이 있음 (1/3)

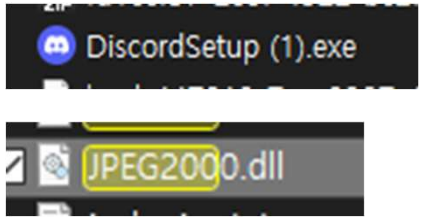


대부분이 GUI기반이며, GUI load는 overhead가 큼

background

- Windows 환경에서의 fuzzing에는 추가 제약사항이 있음 (2/3)

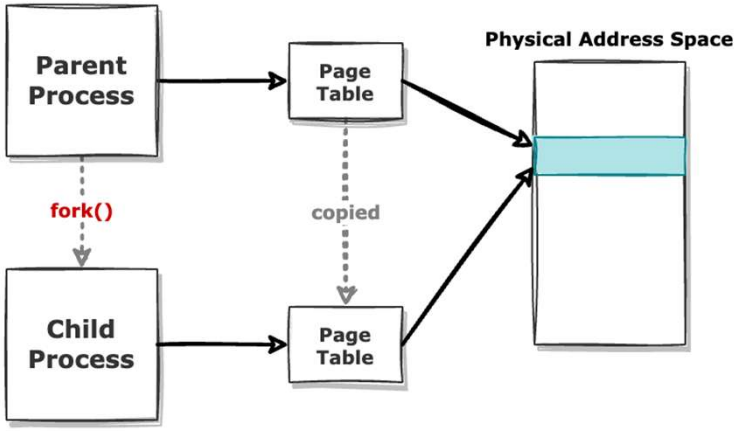




대부분이 closed-source 로 배포된다

background

- Windows 환경에서의 fuzzing에는 추가 제약사항이 있음 (3/3)



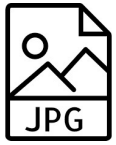
fork api가 제공되지 않는다
매번 create process로 target을 실행해야 한다(아니면 persistence mode)

solution

- GUI 어플리케이션의 일부만 테스팅 할 수 있도록(CLI로) Harness를 작성



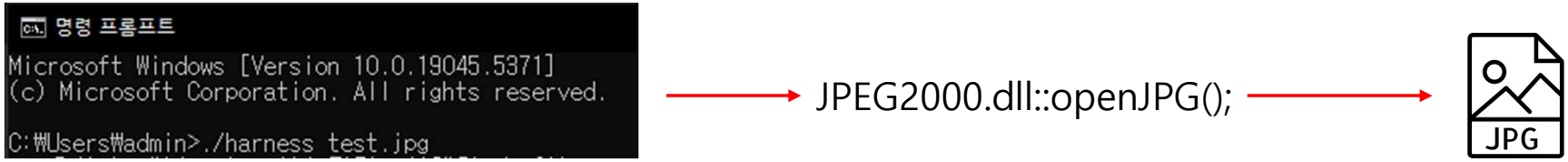
JPEG2000.dll::openJPG();



Harness 작성 과정을 자동화한다.

solution

- GUI 어플리케이션의 일부만 테스팅 할 수 있도록(CLI로) Harness를 작성 (1/3)



Harness 작성 과정을 자동화한다.

(harness를 통해 GUI overhead를 제거한다)

solution

- Coverage 계측 방법 개선 (2/3)

  기존 퍼저

  - Peach : coverage 기반 아님
  - Honggfuzz : windows binary fuzzing 불가능
  - Win-AFL(with dynamoRIO) : windows 10에서 정상 작동 안함(지금은 하던데)
  - Win-AFL(with Intel-PT) : hardware 의존적, 오류가 많아서 cov 실질적 효과가 떨어짐

  WINNE(논문 퍼저)

  - Fullspeed Fuzzing

solution

- **Fullspeed Fuzzing**

  Basic block 기준으로, block 의 첫 byte를 int 3로 패치

  HIT 시 원래 명령어로 복구

  HIT 시에만 overhead 발생, 일반적인 실행에서는 overhead가 없음

  여러번 실행되면서, HIT가 많이 될 수록 int 3 바이트가 줄어들고, overhead가 감소함

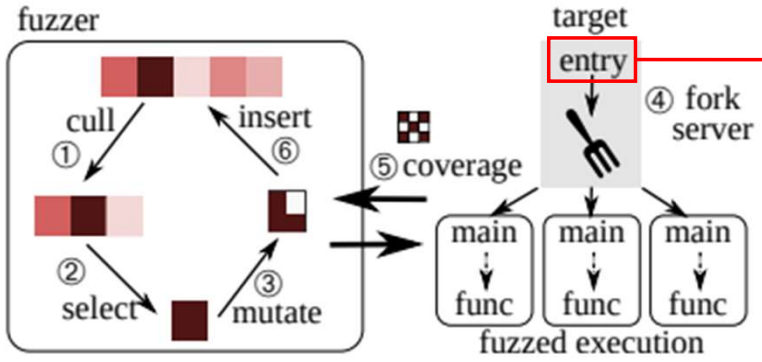
  Breakpoint(int 3)는 자식 프로세스 생성 시 유지되기 때문에 초기에 한번만 작업해두면 중복 작업이 필요없음

  전반적으로 overhead는 고려하지 않아도 될 수준으로 적다고 함

solution

- Coverage 계측 방법 개선 (3/3)

  Unix 계열의 fork



Mutation 된 corpus를 빠르게 testing

solution

- Coverage 계측 방법 개선 (3/3)

  windows의 fork

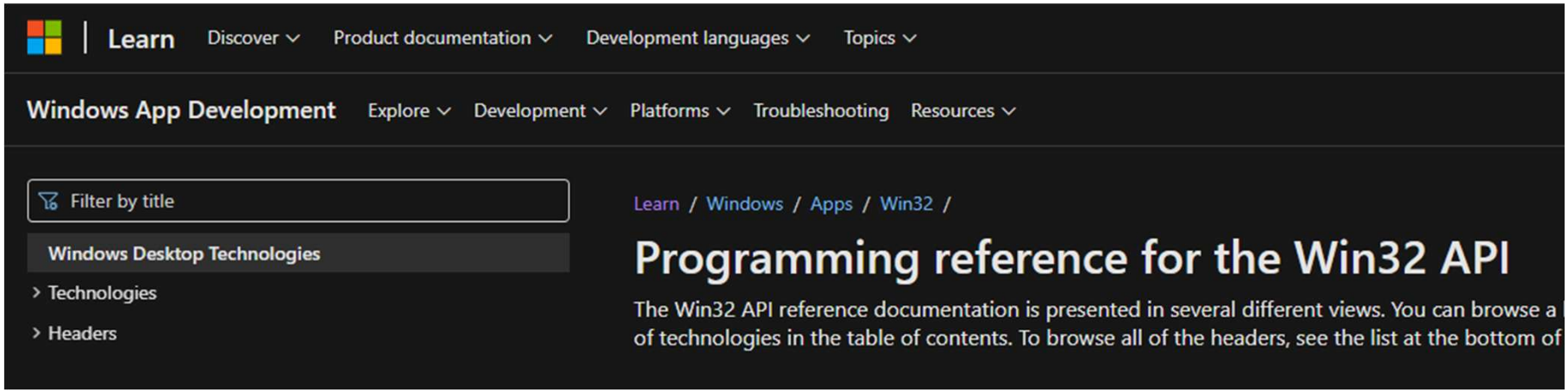


None…

solution

- Coverage 계측 방법 개선 (3/3)

  windows의 fork



  Hidden API로 fork 와 유사한 기능 구현

  https://billdemirkapi.me/abusing-windows-implementation-of-fork-for-stealthy-memory-operations/

  해당 글은 2021년 11월 게시물, 논문도 2021년 게시물 (논문에서 처음 제시한건진 의문)

Technique (2part)

- **Harness generate**

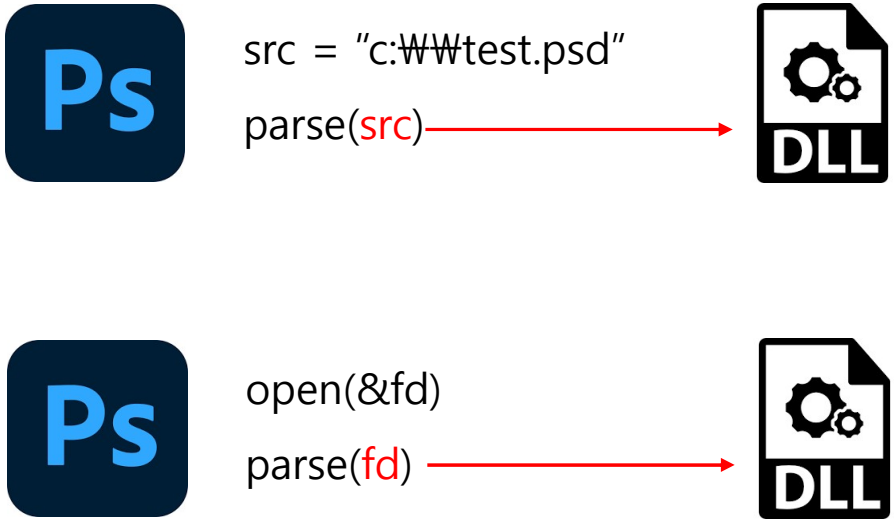
- **Fork server in windows**

Harness generate

1. Target Discovery

Input space를 식별함

1. File name이 인자로 넘어가는지
2. File descriptor를 넘기는지
3. File 내용이 memory에 적재되는지



Harness generate

2. Call-sequence Recovery

동적으로 testing 하며 api 호출 순서를 파악



Harness generate

2. Call-sequence Recovery

동적으로 testing 하며 api 호출 순서를 파악



Thread 간 분리하여 테스트 : 멀티 스레드 상태로 분석 시 영향이 섞일 수 있음

Harness에는 꼭 필요한 함수만 남김 : overhead 최소화

*OpenFile, ReadFile 등의 핵심 api를 사용하는 스레드를 추적

Harness generate

3. Argument Recovery

동적으로 testing 하며 api 호출 인자를 파악

1. 콜백 함수 포인터(ptr) : 휴리스틱한 연구 필요
2. 변수 포인터(ptr)
3. 상수(0,1,~)
4. 자료형(구조체 전달)



decode(**src, 0, 0, %tmp**);

parsing

validation

render

Harness generate

3. Argument Recovery

동적으로 testing 하며 api 호출 인자를 파악

1. 콜백 함수 포인터(ptr) : 휴리스틱한 연구 필요
2. 변수 포인터(ptr)
3. 상수(0,1,~)
4. 자료형(구조체 전달)



decode(**src, 0, 0, %tmp**);

parsing

validation

render

ptr과 const 구분이 어려움 -> Differential Analysis

ASLR 켠 상태로 여러번 실행(같은 seed)

실행 1: function_call(0x7ffeefbff420, 0x10)
실행 2: function_call(0x7ffcfb1aa630, 0x10)
ptr const

Harness generate

3. Argument Recovery

   동적으로 testing 하며 api 호출 인자를 파악

   포인터, 이중 포인터, 삼중 포인터까지 분리(harness에 적절하게 적용)

C: > Users > admin > source > repos > harness > harness > harness.cpp > ...

```cpp
#include <Windows.h>
#include <stdio.h>

#pragma warning(disable:4996)

const wchar_t* GetWC(const char* c)
{
    const size_t cSize = strlen(c) + 1;
    wchar_t* wc = new wchar_t[cSize];
    mbstowcs(wc, c, cSize);
    return wc;
}

typedef DWORD(__stdcall*fuzz_target)(const wchar_t*, INT*, INT, INT, INT);

fuzz_target func;
extern "C" __declspec(dllexport) INT addr = 0x00000000;

extern "C" __declspec(dllexport) __declspec(noinline) int fuzzme(const wchar_t* path)
{
    func(path, &addr, 0x87C, 0, 0);
    return 1;
}

int main(int argc, char** argv)
{
    SetDllDirectoryA("D:\\alsee\\ALSee");
    HMODULE WebP = LoadLibraryA("ALSeeIL.dll");
    func = (fuzz_target)GetProcAddress(WebP, "AIL_FileInfo");
    printf("%p %p\n", WebP, func);
    fuzzme(GetWC(argv[1]));//
    return 0;
}

```

참고자료. 실제로 사용했었던 windows ALSEE.exe harness

Harness generate

4. Control-Flow and Data-Flow Reconstruction

Control-Flow : 특정 분기문 (return, exit 대상으로 분석)

```
int ret = JPM_Document_Start(...);
if (ret) {
    exit(0);
}
```

```
991             else if (tracemode == INDIRECT) {
992                     if (addr_belongs_target(instr_addr) && addr_belongs_module(target_addr)) {
993                             print_ind_address(instr_addr, target_addr, "RETURN");
994                     }
```

코드 분석은 더 해봐야 감이 올 듯 (정적 분석하는 코드는 처음이라 감이 안옴)

Harness generate

4. Control-Flow and Data-Flow Reconstruction

Data-Flow Reconstruction : 리턴 -> 인자

```
int handle = open_file("test.jpg");
parse_file(handle);
```

2번 이상 사용되면 변수로 표기(상수 또한 변수로)

단, 0 같은 자주 사용되는 값은 상수

Harness generate

4. Control-Flow and Data-Flow Reconstruction

Data-Flow Reconstruction : 포인터 간의 관계

```
f0_a0 = JPM_Document_Start(...);
JPM_Document_Parse(f0_a0[0]);
```

f0_a0가 포인터고, JPM_Document_Parse에서 f0_a0에 직접 접근하여 수정하는 경우 하네스에 역참조 반영

Harness generate

5. Harness Validation

   Distant API Calls : 찝찝한거 경고

   API 호출 사이에 처리되지 않은 호출(원래는 있던 호출인데, harnes에 사용하지 않은 경우 warning 처리)

Harness generate

5. Harness Validation

Code Pointer : 찝찝한거 경고

Virtual Method Table 등으로 셋팅된 함수 포인터는 실행 흐름에 따라 달라질 수 있음
따라서 사용자에게 warning을 띄움(사실상 책임을 전가함...)

Harness generate

5. Harness Validation

   File Operations : 안정성 추구

   Open, Close 등 당연한 로직이 실패하면(-1) 예외 핸들링하여, 말도 안되는 crash 방지
   (ex: handler가 없어요!)

Harness generate

5. Harness Validation

Validation : 안정성 추구

Target program으로 정상처리되는 corpus를 harness로 돌려본다.
정상 종료 된다면 저장 (crash 발생 시 harness 폐기)

Harness generate

5. Harness Validation

Coverage check : 안정성 추구

짧게라도 coverage가 올라가는지 테스트 해본다.
안움직이면 core logic이 hit되지 않는다고 판단(harness 폐기)

Harness generate

5. Harness Validation

Execution Speed Evaluation : 효율성 추구

Validation이랑 Coverage check 통과한 것 중에 속도가 가장 빠른 것으로 선택
Fuzzing의 효율을 위한 선택

Harness generate

Program	Harness	GUI	Ratio	Program	Harness	GUI	Ratio
HWP-jpeg	117	4075	34.8×▲	Tiled	28	720	25.7×▲
Gomplayer	15	1105	73.6×▲	ezPDF	184	4397	23.8×▲
ACDSee	16	510	31.8×▲	EndNote	30	1461	23.8×▲

TABLE II: Execution times (ms) with and without GUI. GUI code

Technique (2part)

- ~~**Harness generate**~~

- **Fork server in windows**

Fork server

1. Windows에서 fork가 불가능한 이유

*논문에선 Windows 내부 API 및 서비스들을 리버스 엔지니어링하여 분석

핵심 문제 : CSRSS(Client/Server Runtime Subsystem) 프로세스와의 연결 문제

Windows의 사용자 모드(User-mode) 프로세스로,
Windows GUI, Console 윈도우, 프로세스 및 쓰레드 관리를 담당

새로운 프로세스가 CSRSS 와 연결되어야 windows API를 사용할 수 있음
fork 구현하면 CSRSS와 연결이 깨져서 windows API를 사용할 수 없었음
대부분의 어플리케이션이 windows API 를 사용하기에 CSRSS 연결이 필수적

Fork server

2. Fork 구현

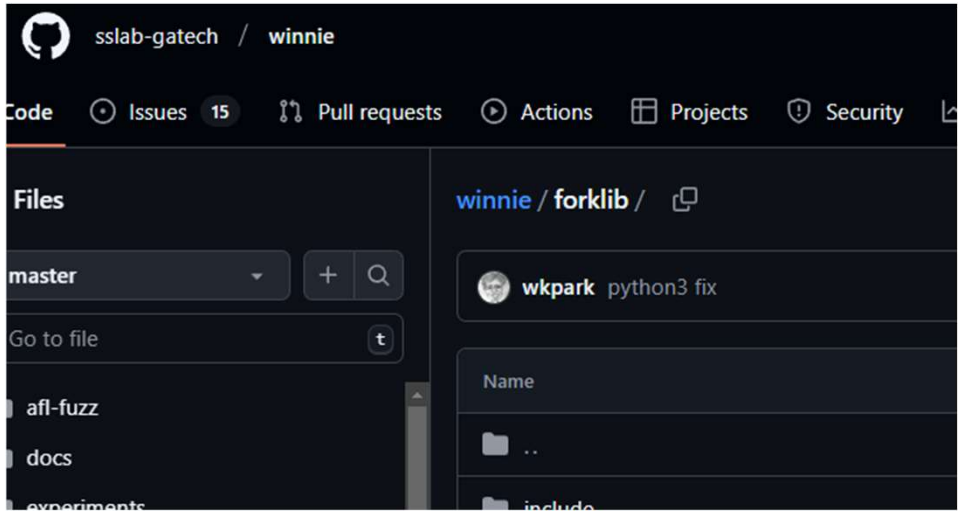
기존엔 CreateProcess 를 사용했음
느리지만 CSRSS와 연결을 지원함

WINNE는 CSRSS가 child process를 인식하도록 수동으로 알림을 전달하여
자체적으로(WINNE가) fork한 child process가 CSRSS에 연결된다

*기술적으론 hidden windows api를 리버싱하여 찾은 것

Fork server

2. Fork 구현



CsrClientCallServer()를 사용하여 CSRSS에 수동으로 새로운 프로세스 정보를 전달.

CsrClientConnectToServer()를 호출하여 자식 프로세스를 CSRSS에 등록하도록 수정

Fork server

2. Fork 구현

```
// (1) 부모 프로세스에서 새로운 프로세스를 생성 (Suspended Mode)
NTSTATUS result = NtCreateUserProcess(
    &hProcess, &hThread, MAXIMUM_ALLOWED, MAXIMUM_ALLOWED,
    NULL, NULL, PROCESS_CREATE_FLAGS_INHERIT_FROM_PARENT
    | PROCESS_CREATE_FLAGS_INHERIT_HANDLES,
    THREAD_CREATE_FLAGS_CREATE_SUSPENDED,
    NULL, &procInfo, NULL
);

if (!result) {  // 부모 프로세스 코드
    // (2) CSRSS에 새로운 프로세스를 생성했다고 알림
    NotifyCsrssParent(hProcess, hThread);

    // (3) 자식 프로세스를 실행 (Resume)
    ResumeThread(hThread);

    // (4) 부모 프로세스는 새로운 자식 프로세스의 PID 반환
    return GetProcessId(hProcess);

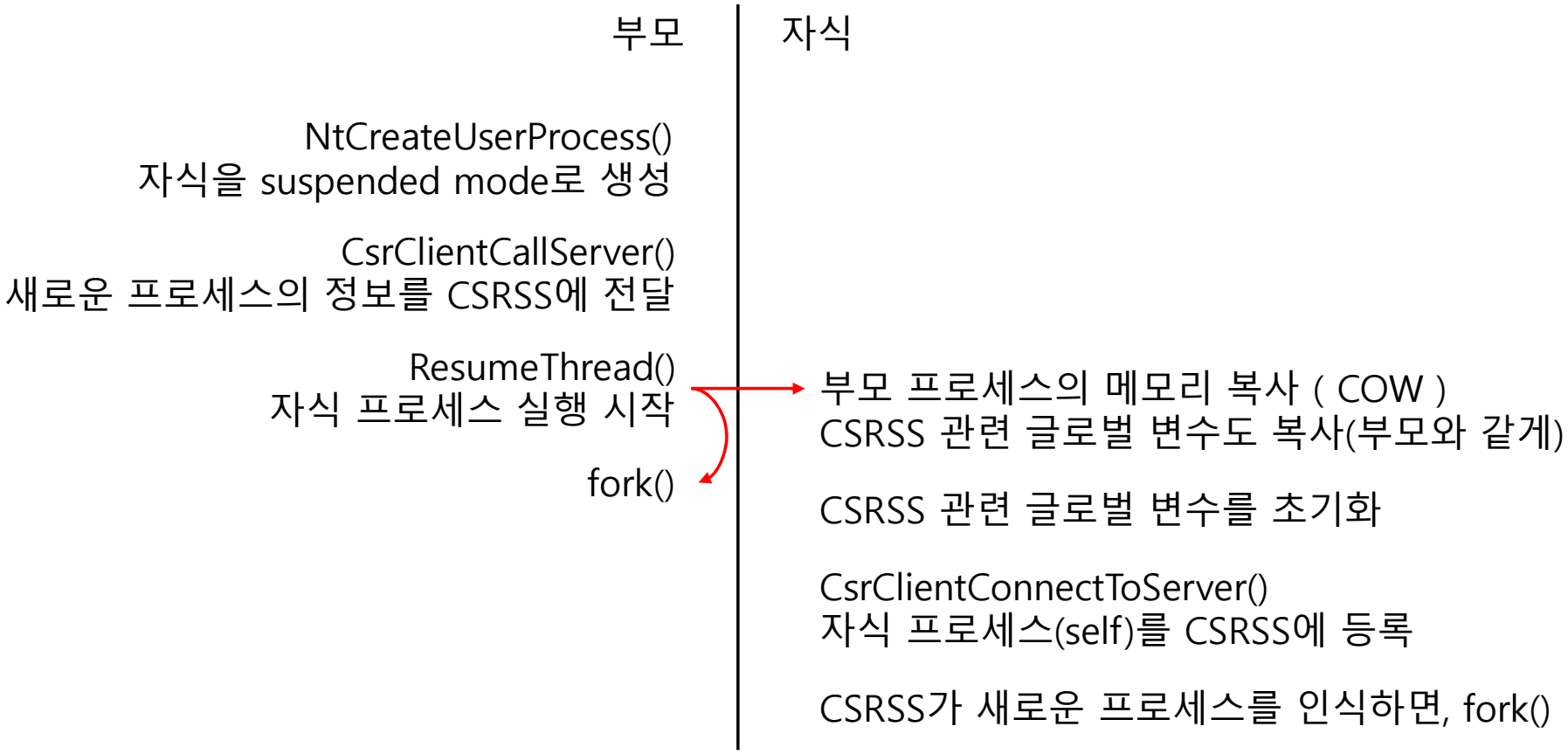
}
else {  // 자식 프로세스 코드
    // (5) 부모 프로세스에서 복사된 CSRSS 관련 데이터를 초기화하여 충돌 방지
    memset(pCsrData, 0, csrDataSize);

    // (6) CSRSS에 자식 프로세스를 등록
    ConnectCsrChild();

    // (7) `fork()` 종료, 자식 프로세스 실행 시작
    return 0;
}
```

Fork server

2. Fork 구현

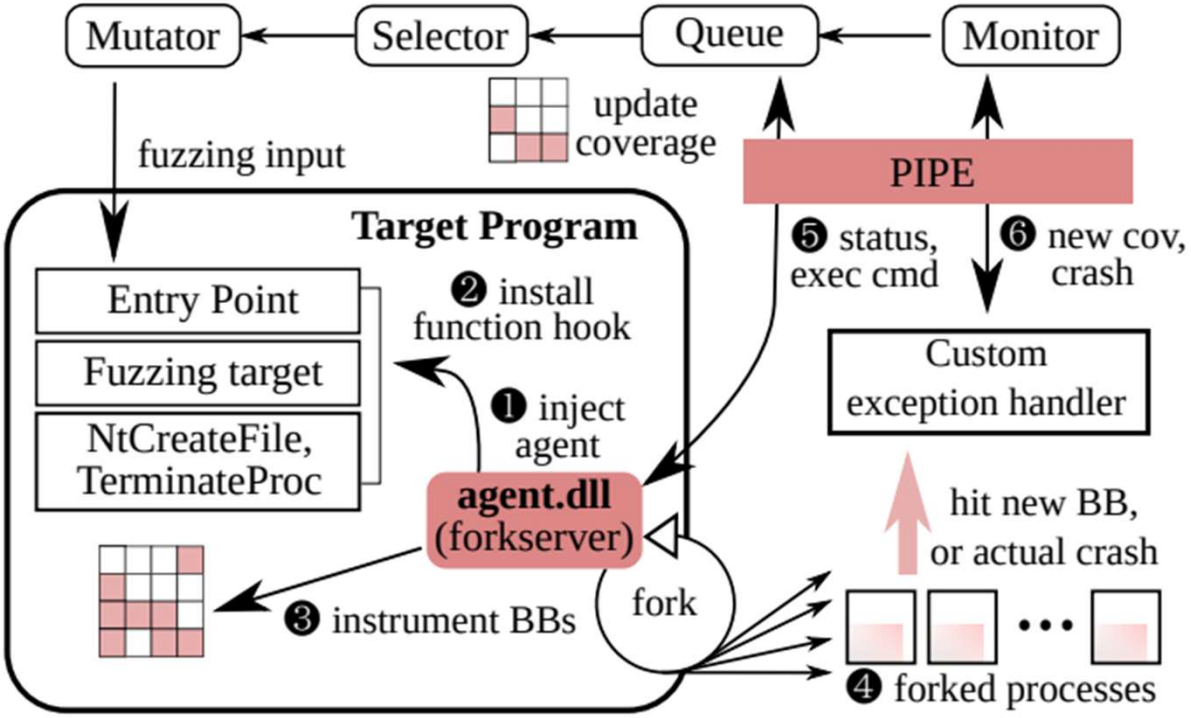


Fork server

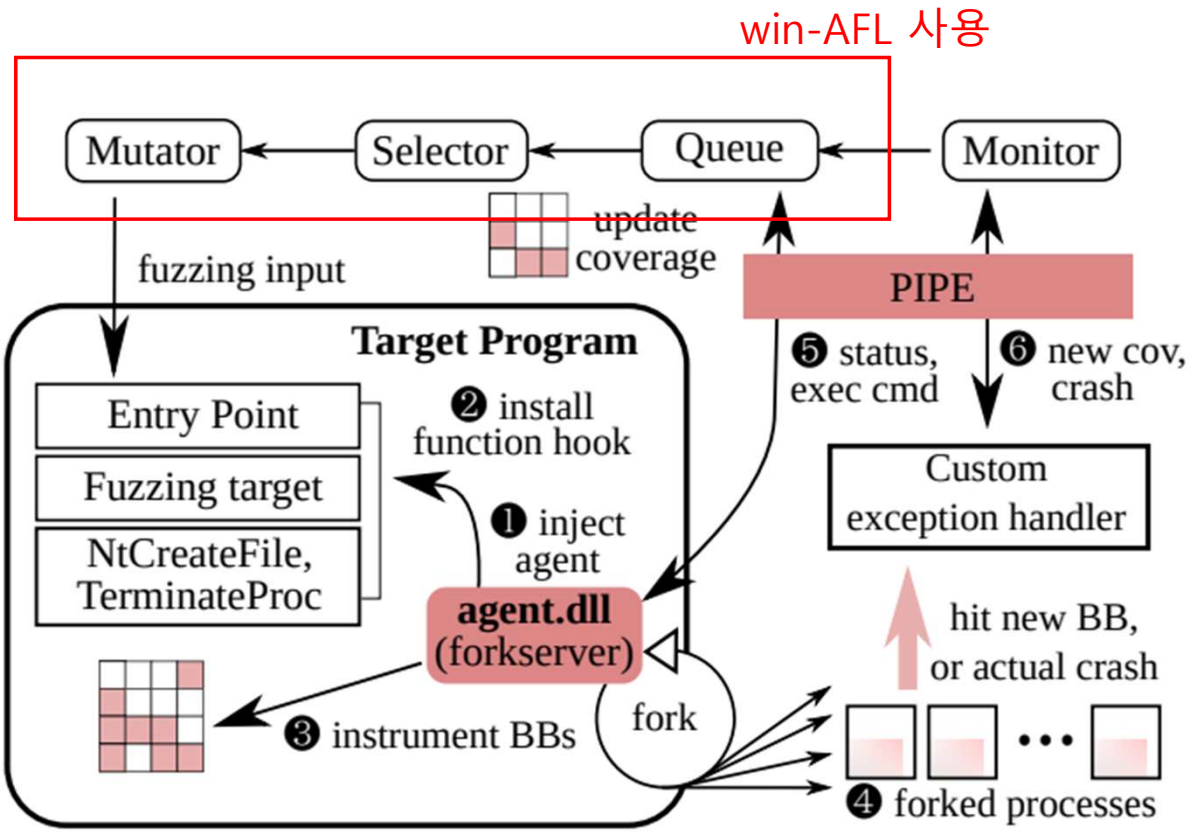
2. Fork 구현

Program	Without Fork				Fork			
	Leak	Hang†	Speed	Cov.	Speed		Coverage	
7z			5.2	1430	49.3	(9.5×▲)	2117	(1.5×▲)
makecab	✗		14.8	576	49.4	(3.3×▲)	1020	(1.8×▲)
GomPlayer		✗	0.4	201	25.9	(64.7×▲)	1496	(7.4×▲)
Hwp-jpeg	✗		4.2	1045	25.9	(6.2×▲)	1847	(1.8×▲)
Hwp-tiff	✗	✗	0.3	1340	26.2	(87.3×▲)	2301	(1.7×▲)
EndNote			5.3	68	89.5	(16.9×▲)	693	(10.2×▲)
Total	3/6	2/6				**(31.3×▲)**		**(4.0×▲)**

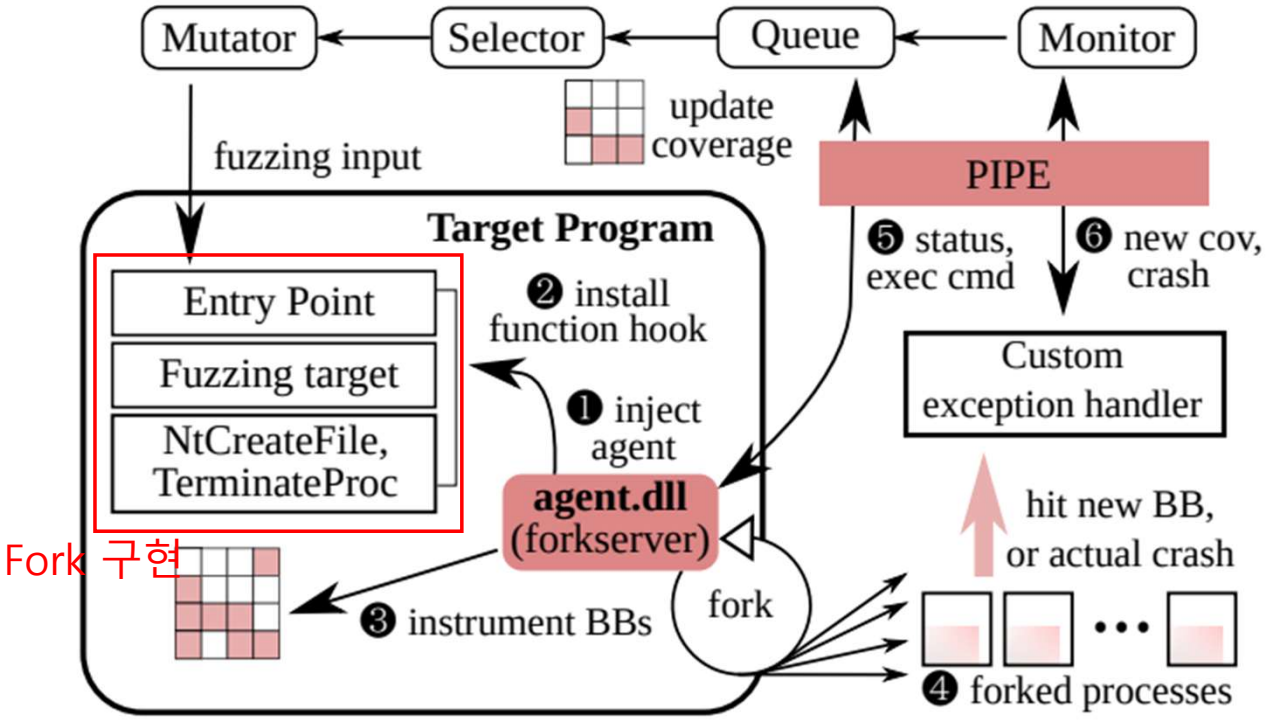
Source code


Mutator
Selector
Queue
Monitor
update
coverage
fuzzing input
PIPE
Target Program
❺ status,
exec cmd
❻ new cov,
crash
Entry Point
Fuzzing target
NtCreateFile,
TerminateProc
❷ install
function hook
Custom
exception handler
❶ inject
agent
agent.dll
(forkserver)
hit new BB,
or actual crash
fork
❸ instrument BBs
❹ forked processes

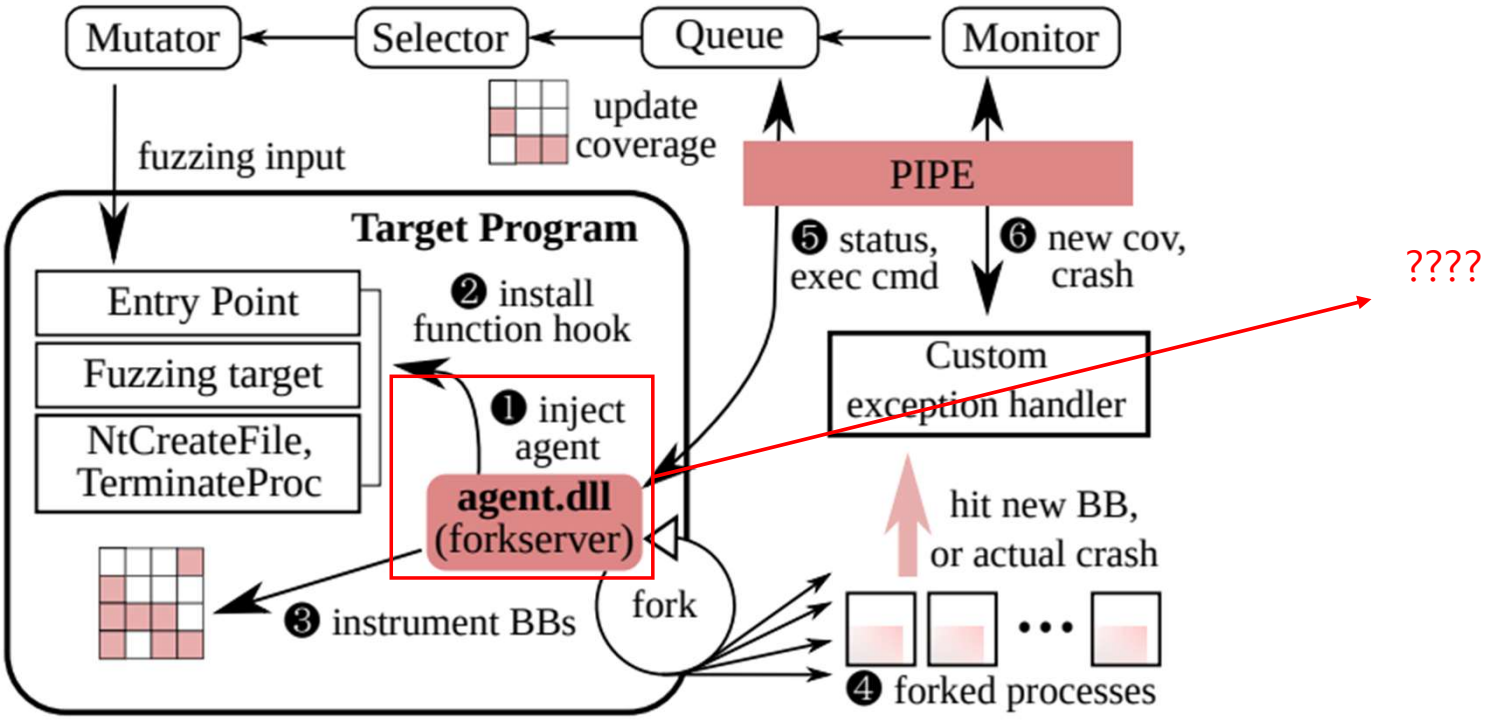
Source code


win-AFL 사용
Mutator
Selector
Queue
Monitor
update
coverage
fuzzing input
PIPE
Target Program
❺ status,
exec cmd
❻ new cov,
crash
Entry Point
Fuzzing target
NtCreateFile,
TerminateProc
❷ install
function hook
Custom
exception handler
❶ inject
agent
agent.dll
(forkserver)
hit new BB,
or actual crash
fork
❸ instrument BBs
❹ forked processes

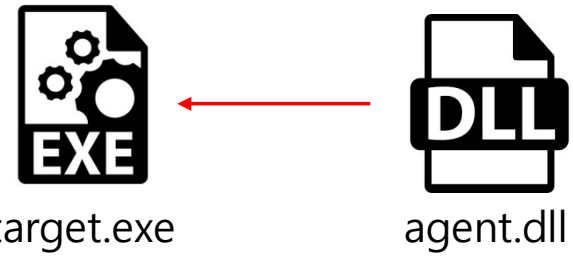
Source code


Mutator
Selector
Queue
Monitor
update
coverage
fuzzing input
PIPE
Target Program
❺ status,
exec cmd
❻ new cov,
crash
Entry Point
Fuzzing target
NtCreateFile,
TerminateProc
❷ install
function hook
Custom
exception handler
❶ inject
agent
agent.dll
(forkserver)
hit new BB,
or actual crash
Fork 구현
fork
❸ instrument BBs
❹ forked processes

Source code


Mutator
Selector
Queue
Monitor
update
coverage
fuzzing input
PIPE
Target Program
❺ status,
exec cmd
❻ new cov,
crash
????
Entry Point
Fuzzing target
NtCreateFile,
TerminateProc
❷ install
function hook
Custom
exception handler
❶ inject
agent
agent.dll
(forkserver)
hit new BB,
or actual crash
fork
❸ instrument BBs
❹ forked processes

Source code

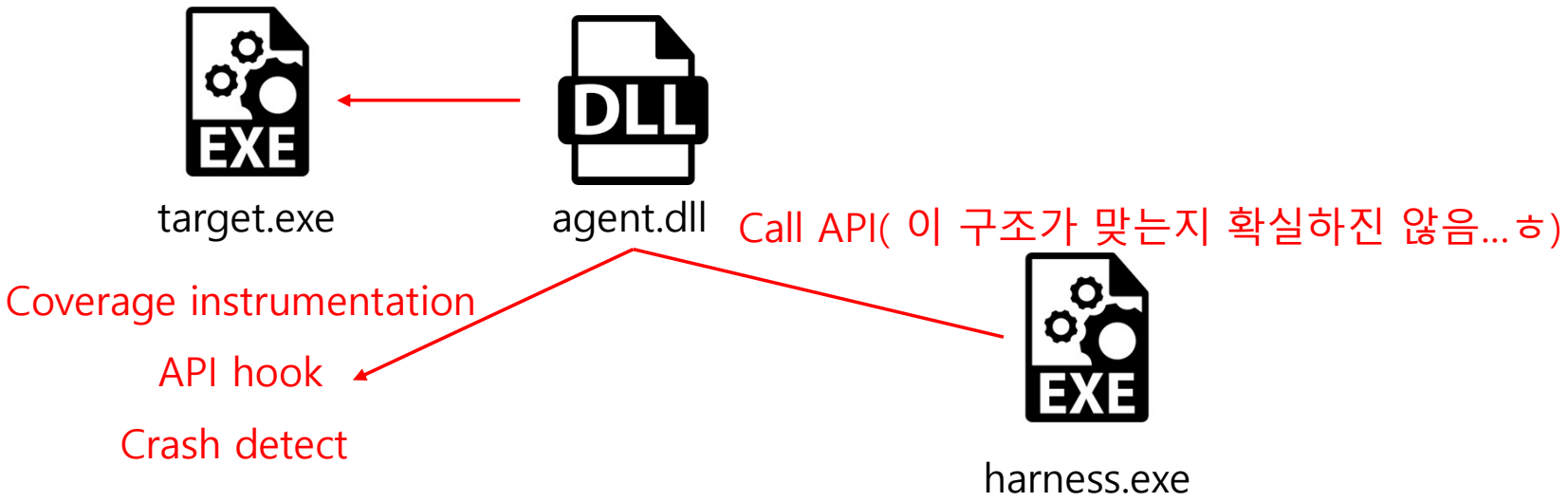


Coverage instrumentation

API hook

Crash detect

Source code


target.exe
agent.dll
Call API(이 구조가 맞는지 확실하진 않음...ㅎ)
Coverage instrumentation
API hook
Crash detect
harness.exe

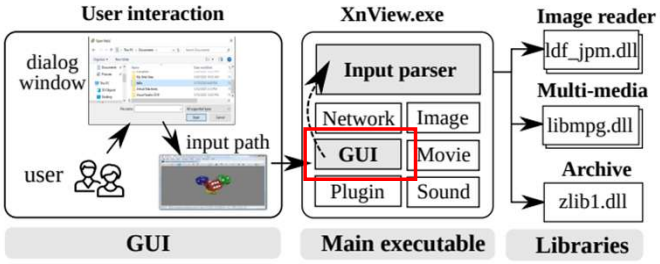
요약

Windows에서 fuzzing이 어려웠음

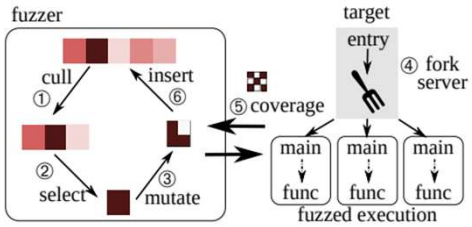
Fuzzer	AFL	WinAFL	HonggFuzz	Peach	WINNIE
Feedback	✔	✔	✗	✗	✔
Forkserver	✔	✗	✗	✗	✔
Open-source	✔	✔	✔	✗	✔
Windows	✗	✔	✔	✔	✔

요약

3가지 issue



heavy



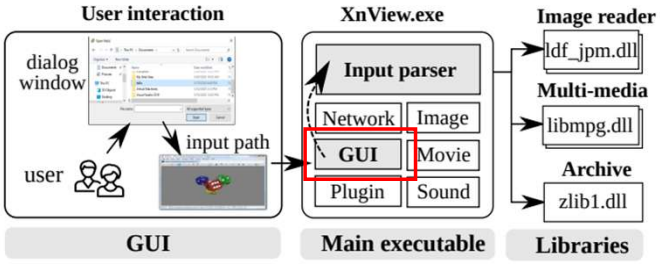
fork 불가능



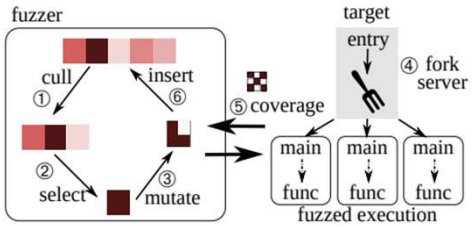
Closed-source 정신

요약

3가지 issue



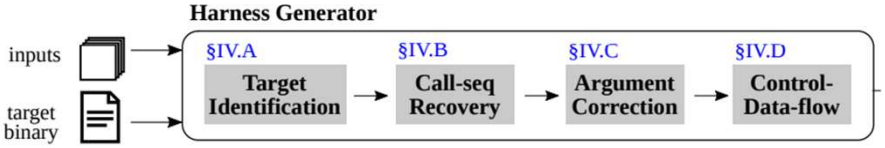
heavy



fork 불가능



Closed-source 정신

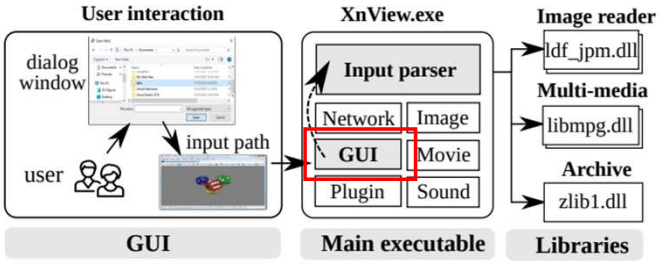


Program	Harness	GUI	Ratio	Program	Harness	GUI	Ratio
HWP-jpeg	117	4075	34.8×▲	Tiled	28	720	25.7×▲
Gomplayer	15	1105	73.6×▲	ezPDF	184	4397	23.8×▲
ACDSee	16	510	31.8×▲	EndNote	30	1461	23.8×▲

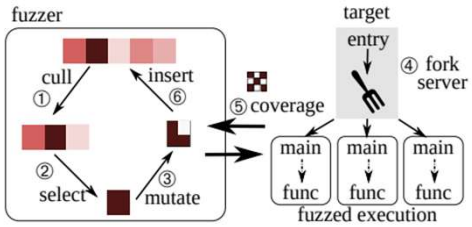
Fast!!!

요약

3가지 issue



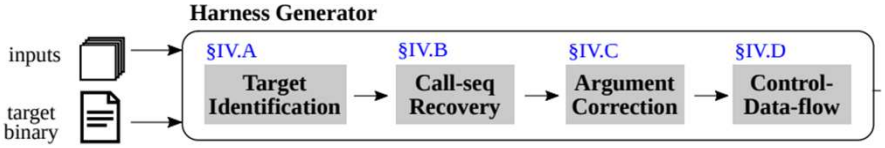
heavy

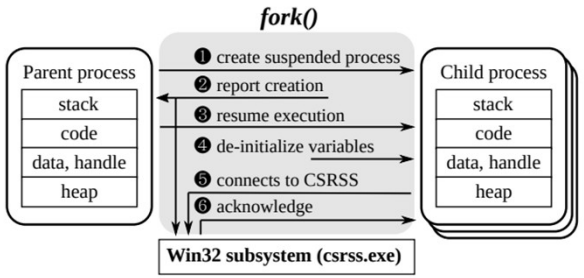


fork 불가능



Closed-source 정신





Program	Harness	GUI	Ratio	Program	Harness	GUI	Ratio
HWP-jpeg	117	4075	34.8×▲	Tiled	28	720	25.7×▲
Gomplayer	15	1105	73.6×▲	ezPDF	184	4397	23.8×▲
ACDSee	16	510	31.8×▲	EndNote	30	1461	23.8×▲

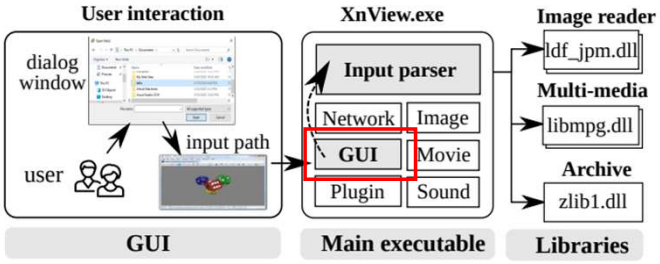
Fast!!!

Fork()	Re-execute	Forkserver				
	CreateProcess	WINNIE	Cygwin	WSL(v1)	WSL(v2)	Linux
Supports PE files?	✔	✔	✔	✗	✗	✗
Copy-on-Write?	✗	✔	✗	✔	✔	✔
Speed (exec/sec)	91.9	310.9	72.8	442.8	405.1	4907.5

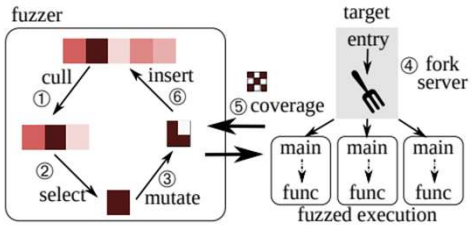
fork 가능

요약

3가지 issue



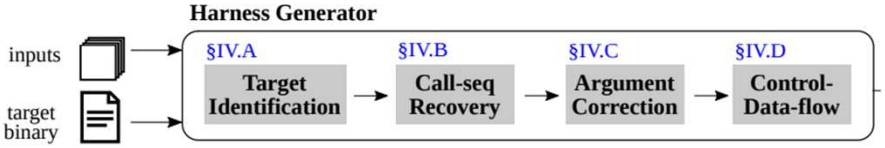
heavy



fork 불가능

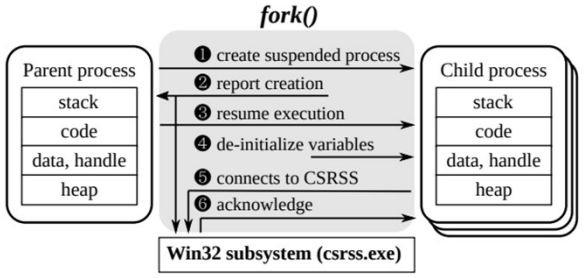


Closed-source 정신



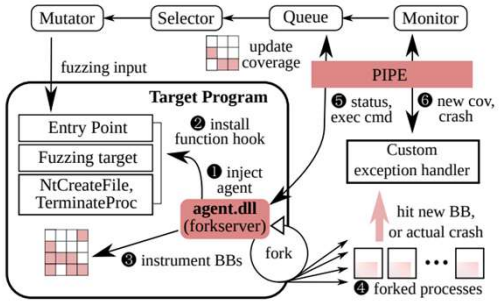
Program	Harness	GUI	Ratio	Program	Harness	GUI	Ratio
HWP-jpeg	117	4075	34.8×▲	Tiled	28	720	25.7×▲
Gomplayer	15	1105	73.6×▲	ezPDF	184	4397	23.8×▲
ACDSee	16	510	31.8×▲	EndNote	30	1461	23.8×▲

Fast!!!



Fork()	Re-execute	Forkserver				
	CreateProcess	WINNIE	Cygwin	WSL(v1)	WSL(v2)	Linux
Supports PE files?	✔	✔	✔	✗	✗	✗
Copy-on-Write?	✗	✔	✗	✔	✔	✔
Speed (exec/sec)	91.9	310.9	72.8	442.8	405.1	4907.5

fork 가능



Fullspeed fuzzing

한계

Harness 자동화는 아직 실전 활용되기엔 어려운 수준이라고 함

구조체, 콜백 함수가 인자로 넘어가는 부분은 연구원이 직접 분석해야함

opinion

설명에선 생략하였지만,
fork의 필요성으로 persistence 모드를 부정한다

fork없이 fuzzing하기 위해, win-AFL에서는 persistence mode를 사용한다
Global variable이 수정될 경우, loop를 다시 처음부터 initialize 해야한다(프로그램 재시작)
Memory leak에 취약하다

그러나 막상 persistence 모드를 사용해보면, 메모리 누수 문제도 상당 부분 해결 가능하며,
실행 속도도 persistence > fork > createProcess 순이다

Coverage 계측도 dynamoRio가 win10을 지원하기에, WINNE보다 win-AFL 연구가 활발한 것으로 판단
Harness자동화 + persistence mode 사용 연구를 진행해보려고 함(기존에 해보다가 포기함)