AI 기반 취약점 자동탐지 트랙

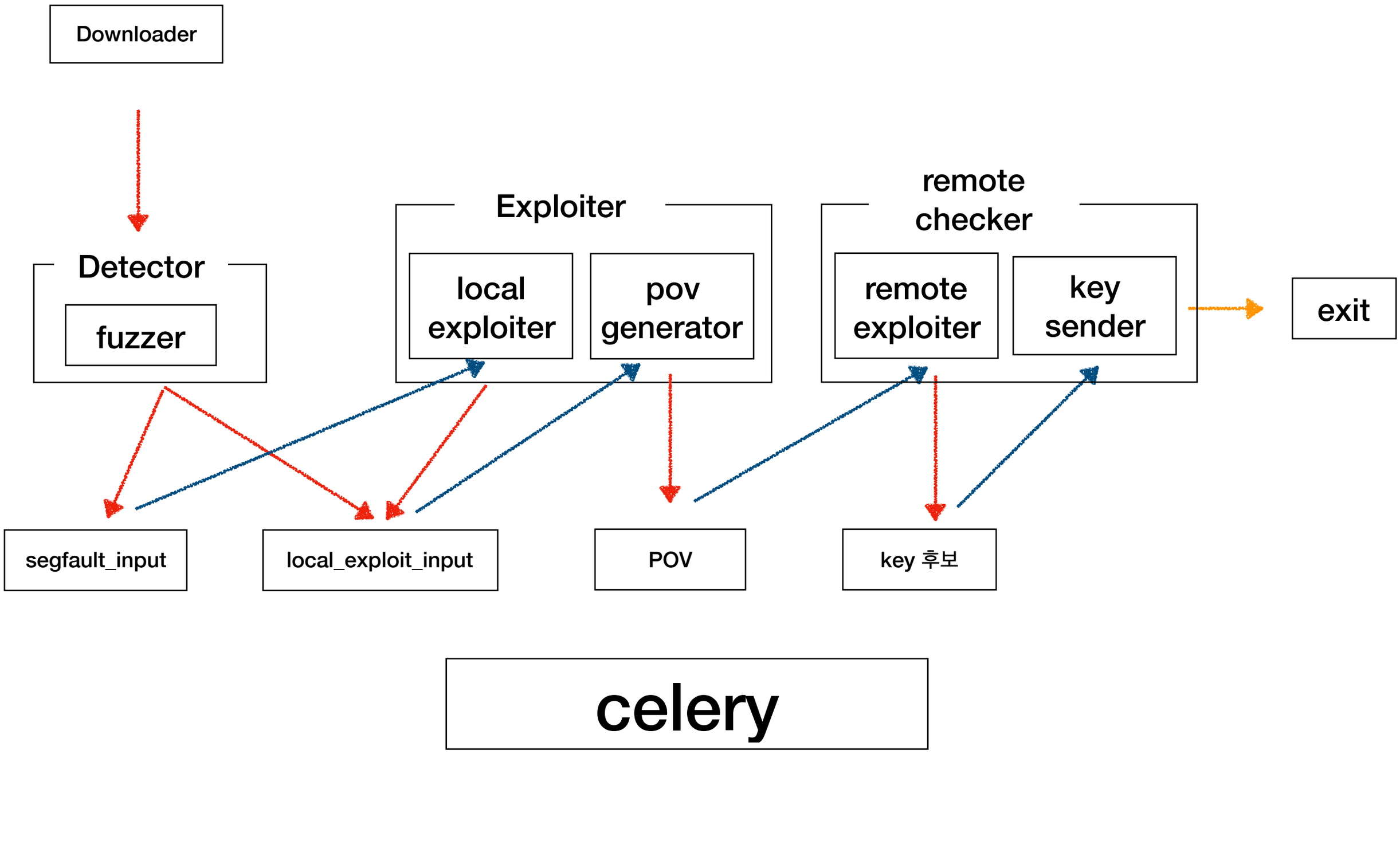
호홍현2

황호, 김재홍, 진수현

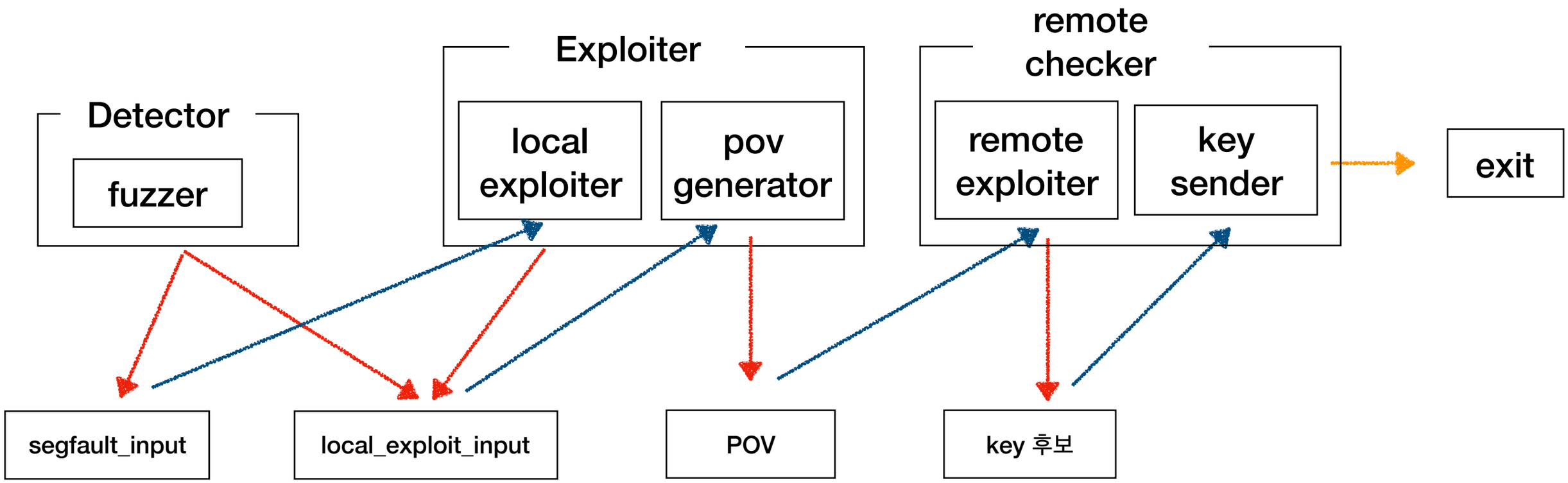
구성

- 구성도
- 알고리즘
- 결과
- 개선점

구성도


Downloader
Detector
fuzzer
Exploiter
local exploiter
pov generator
remote checker
remote exploiter
key sender
exit
segfault_input
local_exploit_input
POV
key 후보
celery

퍼져 구성



탐지 모듈(Detector)

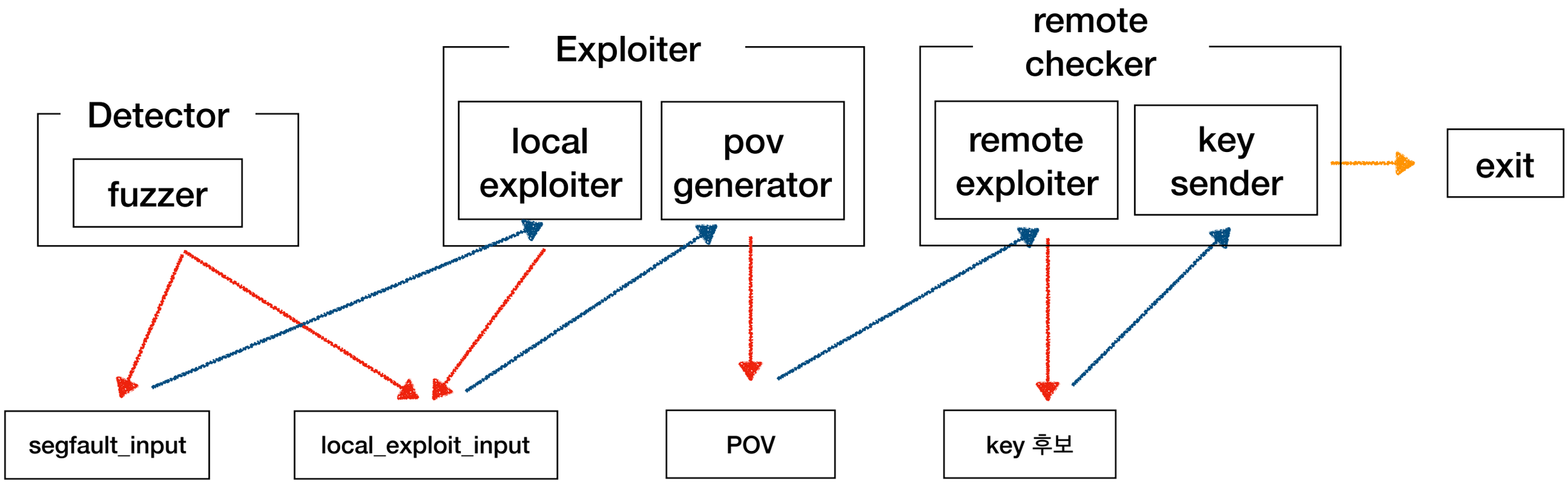
- BOF, Integer Overflow 유형, Out-of-bound, Command injection을 대상
- 대상 프로그램의 시그널을 확인(SIGCHLD, SIGSEGV)
- stdout에 키 값 존재 여부 판단
- 정적분석을 통해 system("/bin/sh") 함수 추출

알고리즘 배경

퍼져 중심 경로 탐색

- symbolic execution은 정확도가 높지만 성능이 매우 떨어짐
- 퍼징은 속도가 빠르지만 깊은 실행경로에 도달하기 어려움
- 퍼징을 강화하여 **빠른 속도**와 함께 **정확도**를 올려보자

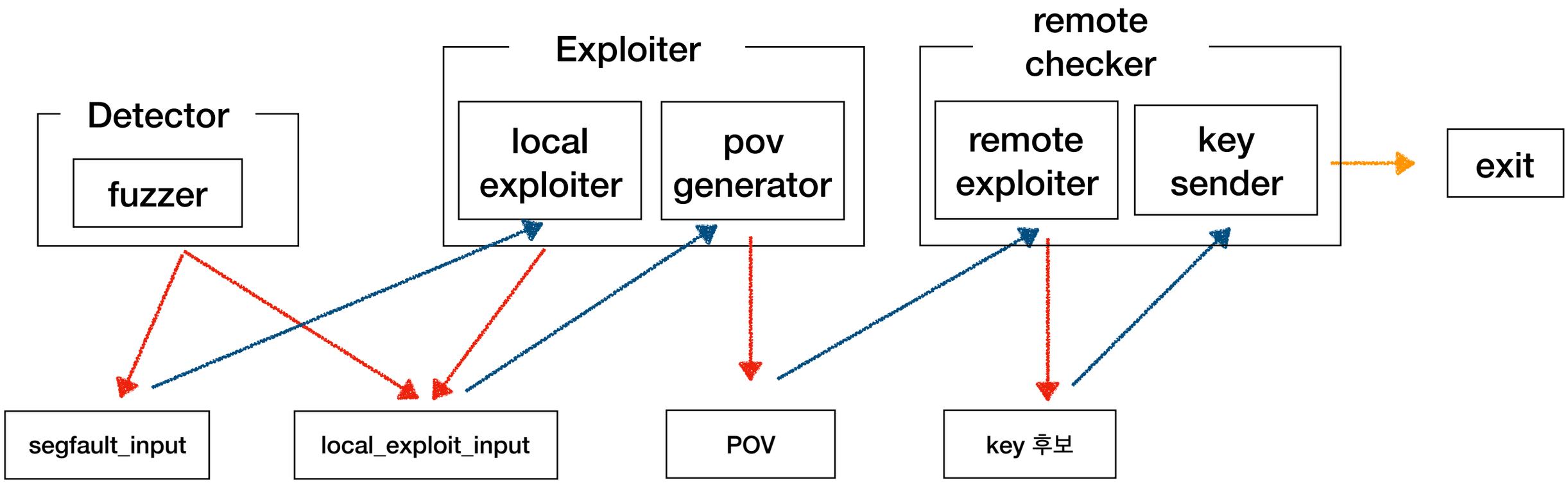
알고리즘



탐지 모듈(Detector)

- radare2이 지원하는 python api 를 이용하여 fuzzer 구현
- BOF, Integer Overflow 유형, Out-of-bound, Command injection, FSB 대상
- 대상 프로그램의 시그널을 확인(SIGCHLD, SIGSEGV)
- stdout에 키 값 존재 여부 판단
- 정적분석을 통해 system("/bin/sh") 함수 추출

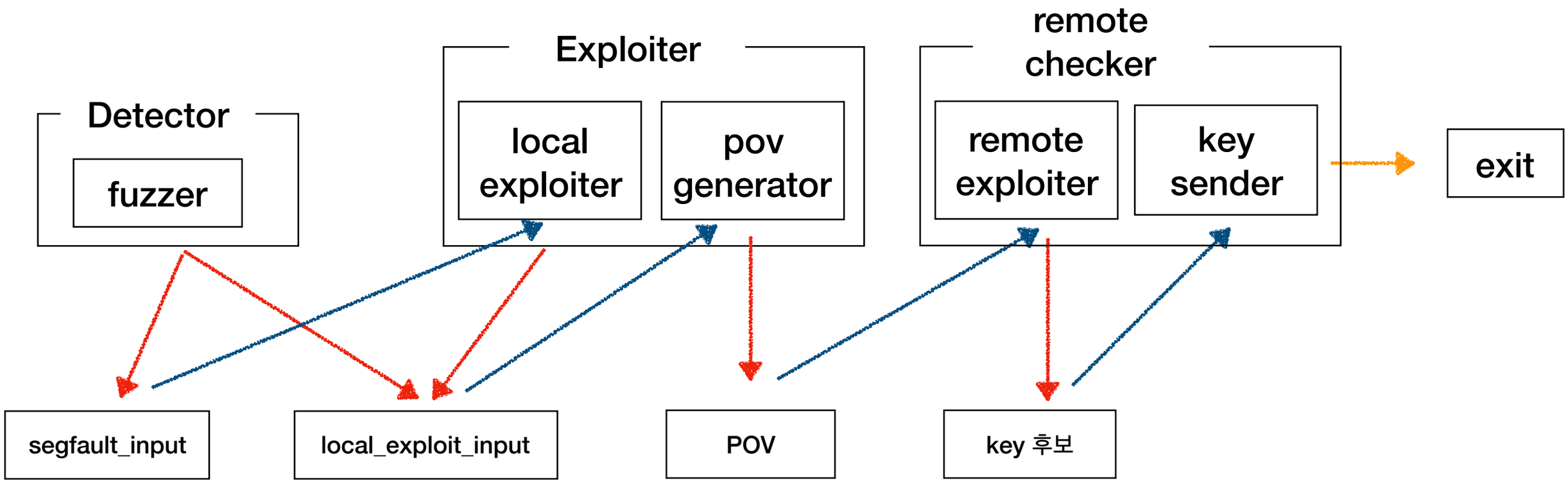
알고리즘



취약점 생성 모듈(Exploiter)

- ltrace를 이용하여 호출된 입력 함수(scanf, read 등)의 구분 및 인자 확인
- system(/bin/sh) 호출 여부를 통해 쉘 획득 여부 확인
  - 쉘은 획득하였지만 cat flag를 못하는 case 판별 가능 -> pwntool의 코드를 차용하여 해결
  - /bin/sh 문자열의 주소, shellcode의 주소, winfunc 주소, win func candidate 주소

알고리즘



원격 확인 모듈(remote checker)

- 대상 바이너리가 실행되는 ip,port에 접속하여 stdout 추출
- stdout내에 키 값 후보 군 추출하여 인증 시도

결과

탐지 가능 케이스

- **CWE-680 : Integer Overflow to Buffer Overflow**
- **CWE-77 : Command Injection**
- **CWE-191 : Integer Underflow**
- **CWE-121 : Stack-based Buffer Overflow**
- **CWE-120(Buffer copy without Checking Size of input)**
- **CWE-134: Use of Externally-Controlled Format String**

결과

```
[DEBUG|fuzzer.py:52]: [134514075]
['/home/root/data/CWE_77_A_1', 9.051464796066284, 5, 'SIGCHLD', True]
['/home/root/data/CWE_120_A_1', 1.5546786785125732, 0, 'SIGSEGV', False]
['/home/root/data/CWE_191_A_2', 11.629221200942993, 7, 'SIGSEGV', False]
['/home/root/data/CWE_191_A_1', 189.77652168273926, 100, 'SIGTRAP', False]
['/home/root/data/CWE_680_A_2', 9.466513633728027, 5, 'SIGSEGV', False]
['/home/root/data/CWE_121_A_1', 1.6087408065795898, 0, 'SIGSEGV', False]
['/home/root/data/CWE_120_B_1', 1.5539708137512207, 0, 'SIGSEGV', False]
['/home/root/data/CWE_680_A_1', 10.621148824691772, 6, 'SIGSEGV', False]
(dataChallenge) root@Server-21:/home/root/h3_fuzzer#
```

탐지 모듈 결과

- menu 케이스 2문제 제외
- 총 8문제 중 7개 성공
  - CWE_77_A_1에서 키 획득 성공
  - 나머지 6개에서 SIGSEGV 성공

결과

```
binary name, win func addr, exploit result
['/home/root/data/CWE_120_A_1', 134514006, True]
['/home/root/data/CWE_191_A_2', 134514038, True]
['/home/root/data/CWE_680_A_2', 134514038, True]
['/home/root/data/CWE_121_A_1', 134513990, False]
['/home/root/data/CWE_120_B_1', 134514086, True]
['/home/root/data/CWE_680_A_1', 134514038, True]
(dataChallenge) root@Server-21:/home/root/jaehong/pov#
```

익스플로잇 생성 모듈 결과

- 키 값을 추출한 케이스 제외, 탐지하지 못한 케이스 1개 제외
- 총 6 케이스 중 5개 성공

본선 결과

1차 - 3 문제 풀이, 15개 SIGSEGV 생성

- 자원 관리 문제로 확인
- 자체 서비스를 돌려 확인해본 결과 대부분 exploitable

2차 - 6 문제 풀이

개선한 점

- ltrace를 이용한 동적 실행 정보 확인
  - 입력을 받는 함수의 이름과 인자, SEGSEGV가 발생하는 주소 파악
- greybox fuzzing
  - 작년 black box 퍼징이 아닌 도달한 basic block 기준 새로운 basic block만 유지
- FSB
  - stdin의 입력이 printf의 인자로 들어가는 지 symbol을 체크
- 시그널별 처리
  - SIGSEGV, SIGCHLD, SIGTRAP, SIGFPE
- exploit을 위해 win function candidate 추출 이용
- 정적 분석을 통한 비교 문자열 추출
  - cmp instruction의 operand 체크, data segment에 사용되는 문자열 추출

부족한 부분

- cmp instruction/ cmp function sensitive
  - cmp instruction에 일부만 일치하면 나머지 부분을 일치시키기 위한 집중퍼징
  - 또는 일부만 일치한 testcase를 저장시켜 추후 퍼징에 사용
- directed fuzzing
  - 정적 분석을 통해 대상 프로그램의 실행경로를 파악하여 여러 실행 경로로 가도록 집중 퍼징
  - 메뉴문제(infinite loop)가 존재하는 경우 실행 경로 파악
    - 예) 1을 입력 후 2을 입력해야 취약점이 발생하는 UAF
    - 예) 덧셈, 뺄셈, 곱셈, 나눗셈을 모두 수행하여 2019를 만들어야 하는 문제

Q & A