**뉴비가 써본 angr 문서 – 상편**

**EGOIST**

Version 1.0

차 례

1. 들어 가기전에
2. Angr 알아보기
   1. Angr 간단소개
   2. Angr 구조비유
   3. VEX 간단정리
   4. IR 이해하기
3. Angr 구성원들
   1. Pyvex와 배경모듈
   2. Simuvex와 Claripy
   3. Angr
4. Angr 활용
   1. Analyses – 그래프도 그려보자
5. 문서 하편에서는..
   1. CTF 예제들
   2. 더 알아볼 것들 – angrop, Driller
6. APPENDIX
   1. 모듈들 설치방법

0. **들어가기 전에…**

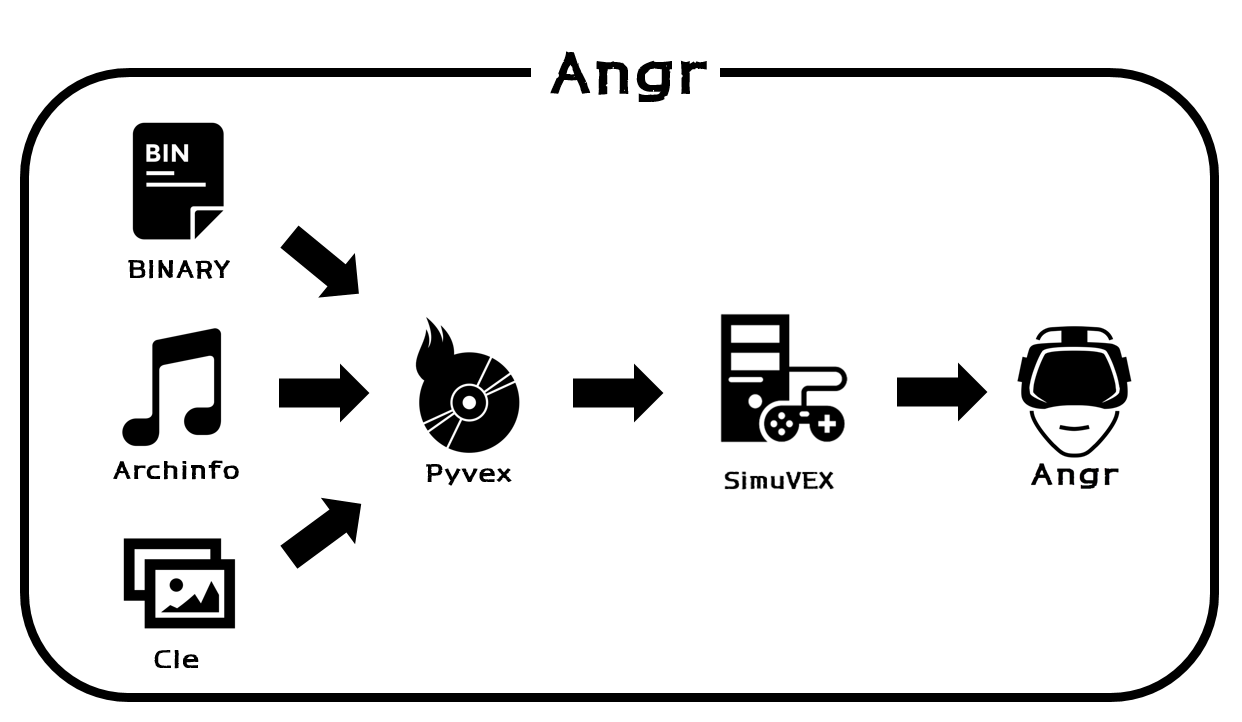
먼저, 이 문서는 angr을 처음 접하고 angr의 구조나 기능을 잘 모르는 분들을 위하여 조금 이해하기 쉬운 비유 등을 이용하여 작성한 문서이다. 그러다 보니 각 API의 상세한 파라미터나 변수들 혹은 세부적인 기능은 이 문서에서 누락되어 있어 이해를 돕기 위해 각 챕터마다 angr-doc페이지에서 소개한 링크를 첨부하였다.

1. **Angr 알아보기**

A. angr 간단소개

Angr은 UC Santa Barbara Computer security lab서 개발할 파이선 기반 바이너리 분석 프레임워크이다. 강력한 Symbolic execution 기능으로 2015 blackhat에서 발표되고 angr을 기반으로 제작된 취약점 분석 모듈 들이 2016 DEFCON이후 공개되어 이슈가 되고 있다. 뿐만 아니라 그래프를 그려줄 수 있는 networkx 모듈과 연동하여 Code-flow를 시각화 할 수 있는 점이나 중간언어인 VEX IR을 이용하여 코드의 동작 형태와 시뮬레 이션 결과를 저장하여 보여주는 점 등, 분석에서의 많은 이점을 보여주고 있다.

B. angr 구조비유

[그림 1]

ANGR의 각 모듈들에 대한 소개와 설명을 하기 전에 그림과 비유를 통하여 조금 이해를 하고 넘어 가면 도움이 될 것 같아 모식도를 그려보았다. [그림 1] 참조.

1. 스크립트(**Binary**), 배경스토리, 음악(**archinfo**), 그래픽(**cle**)등의 요소들을 준비
2. 각 요소들을 빌드하여(**Pyvex**) 게임CD를 제작
3. 제작된 게임CD를 게임기(**Simuvex**)에 넣고 게임 실행
4. 출력되는 화면(**Angr**)을 보면서 상황에 따른 게임 컨트롤

C. VEX는 누구 인가?

VEX는 리눅스 기반의 동적분석 프레임워크인 valgrind에서 사용하는 IR(중간 언어)이다.

아키텍쳐 같은 바이너리의 **배경정보**와 **분석 대상**을 읽어 들인 이후 **레지스터**나 **메모리에 저장되는 상태**, **분기 점,** **리턴 여부** 등으로 상세하게 정보를 분류하여 정해진 규칙의 구문으로 표기하는 형태로 띄고 있다.

IR블록을 생성하는 역할을 하는 모듈인 Pyvex의 설명을 들어가기 전에 아래의 링크를 참조하여 각 명령어나 변수에 대하여 이해를 하고 넘어가는 것을 추천한다.

<https://github.com/angr/pyvex/blob/master/README.md>

D. 그럼 IR은 누구 인가?

IR(Intermediate Representation)은 IR은 컴파일러나 가상머신에서 **소스코드로 재현**하기 위해서 내부적으로 사용하는 자료 구조이다. 소스코드로 변환하였을 때 정보의 손실이 없도록 정확해야 하며, 언어나 아키텍쳐에 구애되지 않고 최적화나 변환에 유용하다는 특징을 가지고 있어야 한다.

이러한 특징 덕분에 IR은 toolchain(대표적으로 llvm) 등을 통하여 매우 다양한 아키텍쳐의 코드로 변환할 수 있는 강력한 이점을 가지고 있고, 최적화 뿐만이 아닌 강력한 코드 난독화도 적용할 수 있는 등의 부가적인 이점도 가지고 있다.

2. **Angr 구성원들**

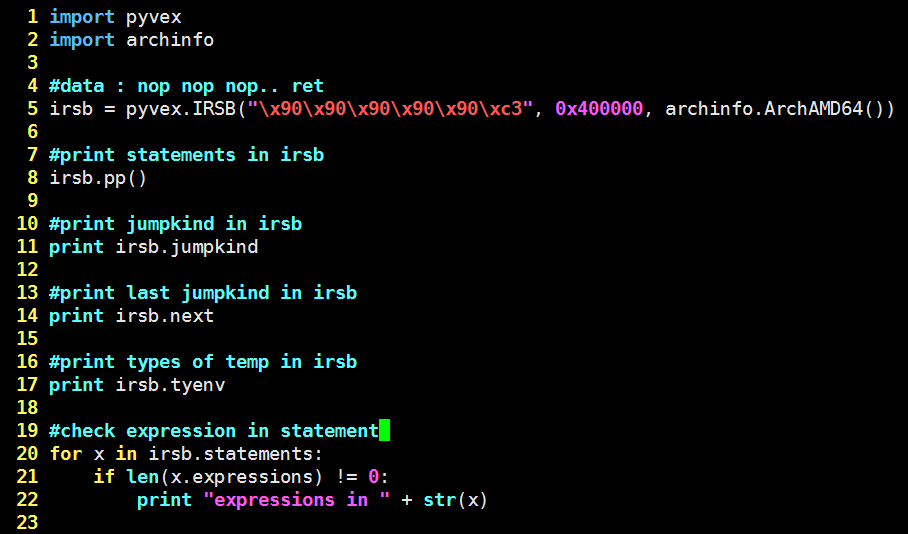
A. Pyvex와 배경정보들

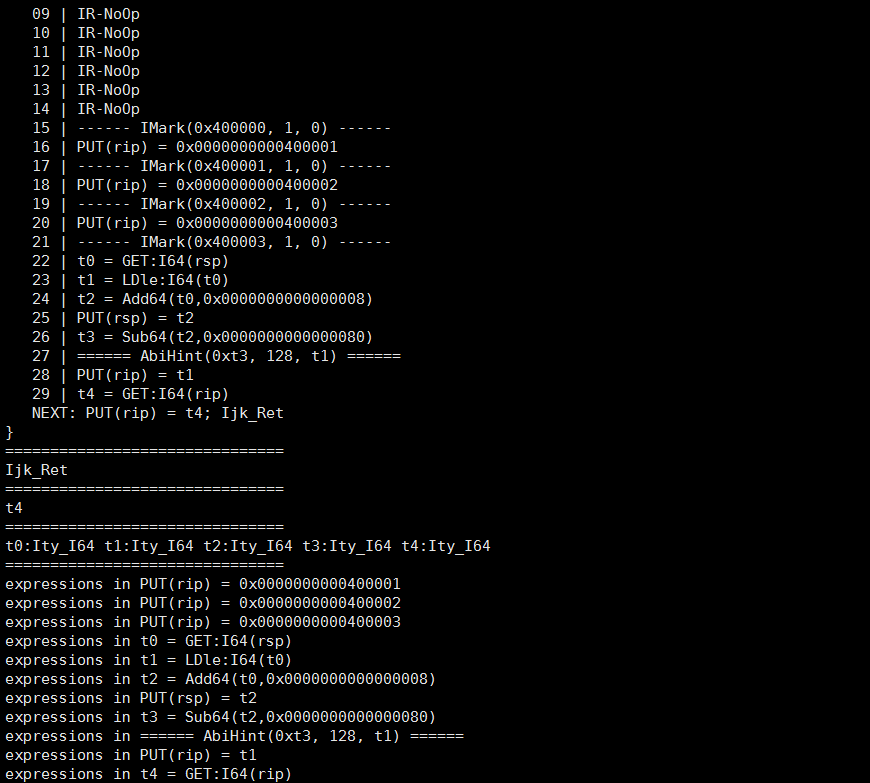
실제로 angr을 사용 하다 보면 angr과 Claripy등을 제외한 나머지 모듈에 대해서는 직접 코드를 만져 볼 일이 많이 없다. 그래서 Pyvex, CLE, Archinfo모듈은 예시보다는 내부구조에 대하여 설명하려고 한다.

A-1. Pyvex

먼저, Pyvex는 앞에서 설명한 VEX를 angr에서 python으로 바인딩한 라이브러리이다. 내부적으로 Lib-VEX라이브러리를 호출하여 어셈블리 코드들을 IR블록으로 변환하는 역할을 맡고 있다.

눈 여겨 볼만한 내부 인터페이스는 opcode와 start-address, 아키텍쳐 정보를 받아 VEX 블록으로 출력해 주는 IRSB정도여서 예제인 [그림 2]와 출력결과인 [그림 3]을 첨부하였다.

[그림 2]

[그림3]

IRSB예제에 대한 설명을 하자면 다음과 같다.

* IRSB의 파라미터 : 어셈블리 명령어(String), 시작주소, 아키텍쳐(archinfo 라이브러리)
* IRSB.pp() : 생성된 IRSB를 출력하여 보여줌
* IRSB.jumpkind : 해당 블록 이후 유추되는 분기점 (해당 예제에서는 \xc3이였으므로 ‘ljk\_Ret’)
* IRSB에서 정보 저장을 위하여 사용된 임시변수의 종류들 : IRSB.tyenv

A-2. archinfo

예제에서 IRSB를 생성하기 전에 파라미터로 archinfo가 들어가는데 이는 이름에서 유추할 수 있듯이 분석하려는 opcode들의 **아키텍쳐 정보**들을 내포하고 있는 라이브러리이다.

아래의 [그림 4]를 보면 코드 내부에서 레지스터와 데이터 타입의 종류, 크기 등등이 정의되어 있는 것과 멀티 플랫폼 기반의 강력한 디스어셈블러 엔진인 Capstone에서 엔디언 분류나 아키텍쳐 종류 여부를 가져와 사용하는 것도 확인할 수 있다.

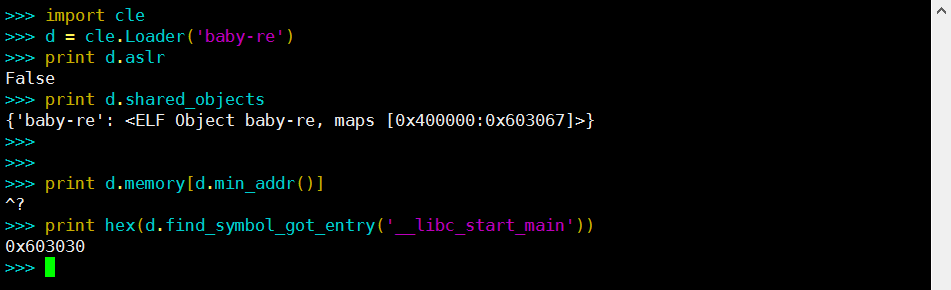
[그림 4]

A- 3. cle

cle는 angr에서 binary와 종속되는 라이브러리를 로딩해주는 loader역할을 수행한다.

Cle.Loader에서 직접 Binary만 load할 경우에는 기존에 정의된 라이브러리와 entry\_point로 설정 되는데, load\_option 을 설정하고자 하면 angr.Project에서 dictionary형태로 옵션을 전달할 수 있다.

loader에서 확인 가능한 값의 예시는 [그림 5 – 1], 설정 가능한 옵션들은 [그림5 – 2] 의 주석 참조.

[그림 5 - 1]

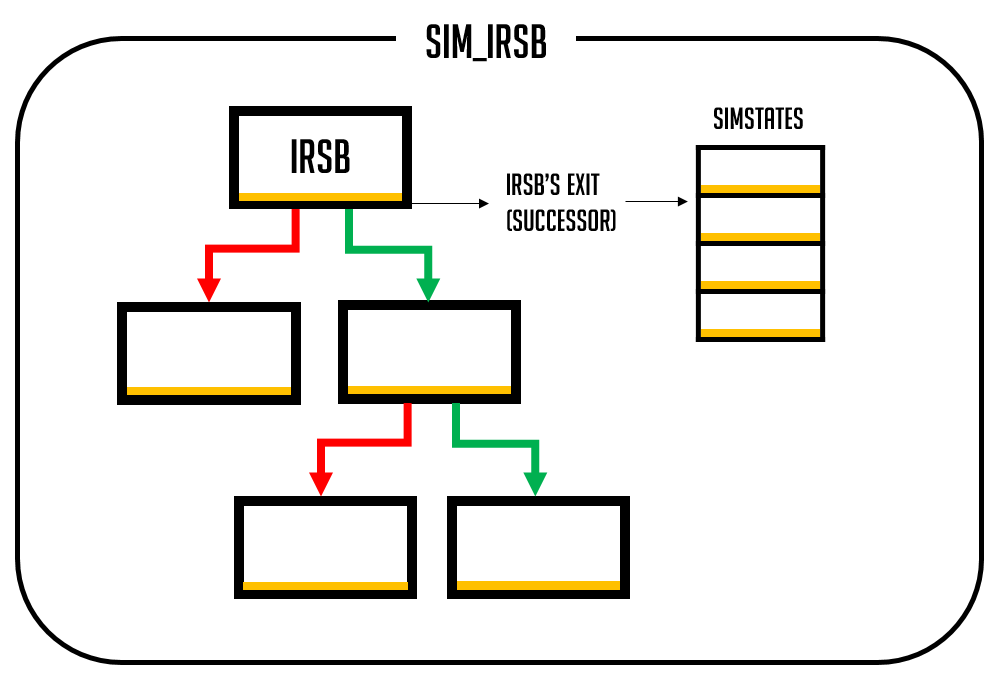
[그림 5 – 2]

main\_opts의 backend 옵션에서 추가설명을 하자면, load되는 대상이 아키텍쳐가 정해져 있지 않은 IDABin이나 Blob일 경우, custom\_arch 옵션으로 타겟 archinfo 오브젝트를 추가하여야 한다.

B. Simuvex와 Claripy

분석을 하다 보면 정적 분석만이 아닌 동적 분석을 통하여 해당 코드의 동작을 분석해야 할 필요가 생긴다. 이러한 점 때문에angr안에서 symbolic VEX emulator 모듈인 SimuVEX를 통하여 IR블록의 시뮬레이션을 진행하고 상태정보를 저장할 수 있도록 되어있다.

angr내부에서 실제 값의 조작은 factory모듈에서 프론트 엔드로써 다루어 지므로 Symbolic execution에 사용 되는 SimState와 SimProcedure 개념에 대하여 조금 설명이 필요할 것 같아 적어 보려고 한다.

[그림 6]

[그림 6]은 simuvex에서 symbolic execution을 지원하기 위하여 제공하는 SimIRSB객체의 모습이다.

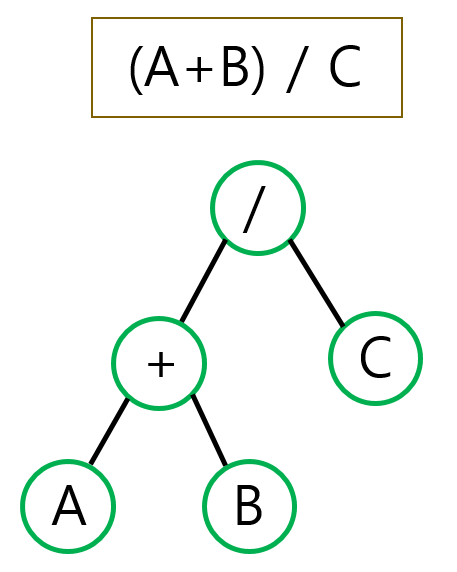
pyvex에서 보았던 irsb와는 다른 점이 있다면 각 irsb의 끝지점이 다음 irsb로 이어지는 successor라고 불리는 값으로 내부에 존재한다. 그리고 이 값은 시뮬레이션을 진행하면서 기준점, 상태정보를 저장하는 SimState라는 값이 되어 SimIRSB내부에 존재하게 된다.

여기 까지만 보았을 때는 SimIRSB를 통해 symbolic execution이 진행되는 것으로 이해된다. 하지만, Sim-IRSB만으로는 시뮬레이션을 진행할 수가 없는데 이는 실제로 바이너리가 실행되는 것이 아닌 IRSB를 통해 진행되므로, 라이브러리 안의 시스템콜은 진행할 수가 없기 때문이다. 때문에 파이선으로 기능을 유사하게 구현한 모듈인 SimProcedure를 통해 진행한다.

Simstate와 SimProcedure가 사용되는 예시를 보고 싶다면 angr에서 factory탭에서 참조.

다음으로 claripy는 Simuvex 안에서 동작하는 angr의 핵심에 해당된다고 볼 수 있는 solver engine을 담당 한다. 실제로 claripy는 동작방식만 이해하고 내부 요소들을 크게 몰라도 angr사용에 무리가 없으므로 여기 에서는 claripy이해를 위해서 알고 넘어가면 좋을 AST와 bit-vector값에 대하여 설명하고 넘어가려고 한다.

claripy에서 값을 구하는 방식은 구해야 할 대상을 AST(추상 구문 트리)의 형태로 두고서 트리의 요소안에 피연산자, 연산자 들을 대입하여 구현된 식으로 solver를 통해 해를 구하게 되는 형식이다. [그림 7]은 AST를 시각화한 모습이다.

[그림 7]

이때 AST에 들어가는 대상은 bit-vector의 형태로 두게 되고, 설정이 끝나면 symbolic execution을 진행 하여 값을 구하게 된다. (내부적으로 SMT resolver모듈인 z3가 사용되는데, 검색하여 이해하기를 권장한다.) 여기서 bit-vector(비트벡터)는 비트 값의 유무를 통해 해당 비트의 자리에 해당하는 자료의 값이 있는 지의 여부를 판단하는 방식을 말한다. 비트벡터의 사용이유 중 하나는 [그림 8]과 같다고 볼 수 있다.

[그림 8]

|  |  |
| --- | --- |
| [그림 8 - 1] | [그림 8 – 2] |
|  |  |

[그림 8 – 1]은 bit-vector로 해당 요소의 유무 판단을 기록한 것이고 [그8 – 2]는 실제 요소를 리스트화 시킨 것이다. 만약 A와 B에서 공통된 요소를 찾는 연산을 할 경우 속도는 비트연산을 통해 진행하는 [그림8 – 1]이 월등하게 속도가 빠를 것이다. 이러한 점 때문에 빠른 연산을 요구하는 symbolic execution내부에서 bitvec-tor로 구현되어 있다.

해당 문서에서는 claripy의 동작 방식의 이해를 위하여 중요한 요소들만 설명을 했는데, 실제 사용되는 요소들은 angr-doc내의 다음 링크를 참조하는 것을 권장한다.

https://goo.gl/KUZYY4

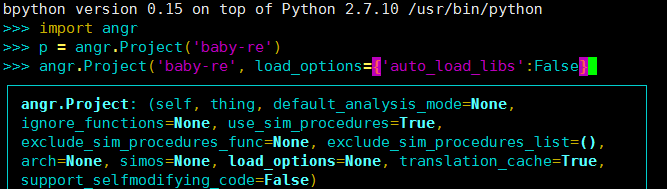
C. Angr

메인 모듈인 만큼 다양한 기능들과 내부 모듈을 가지고 있다. 때문에 해당 모듈은 메인 클래스부터 시작하여 자주 사용하는 기능들을 예시와 함께 차례대로 설명하고자 한다.

C-1. angr.Project

Angr을 사용하기 위해서 분석대상이나 load\_option이 들어가는 메인 클래스에 해당된다. 여기서 나온 Project클래스를 기점으로 hooking이나, symbolic execution, 분석기능을 사용할 수 있다.

몇 가지 파라미터가 있으나, 대부분 바이너리 경로나 로딩 옵션만 들어간다. (로딩옵션은 cle 참조)

[그림 9]

C-2. analyses

이름 그대로 바이너리의 분석을 지원하는 모듈이다.

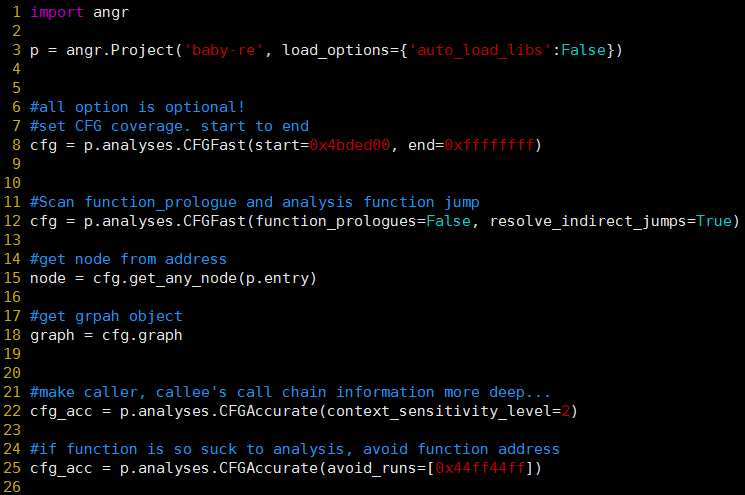
가장 많이 사용하는 CFG는 코드 실행 흐름 그래프를 구성하여 내부 정보들을 보여준다.

한 분기점에서 다른 분기점으로 이동하기 이전까지의 IRSB인 Node와 주소 값으로 다음 Node까지의 연결을 지어주는 Edge로 구성되어 있는데 옵션에 따라 더욱 더 자세하게 보여줄 수 있는 CFGAccurate가 있고, CFG를 생성하고 나면 그래프 정보를 시각화 하여 보여주는 객체도 내부에 존재한다. (3-B챕터 참조)

간단한 예제와 내부변수를 [그림 10]에 참조하였다. 더욱 자세한 내용이나 파라미터는 아래의 링크 확인

(특히 CFGAccurate에서 알아 두어야 할 파라미터가 있으므로 읽어보는 것이 좋다)

http://goo.gl/7HXaYc

[그림 10]

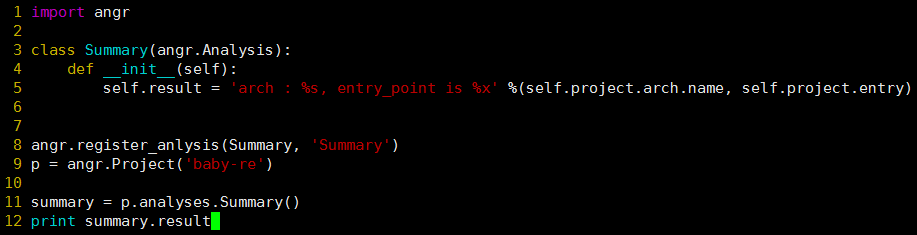
다음으로 DFG는 CFG와는 조금 다르게 레지스터, 변수, 파라미터 등등의 값이 코드를 실행하며 변화한 정보를 구성하여 보여주는 기능이다. CFG와 마찬가지로 그래프를 통한 시각화 표현이 가능하며, 각 node가 IRSB에서 데이터의 흐름만 보여주는 형태를 띠고 있다. [그림 11]은 dfg의 간단한 예시와 출력 결과이다.

[그림 11]

|  |  |
| --- | --- |
| [그림 11 - 1] | [그림 11 – 2] |
|  |  |

이 외에도 참고할 수 있는 다양한 기능들이 있는데 그 중에서 부각되는 또 다른 기능은 직접 분석기능을 만들어 등록할 수 있는 register\_analysis도 있다. 사용법은 [그림 12]

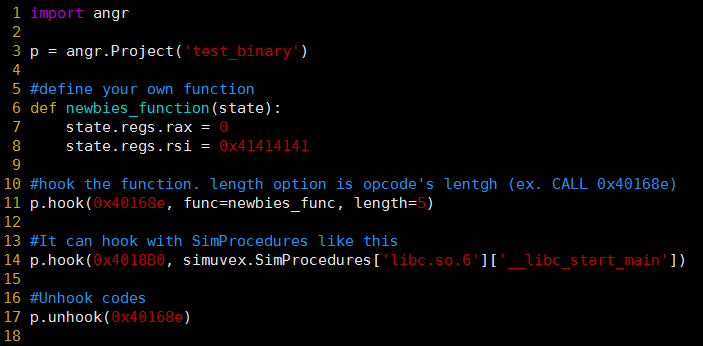
[그림 12]



C-2. hook, factory

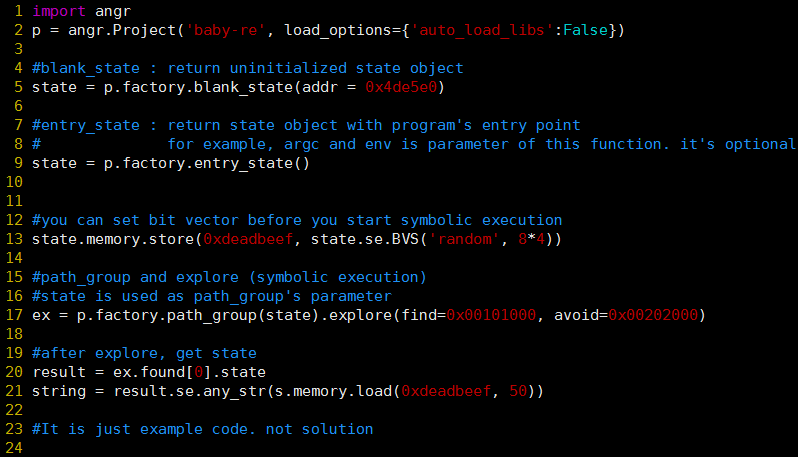
Hook 기능은 Symbolic execution에서 시스템콜 혹은 Code-caving을 원하는 주소에서 원래의 함수 대신 본인이 원하는 함수를 실행하고 싶을 때 해당 함수 대신 파이선에서 정의한 함수로 후킹하여 진행할 수 있게 해주는 모듈이다. Simuvex에서도 언급되었지만 실제 바이너리의 실행이 아닌 Symbolic execution이므로 라이브러리 내의 함수들이 파이선으로 구현되어 있는 SimProcedure모듈이 내부적으로 hook기능을 통해 구동되므로 눈여겨봐야 할 부분이다. [그림 13]

[그림 13]



factory에서 자주 사용하는 큰 기능은 분석을 시작할 주소를 설정하거나 레지스터와 메모리를 bitvector로 설정하여 symbolic execution을 통해 해당 값을 도출할 수 있게 도와주는 state가 있고 이 모듈에서 가장 자주 언급되고 사용되는 symbolic execution을 진행하는 path\_group기능이 있다. 간단한 사용법은 [그림 14]

[그림 14]

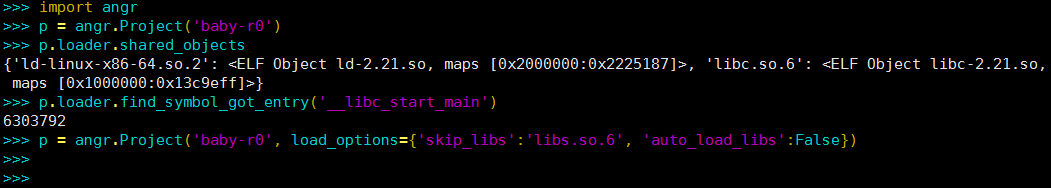


설명을 덧붙이자면, explore함수에서 파라미터find는 원하는 방향으로 진행하고자 하는 주소 값으로 다수의 주소지정을 하고자 하면 리스트로 대입할 수 있으며, avoid는 dead-end로 끝나는 주소로, 이 역시 리스트로 다수의 주소지정이 가능하다.

C-3. loader

Loader는 바이너리를 로딩 할 때 옵션이나 아키텍쳐 환경 등을 지원하는 기능이 들어있는데 앞에서 살펴 보았던 cle가 frontend로 구현된 것이다. [그림 15]

[그림 15]



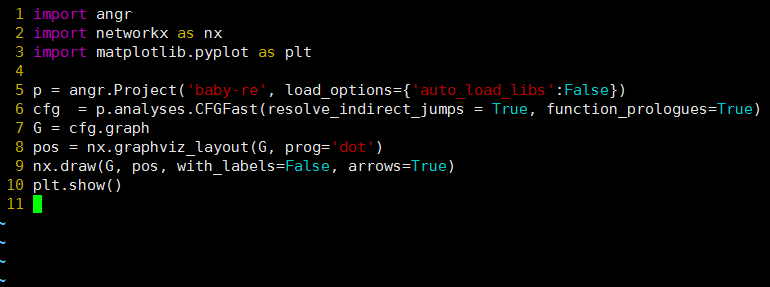
3. angr 활용하기

A. analyses – 그래프도 그려보자

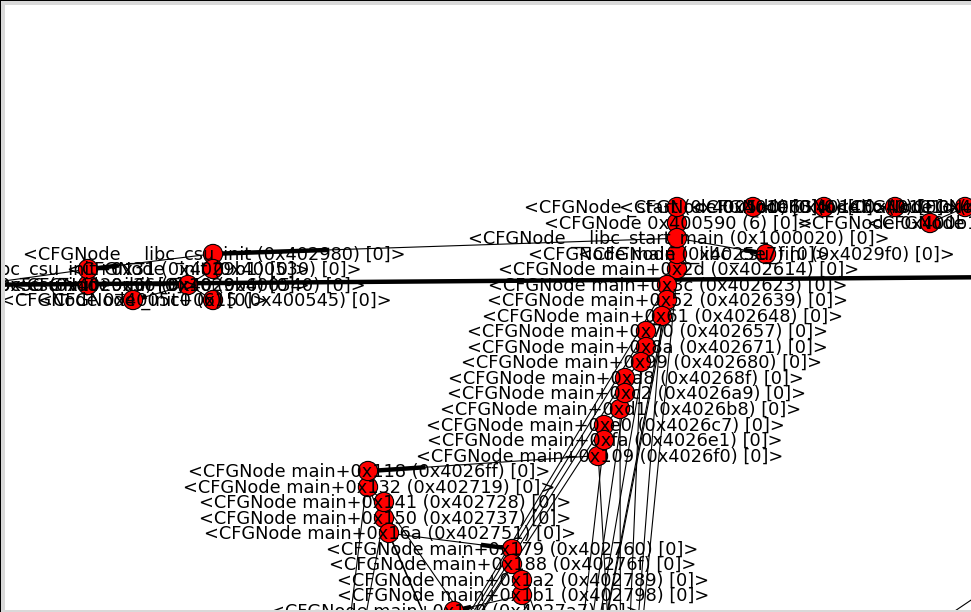
Analyses내의 그래프 객체는 실제로 networkx와 matplotlib같은 시각화 모듈을 이용하여 해당 바이너리의 코드흐름과 분기점을 시각화 하여 보여준다. (angr과는 별개로 두 모듈의 설치가 필요한데, 둘다 pip를 이용하여 간단하게 설치할 수 있다.)

그림을 그리는 간단한 예는 [그림 16]에서 참조

[그림 16]



CFGFast내부에는 점과 선으로 코드 흐름을 표현할 수 있도록 Digraph라는 변수로 저장되어 있다. 이 값을 graphviz\_layout이나 draw등의 함수에서 옵션을 주거나 빼는 식으로 조정하여 더 다양한 그래프를 그려볼 수 있다. 아래의 [그림 17]은 출력된 결과를 일부 확대한 모습이다.

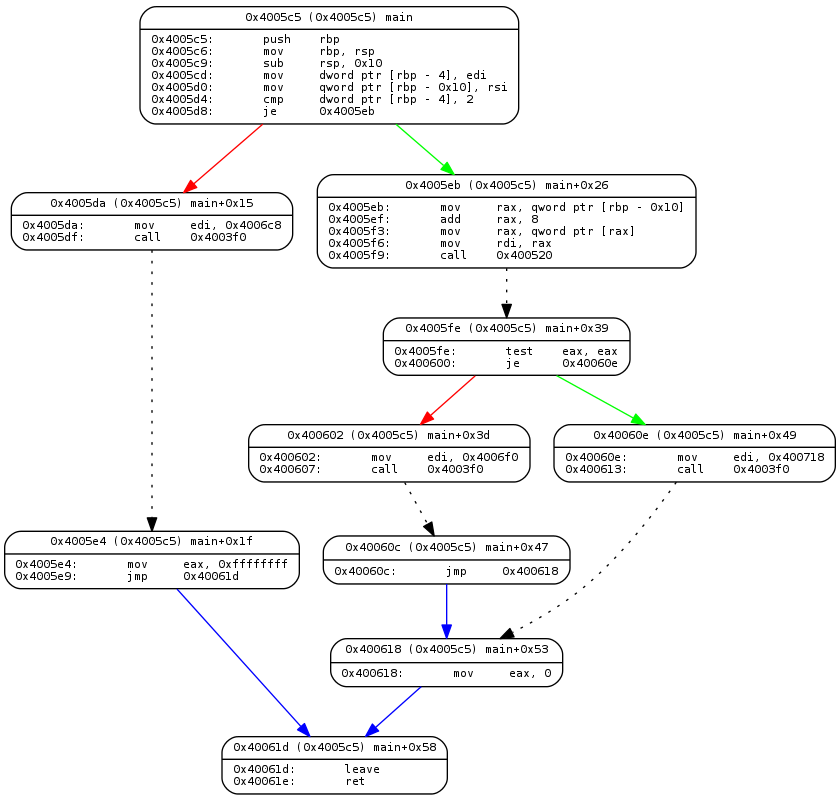
[그림 17]

하지만 [그림 17]을 보다 보면 기대하던 Code-flow 그래프의 형태와는 달리 다소 보기 불편한 형태로 되어있다. 이는 Angr내부에서는 시각화 할 수 있는 Digraph정보만 제공하고 있기 때문인데 그래서 angr을 사용하던 유저들이 직접 모듈을 만들어 제공하는지 조사해보았고 이 글을 작성한 기준으로 angr-util과 egoist 두개가 있는 것을 확인하였다.

Angr-util의 경우 python모듈로 api를 제공하며 github로 제공하고 있었는데 링크는 아래의 주소와 같다.

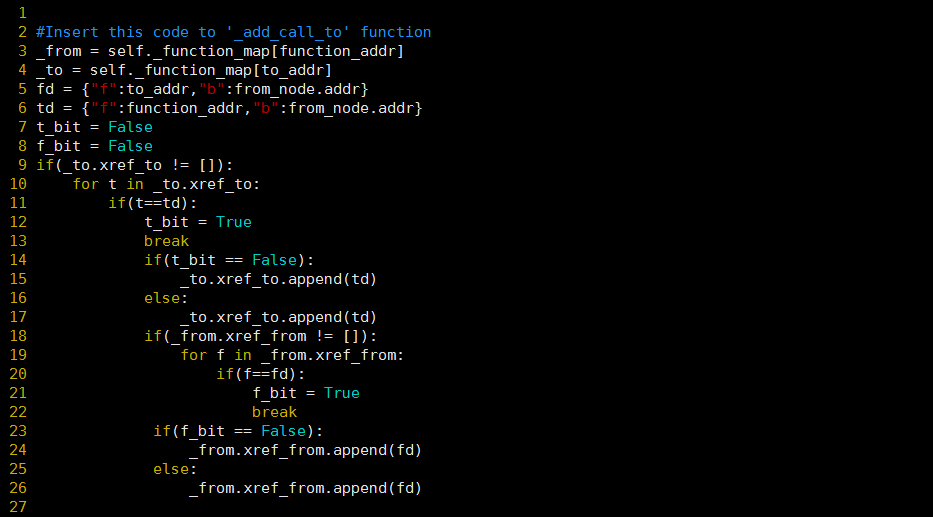
http://goo.gl/7HXaYc

[그림 18]

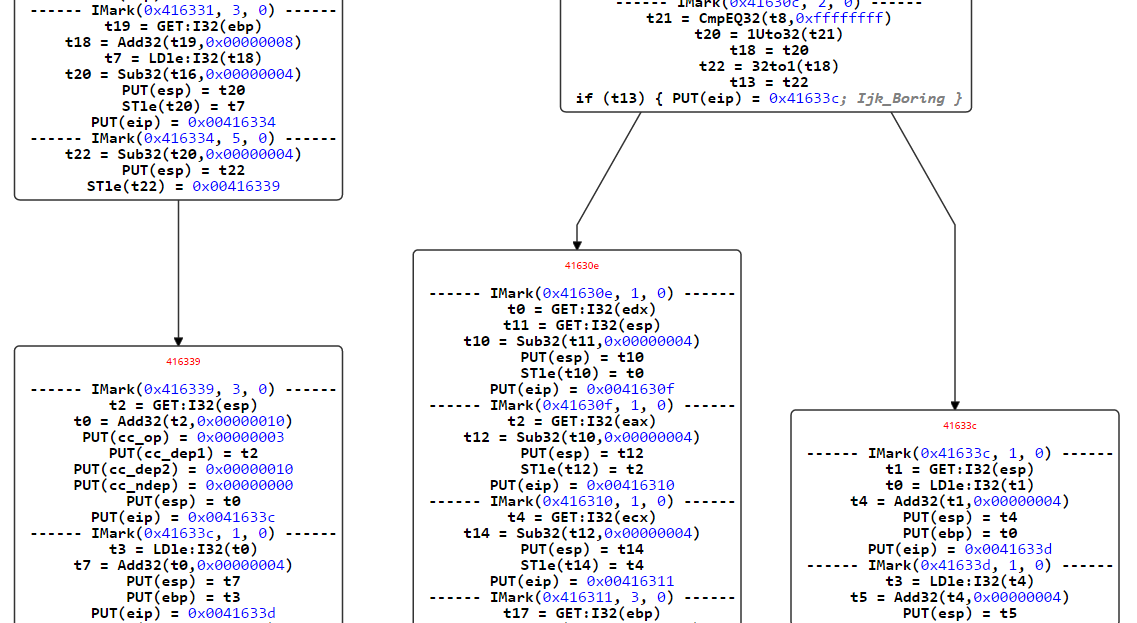


그리고 egoist는 웹으로 바이너리 분석 서비스를 제공하고 있어 해당 서비스 내부에 타겟 바이너리의 코드플로우를 보여주는 탭이 있었다. 개발자에게 질문을 한 결과 ‘angr\angr\knowledge\function\_manager.py’ 경로에 [그림 19]의 코드를 삽입하여 xref를 수집하고 시각화 라이브러리를 따로 구현하였다고 한다

[그림 19]



[그림 20 ]



4. 문서 하편에서는..

A. CTF 예제들

원래 이 문서의 최종 목표로 실제 분석 예제들과 문제들을 풀어가는 과정을 문서에 실으려고 하였으나, 여러가지의 사정이 있어 수록하지 못하였다. 문서를 마무리하면서도 되게 안타까운 부분 중 하나였던 지라 문서 후편에서는 CTF문제들을 풀어가는 과정을 수록하려고 한다.

B. 더 알아볼 것들

해당 문서를 쓰기 시작한 기점인 8월 중순 이후에 Shellphish팀에서 rop가젯을 찾아주고 체인을 만들어주는 모듈인 angrop 와 angr기반으로 만들어진 AFL(American Fuzzing Lip) 강화판인 Driller가 Defcon에서 소개 되었다. 따라서 추후 작성하게 될 문서에서는 해당 문서에서 부족했던 예제의 보충과 동시에 해당 모듈들을 소개하려고 한다.

5. 문서후기

시간문제 때문에 열심히 쓰려고 한 부분도 다소 놓친 부분들이 있습니다. ㅜㅜ

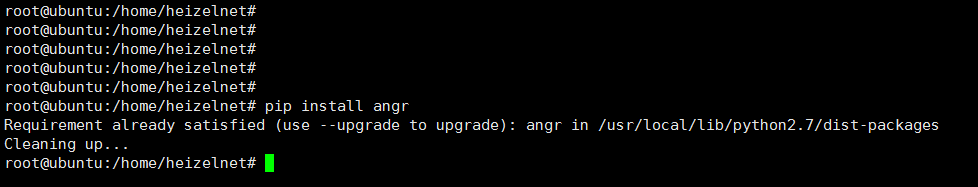
후편에서 더 보충하고 열심히 쓰겠습니다. 검수할 때 도와주신 분들 감사합니다

6. APPENDIX

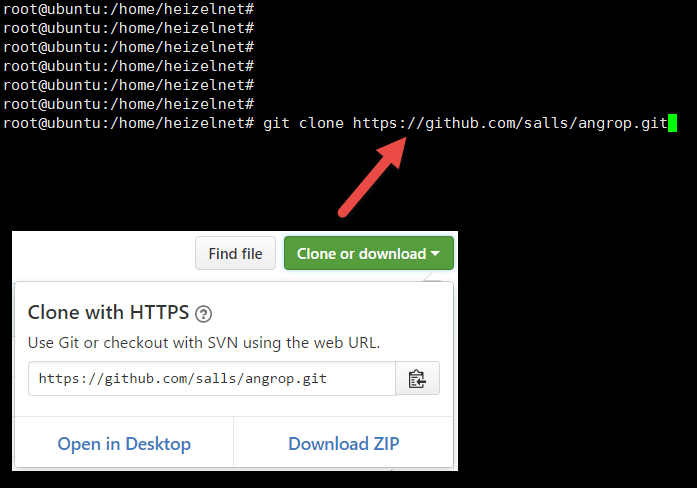
A. 설치방법

파이선 모듈이므로 `pip install angr` 명령어를 통해 angr과 내부의 모듈을 모두 한꺼번에 간단하게 설치할 수 있다. [그림 20] 참조.

[그림 20]



하지만 angr 모듈이 에러이슈 등에 대한 수정이 자주 있는 편인데 이 때문에 이슈가 수정된 버전의 angr 을 이용하여 개발을 하고자 할 경우 [그림 21]처럼 github링크를 복사하여 install하는 것을 추천한다.

[그림 21]