

《漏洞利用及渗透测试基础》实验报告

姓名：张洋 学号：2111460 班级：信安二班

实验名称：

Angr 应用实例

实验要求：

根据课本 8.4.3 章节，复现 sym-write 示例的两种 angr 求解方法，并就如何使用 angr 以及怎么解决一些实际问题做一些探讨。

实验过程：

1. 安装 Angr

Windows 下安装 angr。在 windows 系统下安装 python3，添加 python 环境变量。然后，打开命令控制台，使用 PIP 命令安装 angr: `pip install angr`。

测试安装。输入命令 `python`，进入 python 界面，然后输入 `import angr`，如果成功，则说明安装没有问题。

```
(base) D:\>python
Python 3.9.12 (main, Apr  4 2022, 05:22:27) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import angr
>>> |
```

2. Angr 示例

在 angr-doc 里有各类 Example，展示了 Angr 的用法，比如 `cmu_binary_bomb`、`simple_heap_overflow` 等二进制爆破、堆溢出等漏洞挖掘、软件分析的典型案例。下面，我们以 `sym-write` 为例子，来说明 angr 的用法。

源码 `issue.c`（详见 `angr-doc-master/examples/sym-write/issue.c`）如下：

```
#include <stdio.h>
char u=0;
int main(void)
{
    int i, bits[2]={0,0};
    for (i=0; i<8; i++) {
        bits[(u&(1<<i))!=0]++;
    }
    if (bits[0]==bits[1]) {
        printf("you win!");
    }
    else {
        printf("you lose!");
    }
    return 0;
}
```

`1<<i` 即将 `i` 左移一位。`&`即按位与，for 循环即为将 `u` 的每一位都取出来比较，如果为 0，则 `bit[0]++`，否则 `bit[1]++`。只有当 `u` 的二进制中 1 的个数与 0 的个数相同时才能胜利。

源码 solve.py (详见 angr-doc-master\examples\sym-write\solve.py) 如下:

```
import angr
import claripy

def main():

    # 1. 新建一个工程，导入二进制文件，后面的选项是选择不自动加载依赖项，不会
    # 自动载入依赖的库
    p = angr.Project('./issue', load_options={"auto_load_libs": False})

    # 2. 初始化一个模拟程序状态的 SimState 对象 state，该对象包含了程序的内存、
    # 寄存器、文件系统数据、符号信息等等模拟运行时动态变化的数据
    # blank_state():可通过给定参数 addr 的值指定程序起始运行地址
    # entry_state():指明程序在初始运行时的状态，默认从入口点执行
    # add_options 获取一个独立的选项来添加到某个 state 中，更多选项说明见
    # https://docs.angr.io/appendix/options
    # SYMBOLIC_WRITE_ADDRESSES: 允许通过具体化策略处理符号地址的写操作
    state = p.factory.entry_state(add_options={angr.options.SYMBOLIC_WRITE_ADDRESSES})

    # 3. 创建一个符号变量，这个符号变量以 8 位 bitvector 形式存在，名称为 u
    u = claripy.BVS("u", 8)
    # 把符号变量保存到指定的地址中，这个地址是就是二进制文件中.bss 段 u 的地址
    state.memory.store(0x804a021, u)

    # 4. 创建一个 Simulation Manager 对象，这个对象和我们的状态有关系
    sm = p.factory.simulation_manager(state)

    # 5. 使用 explore 函数进行状态搜寻，检查输出字符串是 win 还是 lose
    # state.posix.dumps(1)获得所有标准输出
    # state.posix.dumps(0)获得所有标准输入
    def correct(state):
        try:
            return b'win' in state.posix.dumps(1)
        except:
            return False
    def wrong(state):
        try:
            return b'lose' in state.posix.dumps(1)
        except:
            return False
    # 进行符号执行得到想要的状态，即得到满足 correct 条件且不满足 wrong 条件的
    # state
    sm.explore(find=correct, avoid=wrong)
```

```
# 也可以写成下面的形式，直接通过地址进行定位
# sm.explore(find=0x80484e3, avoid=0x80484f5)
# 获得到 state 之后，通过 solver 求解器，求解 u 的值
# eval_upto(e, n, cast_to=None, **kwargs) 求解一个表达式指定个数个可能的求解方
案 e - 表达式 n - 所需解决方案的数量
# eval(e, **kwargs) 评估一个表达式以获得任何可能的解决方案。 e - 表达式
# eval_one(e, **kwargs) 求解表达式以获得唯一可能的解决方案。 e - 表达式
return sm.found[0].solver.eval_upto(u, 256)

if __name__ == '__main__':
    # repr()函数将 object 对象转化为 string 类型
    print(repr(main()))
```

上述代码定义了一个 main 函数。

整个 python 程序将执行 `print(repr(main()))` 语句，进而，将 main 函数的返回值打印出来，`repr()` 函数将 object 对象转化为 string 类型。

在上述 Angr 示例中，几个关键步骤如下：

(1) 新建一个 Angr 工程，并且载入二进制文件。`auto_load_libs` 设置为 `false`，将不会自动载入依赖的库，默认情况下设置为 `false`。如果设置为 `true`，转入库函数执行，有可能给符号执行带来不必要的麻烦。

(2) 初始化一个模拟程序状态的 `SimState` 对象 `state`（使用函数 `entry_state()`），该对象包含了程序的内存、寄存器、文件系统数据、符号信息等等模拟运行时动态变化的数据。此外，也可以使用函数 `blank_state()` 初始化模拟程序状态的对象 `state`，在该函数里可通过给定参数 `addr` 的值指定程序起始运行地址。

(3) 将要求解的变量符号化，注意这里符号化后的变量存在二进制文件的存储区。

(4) 创建模拟管理器（Simulation Managers）进行程序执行管理。初始化的 `state` 可以经过模拟执行得到一系列的 `states`，模拟管理器 `sm` 的作用就是对这些 `states` 进行管理。

(5) 进行符号执行得到想要的状态，得到想要的状态。上述程序所表达的状态就是，符号执行后，源程序里打印出的字符串里包含 `win` 字符串，而没有包含 `lose` 字符串。在这里，状态被定义为两个函数，通过符号执行得到的输出 `state.posix.dumps(1)` 中是否包含 `win` 或者 `lose` 的字符串来完成定义。

这里也可以用 `find=0x80484e3`, `avoid=0x80484f5` 来代替，即通过符号执行是否到达特定代码区的地址。使用 IDA 反汇编可知 `0x80484e3` 是 `printf("you win!")` 对应的汇编语句；`0x80484f5` 则是 `printf("you lose!")` 对应的汇编语句。

(6) 获得到 `state` 之后，通过 solver 求解器，求解 `u` 的值。这里有多个函数可以使用，`eval_upto(e, n, cast_to=None, **kwargs)` 求解一个表达式多个可能的求解方案，`e`-表达式，`n`-所需解决方案的数量；`eval(e, **kwargs)` 评估一个表达式以获得任何可能的解决方案；`eval_one(e, **kwargs)` 求解表达式以获得唯一可能的解决方案。

实验验证。在 windows 环境下，选择填写的 solve.py，点右键选择 Edit with IDLE，将弹出界面：

```
File Edit Format Run Options Window Help
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Author: xoreaxeaxeax
Modified by David Manouchehri <manouchehri@protonmail.com>
Original at https://lists.cs.ucsb.edu/pipermail/angr/2016-August/000167.html
The purpose of this example is to show how to use symbolic write addresses.
"""

import angr
import claripy

def main():
    p = angr.Project('./issue', load_options={"auto_load_libs": False})

    # By default, all symbolic write indices are concretized.
    state = p.factory.entry_state(add_options={angr.options.SYMBOLIC_WRITE_A

    u = claripy.BVS("u", 8)
    state.memory.store(0x804a021, u)

    sm = p.factory.simulation_manager(state)

    def correct(state):
        try:
            return b'win' in state.posix.dumps(1)
        except:
            return False
    def wrong(state):
        try:
            return b'lose' in state.posix.dumps(1)
        except:
            return False

    sm.explore(find=correct, avoid=wrong)

    # Alternatively, you can hardcode the addresses.
    # sm.explore(find=0x80484e3, avoid=0x80484f5)
```

Ln: 1 Col: 0

选择 Run -> run model, 界面如下:

```
File Edit Shell Debug Options Window Help
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> = RESTART: E:\Sophomore2\软件安全\tools\angr-doc-master\examples\sym-write\solve
.py
WARNING | 2022-05-02 10:11:34,263 | [32mangr.storage.memory_mixins.default_filler_mixin[0m | [32mThe program is accessing register with an unspecified value. This could indicate unwanted behavior. [0m
WARNING | 2022-05-02 10:11:34,320 | [32mangr.storage.memory_mixins.default_filler_mixin[0m | [32mangr will cope with this by generating an unconstrained symbolic variable and continuing. You can resolve this by: [0m
WARNING | 2022-05-02 10:11:34,346 | [32mangr.storage.memory_mixins.default_filler_mixin[0m | [32m1) setting a value to the initial state [0m
WARNING | 2022-05-02 10:11:34,362 | [32mangr.storage.memory_mixins.default_filler_mixin[0m | [32m2) adding the state option ZERO_FILL_UNCONSTRAINED_(MEMORY,REGISTERS), to make unknown regions hold null [0m
WARNING | 2022-05-02 10:11:34,390 | [32mangr.storage.memory_mixins.default_filler_mixin[0m | [32m3) adding the state option SYMBOL_FILL_UNCONSTRAINED_(MEMORY,REGISTERS), to suppress these messages. [0m
WARNING | 2022-05-02 10:11:34,417 | [32mangr.storage.memory_mixins.default_filler_mixin[0m | [32mFilling register edi with 4 unconstrained bytes referenced from 0x8048521 (__libc_csu_init+0x1 in issue (0x8048521)) [0m
WARNING | 2022-05-02 10:11:34,448 | [32mangr.storage.memory_mixins.default_filler_mixin[0m | [32mFilling register ebx with 4 unconstrained bytes referenced from 0x8048523 (__libc_csu_init+0x3 in issue (0x8048523)) [0m
[51, 57, 240, 60, 75, 139, 78, 197, 23, 142, 90, 29, 209, 154, 99, 212, 163, 102, 108, 166, 172, 105, 169, 114, 120, 53, 178, 184, 71, 135, 77, 83, 89, 202, 147, 86, 92, 153, 150, 156, 141, 101, 106, 165, 43, 113, 232, 226, 177, 116, 46, 180, 45, 58, 198, 15, 195, 201, 85, 204, 30, 149, 210, 27, 216, 39, 225, 170, 228, 54]
>>>
```

蓝色部分, 就是输出的 u 的求解的结果, 因为我们采用了 eval_upto 函数, 给出了多个解, 对每个解我们都可以带入源程序进行验证。

其他解法。对于上述程序, 我们也可以采用下面的代码来进行求解:

```
#!/usr/bin/env python
# coding=utf-8
import angr
import claripy
def hook_demo(state):
    state.regs.eax = 0

p = angr.Project("./issue", load_options={"auto_load_libs": False})
# hook 函数: addr 为待 hook 的地址
# hook 为 hook 的处理函数, 在执行到 addr 时, 会执行这个函数, 同时把当前的 state 对象作为参数传递过去
# length 为待 hook 指令的长度, 在执行完 hook 函数以后, angr 需要根据 length 来跳过这条指令, 执行下一条指令
# hook 0x08048485 处的指令 (xor eax, eax), 等价于将 eax 设置为 0
# hook 并不会改变函数逻辑, 只是更换实现方式, 提升符号执行速度
```



```

p.hook(addr=0x08048485, hook=hook_demo, length=2)
state = p.factory.blank_state(addr=0x0804846B,
add_options={"SYMBOLIC_WRITE_ADDRESSES"})
u = claripy.BVS("u", 8)
state.memory.store(0x0804A021, u)
sm = p.factory.simulation_manager(state)
sm.explore(find=0x080484DB)
st = sm.found[0]

print(repr(st.solver.eval(u)))

```

上述代码与前面的解法有三处区别：

采用了 hook 函数,将 0x08048485 处的长度为 2 的指令通过自定义的 hook_demo 进行替代,功能是一致的,原始 xor eax, eax 和 state.regs.eax = 0 是相同的作用, 这里只是演示,可以将一些复杂的系统函数调用,比如 printf 等,可以进行 hook, 提升符号执行的性能。

进行符号执行得到想要的状态,有变化,变更为 find=0x080484DB。因为源程序 win 和 lose 是互斥的,所以,只需要给定一个 find 条件即可。

最后,eval(u)替代了原来的 eval_upto,将打印一个结果出来。

实验验证:

```

File Edit Format Run Options Window Help
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Author: xoreaxeaxeax
Modified by David Manouchehri <manouchehri@protonmail.com>
Original at https://lists.cs.ucsb.edu/pipermail/angr/2016-August/000167.html

The purpose of this example is to show how to use symbolic write addresses.
"""

import angr
import claripy

def hook_demo(state):
    state.regs.eax=0
p = angr.Project('./issue', load_options={"auto_load_libs": False})

p.hook(addr=0x08048485, hook=hook_demo, length=2)

    # By default, all symbolic write indices are concretized.
state = p.factory.blank_state(addr=0x0804846B, add_options={"SYMBOLIC_WRITE_ADDRE
u = claripy.BVS("u", 8)
state.memory.store(0x804A021, u)
sm = p.factory.simulation_manager(state)
sm.explore(find=0x080484DB)
st=sm.found[0]
print(repr(st.solver.eval(u)))

```

运行结果为：

```
File Edit Shell Debug Options Window Help
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: E:\Sophomore2\软件安全\tools\angr-doc-master\examples\sym-write\solve
.py
WARNING | 2022-05-02 11:32:30,411 | [32mangr.storage.memory_mixins.default_filer_mixin[0m | [32mThe program is accessing memory with an unspecified value. This could indicate unwanted behavior.[0m
WARNING | 2022-05-02 11:32:30,458 | [32mangr.storage.memory_mixins.default_filer_mixin[0m | [32mangr will cope with this by generating an unconstrained symbolic variable and continuing. You can resolve this by:[0m
WARNING | 2022-05-02 11:32:30,479 | [32mangr.storage.memory_mixins.default_filer_mixin[0m | [32ml) setting a value to the initial state[0m
WARNING | 2022-05-02 11:32:30,491 | [32mangr.storage.memory_mixins.default_filer_mixin[0m | [32m2) adding the state option ZERO_FILL_UNCONSTRAINED_{MEMORY,REGISTERS}, to make unknown regions hold null[0m
WARNING | 2022-05-02 11:32:30,512 | [32mangr.storage.memory_mixins.default_filer_mixin[0m | [32m3) adding the state option SYMBOL_FILL_UNCONSTRAINED_{MEMORY,REGISTERS}, to suppress these messages.[0m
WARNING | 2022-05-02 11:32:30,532 | [32mangr.storage.memory_mixins.default_filer_mixin[0m | [32mFilling memory at 0x7fff0000 with 4 unconstrained bytes referenced from 0x8048472 (main+0x7 in issue (0x8048472))[0m
WARNING | 2022-05-02 11:32:30,554 | [32mangr.storage.memory_mixins.default_filer_mixin[0m | [32mFilling register ebp with 4 unconstrained bytes referenced from 0x8048475 (main+0xa in issue (0x8048475))[0m
71
>>>
```

Ln: 13 Col: 0

3. 如何使用 angr

使用 angr 可以分为三个步骤：加载二进制文件、定义分析目标、运行分析。

首先，使用`angr.Project()`函数加载二进制文件。接下来，定义分析目标。angr 支持符号执行、静态分析、动态分析等多种分析方式，具体使用方式可以参考官方文档。最后，运行分析。在符号执行的情况下，可以使用`simulation.explore()`函数进行分析。在静态分析和动态分析的情况下，可以使用不同的函数进行分析。

4. 利用 angr 解决一些实际问题

在实际问题中，angr 可以用于漏洞挖掘、二进制加固、漏洞利用等多个方面。以下是一些实际问题的解决思路：

(1) 漏洞挖掘：使用符号执行技术，通过构造特殊的输入，找到程序中存在的漏洞。例如，可以使用`simulation.found`属性找到满足某个条件的输入。

(2) 二进制加固：通过修改二进制文件中的代码，使其难以被攻击者利用。例如，可以使用 angr 对二进制文件进行静态分析，找到其中存在的漏洞，并进行相应的修复。

(3) 漏洞利用：通过对目标程序进行动态分析，找到程序中存在的漏洞，并构造特殊的输入，从而实现攻击。例如，可以使用 angr 对目标程序进行动态分析，找到某个关键点并修改其行为，从而实现攻击。

心得体会：

本次实验的主要内容是基于 angr 工具实现对二进制程序的动态分析。具体而言，我复现了 sym-write 示例的两种 angr 求解方法。

在实验过程中，我学习到了如何使用 angr 进行符号执行，如何将符号执行与约束求解结合起来进行二进制程序分析。在实际应用中，我们还探讨了如何应对 angr 工具可能遇到的问题，例如路径爆炸问题、符号执行不支持的指令以及路径合并等问题。

通过本次实验，我深刻认识到 angr 作为一款开源工具，可以帮助我们轻松地进行二进制程序分析，尤其在 CTF 比赛等安全领域中，angr 工具是非常有用的。同时，我也认识到 angr 在使用过程中可能会遇到一些问题，这要求我们在使用时更加小心谨慎，避免出现不必要的错误和漏洞。

总之，这次实验让我对 angr 工具有了更深刻的理解，也增加了我在安全领域的实践能力和技能水平。