《软件安全》实验报告

姓名：邢清画 学号：2211999 班级：1023

**实验名称：**

Angr 应用示例

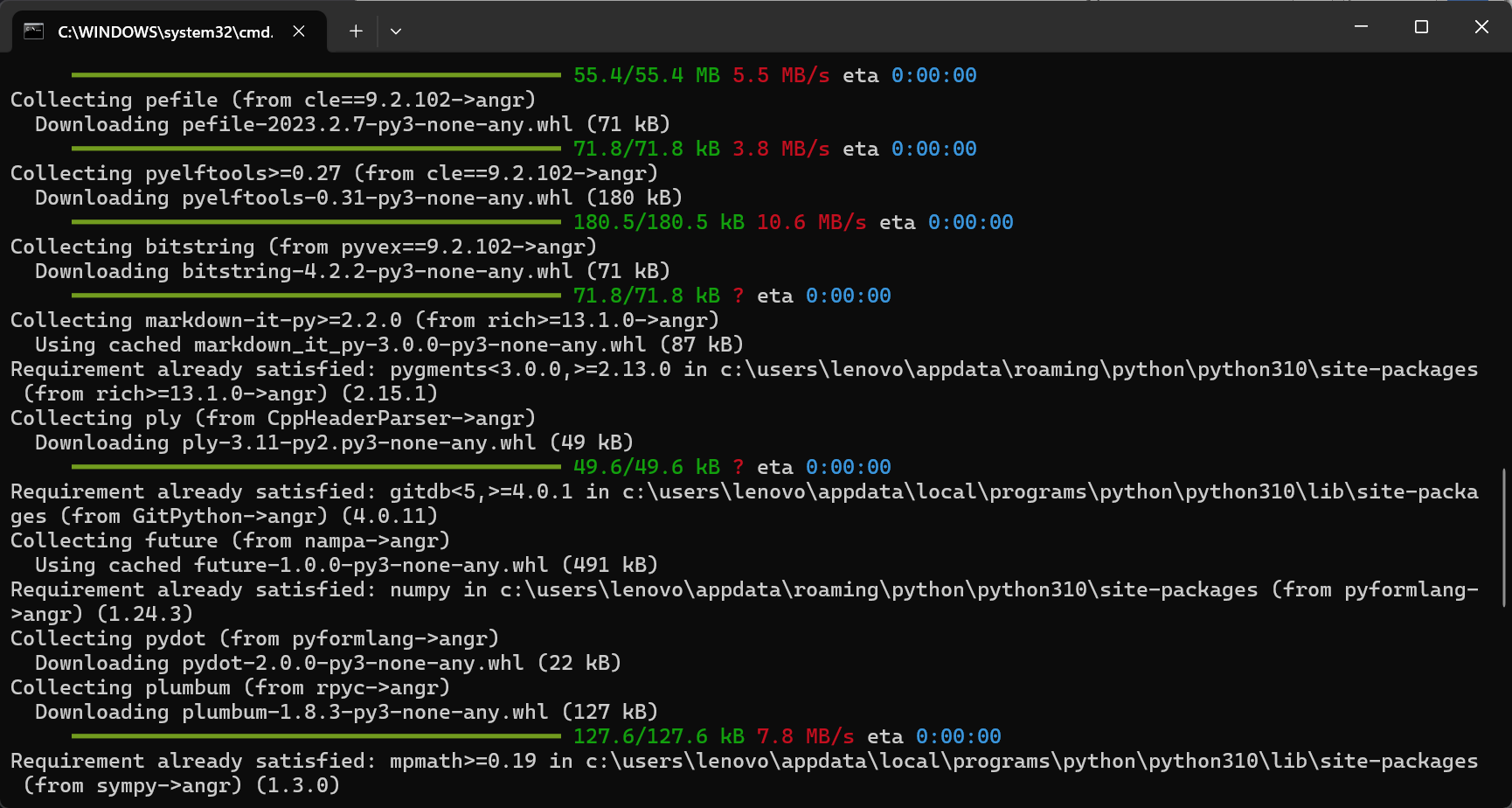
**实验要求：**

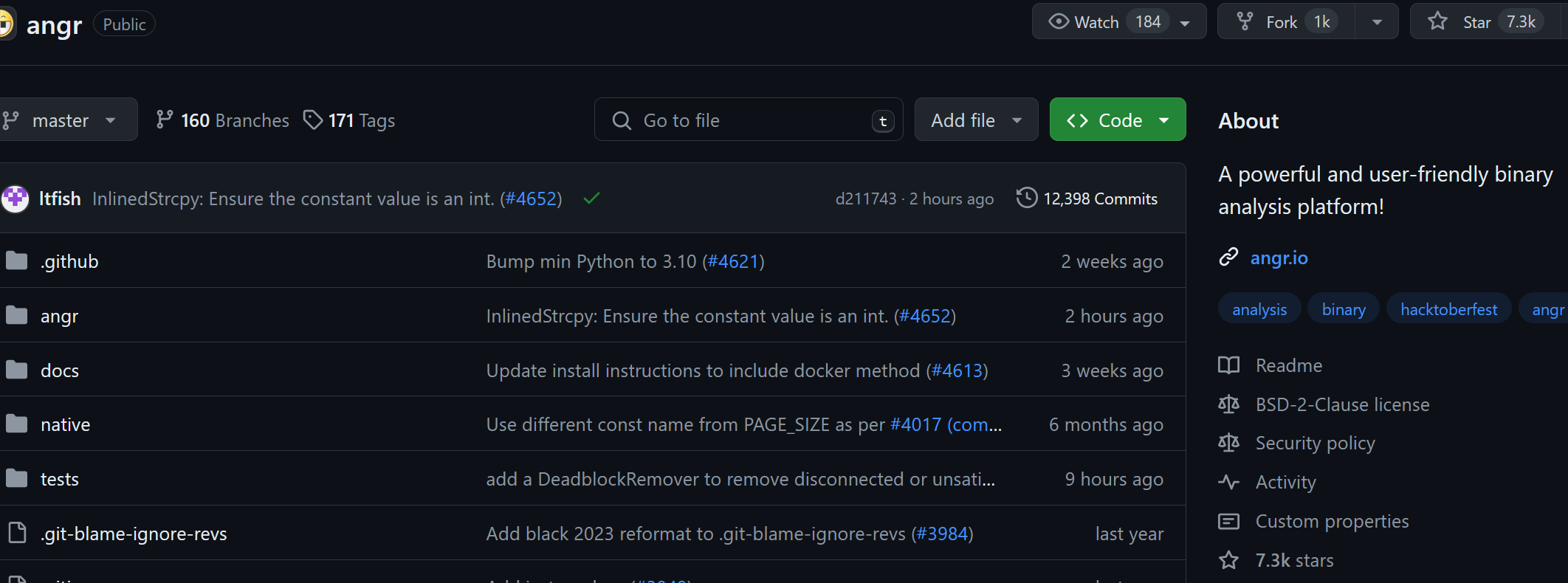
根据课本 8.4.3 章节，复现 sym-write 示例的两种 angr 求解方法，并就如何使用 angr 以及怎么解决一些实际问题做一些探讨。

**实验过程：**

1. 安装Angr

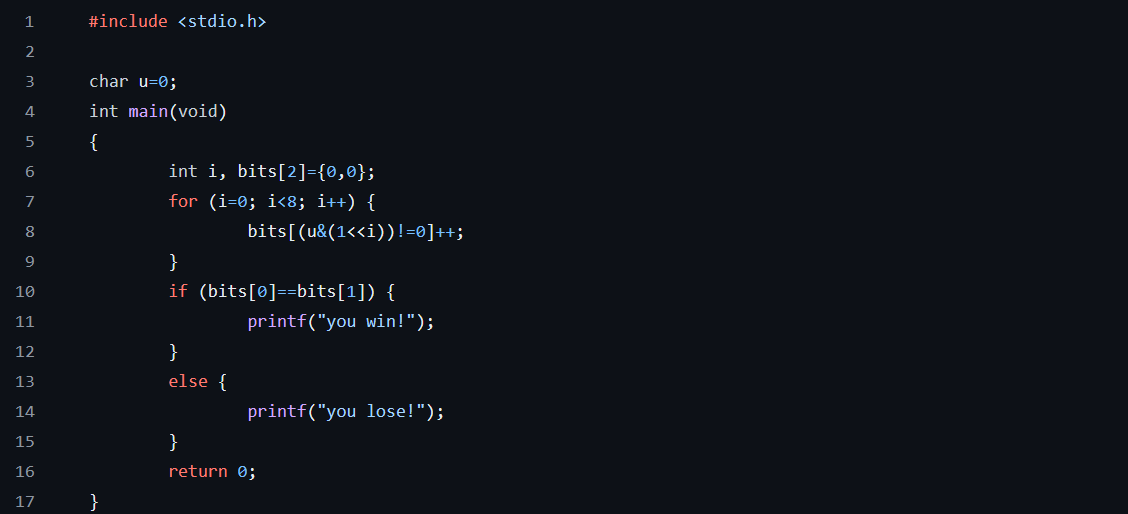
在终端使用命令：pip install angr,在Python环境中安装angr包，并且在GitHub上下载angr官方文档。





**2.angr解法一：solve.py**

在官网下载的angr-doc中有各类的example，在examples>sym-write文件夹中找到issue.c，测试代码如下：



下面是solve.py文件部分代码：  


通过上述angr示例，进行了几个关键步骤：

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() 函数将对象转化为字符串类型。

1. **新建一个 Angr 工程**：

载入二进制文件 './issue'，设置 auto\_load\_libs 为 False，这将不会自动载入依赖的库。默认情况下，auto\_load\_libs 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。

1. **初始化模拟程序状态**：

使用 p.factory.entry\_state(add\_options={angr.options.SYMBOLIC\_

WRITE\_ADDRESSES}) 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。

1. **符号化变量**：

将要求解的变量符号化，注意这里符号化后的变量 u 存在二进制文件的存储区中。使用 claripy.BVS("u", 8) 创建一个 8 位的符号变量 u，并通过 state.memory.store(0x804a021, u) 将其存储在地址 0x804a021。

1. **创建模拟管理器**：

使用 p.factory.simulation\_manager(state) 创建模拟管理器 sm。模拟管理器的作用是对经过模拟执行得到的一系列 states 进行管理。

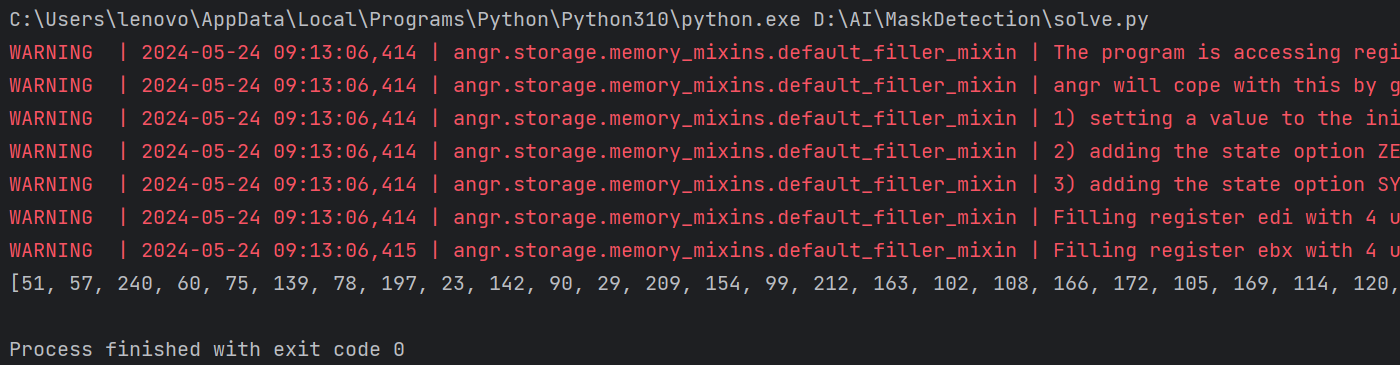
1. **符号执行**：

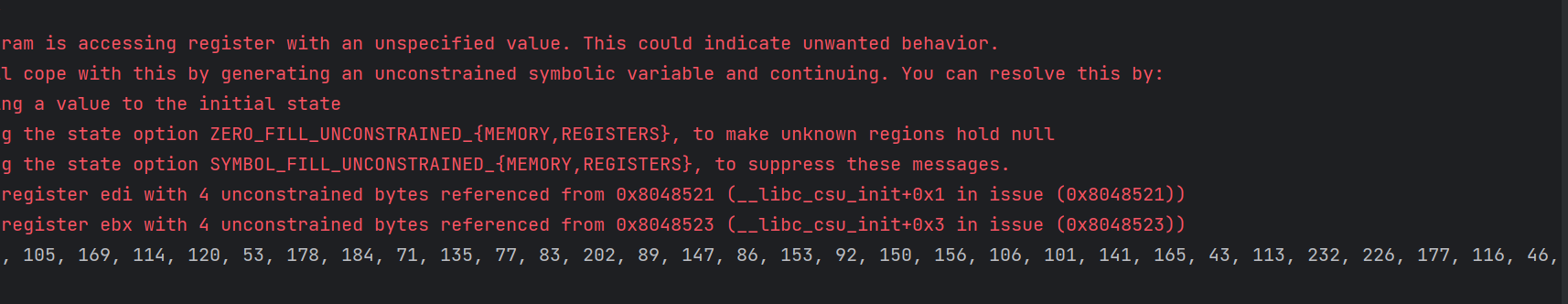
通过 sm.explore(find=correct, avoid=wrong) 进行符号执行以找到符合条件的状态。状态通过两个函数 correct 和 wrong 来定义，通过符号执行得到的输出 state.posix.dumps(1) 中是否包含 win 或 lose 的字符串来完成定义。程序中也提供了另一种定义状态的方法，即通过 find=0x80484e3 和 avoid=0x80484f5 的地址来定义状态，使用 IDA 反汇编可知 0x80484e3 是 printf("you win!") 对应的汇编语句，0x80484f5 则是 printf("you lose!") 对应的汇编语句。

1. **求解变量**：

通过符号执行获得符合条件的 state 之后，使用求解器求解 u 的值。代码中通过 sm.found[0].solver.eval\_upto(u, 256) 求解 u 的所有可能取值。eval\_upto(e, n, cast\_to=None, \*\*kwargs) 方法用于求解一个表达式的多个可能方案，其中 e 是表达式，n 是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。

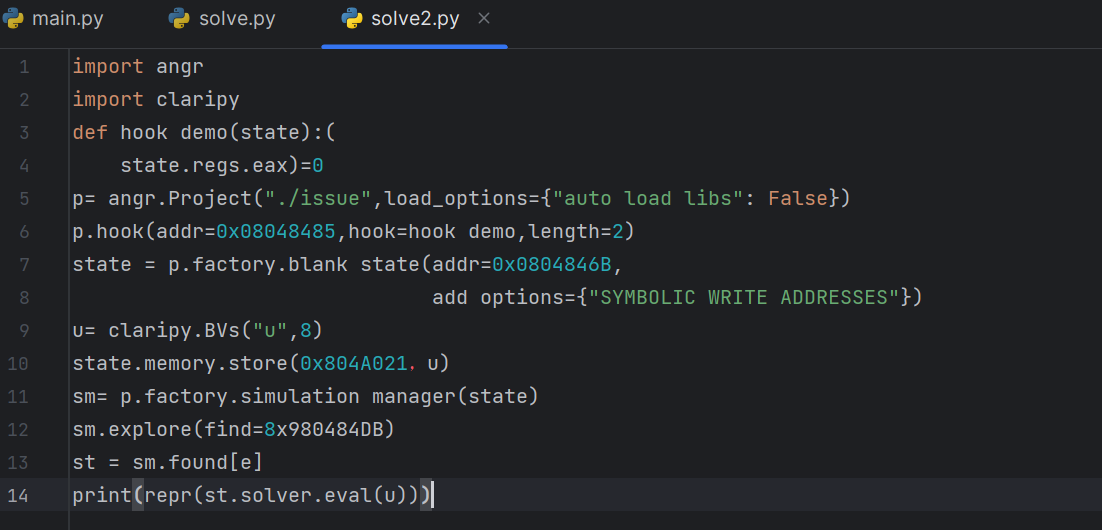
程序运行后，将 solve.py 拖入 IDLE 并运行，即可输出 u 的所有可能取值。该代码通过 Angr 库实现了符号执行，能够在二进制文件中查找特定状态并求解符号化变量的值。

执行上述代码，得到如下结果：  




最下面的列表部分是输出的u的求解的结果，由于采用了evalupto 函数，给出了多个解。

3. **angr解法二：solve2.py**

使用下面的solve2.py代码：  


比较与solve.py的区别：

**（1）采用了 hook 函数**：

在 0x08048485 处的长度为 2 的指令通过自定义的 hook\_demo 进行替代。该 hook 函数将 state.regs.eax 置为 0，功能与原始的 xor eax, eax 指令一致。这仅是一个演示，可以将一些复杂的系统函数调用（例如 printf 等）进行 hook，从而提升符号执行的性能。

**（2）符号执行条件的变化**：

符号执行中寻找的状态条件变更为 find=0x080484DB。由于源程序中的 win 和 lose 是互斥的，因此只需要给定一个 find 条件即可。

**（3）结果求解方法的变化**：

在最终求解符号变量 u 时，使用了 eval(u) 方法替代了原来的 eval\_upto。这意味着程序将只打印一个结果。

最终**只打印一个结果83**

（此处根据实际操作过程，留下具体操作步骤、附加一些自己的理解，即可）

**心得体会：**

通过此次实验，我复现了 sym-write 示例中的两种 angr 求解方法，并深入探讨了如何使用 angr 以及如何解决实际问题。通过 hook 函数提升了符号执行的性能，探索了符号执行的不同状态条件，最终成功求解符号变量，验证了 angr 在二进制分析中的强大功能。

**Angr的使用：**

1. **安装 angr库**： 安装 angr 库非常简单，只需使用 pip 包管理器执行 pip install angr 命令即可。对于某些特定的依赖项，可能需要进行额外的配置。
2. **加载二进制文件**： 使用 angr.Project 类来加载目标二进制文件。通过指定 auto\_load\_libs 选项，可以控制是否自动加载二进制文件所依赖的库文件。
3. **构建模拟管理器**： 创建 SimState 对象表示模拟程序状态，并使用 p.factory.simulation\_manager(state) 来构建模拟管理器 sm，管理模拟执行过程中产生的各种状态。
4. **执行分析**： 使用 sm.explore 方法对程序进行符号执行，指定要查找的目标状态（find）和要避开的状态（avoid）。
5. **提取结果**： 在符号执行完成后，使用求解器 st.solver.eval 提取符号变量的值，以便分析和处理结果。

**Angr 可解决的实际问题：**

1. **漏洞挖掘**： 通过符号执行和约束求解，angr 可以自动化地探索二进制程序的所有可能执行路径，帮助发现潜在的安全漏洞，如缓冲区溢出、格式化字符串漏洞、整数溢出等。这对于提高软件安全性和稳定性非常重要。
2. **逆向工程**： angr 可以用来分析二进制程序的行为，即使没有源代码。这对于理解闭源软件或恶意软件的行为非常有用，帮助安全研究人员或逆向工程师进行深入分析。
3. **自动化测试**： angr 可以生成测试用例，用于覆盖程序的不同执行路径，从而提高软件测试的覆盖率。这种自动化测试方法能够发现程序中的隐藏错误，提高软件质量。
4. **程序理解**： angr 提供了各种分析工具，可以帮助开发者更好地理解程序的执行流程、控制流图、数据依赖等。通过这些工具，开发者可以更高效地进行代码审查和优化。
5. **二进制补丁**： 通过分析，angr 可以帮助开发者找到并理解漏洞的根本原因，从而开发出相应的补丁。对于已知漏洞的二进制程序，angr 可以定位问题并协助修复。
6. **游戏破解**： 尽管这通常不是 angr 的主要用途，但它可以用来分析游戏程序，寻找游戏内的逻辑漏洞或弱点，用于创建作弊工具或游戏修改器。通过符号执行，可以自动化地找到游戏中的漏洞。
7. **安全评估**： 安全研究人员可以使用 angr 对软件进行安全评估，识别和修复潜在的安全问题。通过全面的路径探索和约束求解，angr 可以提供详细的安全评估报告。
8. **竞态条件分析**： angr 可以用来检测多线程程序中的竞态条件，帮助开发者找到并修复可能导致不确定行为的问题。这对于多线程应用的稳定性和可靠性至关重要。
9. **定制分析**： 由于 angr 的可扩展性，开发者可以根据需要编写自定义的分析插件，来解决特定的问题。angr 的插件机制允许开发者扩展其功能，以满足特定的需求。

总的来说，angr 提供了强大的符号执行和约束求解能力，使得漏洞挖掘、逆向工程、自动化测试、程序理解等任务变得更加高效。通过不断的实践和探索，能够更好地掌握 angr 的使用方法，并将其应用到实际问题的解决中。