# Program Analysis Lab Task

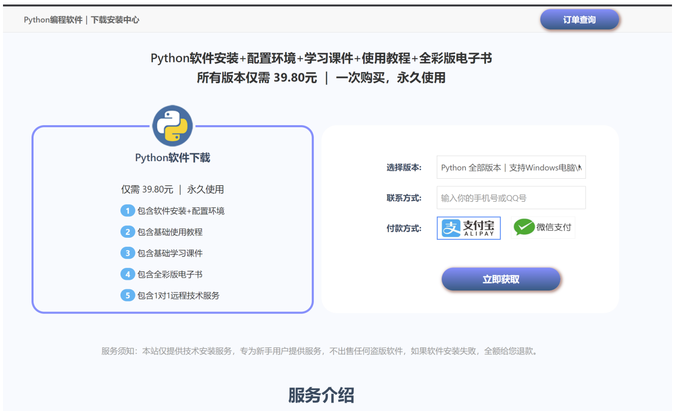
- Symbolic Execution(40pts):
  - Get the CD-KEY for `key_challenge.c` program (20pts)
    - The key is a string of 16 characters
  - Get three vip register keys for `vip_challenge` program (20pts)
    - **Use last 4 digits of your student ID as input sid**
    - The key is a large number
- Fuzzing(60pts):
  - Find a real crash in mcrypt-2.6.5 (60pts)
    - find a crash (50pts)
    - address the crash via gdb (10pts, brief describe where the bug is)
  - Here are some checkpoints if you can't find crash(50pts):
    - setup fuzzer environment (15pts, show the crash found in previous lab by AFL++)
    - pick any task in https://github.com/antonio-morales/Fuzzing101 and use fuzzing to find crash
      - setup target program (10pts)
      - find crash (15pts)
      - address and describe the bug (10pts)
- Bonus: find a bug in a real program by fuzz or symbolic execution and report it, the points depend on the bug you found. If your finding is assigned with a CVE number or accepted by program maintainers, you will get more extra points.
- Bonus: solve the challenge in Task2 reference repository (this may be harder than finding a real CVE)

# Symbolic Execution task instruction

Task 1: Get the CD-KEY for `key_challenge.c` program (20pts)

某一天，你正在自学AI的舍友问你安装Python时候买的激活码是不是还留着，热心的你凑近一看发现舍友百度搜索`Python安装`后了解到需要支付39.9￥才可以收到一个激活码(CD-KEY)用来安装Python，作为程序分析大师，你决定想办法替舍友省下这39.9￥。

我们已经通过逆向(IDA hex-rays插件)获取了程序源代码(`key_challenge.c`)，请找到正确的CD-KEY。



Python软件安装+配置环境+学习课件+使用教程+全彩版电子书
所有版本仅需 39.80元 ｜ 一次购买，永久使用

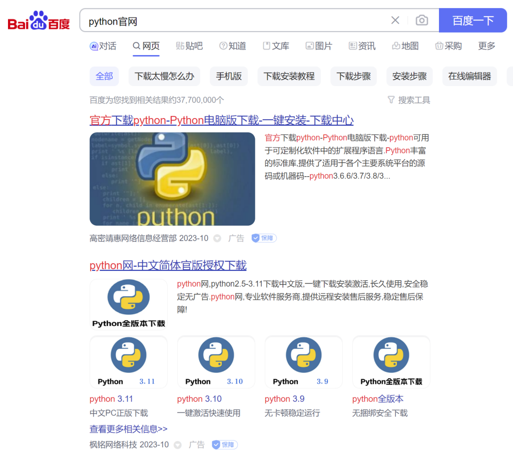
Python软件下载

仅需 39.80元 ｜ 永久使用
1 包含软件安装+配置环境
2 包含基础使用教程
3 包含基础学习课件
4 包含全彩版电子书
5 包含1对1远程技术服务

选择版本: Python 全部版本 | 支持Windows电脑\
联系方式: 输入你的手机号或QQ号
付款方式: 支付宝 ALIPAY　微信支付

服务须知：本站仅提供技术安装服务，专为新手用户提供服务，不出售任何盗版软件，如果软件安装失败，全额给您退款。

服务介绍



Bai百度
python官网
百度一下
对话 网页 贴吧 知道 文库 图片 资讯 地图 采购 更多
全部 下载太慢怎么办 手机版 下载安装教程 下载步骤 安装步骤 在线编辑器

百度为您找到相关结果约37,700,000个　搜索工具

官方下载python-Python电脑版下载-一键安装-下载中心

官方下载python-Python电脑版下载-python可用于可定制化软件中的扩展程序语言.Python丰富的标准库,提供了适用于各个主要系统平台的源码或机器码--python3.6.6/3.7/3.8/3...

高密靖惠网络信息经营部 2023-10　广告　保障

python网-中文简体官版授权下载

Python全版本下载

python网,python2.5-3.11下载中文版,一键下载安装激活,长久使用,安全稳定无广告.python网,专业软件服务商,提供远程安装售后服务,稳定售后保障!

Python 3.11　Python 3.10　Python 3.9　Python全版本下载

python 3.11 中文PC正版下载
python 3.10 一键激活快速使用
python 3.9 无卡顿稳定运行
python全版本 无捆绑安全下载

查看更多相关信息>>

枫铭网络科技 2023-10　广告　保障

Task 2: Get three vip register keys for `vip_challenge` program (20pts)

帮舍友省下一顿KFC疯狂星期四的几天后，你的舍友开始抱怨TA新装的IDE需要付费才能启用全部功能，你凑近一看发现IDE十分眼熟，上面模糊不清地写着C5C-IDE的字样，你决定想办法搞到vip激活码，看看这个“最先进的”IDE究竟有什么会员功能。

这次反汇编的代码非常冗长，更适合直接使用二进制作为符号执行的输入。
Note：本题使用 https://github.com/dtcxzyw/fsubfuscator#ctf-challenge 制作，如果您解出了该仓库内的挑战，您可以汇报并申请bonus。
Credict：感谢 @https://github.com/dtcxzyw 提供支持。

## Setup angr/klee/Triton environment

angr is easy to setup in any system with python3: https://github.com/angr/angr
angr is already installed in our VM, documentation can be found in https://docs.angr.io/

klee need llvm to compile, or use docker to setup klee: https://github.com/klee/klee
Our lab task can be done via online klee demo**: http://klee.doc.ic.ac.uk/

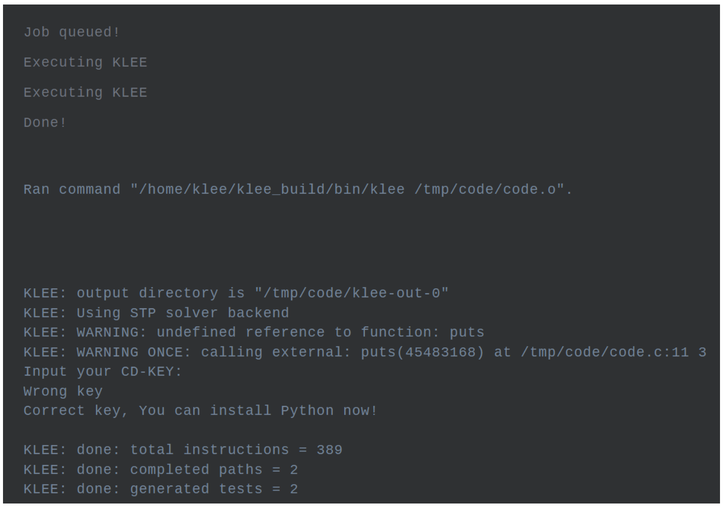
Use Triton to solve symbolic execution task: https://github.com/JonathanSalwan/Triton
document: https://triton-library.github.io/

## Analyse program with source code

This demo using online klee: http://klee.doc.ic.ac.uk/

click left demo files to load demo program, then click `Run KLEE` to start symbolic execution.

klee should start quickly and find key in 3 seconds.



```
Job queued!

Executing KLEE

Executing KLEE

Done!


Ran command "/home/klee/klee_build/bin/klee /tmp/code/code.o".




KLEE: output directory is "/tmp/code/klee-out-0"
KLEE: Using STP solver backend
KLEE: WARNING: undefined reference to function: puts
KLEE: WARNING ONCE: calling external: puts(45483168) at /tmp/code/code.c:11 3
Input your CD-KEY:
Wrong key
Correct key, You can install Python now!

KLEE: done: total instructions = 389
KLEE: done: completed paths = 2
KLEE: done: generated tests = 2
```

## Analyse program without source code

Use angr to solve `key_challenge.c` program:

```python
import angr
import claripy
from angr import SimFileStream

cd_key = claripy.BVS('cd_key', 8 * 16)                 # we know key is 16 bytes

proj = angr.Project("a.out")                           # load program
initial_state = proj.factory.entry_state(stdin=SimFileStream(name='stdin',
content=cd_key, has_end=False))  # replace stdin with cd_key
for i in range(16):                                    # assume key is printable
    initial_state.solver.add(cd_key.get_byte(i) >= 0x20)
    initial_state.solver.add(cd_key.get_byte(i) <= 0x7e)

sim = proj.factory.simulation_manager(initial_state)   # create simulation
manager

# explore by default is like a BFS, find is a filter
sim.explore(find=lambda s: b"Correct" in s.posix.dumps(1), avoid=0xdeadbeef)  #
avoid address 0xdeadbeef
print(sim.found[0].posix.dumps(0))                      # show stdout
print(sim.found[0].solver.eval(cd_key, cast_to=bytes))  # show key found
# print(sim.found[0].solver.constraints)                # show constraints
```

more about angr: https://docs.angr.io/

now try modify the code to solve `vip_challenge` program.

- hint1: angr will conisder each int from scanf("%d") as a bit vector which **length is 11 * 8** ( https://github.com/angr/angr/blob/master/angr/state_plugins/libc.py#L1182 )
  so you may need set first 11 bytes as 0000000xxxx. (xxxx is the last 4 digits of your student ID)
- hint2: use `strings vip_challenge` or drag `vip_challenge` to IDA and press `Shift+F12` to find what will be printed if you input correct key.
- hint3: if program takes 2 integers as input, you can use `claripy.BVS('input', 8 * 2 * 11)` to create a bit vector with length 2 * 11 * 8.
- hint4: angr should take at most 3 minutes to find the key.

# Fuzzing task instruction

mcrypt is a small tool but supports many encryption algorithms. It is widely use in many web applications like `php`. In this lab, we will use AFL++ to find a real crash in mcrypt-2.6.5 (CVE-2012-4409).
**CVE-2012-4409 happens when mcrypt decrypts a file.**

If you find any new crash not on record, please report it to the maintainer.

## Setup AFL++ environment

Note below steps will install shared libs in your Virtual Machine, be careful if you are using your own machine and don't want to install shared libs.

### 1 Compile AFL++:

```
sudo apt update
sudo apt install -y automake cmake ninja-build clang-12 lld-12 llvm-12 llvm-12-dev
sudo apt install -y gcc-$(gcc --version|head -n1|sed 's/.* //'|sed 's/\..*//')-plugin-dev libstdc++-$(gcc --version|head -n1|sed 's/.* //'|sed 's/\..*//')-dev
git clone https://github.com/AFLplusplus/AFLplusplus
export LLVM_CONFIG="llvm-config-12"
cd AFLplusplus
make distrib
sudo make install
cd qemu_mode
CPU_TARGET=i386 ./build_qemu_support.sh
cd ../
sudo make install
```

## 2 Fuzz binary in previous lab

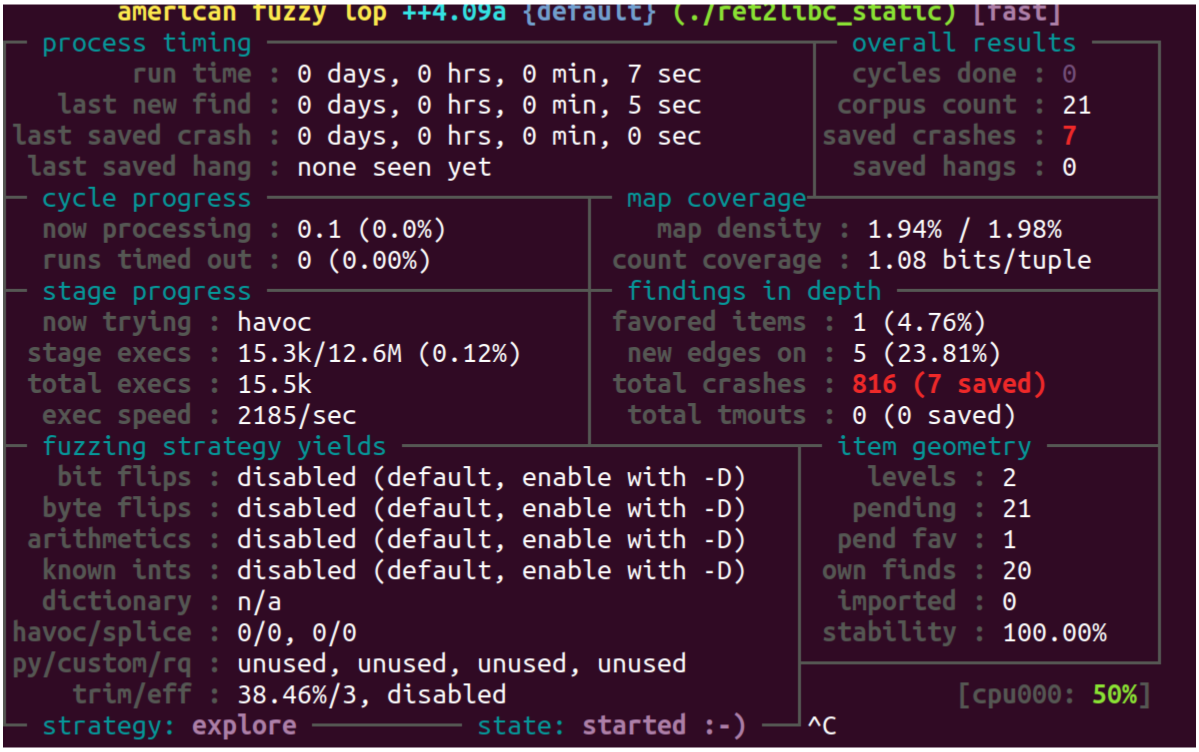
First create a directory for input and output files:

```
sudo su
echo core >/proc/sys/kernel/core_pattern
exit

cd ~/Desktop/lab5
chmod +x ./ret2libc_static
mkdir inputs
mkdir outputs
echo "random input" > inputs/1
afl-fuzz -i inputs/ -o outputs/ -Q ./ret2libc_static
```

AFL may find crash quickly.

The crash can be found in `outputs/default/crashes/`



```
       american fuzzy lop ++4.09a {default} (./ret2libc_static) [fast]
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 0 min, 7 sec       │  cycles done : 0     │
│   last new find : 0 days, 0 hrs, 0 min, 5 sec       │ corpus count : 21    │
│last saved crash : 0 days, 0 hrs, 0 min, 0 sec       │saved crashes : 7     │
│ last saved hang : none seen yet                     │  saved hangs : 0     │
├─ cycle progress ─────────────────────┬─ map coverage┴──────────────────────┤
│  now processing : 0.1 (0.0%)         │    map density : 1.94% / 1.98%      │
│  runs timed out : 0 (0.00%)          │ count coverage : 1.08 bits/tuple    │
├─ stage progress ─────────────────────┼─ findings in depth ─────────────────┤
│  now trying : havoc                  │ favored items : 1 (4.76%)           │
│ stage execs : 15.3k/12.6M (0.12%)    │  new edges on : 5 (23.81%)          │
│ total execs : 15.5k                  │ total crashes : 816 (7 saved)       │
│  exec speed : 2185/sec               │  total tmouts : 0 (0 saved)         │
├─ fuzzing strategy yields ────────────┴─────────────┬─ item geometry ───────┤
│   bit flips : disabled (default, enable with -D)   │    levels : 2         │
│  byte flips : disabled (default, enable with -D)   │   pending : 21        │
│ arithmetics : disabled (default, enable with -D)   │  pend fav : 1         │
│  known ints : disabled (default, enable with -D)   │ own finds : 20        │
│  dictionary : n/a                                  │  imported : 0         │
│havoc/splice : 0/0, 0/0                             │ stability : 100.00%   │
│py/custom/rq : unused, unused, unused, unused       ├───────────────────────┘
│    trim/eff : 38.46%/3, disabled                   │          [cpu000: 50%]
└─ strategy: explore ────────── state: started :-) ──┘^C
```

## 3 compile mcrypt

```
tar -zxvf mhash-0.9.9.9.tar.gz
tar -zxvf libmcrypt-2.5.8.tar.gz
tar -zxvf mcrypt-2.6.5.tar.gz
# z: uncompress with gzip
# x: extract files from archive
# v: verbose
# f: use archive file

# compile mhash first
cd mhash-0.9.9.9/
./configure
make -j
sudo make install  # remeber type your password
```

```
cd -
sudo ln -s /usr/local/lib/libmhash.so /lib/libmhash.so
sudo ln -s /usr/local/lib/libmhash.so.2 /lib/libmhash.so.2

# then compile libmcrypt
cd libmcrypt-2.5.8/
./configure
make -j
sudo make install
sudo ln -s /usr/local/lib/libmcrypt.so /lib/libmcrypt.so
sudo ln -s /usr/local/lib/libmcrypt.so.4 /lib/libmcrypt.so.4

# at last compile mcrypt
cd mcrypt-2.6.5/
CC=~/Desktop/AFLplusplus/afl-clang-fast ./configure --disable-shared
export AFL_USE_CFISAN=1
export LLVM_CONFIG="llvm-config-12"
# change directory to absolute path of your AFLplusplus
make CC=~/Desktop/AFLplusplus/afl-clang-fast CXX=~/Desktop/AFLplusplus/afl-clang-fast++ LD=~/Desktop/AFLplusplus/afl-clang-fast

./src/mcrypt --version
cp ./src/mcrypt ..
```

Now you should have `mcrypt` in `src/` directory (in `mcrypt-2.6.5/src/`).

Firstly, open a new terminal and run:
`while true; do rm outputs/default/.cur_input.dc;done;`

Then use `afl-fuzz -i inputs/ -o outputs/ -- ./mcrypt arg1 arg2` (change agr1 and arg2 to real argument!!) to fuzz `mcrypt` program, if some arguments represent file, you can use `@@` to represent the file.

If you need to stop and re-start the fuzzing, use the same command line options and switch the input directory with a dash (-):
`afl-fuzz -i - -o outputs/ -- ./mcrypt arg1 arg2`

This time, AFL may not find crash quickly, we can create a more **vaild** input for `mcrypt` to help AFL reach more code and find crash.

```
echo "random input" > raw_text
./mcrypt raw_text  # type some password
cp raw_text.nc inputs/2
```