

通用类

I. 代码实现

1.1 加密算法

1.1.1 【必须】避免使用不安全的哈希算法

- DES和3DES已经不再现代应用程序，应改为使用AES。

1.2 程序日志

1.2.1 【建议】对每个重要行为都记录日志

- 确保重要行为都记录日志，且可靠保存6个月以上。

1.2.2 【建议】禁止将未经验证的用户输入直接记录日志

- 当日志条目包含未经净化的用户输入时会引发记录注入漏洞。恶意用户会插入伪造的日志数据，从而让系统管理员以为是系统行为。

1.2.3 【建议】避免在日志中保存敏感信息

- 不能在日志保存密码（包括明文密码和密文密码）、密钥和其它敏感信息

1.3 系统口令

1.3.1 【必须】禁止使用空口令、弱口令、已泄露口令

1.3.2 【必须】口令强度要求

口令强度须同时满足：

1. 密码长度大于14位
2. 必须包含下列元素：大小写英文字母、数字、特殊字符
3. 不得使用各系统、程序的默认初始密码
4. 不能与最近6次使用过的密码重复
5. 不得与其他外部系统使用相同的密码

1.3.3 【必须】口令存储安全

- 禁止明文存储口令
- 禁止使用弱密码学算法（如DES和3DES）加密存储口令
- 使用不可逆算法和随机salt对口令进行加密存储

1.3.4 【必须】禁止传递明文口令

1.3.5 【必须】禁止在不安全的信道中传输口令

II. 配置&环境

2.1 Python版本选择

2.1.1 【建议】使用Python 3.6+的版本

- 新增的项目应使用 Python 3.6+

为什么要这么做？由于 Python 2 在 [2020 年停止维护](#)，相关组件的漏洞不能得到及时修复与维护

2.2 第三方包安全

2.2.2 【必须】禁止使用不安全的组件

2.3 配置信息

2.3.1 【必须】密钥存储安全

- 在使用对称密码算法时，需要保护好加密密钥。当算法涉及敏感、业务数据时，可通过非对称算法协商加密密钥。其他较为不敏感的数据加密，可以通过变换算法等方式保护密钥。

2.3.2 【必须】禁止硬编码敏感配置

- 禁止在源码中硬编码AK/SK、IP、数据库账密等配置信息
- 应使用配置系统或KMS密钥管理系统。

后台类

I. 代码实现

1.1 输入验证

1.1.1 【必须】按类型进行数据校验

- 所有程序外部输入的参数值，应进行数据校验。校验内容包括但不限于：数据长度、数据范围、数据类型与格式。校验不通过，应拒绝。
- 推荐使用组件：[Cerberus](#)、[jsonschema](#)、[Django-Validators](#)

```
# Cerberus示例
v = Validator({'name': {'type': 'string'}})
v.validate({'name': 'john doe'})

# jsonschema示例
schema = {
    "type" : "object",
    "properties" : {
        "price" : {"type" : "number"},
        "name" : {"type" : "string"},
```

```
    },  
}  
  
validate(instance={"name" : "Eggs", "price" : 34.99}, schema=schema)
```

1.2 SQL操作

1.2.1 【必须】使用参数化查询

- 使用参数化SQL语句，强制区分数据和命令，避免产生SQL注入漏洞。

```
# 错误示例  
import mysql.connector  
  
mydb = mysql.connector.connect(  
    ...  
)  
  
cur = mydb.cursor()  
userid = get_id_from_user()  
# 使用%直接格式化字符串拼接SQL语句  
cur.execute("SELECT `id`, `password` FROM `auth_user` WHERE `id`=%s " % (userid,))  
myresult = cur.fetchall()
```

```
# 安全示例  
import mysql.connector  
  
mydb = mysql.connector.connect(  
    ...  
)  
cur = mydb.cursor()  
userid = get_id_from_user()  
# 将元组以参数的形式传入  
cur.execute("SELECT `id`, `password` FROM `auth_user` WHERE `id`=%s " , (userid,))  
myresult = cur.fetchall()
```

- 推荐使用ORM框架来操作数据库，如：使用SQLAlchemy。

```
# 安装sqlalchemy并初始化数据库连接  
# pip install sqlalchemy  
from sqlalchemy import create_engine  
# 初始化数据库连接，修改为你的数据库用户名和密码  
engine = create_engine('mysql+mysqlconnector://user:password@host:port/DATABASE')
```

```
# 引用数据类型
from sqlalchemy import Column, String, Integer, Float
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()
# 定义 Player 对象:
class Player(Base):
    # 表的名字:
    __tablename__ = 'player'

    # 表的结构:
    player_id = Column(Integer, primary_key=True, autoincrement=True)
    team_id = Column(Integer)
    player_name = Column(String(255))
    height = Column(Float(3, 2))
```

```
# 增删改查
from sqlalchemy.orm import sessionmaker
# 创建 DBSession 类型:
DBSession = sessionmaker(bind=engine)
# 创建 session 对象:
session = DBSession()

# 增:
new_player = Player(team_id=101, player_name="Tom", height=1.98)
session.add(new_player)
# 删:
row = session.query(Player).filter(Player.player_name=="Tom").first()
session.delete(row)
# 改:
row = session.query(Player).filter(Player.player_name=="Tom").first()
row.height = 1.99
# 查:
rows = session.query(Player).filter(Player.height >= 1.88).all()

# 提交即保存到数据库:
session.commit()
# 关闭 session:
session.close()
```

1.2.2 【必须】对参数进行过滤

- 将接受到的外部参数动态拼接成SQL语句时，必须对参数进行安全过滤。

```
def sql_filter(sql, max_length=20):
    dirty_stuff = ["'", '"', '/', '*', '(', ')', '=', '-', '#', ';', '<', '>', '+',
                   '&', '$', '(', ')', '%', '@', ',']
    for stuff in dirty_stuff:
```

```
sql = sql.replace(stuff, "x")
return sql[:max_length]
```

1.3 执行命令

1.3.1 【建议】避免直接调用函数执行系统命令

- 相关功能的实现应避免直接调用系统命令（如`os.system()`、`os.popen()`、`subprocess.call()`等），优先使用其他同类操作进行代替，比如：通过文件系统API进行文件操作而非直接调用操作系统命令
- 如评估无法避免，执行命令应避免拼接外部数据，同时进行执行命令的白名单限制。

1.3.2 【必须】过滤传入命令执行函数的字符

- 程序调用各类函数执行系统命令时，如果涉及的命令由外部传入，过滤传入命令执行函数的字符。

```
import os
import sys
import shlex

domain = sys.argv[1]
# 替换可以用来注入命令的字符为空
badchars = "\n&;|'\"$()``-'"
for char in badchars:
    domain = domain.replace(char, " ")

result = os.system("nslookup " + shlex.quote(domain))
```

1.4 XML读写

1.4.1 【必须】禁用外部实体的方法

- 禁用外部实体的方法，来预防XXE攻击。

```
from lxml import etree

xmlData = etree.parse(xmlSource,etree.XMLParser(resolve_entities=False))
```

1.5 文件操作

1.5.1 【必须】文件类型限制

- 通过白名单对上传或者下载的文件类型、大小进行严格校验。仅允许业务所需文件类型上传，避免上传木马、WebShell等文件。

```
import os

ALLOWED_EXTENSIONS = ['txt', 'jpg', 'png']

def allowed_file(filename):
    if ('.' in filename and
        '..' not in filename and
        os.path.splitext(filename)[1].lower() in ALLOWED_EXTENSIONS):

        return filename
    return None
```

1.5.2 【必须】禁止外部文件存储于可执行目录

- 禁止外部文件存储于WEB容器的可执行目录（appBase）。建议使用[tempfile](#)库处理临时文件和临时目录。

1.5.3 【必须】避免路径穿越

- 保存在本地文件系统时，必须对路径进行合法校验，避免目录穿越漏洞

```
import os

upload_dir = '/tmp/upload/' # 预期的上传目录
file_name = '../../etc/hosts' # 用户传入的文件名
absolute_path = os.path.join(upload_dir, file_name) # /tmp/upload/../../etc/hosts
normalized_path = os.path.normpath(absolute_path) # /etc/hosts
if not normalized_path.startswith(upload_dir): # 检查最终路径是否在预期的上传目录中
    raise IOError()
```

1.5.4 【建议】避免路径拼接

- 文件目录避免外部参数拼接。保存文件目录建议后台写死并对文件名进行校验（字符类型、长度）。

1.5.5 【建议】文件名hash化处理

- 建议文件保存时，将文件名替换为随机字符串。

```
import uuid

def random_filename(filename):
    ext = os.path.splitext(filename)[1]
    new_filename = uuid.uuid4().hex + ext
    return new_filename
```

1.6 网络请求

1.6.1 【必须】限定访问网络资源地址范围

当程序需要从用户指定的URL地址获取网页文本内容、加载指定地址的图片、进行下载等操作时，需要对URL地址进行安全校验：

1. 只允许HTTP或HTTPS协议
2. 解析目标URL，获取其host
3. 解析host，获取host指向的IP地址转换成long型
4. 检查IP地址是否为内网IP

```
# 以RFC定义的专有网络为例，如有自定义私有网段亦应加入禁止访问列表。  
10.0.0.0/8  
172.16.0.0/12  
192.168.0.0/16  
127.0.0.0/8
```

5. 请求URL
6. 如果有跳转，跳转后执行1，否则对URL发起请求

1.7 响应输出

1.7.1 【必须】设置正确的HTTP响应包类型

响应包的HTTP头“Content-Type”必须正确配置响应包的类型，禁止非HTML类型的响应包设置为“text/html”。

1.7.2 【必须】设置安全的HTTP响应头

- X-Content-Type-Options

添加“X-Content-Type-Options”响应头并将其值设置为“nosniff”

- HttpOnly 控制用户登鉴权的Cookie字段 应当设置HttpOnly属性以防止被XSS漏洞/JavaScript操纵泄漏。
- X-Frame-Options

设置X-Frame-Options响应头，并根据需求合理设置其允许范围。该头用于指示浏览器禁止当前页面在frame、iframe、embed等标签中展现。从而避免点击劫持问题。它有三个可选的值: DENY: 浏览器会拒绝当前页面加载任何frame页面; SAMEORIGIN:则frame页面的地址只能为同源域名下的页面 ALLOW-FROM origin:可以定义允许frame加载的页面地址。

1.7.3 【必须】对外输出页面包含第三方数据时须进行编码处理

- 当响应“Content-Type”为“text/html”类型时，需要对响应体进行编码处理

```
# 推荐使用mozilla维护的bleach库来进行过滤  
import bleach
```

```
bleach.clean('an <script>evil()</script> example')  
# u'an &lt;script&gt;evil()&lt;/script&gt; example'
```

1.8 数据输出

1.8.1【必须】敏感数据加密存储

- 敏感数据应使用SHA2、RSA等算法进行加密存储
- 敏感数据应使用独立的存储层，并在访问层开启访问控制
- 包含敏感信息的临时文件或缓存一旦不再需要应立刻删除

1.8.2【必须】敏感信息必须由后台进行脱敏处理

- 敏感信息须再后台进行脱敏后返回，禁止接口返回敏感信息交由前端/客户端进行脱敏处理。

1.8.3【必须】高敏感信息禁止存储、展示

- 口令、密保答案、生理标识等鉴权信息禁止展示
- 非金融类业务，信用卡cvv码及日志禁止存储

1.8.4【必须】个人敏感信息脱敏展示

在满足业务需求的情况下，个人敏感信息需脱敏展示，如：

- 身份证只显示第一位和最后一位字符，如3*****1。
- 移动电话号码隐藏中间6位字符，如134*****48。
- 工作地址/家庭地址最多显示到“区”一级。
- 银行卡号仅显示最后4位字符，如*****8639

1.8.5【必须】隐藏后台地址

- 若程序对外提供了登录后台地址，应使用随机字符串隐藏地址。

```
# 不要采取这种方式  
admin_login_url = "xxxx/login"
```

```
# 安全示例  
admin_login_url = "xxxx/ranD0Str"
```

1.9 权限管理

1.9.1【必须】默认鉴权

- 除非资源完全可对外开放，否则系统默认进行身份认证（使用白名单的方式放开不需要认证的接口或页面）。

1.9.2 【必须】授权遵循最小权限原则

- 程序默认用户应不具备任何操作权限。

1.9.3 【必须】避免越权访问

- 对于非公共操作，应当校验当前访问账号进行操作权限（常见于CMS）和数据权限校验。
 1. 验证当前用户的登录态；
 2. 从可信结构中获取经过校验的当前请求账号的身份信息（如：session），禁止从用户请求参数或Cookie中获取外部传入不可信用户身份直接进行查询；
 3. 校验当前用户是否具备该操作权限；
 4. 校验当前用户是否具备所操作数据的权限；
 5. 校验当前操作是否账户是否预期账户。

1.9.4 【建议】及时清理不需要的权限

- 程序应定期清理非必需用户的权限。

1.10 异常处理

1.10.1 【必须】不向对外错误提示

- 应合理使用try/except/finally处理系统异常，避免出错信息输出到前端。
- 对外环境禁止开启debug模式，或将程序运行日志输出到前端。

1.10.2 【必须】禁止异常抛出敏感信息

1.11 Flask安全

1.11.1 【必须】生产环境关闭调试模式

1.11.2 【建议】遵循Flask安全规范

- 参考Flask文档中的安全注意事项 <https://flask.palletsprojects.com/en/latest/security/>

1.12 Django安全

1.12.1 【必须】生产环境关闭调试模式

1.12.2 【建议】保持Django自带的安全特性开启

- 保持Django自带的安全特性开启 <https://docs.djangoproject.com/en/3.0/topics/security/>
- 在默认配置下，Django自带的安全特性对XSS、CSRF、SQL注入、点击劫持等类型漏洞可以起到较好防护效果。应尽量避免关闭这些安全特性。