实用小笔记

- 操作系统
  - 使用`WScript.Shell`创建隐藏窗口
- Python
  - `Python`中`int`转`bytes`的方法
    - 使用`int.to_bytes()`
    - 示例代码
  - 将`Python`对象转换为`JSON`字符串
    - 格式化`JSON`的方法
  - 从`JSON`字符串加载并格式化
  - Triple DES_Decode - Python
  - `Python`实现进度条
    - 主要参数
    - 更新后的第七届省赛CTF决赛签到爆破脚本
  - `Python`中没有总工作量的进度条
    - 代码示例：
  - `Python`中的多进程
    - 主要功能
      1. 创建进程：
         示例代码：
      2. 进程间通信：
         示例代码：
      3. 共享数据
         示例代码：
      4. 进程池
         示例代码：
      5. 同步机制
         示例代码：
    - `multiprocessing`中的`Event`类
      - 代码示例：
  - `Python`中的`string`模块
    - `ascii_letters` -- `Python3`
    - `string.digits`
      - 示例代码：
  - `Python`中的`random`模块
    - 代码示例：
  - `Python`中的`Crypto.Cipher`模块
- MD5
  - 单MD5绕过
  - 双MD5碰撞绕过
  - MD5碰撞脚本
  - MD5碰撞总结
- RCE
  - 命令执行一
    - 涉及函数
      - 常见函数
      - 操作系统命令(输出函数)
      - 命令执行函数
      - 获取文件内容函数
  - 无参数RCE
- Crypto
- 报错学习

实用小笔记

操作系统

使用`WScript.Shell`创建隐藏窗口

你可以使用`WScript.Shell`对象来执行一个命令，并将其窗口设置为隐藏。以下是一个代码示例：

```
Dim objShell
Set objShell = CreateObject("WScript.Shell")
objShell.Run "your_script.bat", 0, False
Set objShell = Nothing
```

在这个例子中，`your_script.bat`是你想要运行的批处理文件。第二个参数`0`表示窗口的状态为隐藏。第三个参数`False`表示该脚本在运行时不会等待该命令完成。

Python

`Python`中`int`转`bytes`的方法

在`Python`中，可以使用`int.to_bytes()`方法将整数转换为字节对象。这个方法允许你指定字节数和字节序`(endianness)`。以下是如何将整数转换为字节的基本用法。

使用`int.to_bytes()`

`int.to_bytes(length, byteorder)`方法的参数：

- `length`：要生成的字节数。如果你希望将一个整数转换为一个特定字节数的字节对象，你需要知道该整数的范围。
- `byteorder`：字节序，可以是`big`或`little`，分别表示大端序和小端序。

示例代码

```
number = 193
num_bytes = number.to_bytes(1, byteorder='big')
print(f"Integer:{number}")
print(f"Bytes:{num_bytes}")
# 转换多个整数为字节
# 将 RGBA 值转换为字节
rgba_values = [193, 147, 232, 33]
bytes_array = bytes(rgba_values) # 使用 bytes() 转换列表为字节
print(f"RGBA as bytes:{bytes_array}")
```

将Python对象转换为JSON字符串

格式化JSON的方法

1. 使用json.dumps()：将Python对象转换为格式化的JSON字符串。
2. 使用indent参数：指定缩进的空格数，使JSON数据更易读。
3. 设置sort_keys参数：如果设置为True，字典的键将按字母顺序排序。

```python
import json
# 示例 JSON 数据（Python 字典）
data = {
    "name": "Alice",
    "age": 30,
    "city": "New York",
    "is_student": False,
    "courses": ["Math", "Science"],
    "address": {
        "street": "123 Main St",
        "zipcode": "10001"
    }
}
# 将 Python 字典转换为格式化的 JSON 字符串
formatted_json = json.dumps(data, indent=4, sort_keys=True)
# 打印格式化后的 JSON 字符串
print(formatted_json)
```

从JSON字符串加载并格式化

如果你有一个JSON字符串，你可以使用json.loads()加载它，然后再使用json.dumps()格式化：

```python
json_string = '{"name": "Bob", "age": 25, "city": "Los Angeles"}'
# 从 JSON 字符串加载
data = json.loads(json_string)
# 格式化 JSON 数据
formatted_json = json.dumps(data, indent=4)
# 打印格式化后的 JSON 字符串
print(formatted_json)
```

Triple DES_Decode - Python

```python
from Crypto.Cipher import DES3
from tqdm import trange
cpt = 'ABCD'
c = long_to_bytes(0x570fc2416dad7569c13356820ba67ba628c6a5fcbc73f1c8689612d23c3a779befeacf678f93ff5eb4b58dc09dcb9a89)

for i in range(2^18):
    k = ''
    n = i
    for _ in range(9):
```

```
        k += cpt[n%4]
        n //= 4
    des3 = DES3.new(mode = DES3.MODE_CBC, iv = b'12345678', key =
b'D'+k.encode()+b'000000')
    if b'DASCTF' in des3.decrypt(c):
        des3 = DES3.new(mode = DES3.MODE_CBC, iv = b'12345678', key =
b'D'+k.encode()+b'000000')
        print(des3.decrypt(c))
        break
```

Python实现进度条

在Python中可以使用tqdm库来实现进度条，这个库不仅可以轻松地创建进度条，而且一般情况下它的性能也很高，不会对程序地效率产生显著的影响。
下面是一个简单的示例，模拟爆破(破解密码)过程，并使用tqdm显示进度条

```python
import time
import random
from tqdm import tqdm
def brute_force_password(target_password, charset, max_length):
    attempts = 0
    total_attempts = len(charset) ** max_length ## 计算总的尝试次数
    # 使用 tqdm 创建进度条
    with tqdm(total=total_attempts, desc="Brute forcing", unit="attempt") as
pbar:
        for length in range(1, max_length + 1):
            for attempt in generate_attempts(charset, length):
                attemps += 1
                # 模拟密码检查
                if attempt == target_password:
                    print(f"Password found: {attempt} in {attempts}
attempts")
                    return attempts
                pbar.update(1) # 更新进度条
                time.sleep(0.01) # 添加延迟以模拟工作负荷
    print("Password not found.")
    return None

def generate_attempts(charset, length):
    """ 生成所有可能的尝试（简单示例，未实现完整的组合生成） """
    from itertools import product
    for attempt in product(charset, repeat=length):
        yield ''.join(attempt)
# 示例设置
target_password = "abc"
charset = "abc"
max_length = 3
brute_force_password(target_password, charset, max_length)
```

主要参数

1. iterable:
   类型: 可迭代对象（如列表、元组、范围等）。
   描述: 待迭代的对象，tqdm 将根据这个对象的长度生成进度条。
2. desc:
   类型: 字符串。
   描述: 进度条的前缀描述，显示在进度条的左侧，便于了解当前进度的上下文。
3. total:
   类型: 整数。
   描述: 指定进度条的总数，通常用于不可迭代的情况下（例如，你可能在使用生成器时想要设定一个总数）。
4. leave:
   类型: 布尔值（True 或 False）。
   描述: 是否在循环结束后保留进度条。默认值是 False，如果设置为 True，进度条将在完成后显示在终端中。
5. ncols:
   类型: 整数或 None。
   描述: 指定进度条的宽度，以字符为单位。如果设置为 None，则会自动调整以适应终端宽度。
6. mininterval:
   类型: 浮点数。
   描述: 进度条更新的最小时间间隔（单位：秒）。如果更新频率过高，则可能会减少更新频率，以避免过多的计算开销。
7. maxinterval:
   类型: 浮点数。
   描述: 进度条更新的最大时间间隔（单位：秒）。如果进度条更新过慢，则会增加更新频率。
8. ascii:
   类型: 布尔值或字符串。
   描述: 如果设置为 True，则使用 ASCII 字符来绘制进度条；如果设置为 False，则使用 Unicode 字符。
9. unit:
   类型: 字符串。
   描述: 用于表示每个迭代单位的标签，默认为 iter，可以根据需要更改，例如 file、item 等。
10. unit_scale:
    类型: 布尔值。
    描述: 如果设置为 True，则会自动缩放显示的数量。例如，如果总数是 1000，将显示为 1.0k。
11. bar_format:
    类型: 字符串。
    描述: 自定义进度条的格式，允许用户控制进度条的显示方式。
12. color:
    类型: 字符串。
    描述: 设置进度条的颜色，通常用于终端支持的颜色名称。

更新后的第七届省赛CTF决赛签到爆破脚本

```python
#!/usr/bin/env python
from Crypto.Cipher import DES3
from tqdm import tqdm
from Crypto.Util.number import long_to_bytes
cpt = 'ABCD'
```

```
c = long_to_bytes(0x570fc2416dad7569c13356820ba67ba628c6a5fcbc73f1c8689612d23c3a779befeacf678f93ff5eb4b58dc09dcb9a89)
# print(f"c: {c}")
with tqdm(total=2**18, desc="Process") as pbar:
    for i in range(2**18):
        # print(f"i: {i}")
        k = ''
        n = i
        for _ in range(9):
            k += cpt[n%4]
            n //= 4
        # print(f"k: {k}, i")
        des3 = DES3.new(mode = DES3.MODE_CBC, iv = b'12345678', key = b'D'+k.encode()+b'000000')
        if b'DASCTF' in des3.decrypt(c):
            # print(f"Get! ==> {des3.decrypt(c)}")
            des3 = DES3.new(mode = DES3.MODE_CBC, iv = b'12345678', key = b'D'+k.encode()+b'000000')
            print(f"Get! ==> {des3.decrypt(c)}")
            # print(f"Get! ==> {des3.decrypt(c)}")
            break
        pbar.update(1)
```

`Python`中没有总工作量的进度条

> 当我们的爆破工作没有特定的工作量，完全随机的情况下，我们可以通过展示当前正在处理的结果来代替进度条。如果我们想在终端中只显示一行，并且每次刷新时展示当前爆破的进度过程，可以使用`ANSI`转义序列来覆盖这一行的内容。具体来说，可以使用`\\r`来返回首行，覆盖当前行的内容。

代码示例：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import multiprocessing
import hashlib
import random
import string
import sys
# from tqdm import tqdm

CHARS = string.ascii_letters + string.digits
def cmp_md5(substr, stop_event, str_len, start=0, size=20):
    global CHARS
    while not stop_event.is_set():
        rnds = ''.join(random.choice(CHARS) for _ in range(size)).encode('utf-8') # TypeError: Strings must be encoded before hashing (添加.encode('utf-8'))
        md5 = hashlib.md5(rnds)
        value = md5.hexdigest()
        # print(f"value:{value}") # 测试代码
        if value[start: start+str_len] == substr:
```

```
            # print(rnds)
            # stop_event.set() # 单 MD5碰撞
            #碰撞 双MD5等于单MD5
            md5 = hashlib.md5(value.encode('utf-8')) # TypeError: Strings must be encoded before hashing (添加.encode('utf-8'))
            if md5.hexdigest()[start: start+str_len] == substr:
                print(f"\rFound! ==> {rnds.decode('utf-8')} => {value} => {md5.hexdigest()}\n")
                stop_event.set()
        if random.random() < 0.01: # 这种不知道总爆破次数的爆破，可以使用显示当前爆破情况来代替进度条，具体的实现方式通过"\r"让光标返回到当前行的开头，从而覆盖之前的内容即可。
            print(f"\rCurrent: {rnds.decode('utf-8')}", end='')

if __name__ == '__main__':
    substr = sys.argv[1].strip()
    # print(f"substr:{substr}") # 测试代码
    start_pos = int(sys.argv[2]) if len(sys.argv) > 1 else 0
    # print(f"start_pos:{start_pos}") # 测试代码
    str_len = len(substr)
    cpus = multiprocessing.cpu_count()
    # print(f"cpus:{cpus}") # 测试代码
    stop_event = multiprocessing.Event()
    # print(f"stop_event:{stop_event}")
    processes = [multiprocessing.Process(target=cmp_md5, args=(substr, stop_event, str_len, start_pos)) for i in range(cpus)]
    for p in processes:
        p.start()
    for p in processes:
        p.join()
```

`Python`中的多进程

在`Python`中使用`multiprocessing`模块进行多进程编程，提供了创建和管理多个进程的能力。它能够让你利用多核处理器的优势，提高程序的并行处理能力。相比于传统的线程模型，`multiprocessing`使用的是进程而非线程，因此能够有效的避免全局解释器锁(GIL)的问题，从而达到真正的并行。

主要功能

1. 创建进程：

`multiprocessing`提供了`Process`类，允许你创建新的进程。每个进程都有自己的内存空间，彼此之间不会干扰。

示例代码：

```
from multiprocessing import Process

def worker():
    p = Process(target=worker)
    p.start()
    p.join() # 等待进程结束
```

2. 进程间通信：

`multiprocessing`提供了多种方式来在进程之间通信，包括`Queue`(队列)、`Pipe`(管道)和共享内存等。

示例代码：

```python
from multiprocessing import Process, Queue
def worker(queue):
    queue.put("Hello from the worker!")

if __name__ == "__main__":
    queue = Queue()
    p = Process(target=worker, args=(queue,))
    p.start()
    print(queue.get()) # 获取进程发送的数据
    p.join()
```

3. 共享数据

可以使用`Value`和`Array`来创建共享的数据对象，这样多个进程可以共享数据。

示例代码：

```python
from multiprocessing import Process, Value

def worker(shared_value):
    shared_value.value += 1

if __name__ == "__main__":
    counter = Value('i', 0) # 创建共享整型变量
    processes = [Process(target=worker, args=(counter,)) for _ in range(10)]
    for p in processes:
        p.start()
    for p in processes:
        p.join()
    print(counter.value)
```

4. 进程池

`Pool`类允许你创建一个进程池，用于管理多个进程。它提供了一个简单的`API`来并行处理任务。

示例代码：

```python
from multiprocessing import Pool
def square(x):
    return x * x

if __name__ == "__main__":
    with Pool(processes=4) as pool: # 创建一个包含 4 个进程的进程池
        results = pool.map(square, range(10))
    print(results) # 输出每个数字的平方
```

5. 同步机制

`multiprocessing`提供了多种同步机制，如`Lock`、`Event`、`Semaphore`和`Condition`，以帮助管理进程之间的同步。

示例代码：

```python
from multiprocessing import Process, Lock
def worker(lock):
    with lock:
        print("Lock acquired by worker")
if __name__ == "__main__":
    lock = Lock()
    processes = [Process(target=worker, args=(lock,)) for _ in range(5)]
    for p in processes:
        p.start()
    for p in processes:
        p.join()
```

`multiprocessing`中的`Event`类

`multiprocessing.Event`是`Python`的`multiprocessing`模块中的一个类，用于在多个进程之间进行简单的信号传递和同步。`Event`对象可以用于多个进程之间的通信，以指示某些条件已经发生或标志某种状态。它为进程之间提供了一种协调机制。

主要功能：

1. 设置状态：`set()`方法用于将事件的状态设置为"已设置"，这通常表示某个条件已经满足。
2. 清除状态：`clear()`方法用于将事件的状态设置为"未设置"，这表示条件不再满足。
3. 检查状态：`is_set()`方法用于检查事件的当前状态，返回`True`表示事件已设置，返回`False`表示事件未设置。
4. 等待事件：`wait(timeout=None)`方法用于阻塞当前进程，直到事件被设置。如果在指定的超时(可选)时间内事件未被设置，`wait()`会返回`False`；如果事件被设置，则返回`True`。

代码示例：

```python
from multiprocessing import Process, Event
import time
def worker(event):
    print("Worker: 等待事件...")
    event.wait()  # 等待事件被设置
    print("Worker: 事件已设置，开始工作！")

if __name__ == "__main__":
    event = Event()  # 创建一个事件对象
    p = Process(target=worker, args=(event,))
    p.start()  # 启动进程
    time.sleep(3)  # 主进程等待3秒
    print("主进程: 设置事件！")
    event.set()  # 设置事件，通知工作进程可以继续
    p.join()  # 等待工作进程结束
    print("主进程: 工作进程已结束。")
```

`Python`中的`string`模块

`ascii_letters` -- `Python3`

- 定义：`string.ascii_letters`是一个字符串，包含所有`ASCII`字母(大写和小写)。
- 内容：这个常量的内容如下

```
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

- 用途：
  `ascii_letters`常用于需要处理字母字符的情况，比如生成随机字符串、验证用户名、密码等。

`string.digits`

- 定义：`string.digits`是一个字符串，包含所有`ASCII`数字字符。
- 内容：这个常量的内容为：

```
'0123456789'
```

- 用途：
  `digits`常用于需要处理数字字符的情况，比如生成随机数字、验证输入的数字等。

示例代码：

```python
import string
import random
# 生成一个包含字母和数字的随机字符串
def generate_random_string(length):
    characters = string.ascii_letters + string.digits # 包含字母和数字
    return ''.join(random.choice(characters) for _ in range(length))
# 生成一个长度为 10 的随机字符串
random_string = generate_random_string(10)
print(random_string)
```

`Python`中的`random`模块

在`Python`的`random`模块中，`choice`方法用于从一个非空序列(如列表、元组或字符串)中随机选择一个元素。它提供了一种便捷的方式来获取随机数据。

代码示例：

```python
import random
fruits = ['apple', 'banana', 'cherry', 'date']
letters = 'abcdefg'
numbers = (1, 2, 3, 4, 5)
random_fruit = random.choice(fruits)
random_letter = random.choice(letters)
random_number = random.choice(numbers)
print(random_fruit)  # 输出一个随机选择的水果
print(random_letter)  # 输出一个随机选择的字母
print(random_number)  # 输出一个随机选择的数字
```

`Python`中的`Crypto.Cipher`模块

文档地址

MD5

单MD5绕过

`CTF`中`md5`判等可使用`0e`绕过，但是如果是双md5该如何绕过呢？本文将教你如何绕过`md5(md5($_GET['a'])) == md5($_GET['b'])`。

例子

```php
<?php

if (isset($_GET['a']) && isset($_GET['b'])) {
    $a = $_GET['a'];
    $b = $_GET['b'];
    if ($a != $b && md5($a) == md5($b)) {
        echo "flag{XXXXX}";
    } else {
        echo "wrong!";
    }

} else {
    echo 'wrong!';
}
?>
```

上面只要传入参数`a=s1885207154a`，`b=s1836677006a`，即可，为什么呢？看一下这两个字符串的md5值可以返现分别如下：

```
MD5值：
md5("s1885207154a") => 0e509367213418206700842008763514
md5("s1836677006a") => 0e481036490867661113260034900752
```

二者都是`0e`开头，在`php`中`0e`会被当做科学计数法，就算后面有字母，其结果也是`0`，所以上面的if判断结果使`true`，成功绕过！

双MD5碰撞绕过

例子

```php
<?php

if (isset($_GET['a']) && isset($_GET['b'])) {
    $a = $_GET['a'];
    $b = $_GET['b'];
    if ($a != $b && md5($a) == md5(md5($b)) {
        echo "flag{XXXXX}";
    } else {
```

```
        echo "wrong!";
    }

} else {
    echo 'wrong!';
}
?>
```

双面的判断出现了`md5(md5($b)`，有了前面的铺垫，这里我们第一感觉就是找到一个字符串其`MD5`值的`MD5`仍然是`0e`开头的那就好了。开始的时候我不敢相信，那几率得多小啊，但是在昨天做一道`md5`截断碰撞的时候我就来了灵感，何不尝试一下，结果发现原来这种字符串使真的存在，并且碰撞`0e`开头的时候不到一秒钟就能碰撞到。各位观众，下面请看：

```
MD5值：
md5("V5VDSHva7fjyJoJ33IQl") => 0e18bb6e1d5c2e19b63898aeed6b37ea
md5("0e18bb6e1**********") => 0e0a710a092113dd5ec9dd47d4d7b86f
```

原来真的存在`0e`开头的`MD5`值其`md5`结果也是`0e`开头，所以此题答案便出来了。`a=s1885207154a`，`b=V5VDSHva7fjyJoJ33IQl`即可绕过`if`判断。
其实上面的这种双`md5`值`0e`开头的字符串有很多，但是网上似乎很见到，几乎没有，下面发布一些。

0x03 双MD5结果仍为0e开头字符串大全

```
MD5大全：
CbDLytmyGm2xQyaLNhWn
md5(CbDLytmyGm2xQyaLNhWn) => 0ec20b7c66cafbcc7d8e8481f0653d18
md5(md5(CbDLytmyGm2xQyaLNhWn)) => 0e3a5f2a80db371d4610b8f940d296af
77OhQgrBOjrcqftrlaZk
md5(77OhQgrBOjrcqftrlaZk) => 0e689b4f703bdc753be7e27b45cb3625
md5(md5(77OhQgrBOjrcqftrlaZk)) => 0e2756da68ef740fd8f5a5c26cc45064
7r4lGXCH2Ksu2JNT3BYM
md5(7r4lGXCH2Ksu2JNT3BYM) => 0e269ab12da27d79a6626d91f34ae849
md5(md5(7r4lGXCH2Ksu2JNT3BYM)) => 0e48d320b2a97ab295f5c4694759889f
```

MD5碰撞脚本

```
# -*- coding: utf-8 -*-
import multiprocessing
import hashlib
import random
import string
import sys
CHARS = string.letters + string.digits
def cmp_md5(substr, stop_event, str_len,. start=0, size=20):
    global CHARS
    while not stop_event.is_set():
        rnds = ''.join(random.choice(CHARS) for _ in range(size))
        md5 = hashlib.md5(rnds)
        value = md5.hexdigest()
        if value[start: start+str_len] == substr:
            print rnds
            stop_event.set()
            '''
```

```
            #碰撞双md5
            md5 = hashlib.md5(value)
            if md5.hexdigest()[start: start+str_len] == substr:
                print rnds+ "=>" + value+"=>"+ md5.hexdigest()  + "\n"
                stop_event.set()
            '''

if __name__ == '__main__':
    substr = sys.argv[1].strip()
    start_pos = int(sys.argv[2]) if len(sys.argv) > 1 else 0
    str_len = len(substr)
    cpus = multiprocessing.cpu_count()
    stop_event = multiprocessing.Event()
    processes = [multiprocessing.Process(target=cmp_md5, args=(substr,
                                         stop_event, str_len, start_pos))
                 for i in range(cpus)]
    for p in processes:
        p.start()
    for p in processes:
        p.join()
```

上面脚本注释部分是双`MD5`碰撞，取消注释然后注释掉`16`行即可。
使用方法：`python md5Crack.py "你要碰撞的字符串"字符串的起始位置`
例如：`python md5Crack.py "0e" 0`
将产生`MD5`值为`0e`开头的字符串。

MD5碰撞总结

RCE

命令执行一

涉及函数

常见函数

```
isset()函数：用于检测变量是否已设置并且非 NULL。
highlight_file()函数：对文件进行 PHP 语法高亮显示。语法通过使用 HTML 标签进行高亮。
show_source()是 highlight_file() 的别名。
var_dump:该函数用于打印显示，一个变量的内容与结构，以及类型的信息。该函数有一个参数第一个参数（必填）第二个参数（选填参数,N）可以多个参数。
var_export:此函数返回关于传递给该函数的变量的结构信息，它和var_dump() 类似，不同的是其返回的表示是合法的 PHP 代码
eval()函数:用来执行一个字符串表达式，并返回表达式的值。
next() 将内部指针指向数组中的下一个元素
glob() 函数返回匹配指定模式的文件名或目录
array_reverse()：将数组逆序排列
array_rand()：随机返回数组的键名
array_flip()：交换数组的键和值
session_start()：告诉PHP使用session;
session_id()：获取到当前的session_id值；
```

rev():将文件中的每行内容以字符为单位反序输出，即第一个字符最后输出，最后一个字符最先输出，依次类推。
localeconv() 函数返回一包含本地数字及货币格式信息的数组。
current() 函数返回数组中的当前元素（单元）,默认取第一个值，和pos()一样
get_defined_vars
gettext拓展的使用：
_()是一个函数 ()==gettext() 是gettext()的拓展函数，开启text扩展。需要php扩展目录下有php_gettext.dll

?c=print_r(scandir(current(localeconv())));打印出当前目录下文件
show_source(next(array_reverse(scandir(pos(localeconv())))));
pos()是PHP中的内置函数,用于返回内部指针当前指向的数组中元素的值。返回值后,pos()函数不会递增或递减内部指针。

使用print_r()函数，将已经定义好的变量组成的数组进行打印输出。payload:
?c=print_r(get_defined_vars())
见ctfshow-web40*

操作系统命令(输出函数)

```
cat函数 由第一行开始显示内容，并将所有内容输出
tac函数 从最后一行倒序显示内容，并将所有内容输出
nl      类似于cat -n，显示时输出行号
more    根据窗口大小，一页一页的现实文件内容
less    和more类似，但其优点可以往前翻页，而且进行可以搜索字符
head    只显示头几行
tail    只显示最后几行
xxd     二进制显示
rev     逆序
```

命令执行函数

```
system() 输出并返回最后一行shell结果。
exec() 不输出结果，返回最后一行shell结果，所有结果可以保存到一个返回的数组里面。
passthru() 只调用命令，把命令的运行结果原样地直接输出到标准输出设备上。（替换system）
```

获取文件内容函数

```
pos()是current()的别名
pos():返回数组中当前元素的值
scandir()：函数返回一个数组，其中包含指定路径中的文件和目录(获取目录下的文件)
localeconv()：返回一包含本地数字及货币格式信息的数组。其中数组中的第一个为点号(.)
```

无参数RCE

> 2024年浙江省赛出现了，当时做的很慢，很久没碰了

在杨杰的笔记中找到了相关的操作：

```
?c =
print_r(scandir(current(localeconv()));show_source(next(array_reverse(scandir(pos(localeconv())))))
# 查找定义的变量传参执行：
?c = print_r(get_defined_vars())
?c = print_r(next(get_defined_vars()))
?c = eval(array_pop(next(get_defined_vars())));daigua=system('cat flag.php');
```

Crypto

报错学习

`AttributeError: module 'string' has no attribute 'letters'`
你遇到的错误信息表明在 `Python` 的 `string` 模块中没有 `letters` 这个属性。这是因为在 `Python 3` 中，`string.letters` 已被移除。相应的，`string` 模块提供了 `string.ascii_letters` 来表示所有字母字符。

--

比赛WP记录

2023浙江省大学生网络与信息安全决赛-Misc篇