



# Binary Exploitation: basic 64-bit Buffer Overflow

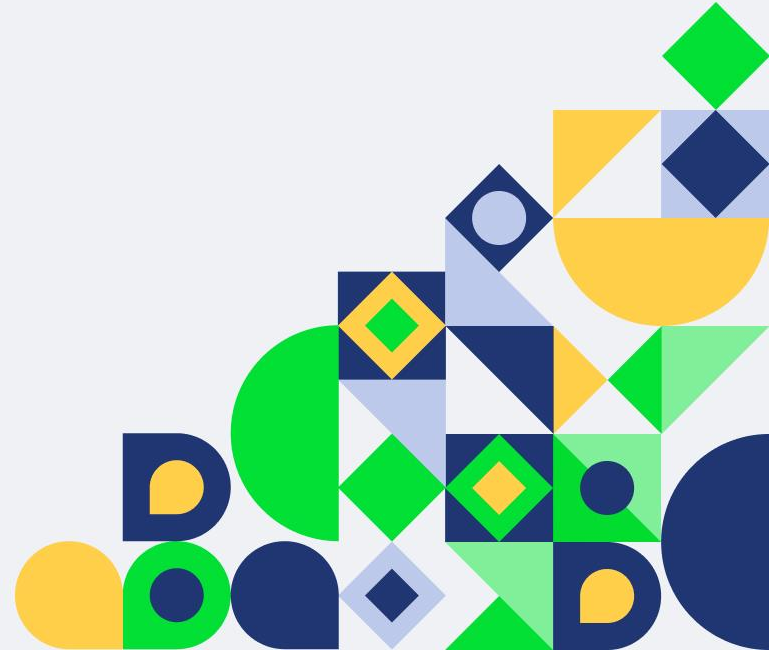




CYBERKARTA



# \$ whoami





# \$ whoami

---



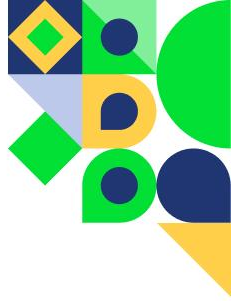
**Moh Rizky Arif** – Undergraduate Student at Universitas Gadjah Mada, majoring in information engineering



[dundorma.live](https://dundorma.live)

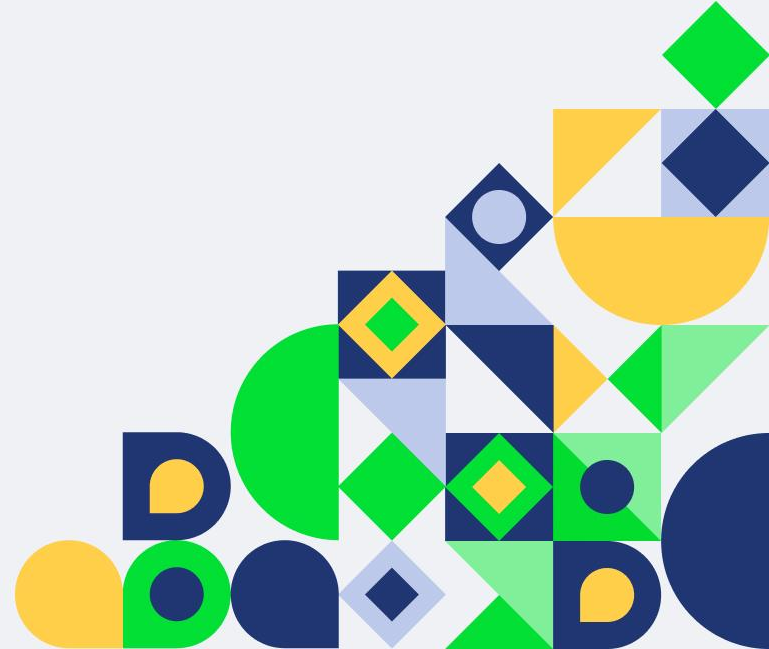


[github.com/dundorma](https://github.com/dundorma)





# Buffer Overflow



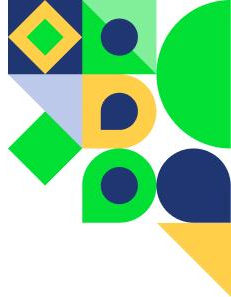


# Buffer Overflow

---

Buffer overflow adalah sebuah kerentanan di mana sebuah program menulis data ke buffer (tempat penyimpanan data sementara di memori) melebihi kapasitas yang dapat ditampung buffer tersebut yang menyebabkan sebuah program melakukan sesuatu yang tidak diinginkan.

Buffer overflow biasanya terjadi di program yang dibuat menggunakan bahasa pemrograman yang memerlukan manajemen memori secara manual seperti C dan C++. Bahasa-bahasa pemrograman ini memungkinkan program untuk memanipulasi memori secara langsung, yang dapat menyebabkan kesalahan dalam menangani buffer.



# Good Mitigation

---

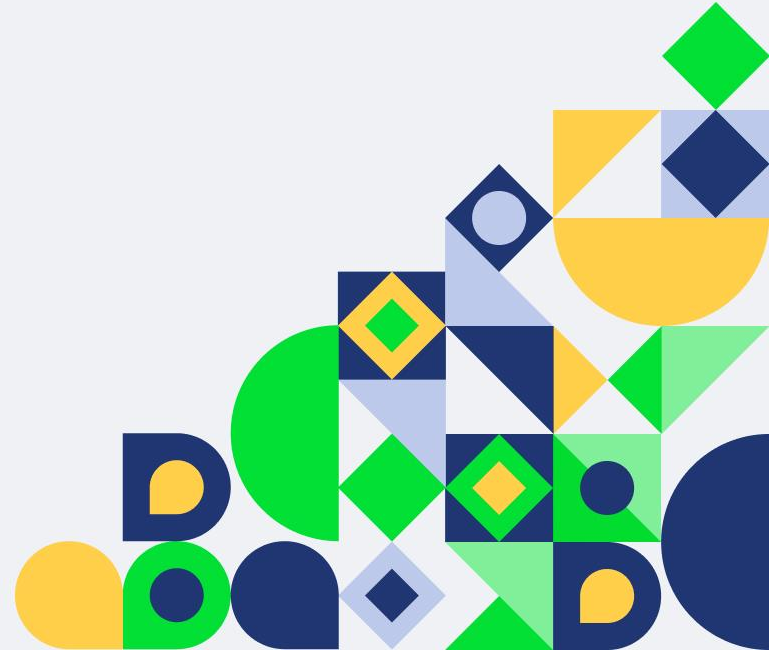
- Shadow Stack
- Control-flow Enforcement Technology (CET)
- Data Execution Prevention (DEP)
- Address space layout randomization (ASLR)
- Stack Canary
- etc



CYBERKARTA



# Demo





# Exploitable binary

---

The binary we'll be using for this session:

[https://github.com/cyberkarta/ctf\\_public/blob/main/BinaryExploit/Cyberkarta\\_Binary%20Exploitation%3A%20basic-64-bit-Buffer-Overflow\\_2\\_DEMO](https://github.com/cyberkarta/ctf_public/blob/main/BinaryExploit/Cyberkarta_Binary%20Exploitation%3A%20basic-64-bit-Buffer-Overflow_2_DEMO)





# Running the binary

```
~/repo/cyberkarta/demo/bufferoverflow
```

```
./cyberkarta_bufferoverflow_demo
```

Masukkan sebuah string:

Cyberkarta

panjang string: 11

```
~/repo/cyberkarta/demo/bufferoverflow
```

```
python -c 'print("A"*1000)' | ./cyberkarta_bufferoverflow_demo
```

Masukkan sebuah string:

panjang string: 200

```
[1] 5608 done python -c 'print("A"*1000)' |  
5609 segmentation fault ./cyberkarta_bufferoverflow_demo
```



# Running the binary

---

Ketika kita memasukkan string yang relatif pendek, program berjalan dengan lancar. Namun ketika kita memasukkan string dengan panjang 1000 karakter, terjadi *segmentation fault*. Segmentation Fault adalah salah satu tanda dari kerentanan buffer overflow. Segmentation fault bisa terjadi karena 'read access violation' atau 'write access violation' yang berarti bahwa kita menuju ke memory address yang tidak dipetakan atau kita tidak memiliki hak untuk 'read' atau 'write' ke memory address tersebut.

**Note:** Perlu diingat bahwa jika terjadi segmentation fault bukan berarti sebuah aplikasi vulnerable terhadap buffer overflow.



# Diving into the debugger

gdb plugin I'll be using:

gef: <https://github.com/hugsy/gef>

Another cool gdb plugins for pwning:

pwndbg: <https://github.com/pwndbg/pwndbg>

peda: <https://github.com/longld/peda>



# Diving into the debugger

---

List all functions: `gef> info functions`

Set breakpoint: `gef> break <address>`

Step instruction (steps through your program's instructions one at a time): `gef> si`

Next Instruction (similar to `si` but skip over function calls): `gef> ni`

Examine the memory at given address: `gef> x /gx <address(es)>`

Disassemble function(machine code to ASM): `gef> disassemble <function name>`



# Diving into the debugger

gef> info functions

```
gef> info functions
All defined functions:

Non-debugging symbols:
0x0000000000401000  _init
0x0000000000401030  puts@plt
0x0000000000401040  printf@plt
0x0000000000401050  read@plt
0x0000000000401060  _start
0x0000000000401090  _dl_relocate_static_pie
0x0000000000401146  overflow
0x0000000000401186  main
0x00000000004011aa  win
0x00000000004011c5  RCE
0x00000000004011d0  _fini
```



# Diving into the debugger

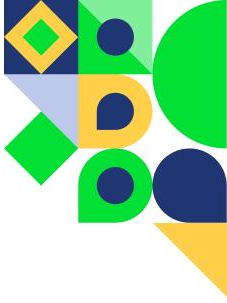
gef> disassemble main

gef> disassemble main

Dump of assembler code for function main:

```
0x0000000000401186 <+0>:    push    rbp
0x0000000000401187 <+1>:    mov     rbp, rsp
0x000000000040118a <+4>:    lea     rax, [rip+0xe8c]
0x0000000000401191 <+11>:   mov     rdi, rax
0x0000000000401194 <+14>:   call    0x401030 <puts@plt>
0x0000000000401199 <+19>:   mov     eax, 0x0
0x000000000040119e <+24>:   call    0x401146 <overflow>
0x00000000004011a3 <+29>:   mov     eax, 0x0
0x00000000004011a8 <+34>:   pop     rbp
0x00000000004011a9 <+35>:   ret
```

End of assembler dump.



# Diving into the debugger

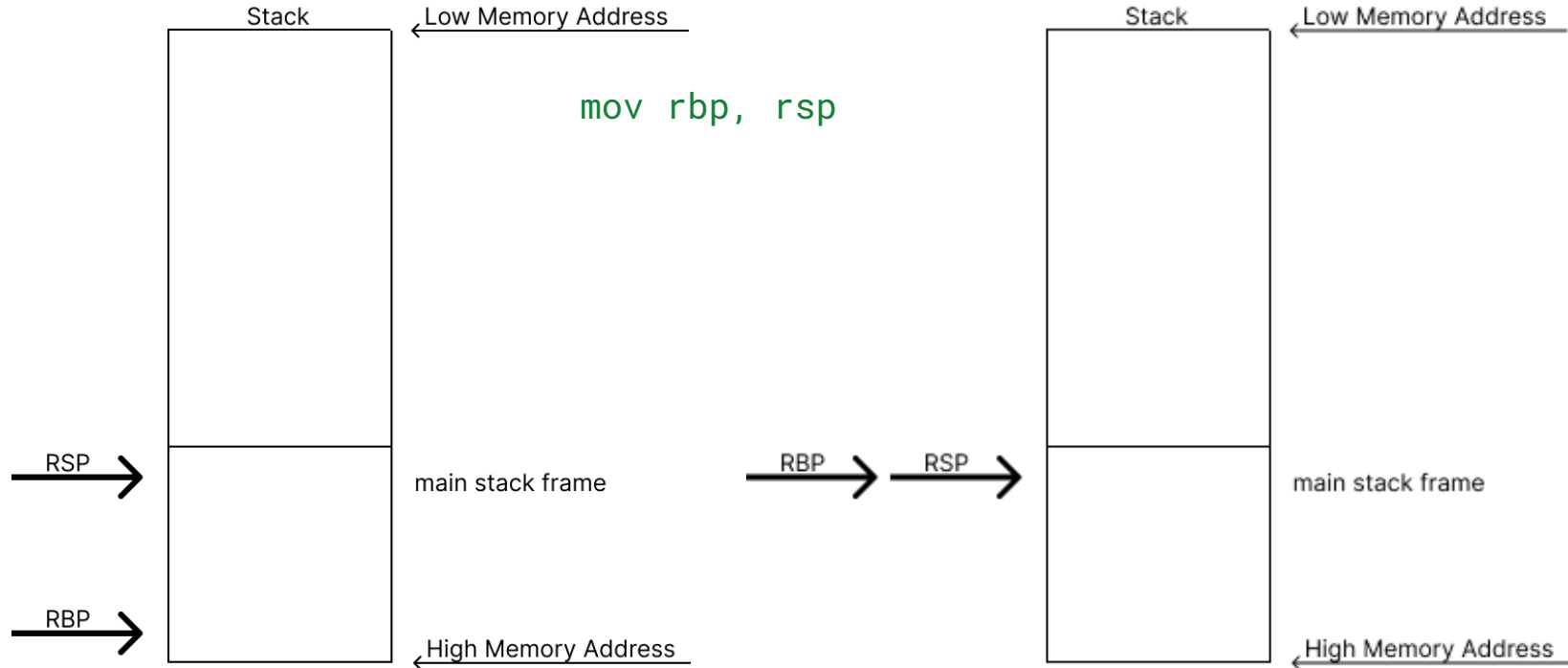
---

Dalam pemrograman bahasa assembly, ***Function Prologue*** adalah beberapa baris kode di awal suatu fungsi, yang mempersiapkan stack dan register untuk digunakan dalam function tersebut.

```
push    rbp
mov     rbp, rsp
```



# Diving into the debugger







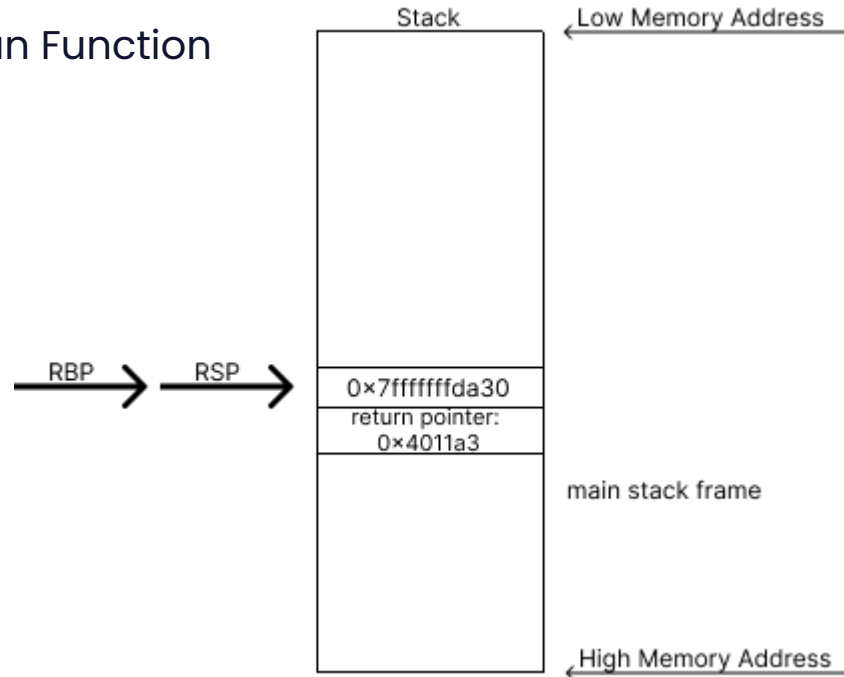
# Diving into the debugger

Call instruction: instruksi `call` melakukan 2 hal, yang pertama melakukan `push` next instruction address ke dalam stack kemudian `jmp` ke fungsi yang akan dipanggil.

```
gef> disassemble main
Dump of assembler code for function main:
0x0000000000401186 <+0>:      push    rbp
0x0000000000401187 <+1>:      mov     rbp, rsp
0x000000000040118a <+4>:      lea     rax, [rip+0xe8c]          # 0x40201d
0x0000000000401191 <+11>:     mov     rdi, rax
0x0000000000401194 <+14>:     call   0x401030 <puts@plt>
0x0000000000401199 <+19>:     mov     eax, 0x0
0x000000000040119e <+24>:     call   0x401146 <overflow>
0x00000000004011a3 <+29>:     mov     eax, 0x0
0x00000000004011a8 <+34>:     pop     rbp
0x00000000004011a9 <+35>:     ret
End of assembler dump.
```

# Diving into the debugger

Setelah instruksi `call <overflow>` dan Function Prologue telah tereksekusi.





# Diving into the debugger

Letak vulnerability buffer overflow dapat dilihat dari instruksi `sub rsp, 0x70` dan `mov edx, 0xc8`. Pada instruksi tersebut buffer diset sebanyak 0x70 bytes atau sama dengan 112 bytes, sedangkan kita maksimum input string diset sebanyak 0xc8 bytes atau sama dengan 200 bytes.

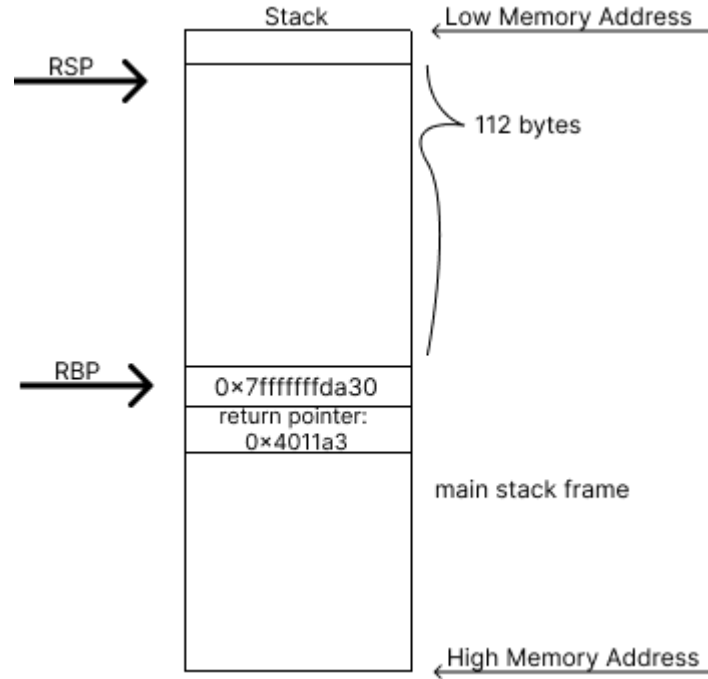
**Note:** fungsi `read@plt` menerima nilai register `edx` sebagai panjang maksimum input

```
gef> disassemble overflow
Dump of assembler code for function overflow:
=> 0x0000000000401146 <+0>:      push    rbp
0x0000000000401147 <+1>:      mov     rbp, rsp
0x000000000040114a <+4>:      sub     rsp, 0x70
0x000000000040114e <+8>:      lea     rax, [rbp-0x70]
0x0000000000401152 <+12>:     mov     edx, 0xc8
0x0000000000401157 <+17>:     mov     rsi, rax
0x000000000040115a <+20>:     mov     edi, 0x0
0x000000000040115f <+25>:     call   0x401050 <read@plt>
0x0000000000401164 <+30>:     mov     QWORD PTR [rbp-0x8], rax
0x0000000000401168 <+34>:     mov     rax, QWORD PTR [rbp-0x8]
0x000000000040116c <+38>:     mov     rsi, rax
0x000000000040116f <+41>:     lea     rax, [rip+0xe92]
0x0000000000401176 <+48>:     mov     rdi, rax
0x0000000000401179 <+51>:     mov     eax, 0x0
0x000000000040117e <+56>:     call   0x401040 <printf@plt>
0x0000000000401183 <+61>:     nop
0x0000000000401184 <+62>:     leave
0x0000000000401185 <+63>:     ret
End of assembler dump.
```



# Diving into the debugger

Set buffer to 112 bytes.





# Diving into the debugger

---

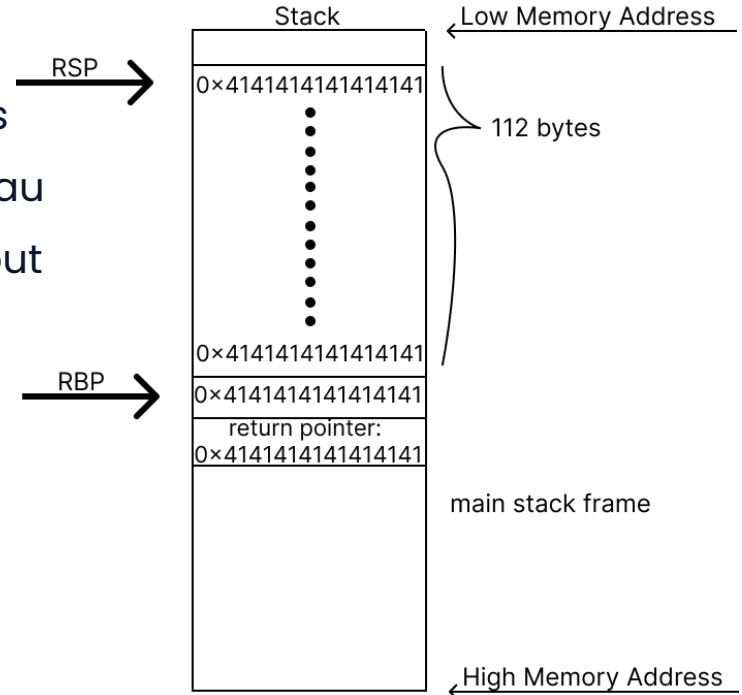
Karena buffer yang diset hanyalah 112 bytes sedangkan kita bisa input 200 bytes karakter, maka kita bisa melakukan overwrite kepada return pointer.





# Diving into the debugger

Apa yang terjadi bila kita input tepat 128 bytes karakter "A". Nah apa kita bisa mengganti 8 bytes terakhir sesuai address dari instruksi yang kita mau untuk nanti akhirnya bisa jump ke instruksi tersebut

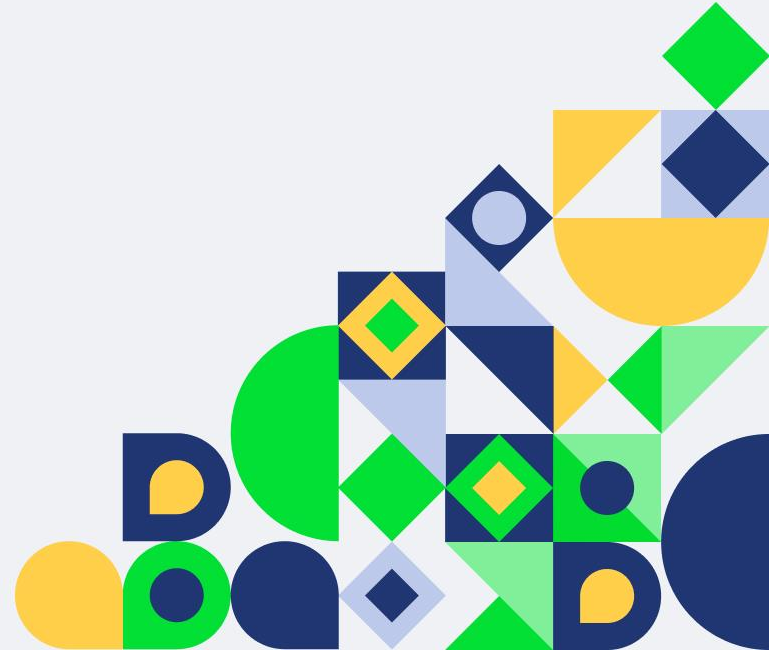




CYBERKARTA



**Try it yourself ! 😁**





# PWN Challenge

```
nc 8.222.221.132 5000
```

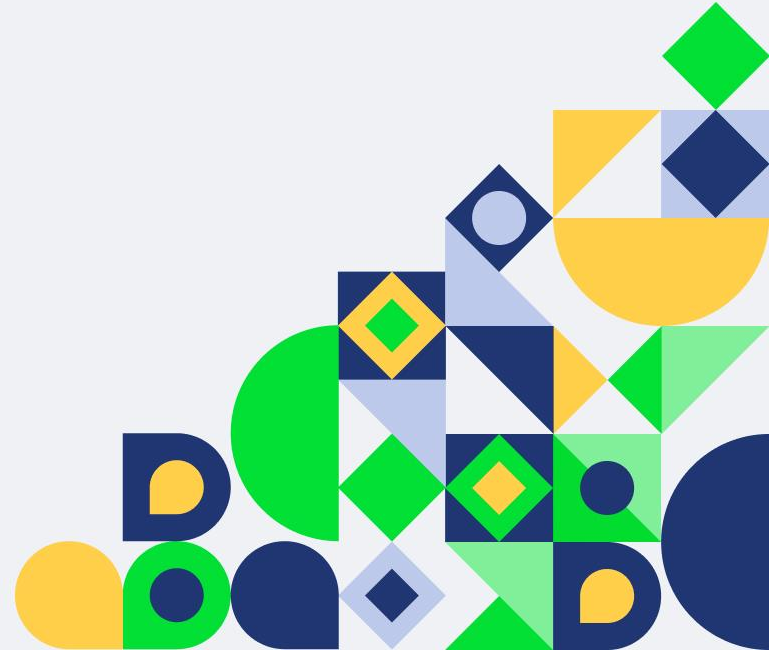
Binary:

[https://github.com/cyberkarta/ctf\\_public/blob/main/BinaryExploit/Cyberkarta\\_Binary%20Exploitation%3A%20basic-64-bit-Buffer-Overflow\\_3\\_CHALLENGE](https://github.com/cyberkarta/ctf_public/blob/main/BinaryExploit/Cyberkarta_Binary%20Exploitation%3A%20basic-64-bit-Buffer-Overflow_3_CHALLENGE)





# Further Reading





# Further Reading

---

Pwn.College: <https://pwn.college>

How2Heap: <https://github.com/shellphish/how2heap>

NightMare: <https://guyinatuxedo.github.io>





CYBERKARTA

# Check Out Our Class

<https://www.cyberkarta.com/>



Premium

★★★★★ (72)

## Dasar Linux Untuk Cyber Security

Total 3 Jam 45 Menit

Beginner



Premium

★★★★★ (104)

## Basic Web Security For Pentester And Bug Bounty...

Total 10 Jam, 21 Menit

All Levels



Premium

★★★★★ (56)

## Zero Cost SIEM Menggunakan Wazuh

Total 4 Jam 18 Menit

Intermediate



CYBERKARTA

**Terima kasih**

<https://www.linkedin.com/in/moh-rizky-arif/>

