Shellcode para desenvolvimento de exploits

https://github.com/helviojunior/Presentations/tree/master/2020/FIAP

Helvio B. D. De Carvalho Junior

Malware and Security Researcher | OSCE | OSCP | eMAPT | CEHv9

- E-mail: helvio_junior@hotmail.com
- Instagram: @helvio_m4v3r1ck
- Mais de 20 anos de atuação com TI
- Foco de estudo e pesquisa:
  - Segurança ofensiva (Red Team)
  - Criação de exploits
  - Bug hunting, Cyber threat hunting
  - Criação e engenharia reversa de Malware
- Especialista em Cyber Security no Banco Original
- https://www.helviojunior.com.br/
- https://github.com/helviojunior
- https://treinamentos.helviojunior.com.br/

Agenda

- PoC Exploit
- Assembly - Passagem de parâmetros para função
- Entendendo uma aplicação em C
- Reproduzindo Shellcode
- Integrando o shellcode no exploit

Hora da maldade

Poc Exploit

Assembly - PUSH

PUSH

Armazena um conteúdo na pilha.

Exemplo: PUSH 0x04030201, armazena o conteúdo 01020304 na pilha. Lembrando que a ordem é inversa em virtude do little-endian.

O Endereçamento do topo da pilha Sempre é armazenado no registrador ESP

Assembly - POP

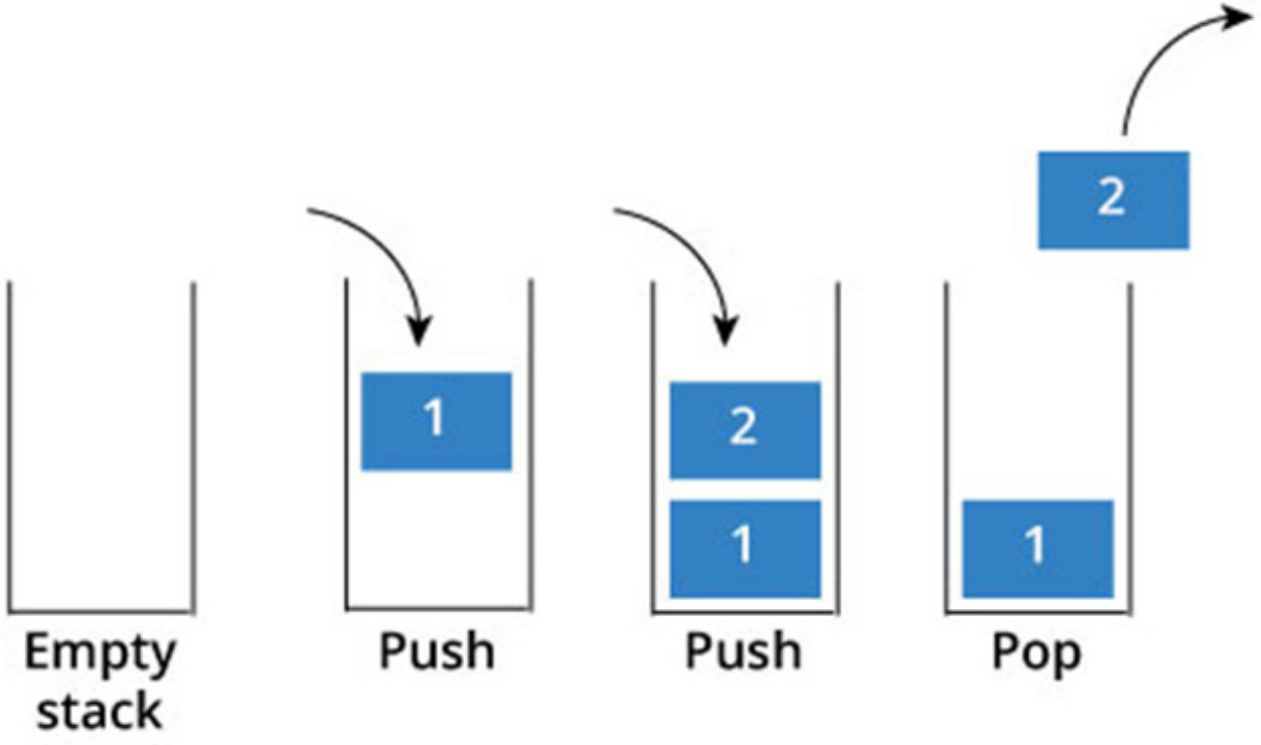
POP

Remove um conteúdo armazenado na pilha o colocando em um registrador

Exemplo: POP EAX, remove o conteúdo da pilha e o armazena no reistrador EAX.

Supondo que antes da execução do POP o endereçamento do topo da pilha seja 0x0000000A (decimal 10) após o POP o topo da pilha é incrementado no mesmo tamanho do dado que foi removido. Neste nosso caso removemos 4 bytes, então o novo top da pilha passará a ser 0x0000000E (decimal 14), ou seja, 4 bytes a mais.

Assembly - PUSH/POP



Fonte da imagem: http://bluegalaxy.info/codewalk/2018/08/12/python-how-to-implement-a-lifo-stack/

PUSH

```
                ...
==> 0x7F200100    push 0x22002200
    0x7F200105    push 0x00110011
    0x7F20010A    pop  eax
    0x7F20010B    pop  edx
    0x7F20010C    ...
                  ...
```

EAX	0x00000000
ECX	0x00000000
EDX	0x00000000
EIP	0x7F200100

0x00803FF0		0x00000000
0x00803FF4		0x00000000
0x00803FF8		0x00000000
0x00803FFC		0x00000000
0x00804000	==>	0x00C0FFEE

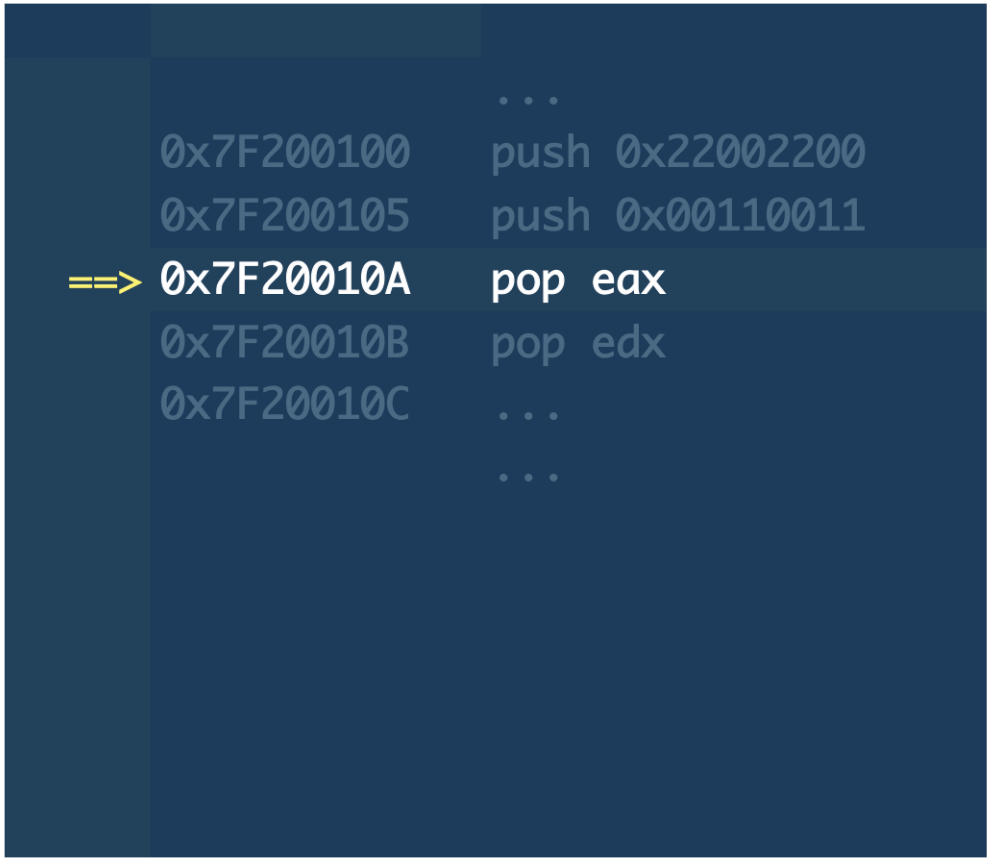
PUSH

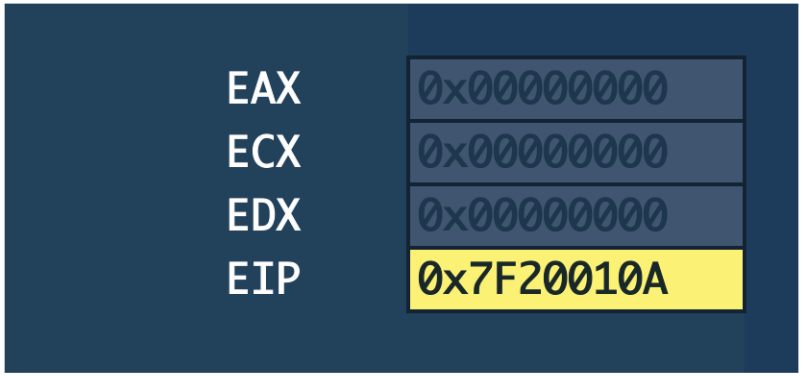
```
            ...
    0x7F200100    push 0x22002200
==> 0x7F200105    push 0x00110011
    0x7F20010A    pop  eax
    0x7F20010B    pop  edx
    0x7F20010C    ...
            ...
```

EAX	0x00000000
ECX	0x00000000
EDX	0x00000000
EIP	0x7F200105

0x00803FF0		0x00000000
0x00803FF4		0x00000000
0x00803FF8		0x00000000
0x00803FFC	==>	0x22002200
0x00804000		0x00C0FFEE

POP


...
0x7F200100 push 0x22002200
0x7F200105 push 0x00110011
==> 0x7F20010A pop eax
0x7F20010B pop edx
0x7F20010C ...
...


EAX 0x00000000
ECX 0x00000000
EDX 0x00000000
EIP 0x7F20010A


0x00803FF0 0x00000000
0x00803FF4 0x00000000
0x00803FF8 ==> 0x00110011
0x00803FFC 0x22002200
0x00804000 0x00C0FFEE

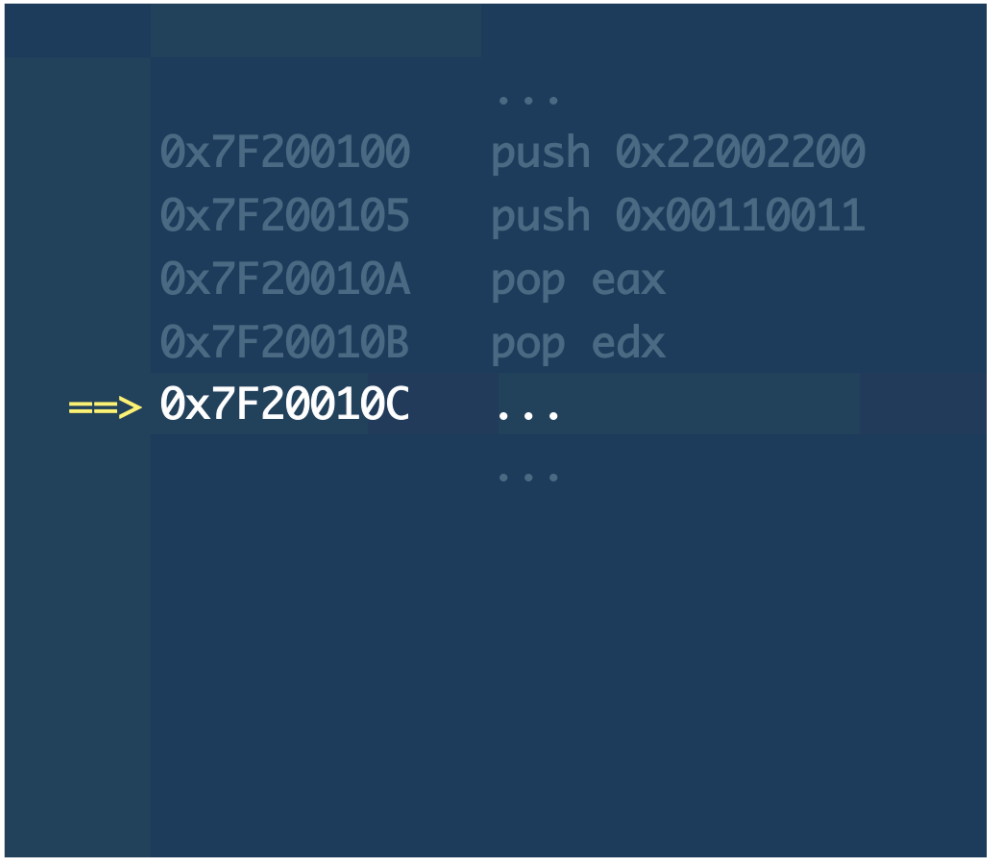
POP

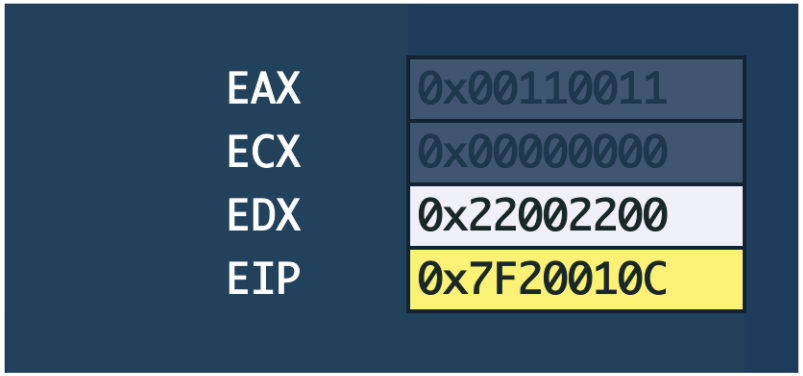
```
                ...
    0x7F200100  push 0x22002200
    0x7F200105  push 0x00110011
    0x7F20010A  pop eax
==> 0x7F20010B  pop edx
    0x7F20010C  ...
                ...
```

EAX	0x00110011
ECX	0x00000000
EDX	0x00000000
EIP	0x7F20010B

0x00803FF0		0x00000000
0x00803FF4		0x00000000
0x00803FF8		0x00110011
0x00803FFC	==>	0x22002200
0x00804000		0x00C0FFEE

POP

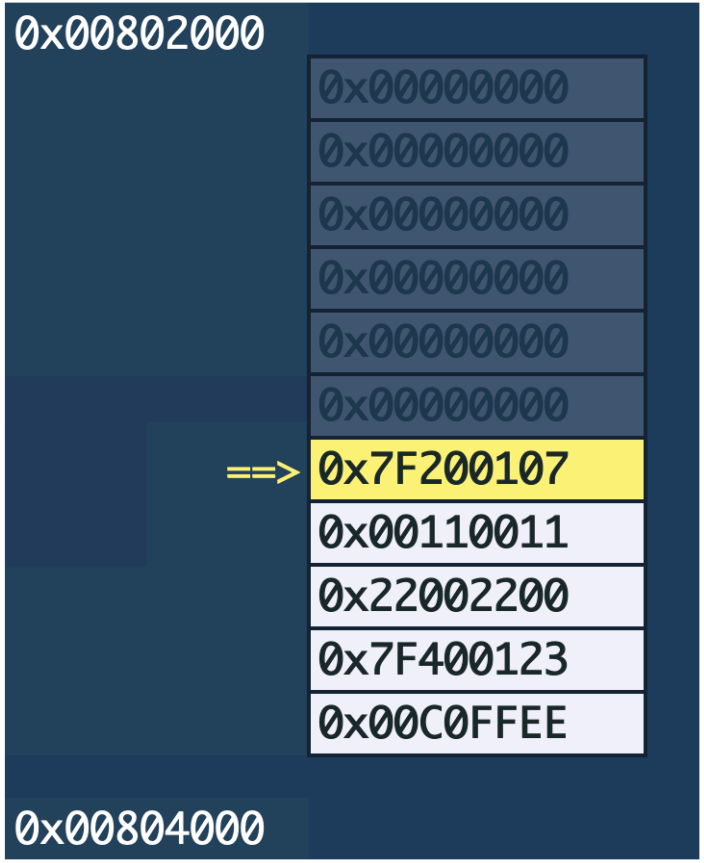

...
0x7F200100 push 0x22002200
0x7F200105 push 0x00110011
0x7F20010A pop eax
0x7F20010B pop edx
==> 0x7F20010C ...
...


EAX 0x00110011
ECX 0x00000000
EDX 0x22002200
EIP 0x7F20010C


0x00803FF0 0x00000000
0x00803FF4 0x00000000
0x00803FF8 0x00110011
0x00803FFC 0x22002200
0x00804000 ==> 0x00C0FFEE

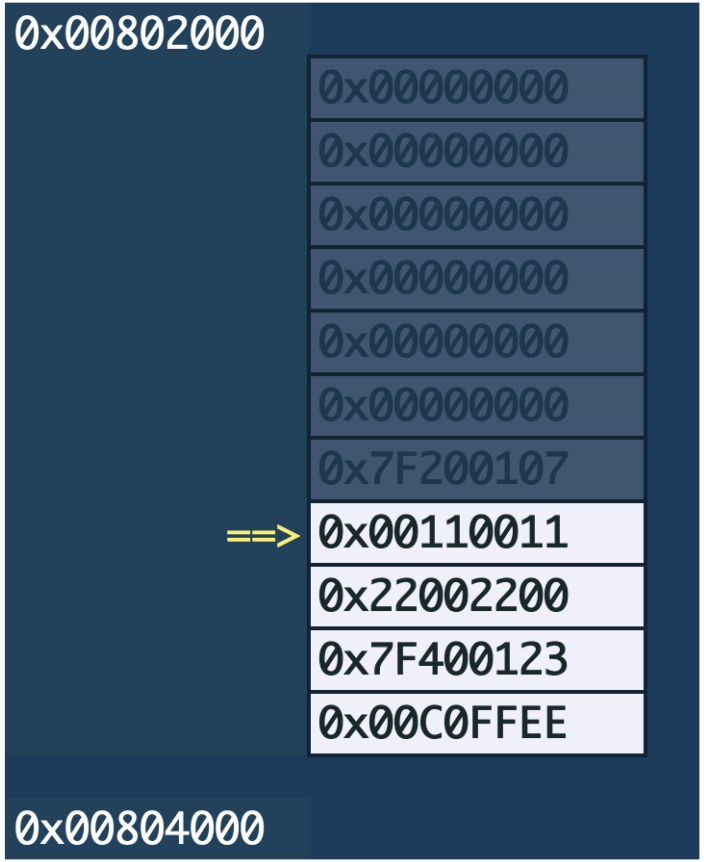
Assembly - CALL

```
    FuncFirst:
                    ...
    0x7F200100      push 0x22002200
    0x7F200101      push 0x00110011
    0x7F200102      call FuncSecond
    0x7F200107      ...

    FuncSecond:
    0x7F204C00      sub esp, 0x8
    0x7F204C03      ...
    0x7F204D19      add esp, 0x8
==> 0x7F204D1C      ret
```



Assembly - CALL

```
    FuncFirst:
                    ...
    0x7F200100      push 0x22002200
    0x7F200101      push 0x00110011
    0x7F200102      call FuncSecond
==> 0x7F200107      ...

    FuncSecond:
    0x7F204C00      sub esp, 0x8
    0x7F204C03      ...
    0x7F204D19      add esp, 0x8
    0x7F204D1C      ret
```



Shellcoding

Passagem de parâmetros

- Arquitetura 32 bits
  - esp+0x00 : Primeiro parâmetro
  - esp+0x04 : Segundo parâmetro
  - esp+0x08 : Terceiro parâmetro
  - esp+0x0C : Quarto parâmetro
  - esp+0x10 : Quinto parâmetro
  - ...
- Resultado da função, quando existir, ocorre em EAX
- Pseudo-código

```
eax = Func1(esp+00, esp+04, esp+08, esp+0C, esp+10, …)
```

Hora da maldade

- Entendendo uma aplicação em C
- Reproduzindo Shellcode
- Integrando o shellcode no exploit

Disassembly - System

system, _wsystem

Executes a command.

Syntax

C

Copy

```
int system(
   const char *command
);
int _wsystem(
   const wchar_t *command
);
```

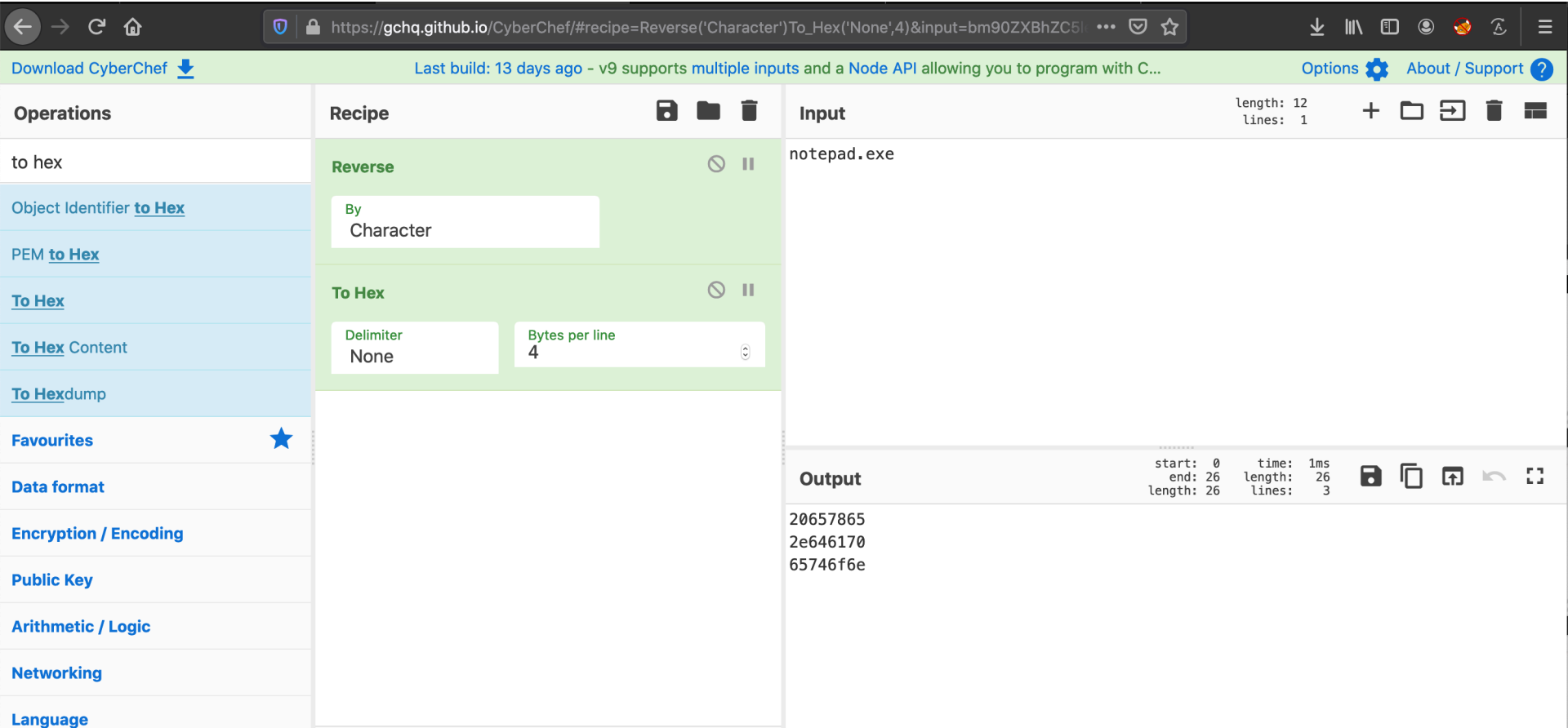
Parameters

command

The command to be executed.

https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/system-wsystem?view=vs-2019

CyberChef


https://gchq.github.io/CyberChef/#recipe=Reverse('Character')To_Hex('None',4)&input=bm90ZXBhZC5l
Download CyberChef
Last build: 13 days ago - v9 supports multiple inputs and a Node API allowing you to program with C...
Options
About / Support
Operations
to hex
Object Identifier to Hex
PEM to Hex
To Hex
To Hex Content
To Hexdump
Favourites
Data format
Encryption / Encoding
Public Key
Arithmetic / Logic
Networking
Language
Recipe
Reverse
By
Character
To Hex
Delimiter
None
Bytes per line
4
Input
length: 12
lines: 1
notepad.exe
Output
start: 0
end: 26
length: 26
time: 1ms
length: 26
lines: 3
20657865
2e646170
65746f6e

Obrigado!

helvio_junior@hotmail.com
@helvio_m4v3r1ck