Intro till assembly

Spooktober 2020

Presentatör: Mattias Grenfeldt
Slides: Asta Olofsson och Mattias Grenfeldt

KODSPORT SVERIGE

Schema

Lördag 17 oktober

- Intro till assembly - 14:00 (Du är här)
- Intro till reverse engineering - 15:00
- Intro till pwn - 16:00

Söndag 18 oktober

- Mini-CTF - 10:00-18:00 - ctf.sakerhetssm.se

Den här föreläsningen antar att du kan grundläggande programmering

Vad är assembly?

- De instruktioner som körs av din CPU. Väldigt små, enkla instruktioner.

```
for(int i = 0; i < 10; i++)
{
    printf("%d", i);
}
```

Kompilering

Assembly:

```
401142: mov     DWORD PTR [rbp-0x4],0x0
401149: jmp     401165
40114b: mov     eax,DWORD PTR [rbp-0x4]
40114e: mov     esi,eax
401150: lea     rdi,[rip+0xead]
401157: mov     eax,0x0
40115c: call    401040 <printf@plt>
401161: add     DWORD PTR [rbp-0x4],0x1
401165: cmp     DWORD PTR [rbp-0x4],0x9
401169: jle     40114b
```

=

Maskinkod:

```
11000111010001011111110000000
00000000000000000000000000011
10101100011010100010110100010
11111110010001001110001100100
10001000110100111101101011010
00011100000000000000000101110
00000000000000000000000000000
01110100011011111111111101111
11111111111110000011010001011
11111000000000011000001101...
```

Blir alla program till assembly innan de “körs”?

- Python
  - Interpreterat språk
  - NEJ
- Java
  - Kompilerar till JVM bytekod som sedan körs på JVM
  - NEJ
- C, C++, Rust, Go
  - Kompilerar direkt till maskinkod
  - JA

- JVM skriven i C++, Python skriven i C => Allt blir maskinkod till slut!

Förkunskaper

Talbaser

- Matte 1b och 1c i gymnasiet

$$1101_2 = 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 13$$

Bas	Namn	Tecken	Exempel
2	Binärt	0, 1	10100111001
8	Oktalt	0-7	2471
10	Decimalt	0-9	1337
16	Hexadecimalt / Hex	0-9, a, b, c, d, e, f	539

Talbaser

Vanliga notationer:

- Binärt - `0b101000101`
- Oktalt - `0o6256, 06256`
- Hex - `0xea67fcd8, ea67fcd8h`

Lätt att konvertera mellan binärt och hex:

- `0b1111 = 0xf`

Minne

- Bit: 0 eller 1
- Byte: den minsta enheten som lagras i minne
  - `0 - 255`
  - `0b00000000 - 0b11111111`
  - `0x00 - 0xff`

Minne är en stor array / lista av bytes

- array[index]
- memory[address]

Minne - Endian

Hur sparas tal större än 255?

- Lägg bytes på rad!
- 2172726 => 0x212736 => 0x21, 0x27, 0x36 => …?

Big-Endian: stora änden först

- memory = [0x21, 0x27, 0x36] = [21 27 36]

Little-Endian: lilla änden först

- memory = [0x36, 0x27, 0x21] = [36 27 21]

Minne - text och flyttal

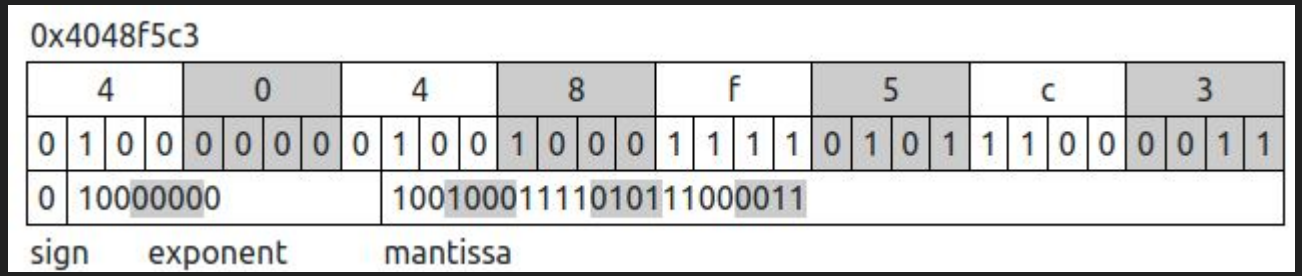
Hur spara text som bytes?

- Använd en stor tabell!
- ASCII
  - “hej” => 104, 101, 106 => [68 65 6a]
- Unicode och UTF-8
  - “hej 🖥“ => [68 65 6a 20 f0 9f 96 a5]

Byte-värde	Tecken
...	...
65	A
66	B
67	C
...	...

Flyttal: tal med decimaler

- 3.14 =>



Minne - Maskinkod

Vi kan ju självklart också spara maskinkod som bytes

Assembly:

```
401142: mov     DWORD PTR [rbp-0x4],0x0
401149: jmp     401165
40114b: mov     eax,DWORD PTR [rbp-0x4]
40114e: mov     esi,eax
401150: lea     rdi,[rip+0xead]
401157: mov     eax,0x0
40115c: call    401040 <printf@plt>
401161: add     DWORD PTR [rbp-0x4],0x1
401165: cmp     DWORD PTR [rbp-0x4],0x9
401169: jle     40114b
```

=

Maskinkod:

```
11000111010001011111110000000
00000000000000000000000000011
10101100011010100010110100010
11111110010001001110001100100
10001000110100111101101011010
00011100000000000000000101110
00000000000000000000000000000
01110100011011111111111101111
11111111111110000011010001011
11111000000000011000001101...
```

Grundläggande assembly

Nu assembly!

Men vilken?

Beror på CPU:

- x86, ARM, MIPS, AVR, SPARC …
- 16 bitar, 32 bitar, 64 bitar
- Little-Endian eller Big-Endian

Vi kommer att kolla på Little-Endian 64-bitars x86. (x86-64).

Syntax

Intel	AT&T
`401142: mov  DWORD PTR [rbp-0x4],0x0`	`401142: movl  $0x0,-0x4(%rbp)`
`401149: jmp  401165`	`401149: jmp  401165`
`40114b: mov  eax,DWORD PTR [rbp-0x4]`	`40114b: mov  -0x4(%rbp),%eax`
`40114e: mov  esi,eax`	`40114e: mov  %eax,%esi`
`401150: lea  rdi,[rip+0xead]`	`401150: lea  0xead(%rip),%rdi`
`401157: mov  eax,0x0`	`401157: mov  $0x0,%eax`
`40115c: call  401040 <printf@plt>`	`40115c: callq  401040 <printf@plt>`
`401161: add  DWORD PTR [rbp-0x4],0x1`	`401161: addl  $0x1,-0x4(%rbp)`
`401165: cmp  DWORD PTR [rbp-0x4],0x9`	`401165: cmpl  $0x9,-0x4(%rbp)`
`401169: jle  40114b`	`401169: jle  40114b`

Vi kommer kolla på Intel-syntax

Exempel

C	x86
`int a = 5;`	`mov rax, 5`
`int b = 7;`	`mov rbx, 7`
`b += b;`	`add rbx, rbx`
`b -= a;`	`sub rbx, rax`
`int c = b;`	`mov rcx, rbx`
`c ^= a;`	`xor rcx, rax`

- Variabler kallas register
- Hur vet CPUn var den är?
- Ser bara: `b805000000bb070000004801db4829c34889d94831c1`

Exempel

C	x86
`int a = 5;`	`401140: b8 05 00 00 00 mov rax, 5`
`int b = 7;`	`401145: bb 07 00 00 00 mov rbx, 7`
`b += b;`	`40114a: 48 01 db add rbx, rbx`
`b -= a;`	`40114d: 48 29 c3 sub rbx, rax`
`int c = b;`	`401150: 48 89 d9 mov rcx, rbx`
`c ^= a;`	`401153: 48 31 c1 xor rcx, rax`

- Variabler kallas register
- Hur vet CPUn var den är?
- Ser bara: `b805000000bb070000004801db4829c34889d94831c1`
- Instruktionspekaren: `rip`

Exempel

C	x86
`1. int a = 5;`	`401140: b8 05 00 00 00 mov eax, 5`
`2. int b = 7;`	`401145: bb 07 00 00 00 mov ebx, 7`
`3. b += b;`	`40114a: 48 01 db add rbx, rbx`
`4. b -= a;`	`40114d: 48 29 c3 sub rbx, rax`
`5. int c = b;`	`401150: 48 89 d9 mov rcx, rbx`
`6. c ^= a;`	`401153: 48 31 c1 xor rcx, rax`

- Variabler kallas register
- Hur vet CPUn var den är?
- Ser bara: `b805000000bb070000004801db4829c34889d94831c1`
- Instruktionspekaren: `rip`

Register

Globala variabler i CPUn

64 bitar
`rax`
`rbx`
`rcx`
`rdx`

Register

Globala variabler i CPUn

64 bitar	Lägsta 32 bitarna
`rax`	`eax`
`rbx`	`ebx`
`rcx`	`ecx`
`rdx`	`edx`

Register

Globala variabler i CPUn

64 bitar	Lägsta 32 bitarna	Lägsta 16 bitarna
`rax`	`eax`	`ax`
`rbx`	`ebx`	`bx`
`rcx`	`ecx`	`cx`
`rdx`	`edx`	`dx`

Register

Globala variabler i CPUn

64 bitar	Lägsta 32 bitarna	Lägsta 16 bitarna	Lägsta 8 bitarna
`rax`	`eax`	`ax`	`al`
`rbx`	`ebx`	`bx`	`bl`
`rcx`	`ecx`	`cx`	`cl`
`rdx`	`edx`	`dx`	`dl`

Register

Globala variabler i CPUn

64 bitar	Lägsta 32 bitarna	Lägsta 16 bitarna	Lägsta 8 bitarna
`rax`	`eax`	`ax`	`al`
`rbx`	`ebx`	`bx`	`bl`
`rcx`	`ecx`	`cx`	`cl`
`rdx`	`edx`	`dx`	`dl`

Alla register

Några extra viktiga: `rip`, `rsp` (stackpekaren), `rbp` (baspekaren)

Några instruktioner

- `jmp 401165`
  - Samma som: `mov rip, 401165`
- `cmp rax, rbx`
  `je 401165`
  - `if(rax == rbx) { rip = 401165; }`
- `cmp rax, rbx`
  `jl 401165`
  - `if(rax < rbx) { rip = 401165; }`

Finns många olika jump: `je, jne, jb, jnb, ja, jg, jge, jle, ...`

Några instruktioner

- `mov rax, rbx`
- `mov rax, [rbx]`
  - `rax = *rbx;`
- `mov rax, [rbx + 8]`
  - `rax = *(rbx + 8);`
- `lea rax, [rbx + 8]`
  - Load effective address
  - `rax = rbx + 8;`
- `mov [rax], rbx`
  - `*rax = rbx;`

Några instruktioner

- `mov QWORD PTR [rax], rbx`
- `mov rax, QWORD PTR [rbx]`

Instruktion	Storlek
BYTE PTR	8 bit / 1 byte
WORD PTR	16 bitar / 2 bytes
DWORD PTR	32 bitar / 4 bytes
QWORD PTR	64 bitar / 8 bytes

Funktioner och stacken

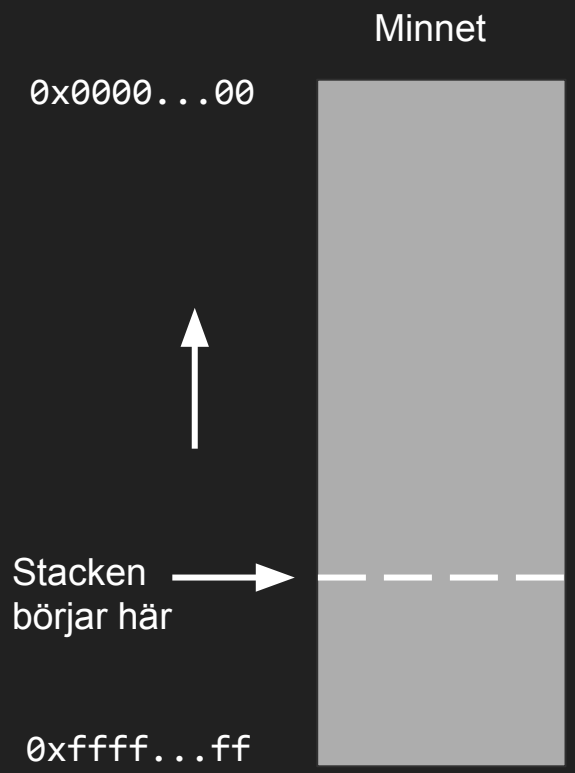
En stack

- En hög med saker
- Två operationer
  - Push
  - Pop

- Hur använder funktioner stacken?
  - Spara lokala variabler och parametrar
  - Spara returaddress och baspekare
  - Push vid anrop, pop vid retur
  - Stack frame

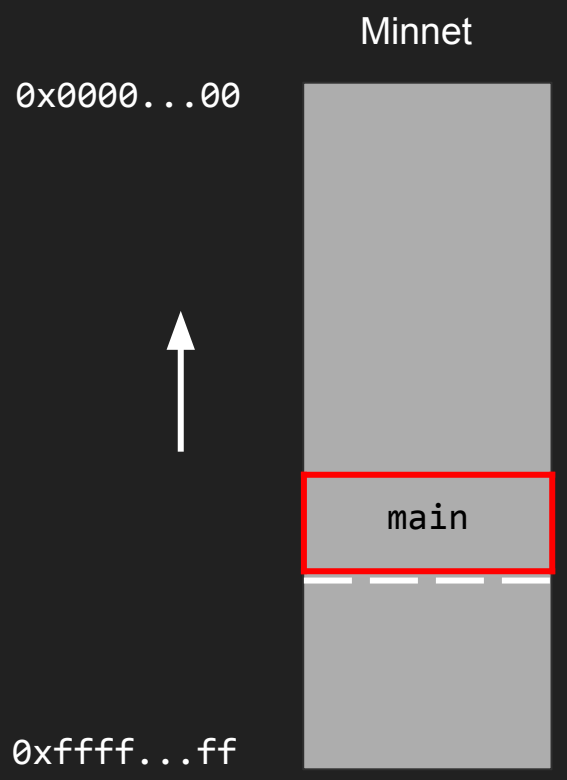
Exempel

```c
int add(int a, int b) {
    return a + b;
}

void print(int a, int b) {
    int r = add(a, b);
    printf("%d\n", r);
}

int main() {
    int x = 5;
    print(x, 6);
}
```



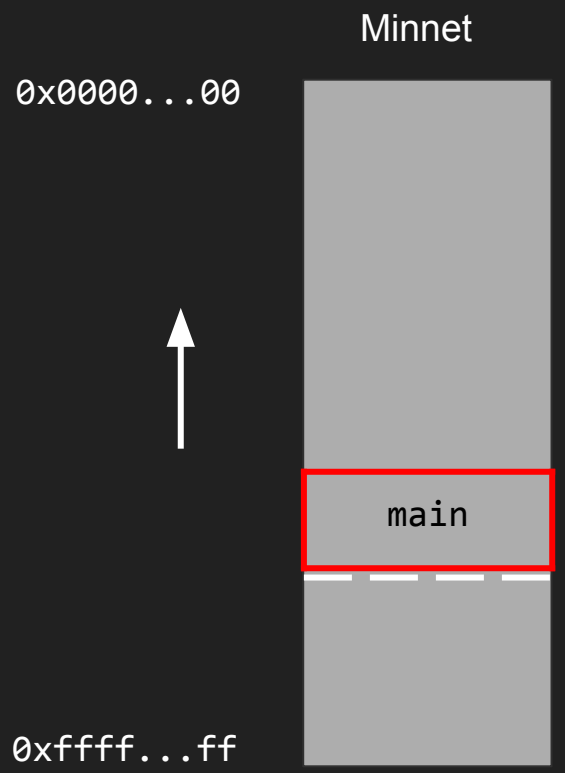
Exempel

```c
int add(int a, int b) {
    return a + b;
}

void print(int a, int b) {
    int r = add(a, b);
    printf("%d\n", r);
}

int main() {
    int x = 5;
    print(x, 6);
}
```



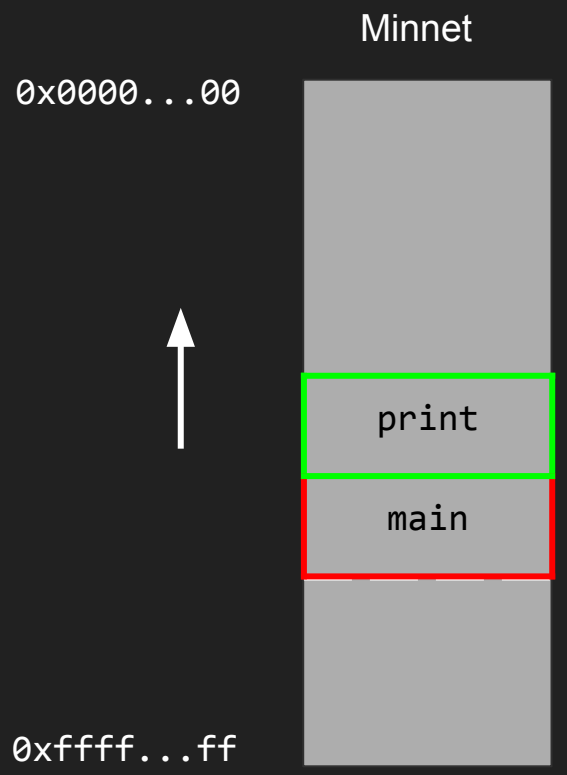
Exempel

```
int add(int a, int b) {
    return a + b;
}

void print(int a, int b) {
    int r = add(a, b);
    printf("%d\n", r);
}

int main() {
    int x = 5;
    print(x, 6);
}
```



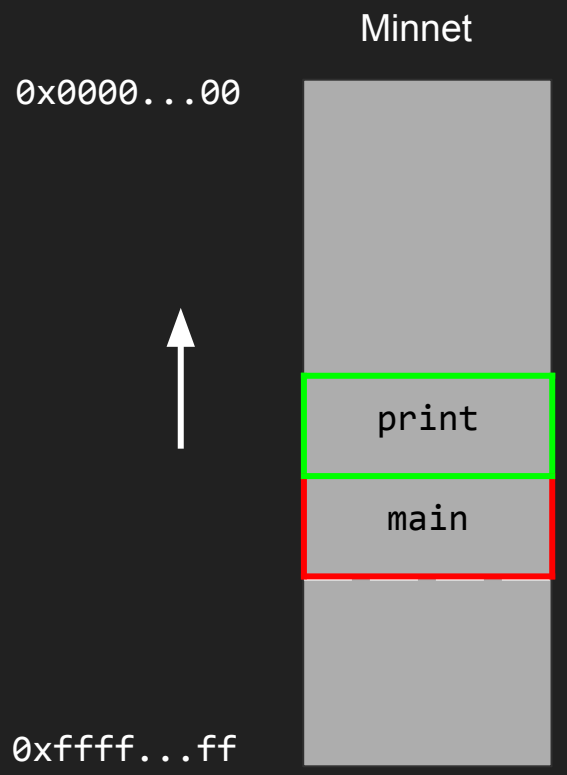
Exempel

```c
int add(int a, int b) {
    return a + b;
}

void print(int a, int b) {
    int r = add(a, b);
    printf("%d\n", r);
}

int main() {
    int x = 5;
    print(x, 6);
}
```



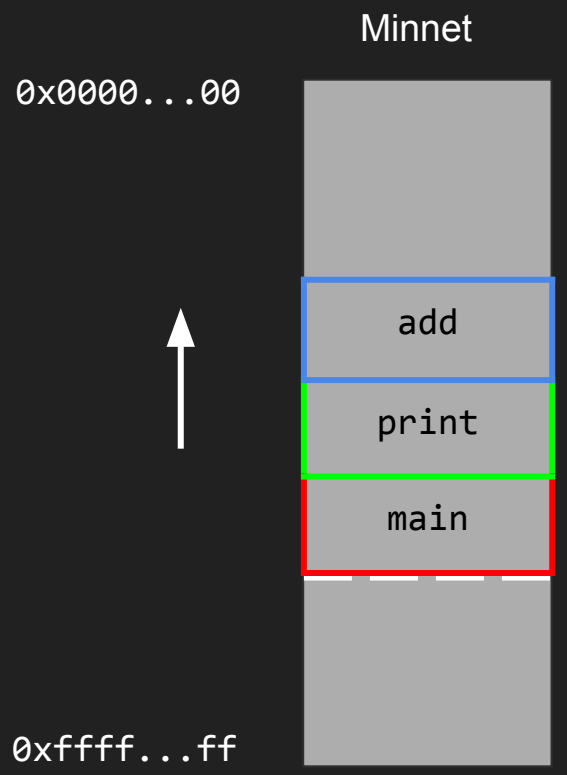
Exempel

```c
int add(int a, int b) {
    return a + b;
}

void print(int a, int b) {
    int r = add(a, b);
    printf("%d\n", r);
}

int main() {
    int x = 5;
    print(x, 6);
}
```



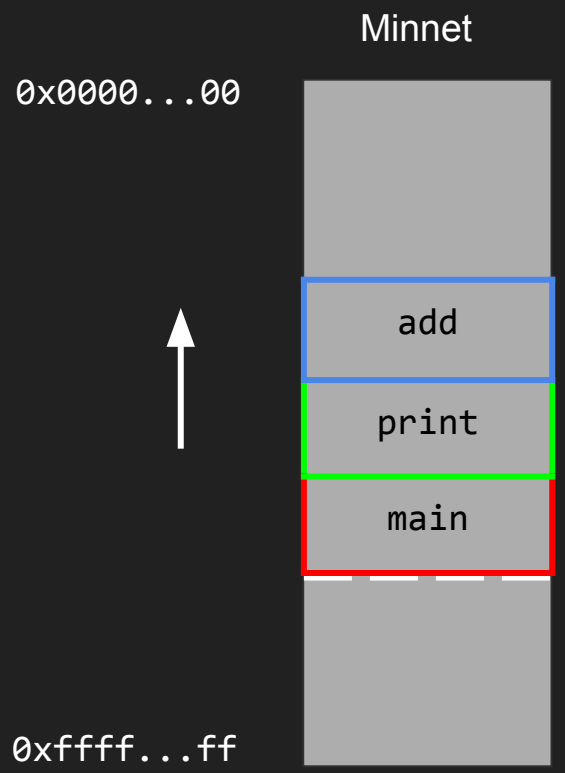
Exempel

```c
int add(int a, int b) {
    return a + b;
}

void print(int a, int b) {
    int r = add(a, b);
    printf("%d\n", r);
}

int main() {
    int x = 5;
    print(x, 6);
}
```



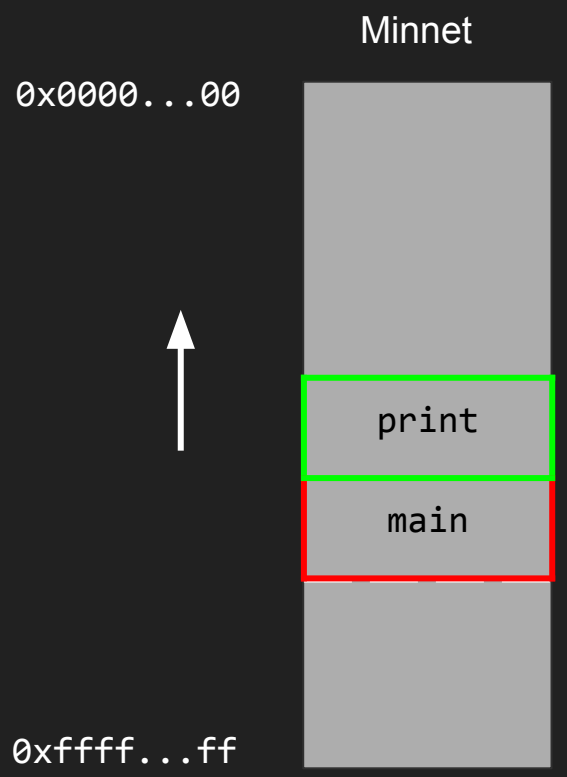
Exempel

```
int add(int a, int b) {
    return a + b;
}

void print(int a, int b) {
    int r = add(a, b);
    printf("%d\n", r);
}

int main() {
    int x = 5;
    print(x, 6);
}
```



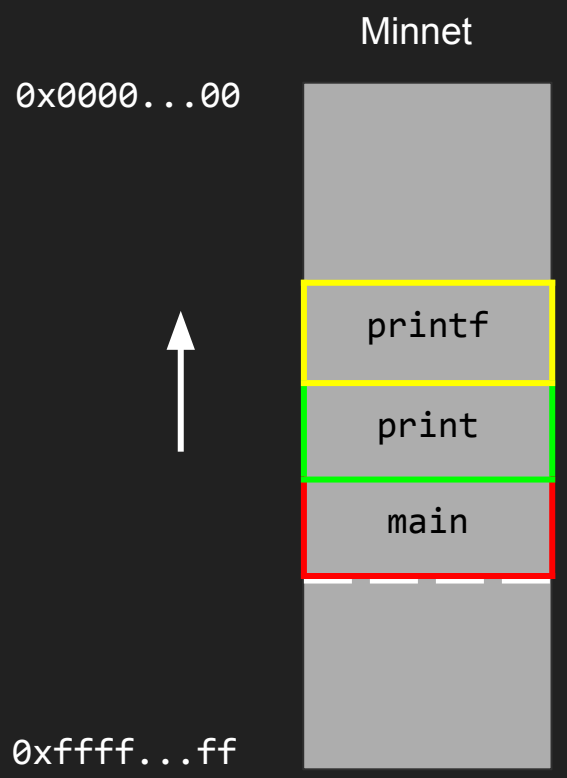
Exempel

```c
int add(int a, int b) {
    return a + b;
}

void print(int a, int b) {
    int r = add(a, b);
    printf("%d\n", r);
}

int main() {
    int x = 5;
    print(x, 6);
}
```



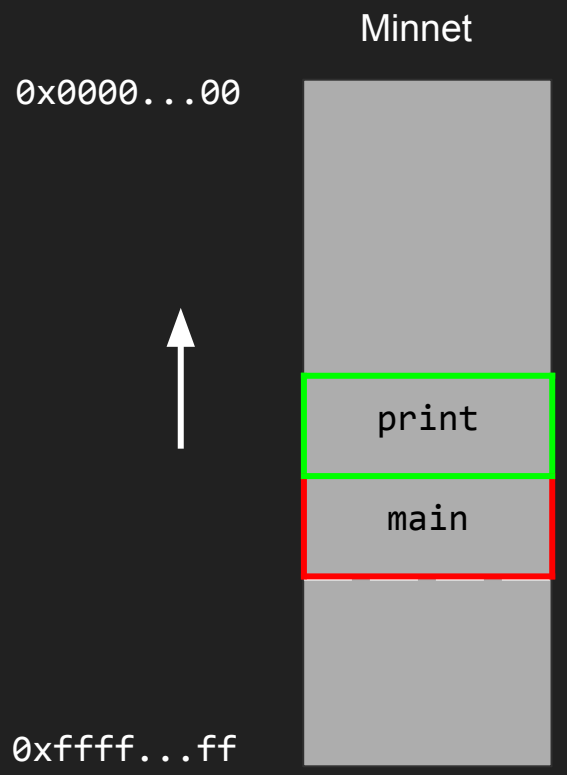
Exempel

```c
int add(int a, int b) {
    return a + b;
}

void print(int a, int b) {
    int r = add(a, b);
    printf("%d\n", r);
}

int main() {
    int x = 5;
    print(x, 6);
}
```



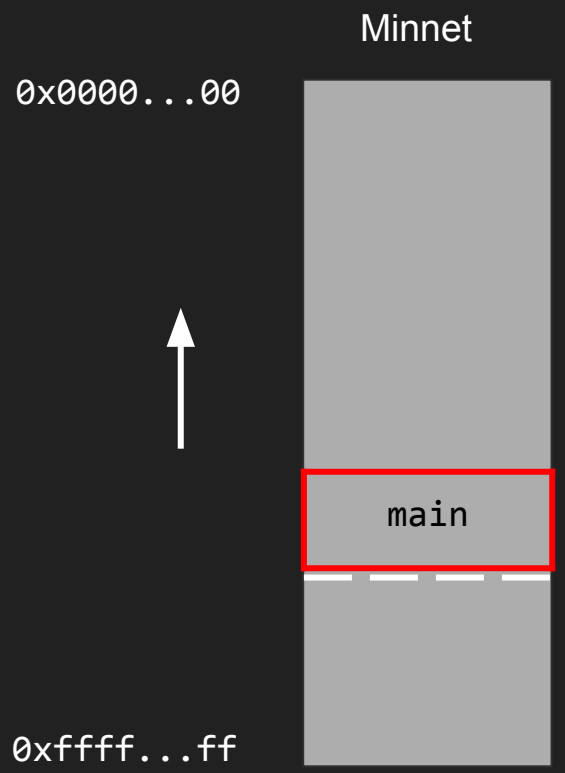
Exempel

```c
int add(int a, int b) {
    return a + b;
}

void print(int a, int b) {
    int r = add(a, b);
    printf("%d\n", r);
}

int main() {
    int x = 5;
    print(x, 6);
}
```



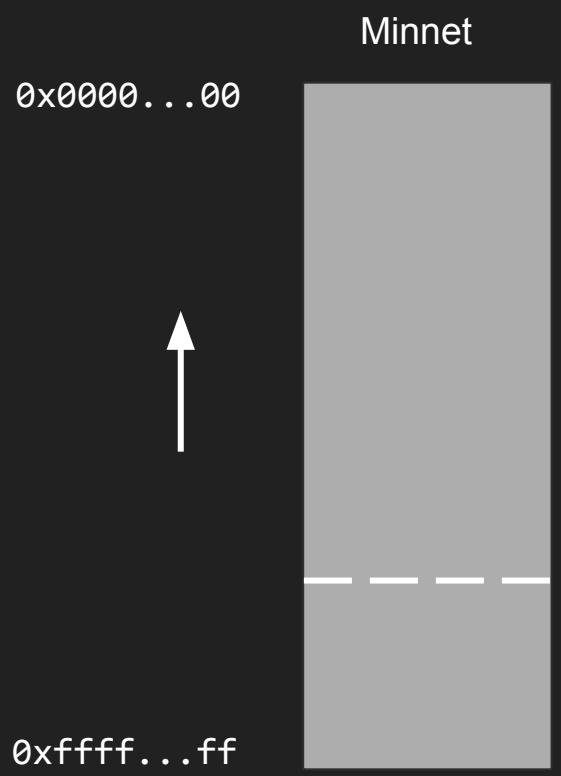
Exempel

```
int add(int a, int b) {
    return a + b;
}

void print(int a, int b) {
    int r = add(a, b);
    printf("%d\n", r);
}

int main() {
    int x = 5;
    print(x, 6);
}
```



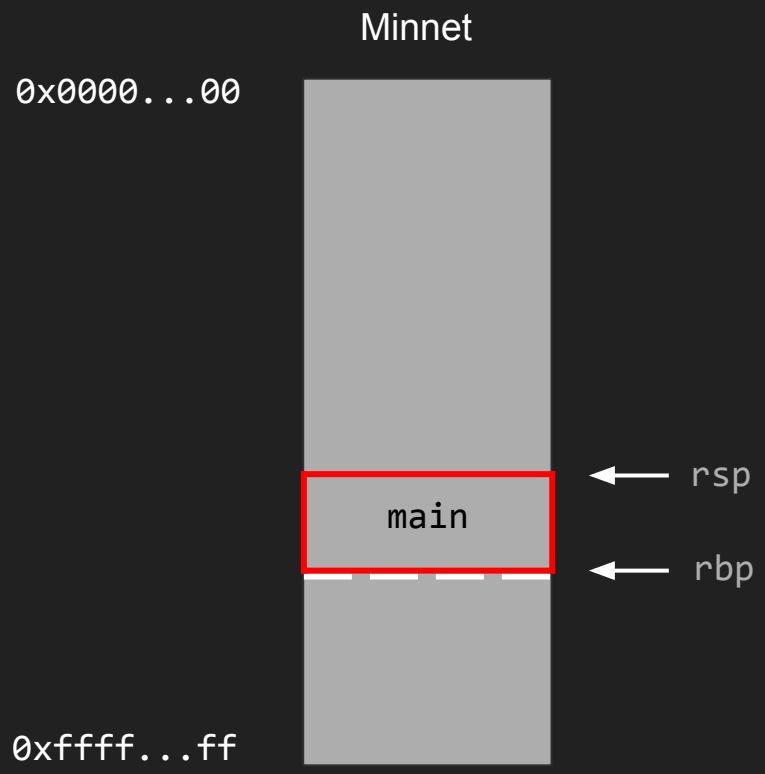
Exempel

```c
int add(int a, int b) {
    return a + b;
}

void print(int a, int b) {
    int r = add(a, b);
    printf("%d\n", r);
}

int main() {
    int x = 5;
    print(x, 6);
}
```



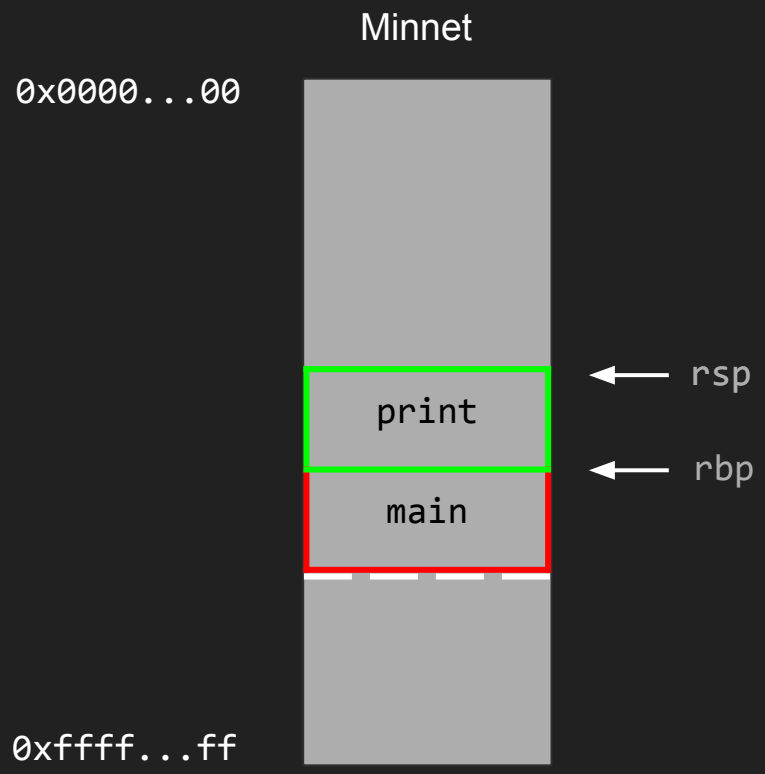
Stack- och baspekaren

- rsp
  - Pekar på toppen av stacken
- rbp
  - Pekar på botten av översta stack framen



Stack- och baspekaren

- rsp
  - Pekar på toppen av stacken
- rbp
  - Pekar på botten av översta stack framen



Stackrelaterade instruktioner

- `push rax`
  - `sub rsp, 8`
    `mov [rsp], rax`
- `pop rax`
  - `mov rax, [rsp]`
    `add rsp, 8`
- `call 401040`
  - `push [address till nästa instruktion]`
    `jmp 401040`
- `ret`
  - `pop rip`

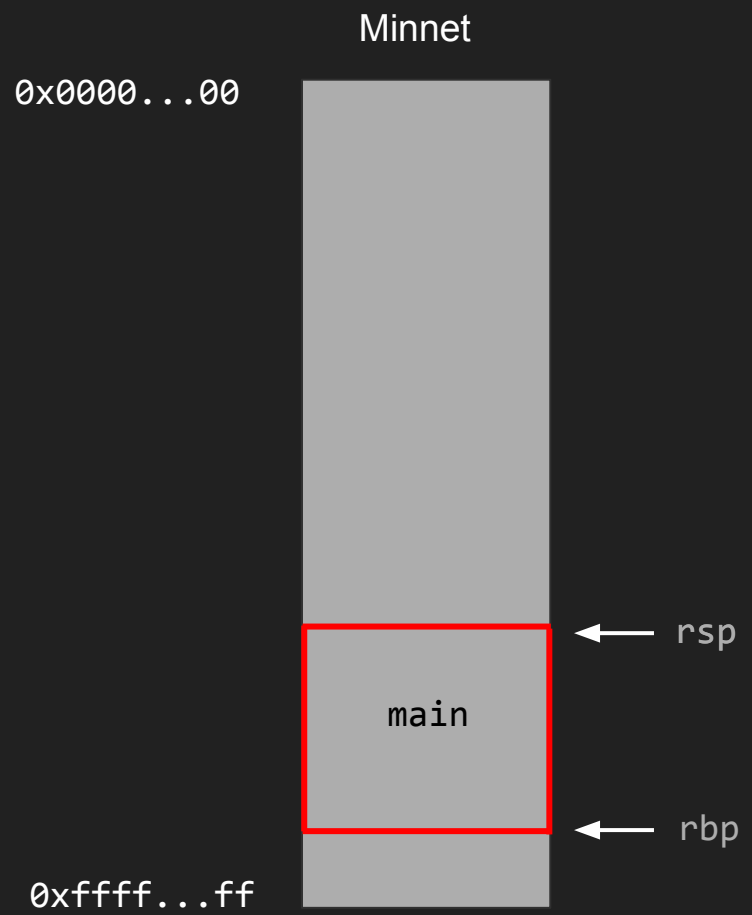
Calling conventions

- Hur händer ett anrop i detalj?
- Beror på calling convention
- Vi kollar på Linux, GCC, 64 bit, calling convention:
  - Argument skickas via: `rdi, rsi, rdx, rcx, r8, r9`
  - Om fler, lägg på stacken
  - ...

- OBS: vid 32 bitar skickas alla argument på stacken

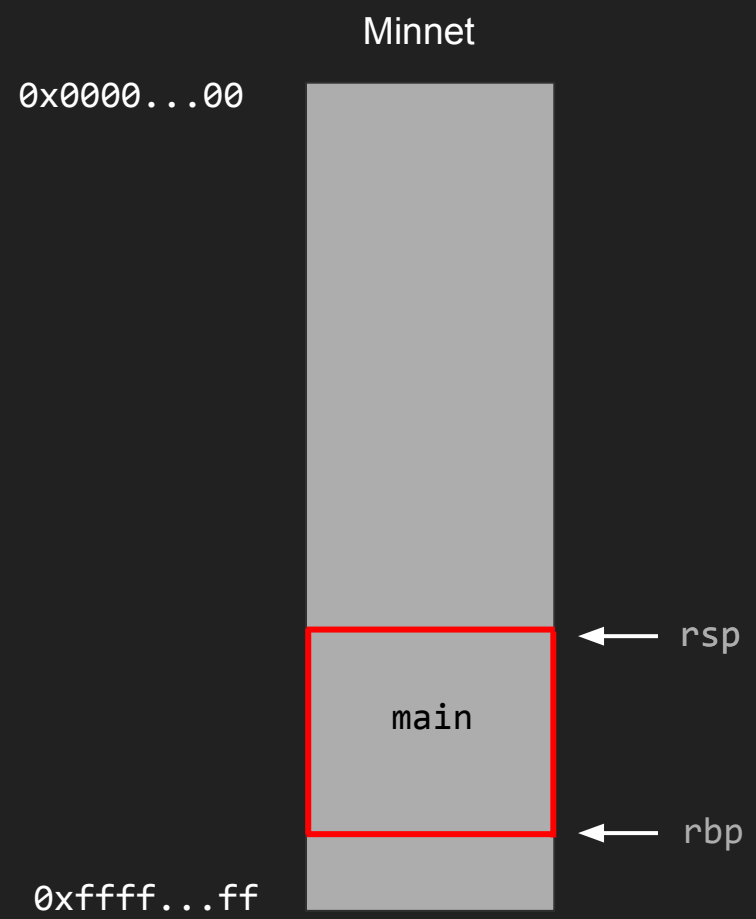
Ett anrop i detalj

```
main:
 ...
 40119e: mov    eax,DWORD PTR [rbp-0x4]
 4011a1: mov    esi,0x6
 4011a6: mov    edi,eax
 4011a8: call   40114e <print>
 4011ad: ...

print:
 40114e: endbr64
 401152: push   rbp
 401153: mov    rbp,rsp
 401156: sub    rsp,0x20
 40115a: mov    DWORD PTR [rbp-0x14],edi
 40115d: mov    DWORD PTR [rbp-0x18],esi
 ...
 401189: leave
 40118a: ret
```



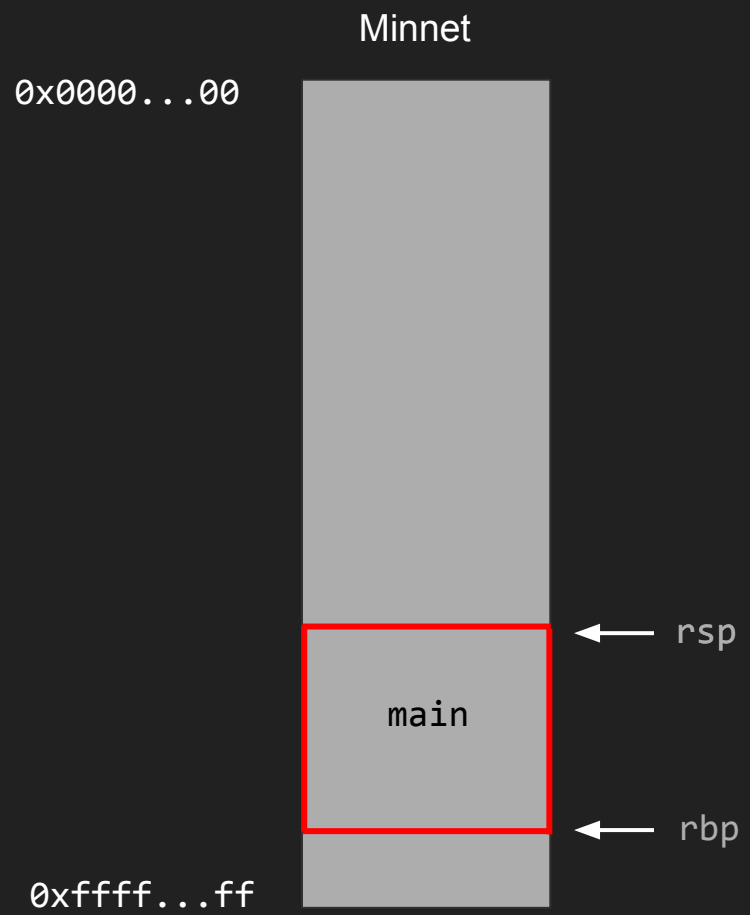
Ett anrop i detalj

```
main:
 ...
 40119e: mov    eax,DWORD PTR [rbp-0x4]
 4011a1: mov    esi,0x6
 4011a6: mov    edi,eax
 4011a8: call   40114e <print>
 4011ad: ...

print:
 40114e: endbr64
 401152: push   rbp
 401153: mov    rbp,rsp
 401156: sub    rsp,0x20
 40115a: mov    DWORD PTR [rbp-0x14],edi
 40115d: mov    DWORD PTR [rbp-0x18],esi
 ...
 401189: leave
 40118a: ret
```



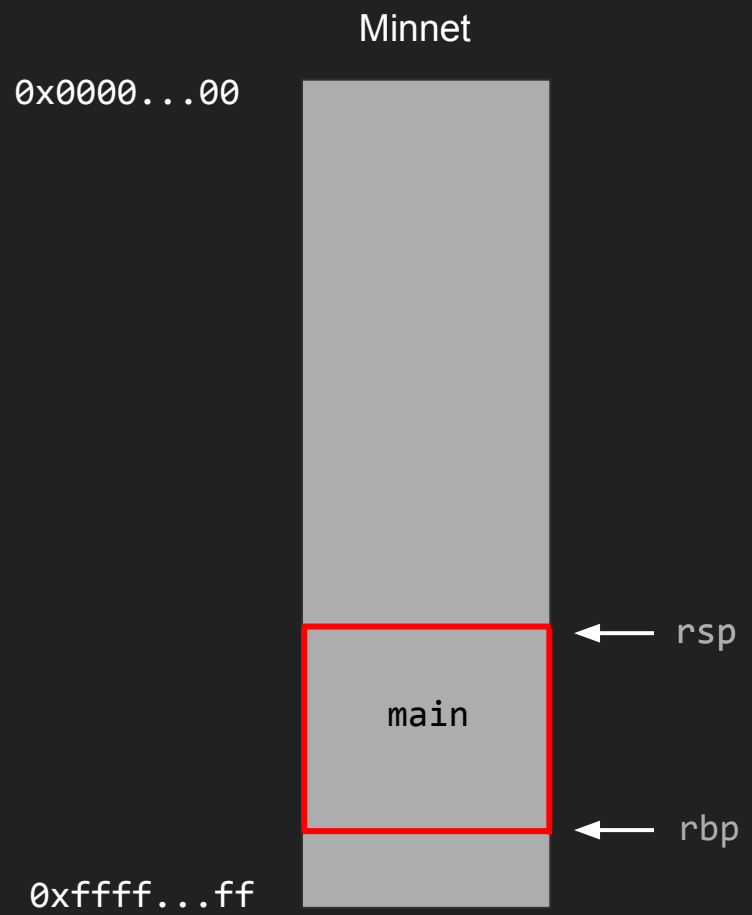
Ett anrop i detalj

```
main:
 ...
 40119e: mov    eax,DWORD PTR [rbp-0x4]
 4011a1: mov    esi,0x6
 4011a6: mov    edi,eax
 4011a8: call   40114e <print>
 4011ad: ...

print:
 40114e: endbr64
 401152: push   rbp
 401153: mov    rbp,rsp
 401156: sub    rsp,0x20
 40115a: mov    DWORD PTR [rbp-0x14],edi
 40115d: mov    DWORD PTR [rbp-0x18],esi
 ...
 401189: leave
 40118a: ret
```



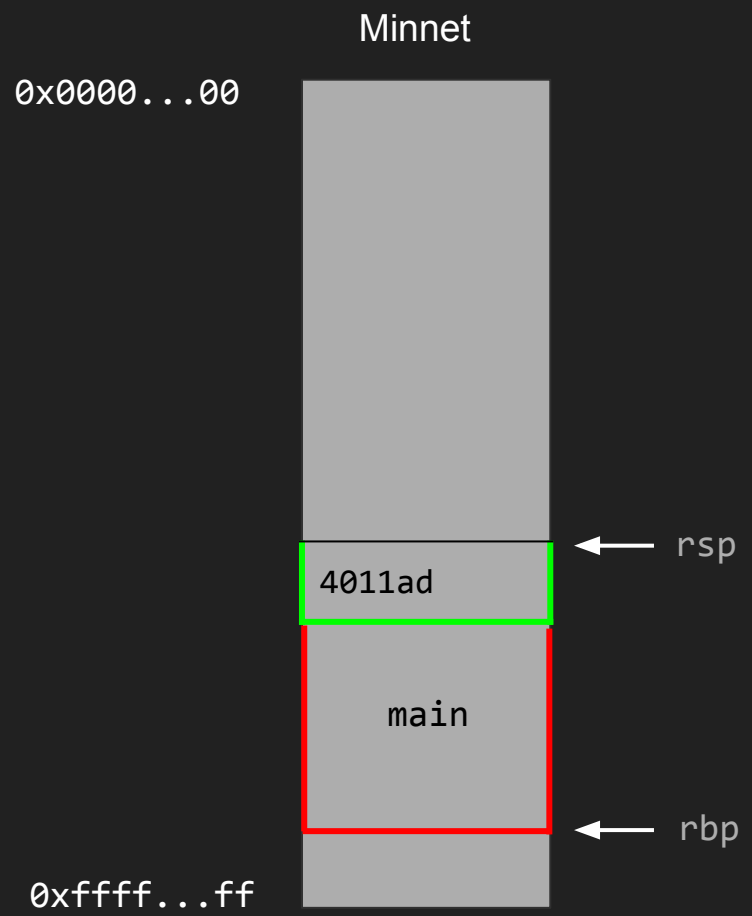
Ett anrop i detalj

```
main:
 ...
 40119e: mov    eax,DWORD PTR [rbp-0x4]
 4011a1: mov    esi,0x6
 4011a6: mov    edi,eax
 4011a8: call   40114e <print>
 4011ad: ...

print:
 40114e: endbr64
 401152: push   rbp
 401153: mov    rbp,rsp
 401156: sub    rsp,0x20
 40115a: mov    DWORD PTR [rbp-0x14],edi
 40115d: mov    DWORD PTR [rbp-0x18],esi
 ...
 401189: leave
 40118a: ret
```



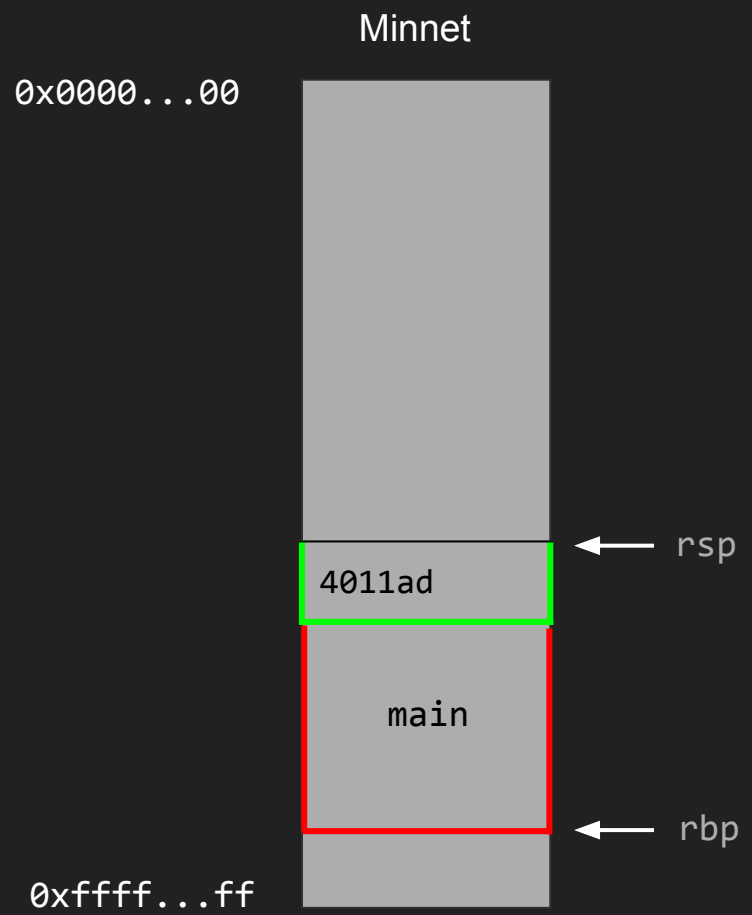
Ett anrop i detalj

```
main:
 ...
 40119e: mov    eax,DWORD PTR [rbp-0x4]
 4011a1: mov    esi,0x6
 4011a6: mov    edi,eax
 4011a8: call   40114e <print>
 4011ad: ...

print:
 40114e: endbr64
 401152: push   rbp
 401153: mov    rbp,rsp
 401156: sub    rsp,0x20
 40115a: mov    DWORD PTR [rbp-0x14],edi
 40115d: mov    DWORD PTR [rbp-0x18],esi
 ...
 401189: leave
 40118a: ret
```



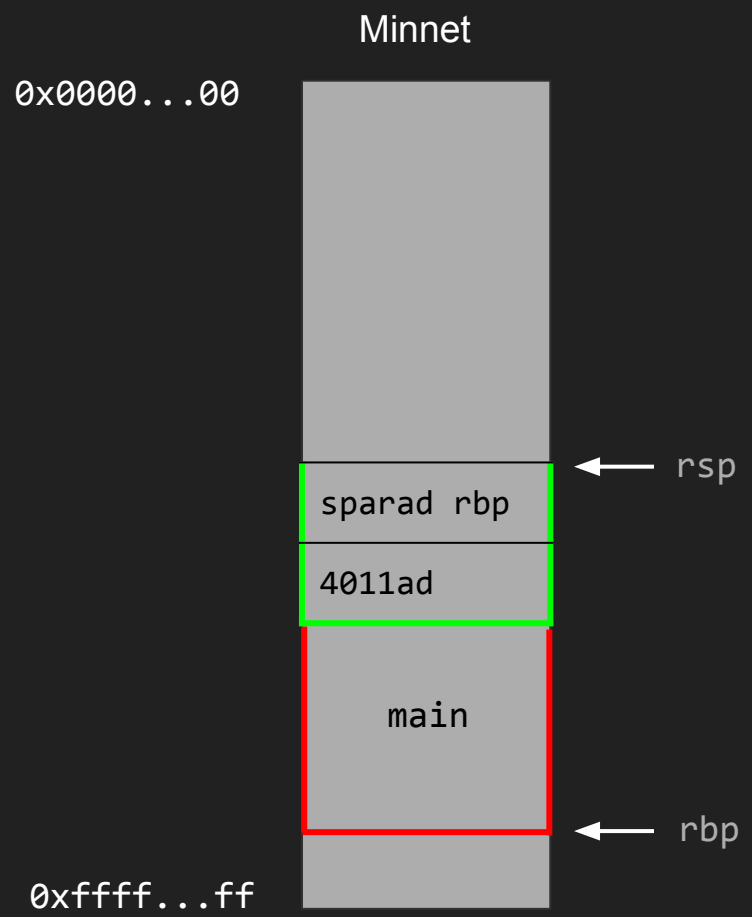
Ett anrop i detalj

```
main:
 ...
 40119e: mov    eax,DWORD PTR [rbp-0x4]
 4011a1: mov    esi,0x6
 4011a6: mov    edi,eax
 4011a8: call   40114e <print>
 4011ad: ...

print:
 40114e: endbr64
 401152: push   rbp
 401153: mov    rbp,rsp
 401156: sub    rsp,0x20
 40115a: mov    DWORD PTR [rbp-0x14],edi
 40115d: mov    DWORD PTR [rbp-0x18],esi
 ...
 401189: leave
 40118a: ret
```



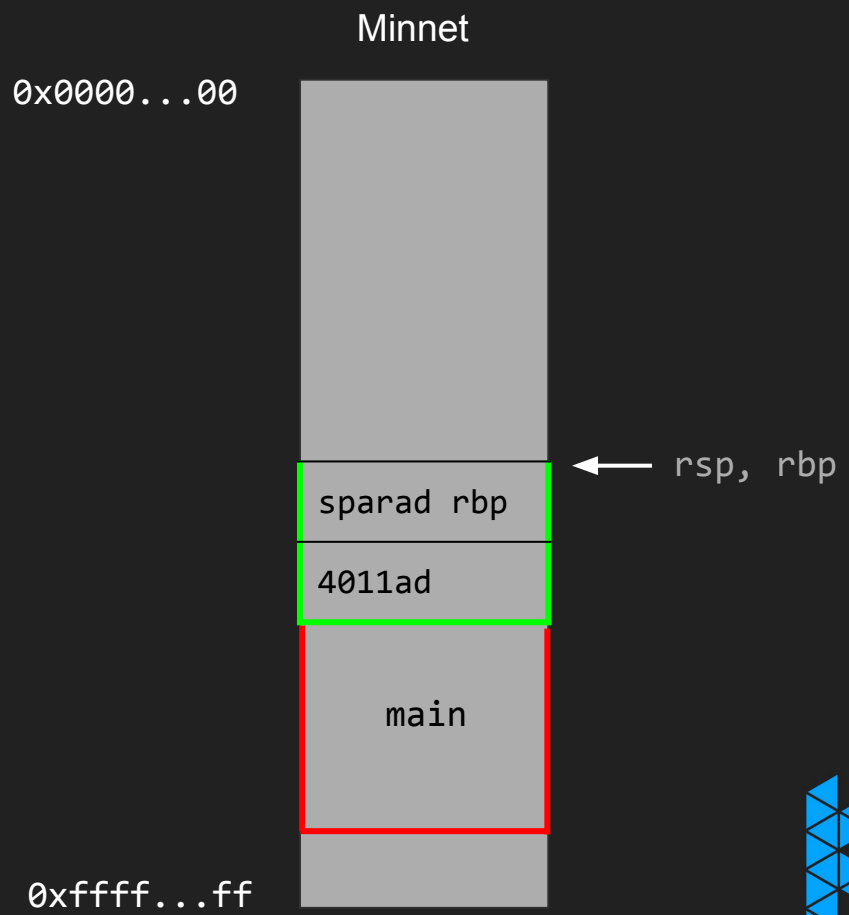
Ett anrop i detalj

```
main:
 ...
 40119e: mov    eax,DWORD PTR [rbp-0x4]
 4011a1: mov    esi,0x6
 4011a6: mov    edi,eax
 4011a8: call   40114e <print>
 4011ad: ...

print:
 40114e: endbr64
 401152: push   rbp
 401153: mov    rbp,rsp
 401156: sub    rsp,0x20
 40115a: mov    DWORD PTR [rbp-0x14],edi
 40115d: mov    DWORD PTR [rbp-0x18],esi
 ...
 401189: leave
 40118a: ret
```



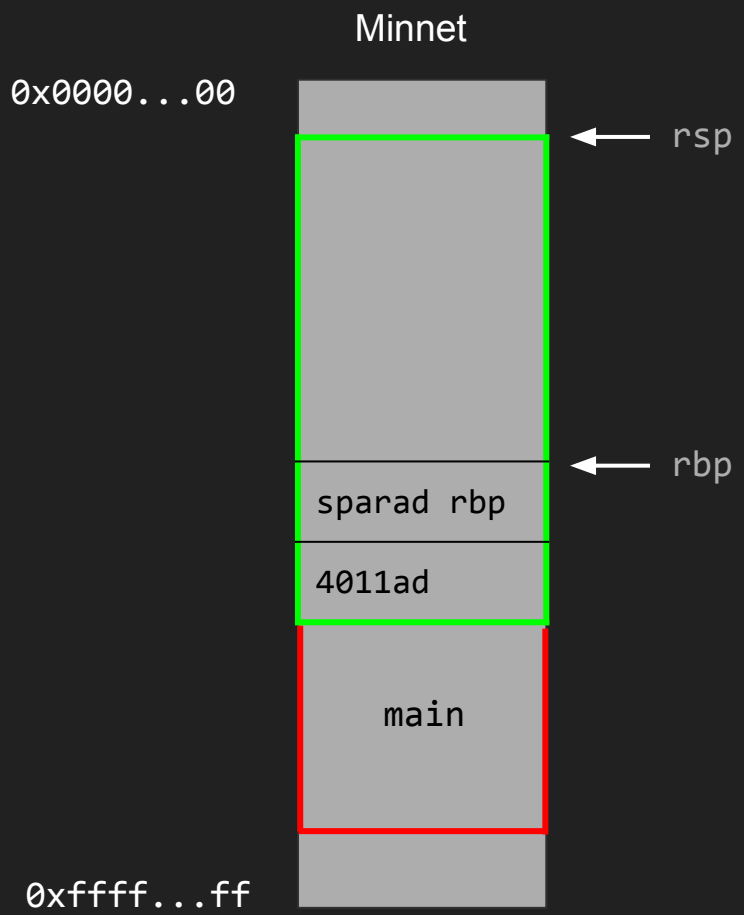
Ett anrop i detalj

```
main:
 ...
 40119e: mov    eax,DWORD PTR [rbp-0x4]
 4011a1: mov    esi,0x6
 4011a6: mov    edi,eax
 4011a8: call   40114e <print>
 4011ad: ...

print:
 40114e: endbr64
 401152: push   rbp
 401153: mov    rbp,rsp
 401156: sub    rsp,0x20
 40115a: mov    DWORD PTR [rbp-0x14],edi
 40115d: mov    DWORD PTR [rbp-0x18],esi
 ...
 401189: leave
 40118a: ret
```



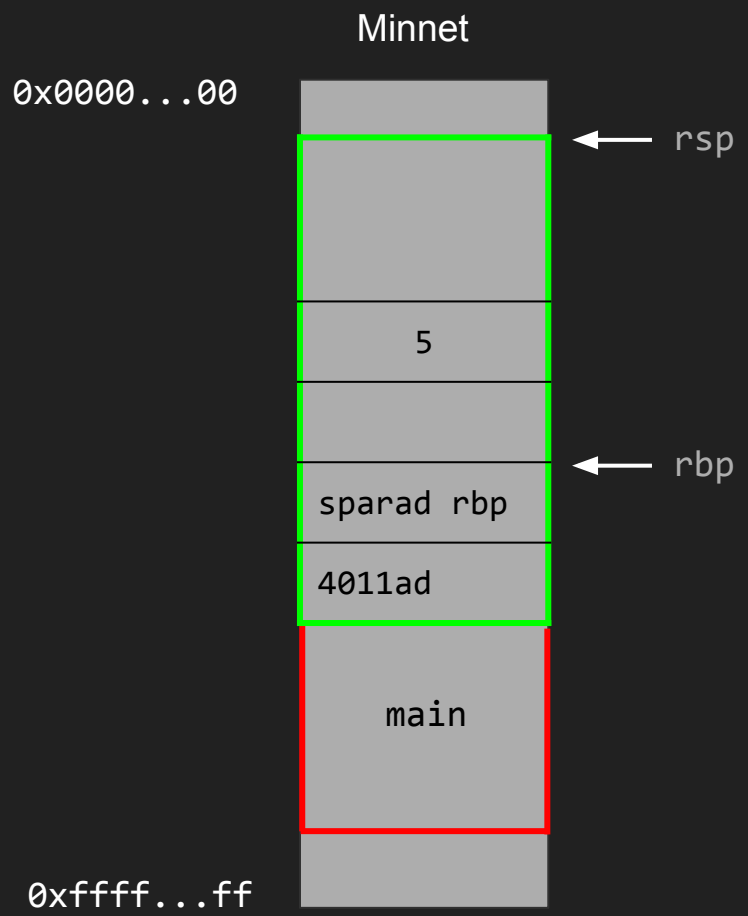
Ett anrop i detalj

```
main:
 ...
 40119e: mov    eax,DWORD PTR [rbp-0x4]
 4011a1: mov    esi,0x6
 4011a6: mov    edi,eax
 4011a8: call   40114e <print>
 4011ad: ...

print:
 40114e: endbr64
 401152: push   rbp
 401153: mov    rbp,rsp
 401156: sub    rsp,0x20
 40115a: mov    DWORD PTR [rbp-0x14],edi
 40115d: mov    DWORD PTR [rbp-0x18],esi
 ...
 401189: leave
 40118a: ret
```



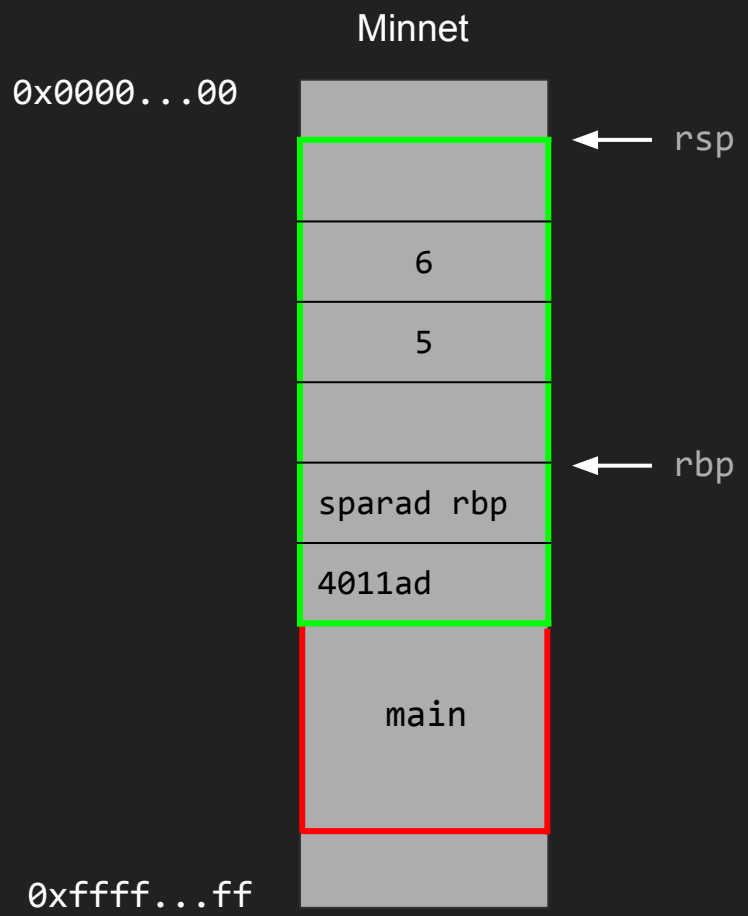
Ett anrop i detalj

```
main:
 ...
 40119e: mov    eax,DWORD PTR [rbp-0x4]
 4011a1: mov    esi,0x6
 4011a6: mov    edi,eax
 4011a8: call   40114e <print>
 4011ad: ...

print:
 40114e: endbr64
 401152: push   rbp
 401153: mov    rbp,rsp
 401156: sub    rsp,0x20
 40115a: mov    DWORD PTR [rbp-0x14],edi
 40115d: mov    DWORD PTR [rbp-0x18],esi
 ...
 401189: leave
 40118a: ret
```



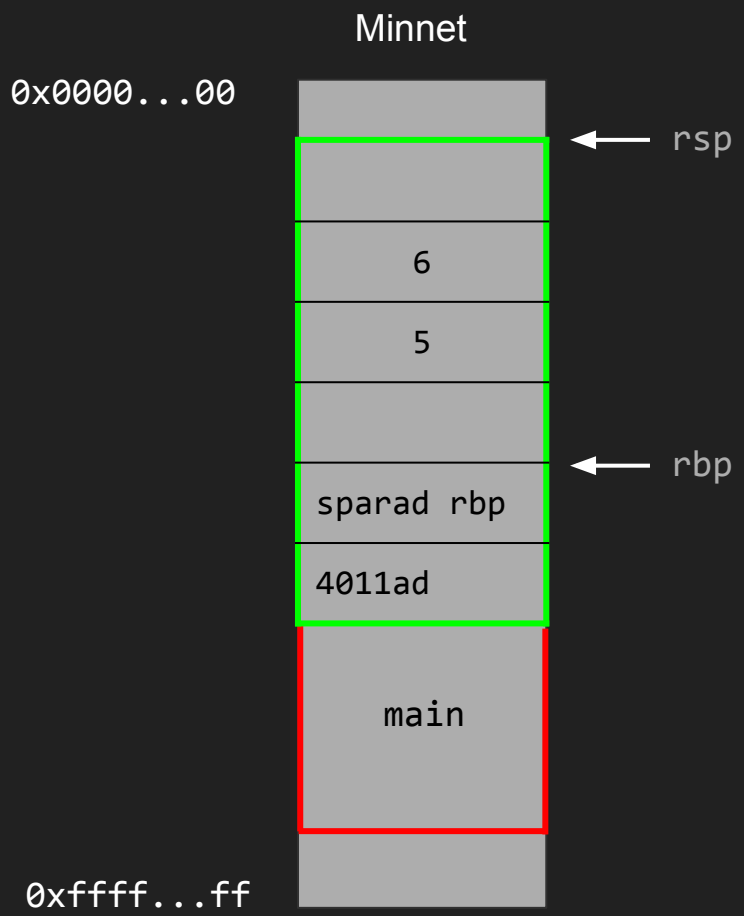
Ett anrop i detalj

```
main:
 ...
 40119e: mov    eax,DWORD PTR [rbp-0x4]
 4011a1: mov    esi,0x6
 4011a6: mov    edi,eax
 4011a8: call   40114e <print>
 4011ad: ...

print:
 40114e: endbr64
 401152: push   rbp
 401153: mov    rbp,rsp
 401156: sub    rsp,0x20
 40115a: mov    DWORD PTR [rbp-0x14],edi
 40115d: mov    DWORD PTR [rbp-0x18],esi
 ...
 401189: leave
 40118a: ret
```



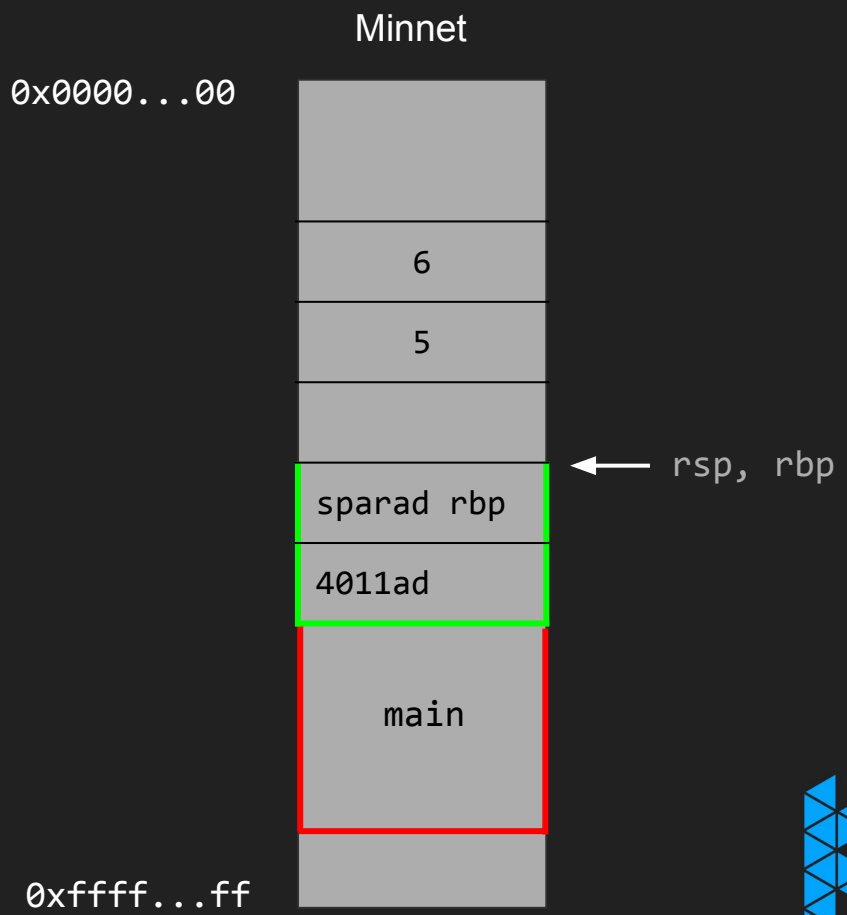
Ett anrop i detalj

```
main:
 ...
 40119e: mov    eax,DWORD PTR [rbp-0x4]
 4011a1: mov    esi,0x6
 4011a6: mov    edi,eax
 4011a8: call   40114e <print>
 4011ad: ...

print:
 40114e: endbr64
 401152: push   rbp
 401153: mov    rbp,rsp
 401156: sub    rsp,0x20
 40115a: mov    DWORD PTR [rbp-0x14],edi
 40115d: mov    DWORD PTR [rbp-0x18],esi
 ...
 401189: leave
 40118a: ret
```



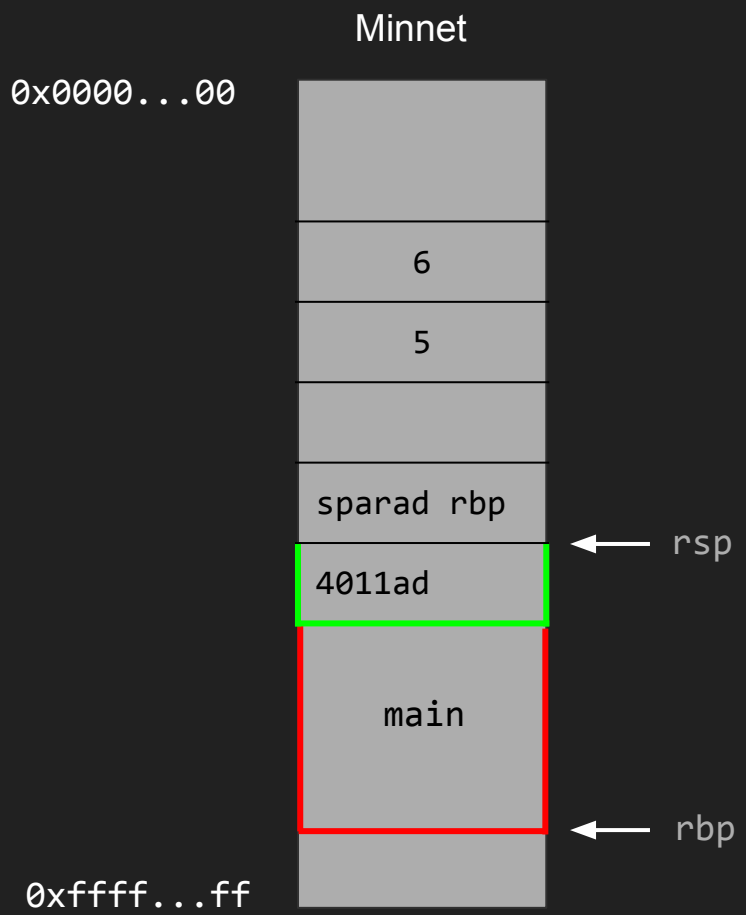
Ett anrop i detalj

```
main:
 ...
 40119e: mov    eax,DWORD PTR [rbp-0x4]
 4011a1: mov    esi,0x6
 4011a6: mov    edi,eax
 4011a8: call   40114e <print>
 4011ad: ...

print:
 40114e: endbr64
 401152: push   rbp
 401153: mov    rbp,rsp
 401156: sub    rsp,0x20
 40115a: mov    DWORD PTR [rbp-0x14],edi
 40115d: mov    DWORD PTR [rbp-0x18],esi
 ...
 401189: leave
 40118a: ret
```



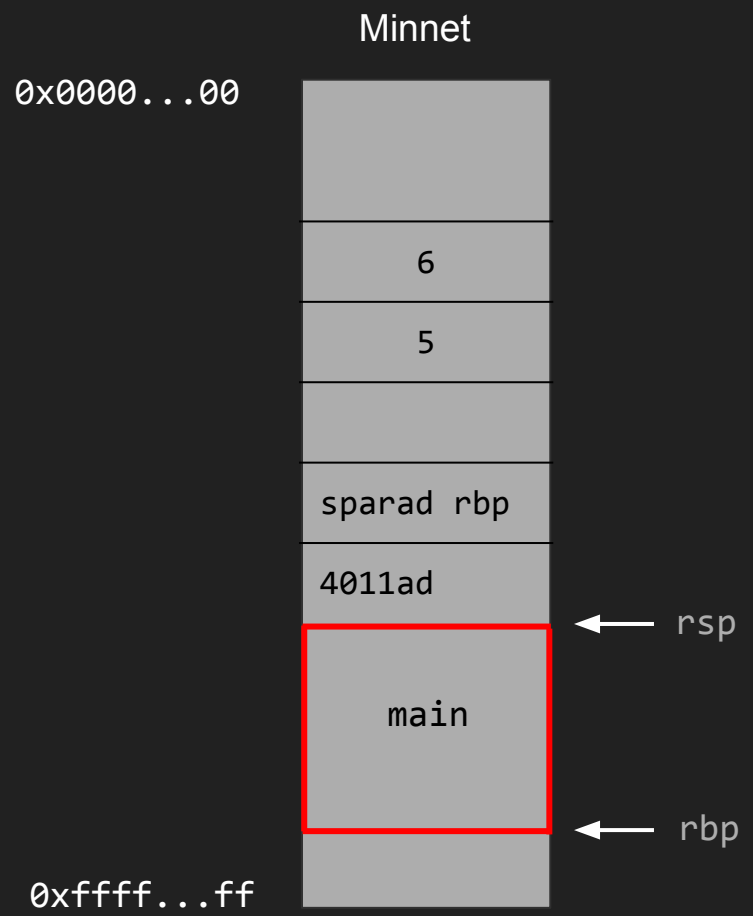
Ett anrop i detalj

```
main:
 ...
 40119e: mov    eax,DWORD PTR [rbp-0x4]
 4011a1: mov    esi,0x6
 4011a6: mov    edi,eax
 4011a8: call   40114e <print>
 4011ad: ...

print:
 40114e: endbr64
 401152: push   rbp
 401153: mov    rbp,rsp
 401156: sub    rsp,0x20
 40115a: mov    DWORD PTR [rbp-0x14],edi
 40115d: mov    DWORD PTR [rbp-0x18],esi
 ...
 401189: leave
 40118a: ret
```



Ett anrop i detalj

```
main:
 ...
 40119e: mov    eax,DWORD PTR [rbp-0x4]
 4011a1: mov    esi,0x6
 4011a6: mov    edi,eax
 4011a8: call   40114e <print>
 4011ad: ...

print:
 40114e: endbr64
 401152: push   rbp
 401153: mov    rbp,rsp
 401156: sub    rsp,0x20
 40115a: mov    DWORD PTR [rbp-0x14],edi
 40115d: mov    DWORD PTR [rbp-0x18],esi
 ...
 401189: leave
 40118a: ret
```



Syscalls

Syscalls - eller, hur saker händer

- Hur kan man skriva till skärmen?
- Hur skriver `printf` till skärmen?
- Syscalls!
- Man ger kommandon till OS:et
- Finns många olika:
  - Skriv till skärmen
  - Hantera filer
  - Nätverk
  - Andra processer
  - …
- [Lista över syscalls]

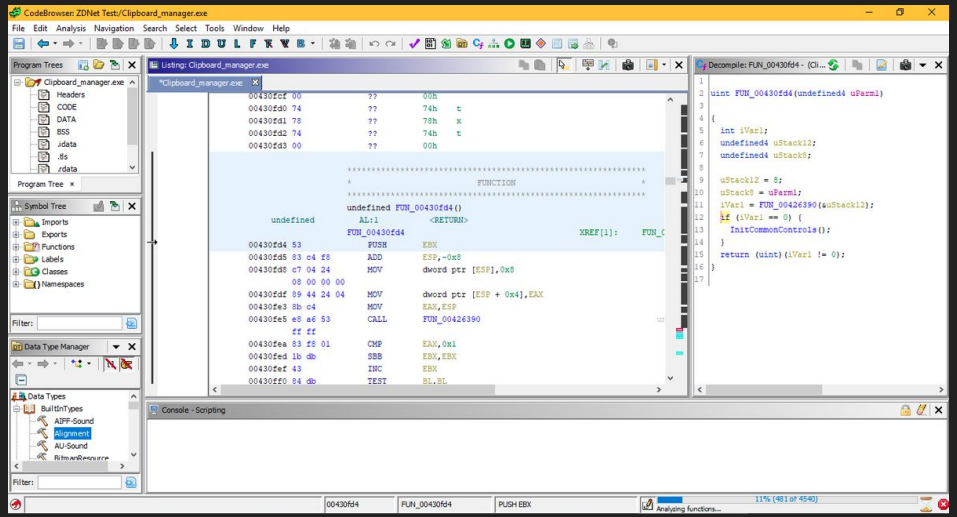
```
; write
401000: mov    eax,0x1
401005: mov    edi,0x1
40100a: movabs rsi,0x402000
401014: mov    edx,0x15
401019: syscall

; exit
40101b: mov    eax,0x3c
401020: xor    rdi,rdi
401023: syscall
```

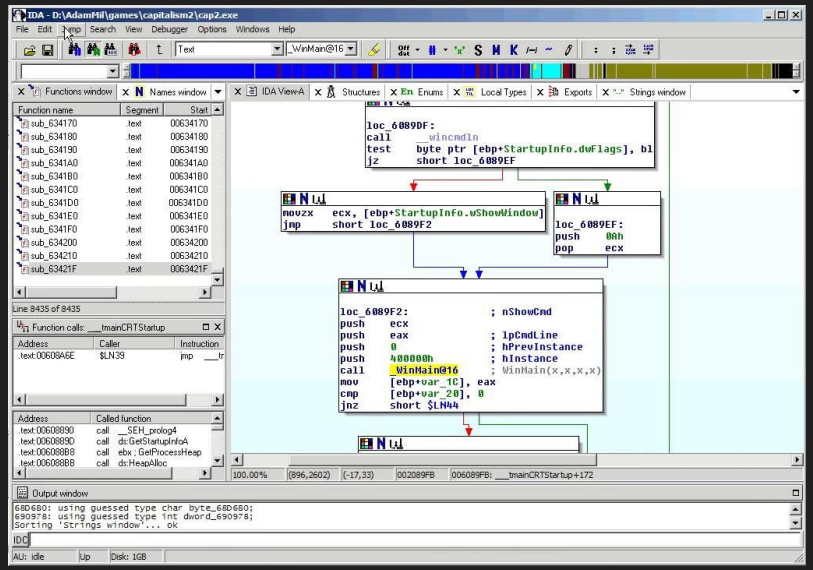
Verktyg

objdump - en disassembler (bland annat)

```
$ gcc main.c -o program
$ objdump -d -M intel ./program
...
000000000040118b <main>:
  40118b:    f3 0f 1e fa          endbr64
  40118f:    55                   push   rbp
  401190:    48 89 e5             mov rbp,rsp
  401193:    48 83 ec 10          sub rsp,0x10
  401197:    c7 45 fc 05 00 00 00 mov DWORD PTR [rbp-0x4],0x5
  40119e:    8b 45 fc             mov eax,DWORD PTR [rbp-0x4]
  4011a1:    be 06 00 00 00       mov esi,0x6
  4011a6:    89 c7                mov edi,eax
  4011a8:    e8 a1 ff ff ff       call   40114e <print>
...
```

Bättre disassemblers

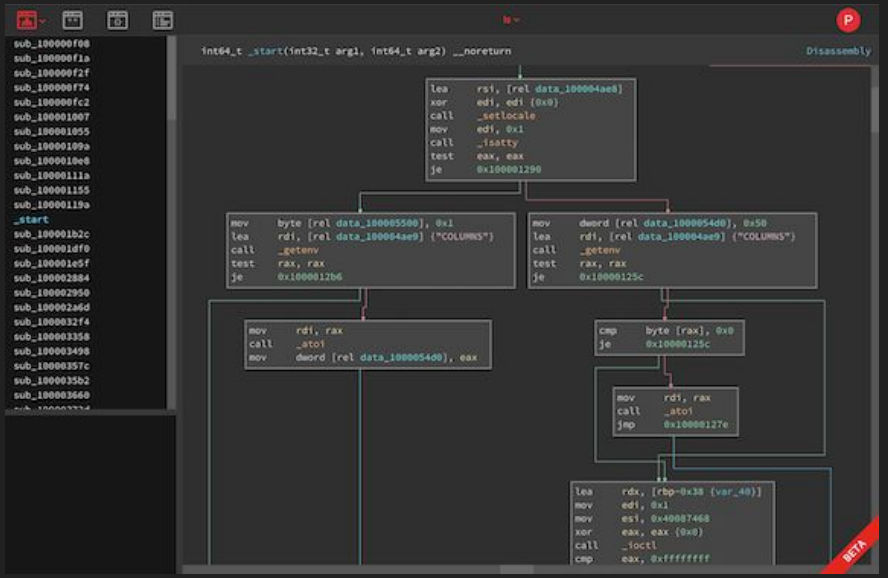


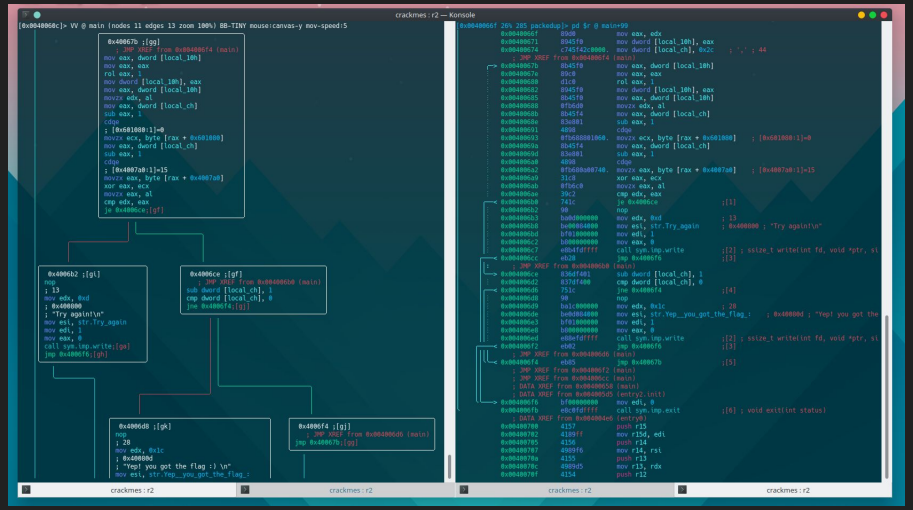


Ghidra

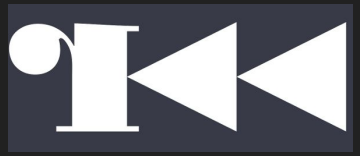
IDA

Bättre disassemblers





[Binary Ninja]



[radare2]

godbolt

C++ source #1

C++

```cpp
// Type your code here, or load an exampl
int square(int num) {
    int x = num;
    return x * num;
}
```

x86-64 gcc 10.2 (Editor #1, Compiler #1) C++

x86-64 gcc 10.2

Compiler options...

```asm
square(int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-20], edi
        mov     eax, DWORD PTR [rbp-20]
        mov     DWORD PTR [rbp-4], eax
        mov     eax, DWORD PTR [rbp-4]
        imul    eax, DWORD PTR [rbp-20]
        pop     rbp
        ret
```

godbolt.org

gdb - GNU debugger

```
$ gcc main.c -o program
$ gdb ./program
(gdb) b main
Breakpoint 1 at 0x40118b
(gdb) r
Starting program: /home/mkg/program

Breakpoint 1, 0x000000000040118b in main ()
(gdb) info registers
rax            0x40118b           4198795
rbx            0x4011c0           4198848
rcx            0x4011c0           4198848
rdx            0x7fffffffe338     140737488347960
...
```

gdb - Några kommandon

Kommando	Beskrivning
`run` eller `r`	Startar programmet
`b *0x40118b`	Sätter breakpoint vid adress
`b main`	Sätter breakpoint vid funktion
`continue` eller `c`	Fortsätter till nästa breakpoint
`backtrace` eller `bt`	Visar ett stacktrace
`si`	“Step instruction”
`ni`	“Next instruction”, hoppar över `call`
`info registers` eller `i r`	Skriver ut alla register

gdb - Några kommandon

Kommando	Beskrivning
`i r rax`	
`set $rax = 5`	
`quit` eller `q`	Gå ur gdb
`x/4gx $rsp`	Visa 4 stycken 64-bitarstal i hex som ligger vid `rsp`
`x/20i $rip`	Visa 20 instruktioner från `rip` och framåt
`x/s $rbx - 0x20`	Visa strängen som ligger vid `rbx - 0x20`

[Många fler sätt att använda `x/…`]

Tips: installera [pwndbg]

Vidare läsning / Saker jag inte hann ta upp

- Symboler
- Minnessektioner
- Virtuellt minne
- Heapen
- Disassembly vs. dekompilering
- …
- Säkert många mer saker

Frågor?

Slides: bit.ly/spooktober-2020-intro-assembly