Foundations of Cybersecurity
C and C++ Secure Coding

Gianluca Dini
Dept. of Information Engineering
University of Pisa
Email: gianluca.dini@unipi.it
Version: 2022-03-07

1

Credits

- These slides come from a version originally produced by Dr. Pericle Perazzo

Mar-22 Secure coding - strings 2

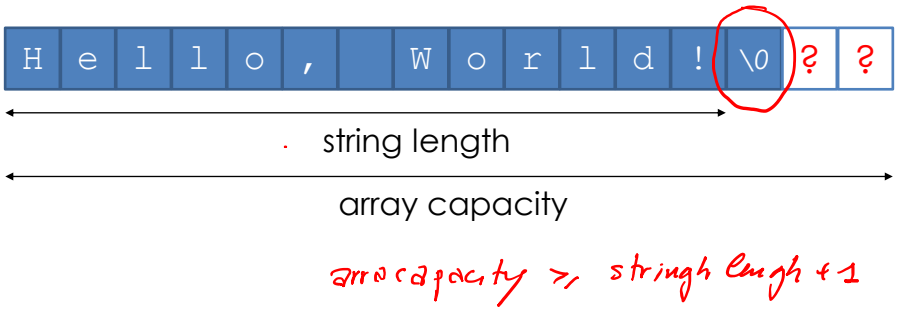
2

C and C++ Secure Coding

STRINGS

3

Strings



array capacity >= strings length + 1

4

gets()

```
void func() {
  char buf[1024];
  if (gets(buf) == NULL) {
    /* Handle error */
  }
}
```

* all code snippets taken or elaborated from: Robert C. Seacord, «Secure Coding in C and C++ (2° ed.)», Addison-Wesley

is this vulnerability exploitable?

in most cases, NO

5

gets()

- User attacking process is the same that lanched it → Attacker is attacking herself
- Dangerous cases:
  - Standard input redirected to untrusted source (e.g., socket)
  - Program used by anonymous users (e.g., public computers)
- Should we leave it unfixed?
  - A bug that seems not to be exploitable could become exploitable in future (program changes, system reconfigurations, etc.)
- You cannot leave an armed bomb in your system!

6

gets()

UNIVERSITÀ DI PISA

- User types:

Hello, World!*<enter>*

- gets(char* s):

H	e	l	l	o	,		W	o	r	l	d	!	\0	?	?

- fgets(char *s, int size, FILE *stream):

H	e	l	l	o	,		W	o	r	l	d	!	\n	\0	?

!!!

7

gets()

UNIVERSITÀ DI PISA

```
void func() {
  char buf[1024];
  if (fgets(buf, 1024, stdin) == NULL) {
    /* Handle error */
  }
  char* p = strchr(buf, '\n');
  if (p) { *p = '\0'; }
}
```

must be >0

```
void func() {
  char buf[1024];
  if (gets_s(buf, 1024) == NULL) {
    /* Handle error */
  }
}
```

NOT AVAILABLE IN SOME COMPILERS!

8

gets()

```
void func() {
  std::string buf;
  std::getline(std::cin, buf);
  if (!std::cin) {
    /* Handle error */
  }
}
```

9

gets()

```
#include <iostream>
#include <string>

int main(void) {
  std::string buf;
  std::cin >> buf;
  if (!std::cin) {
    // Handle error
  }
}
```

10

C and C++ secure coding

FORMATTED OUTPUT AND VARIADIC FUNCTIONS

11

Variadic Functions

UNIVERSITÀ DI PISA

- Number/type of args determined at runtime from a format string
- Output:

```
int a = 10, b = 20;
char s[] = "Hello, world!";

printf("a is: %d, b is: %d, s is: %s\n", a, b, s);
```

format string

%d = signed int

%s = C-string

```
a is: 10, b is: 20, s is: Hello, world!
```


12

Variadic Functions

UNIVERSITÀ DI PISA

- Some common format specifiers:

Format specifier	Meaning	Example
%i	Integer in decimal digits	12
%x	Unsigned integer in hexadecimal digits	3f
%f	Floating-point number	3.14
%c	Single character	a
%s	C-string	Hello
%%	Literal '%'	%

Mar-22 Secure coding - strings 13

13

Variadic Functions

UNIVERSITÀ DI PISA

- Input:

```
int a, b;
float c;
int ret;
if (scanf("%d %d %f", &a, &b, &c) != 3) {
 /* Handle error */
}
```

%d = signed int %f = float

```
30  40  3.14
```

a b c

Mar-22 Secure coding - strings 14

14

scanf() + String Argument

UNIVERSITÀ DI PISA

```
void func() {
  char buf[1024];
  if (scanf("%s", buf) != 1) {
    /* Handle error */
  }
}
```

Mar-22 Secure coding - strings 15

15

scanf() + String Argument

UNIVERSITÀ DI PISA

```
void func() {
  char buf[1024];
  if (scanf("%1023s", buf) != 1) {
    /* Handle error */
  }
}
```

ERROR-PRONE!

```
void func() {
  std::string buf;
  std::cin >> buf;
  if (!std::cin) {
    /* Handle error */
  }
}
```

Mar-22 Secure coding - strings

16

sprintf()

```
void func(const char *name) {
  char filename[128];
  sprintf(filename, "%s.txt", name);
}
```

17

sprintf()

```
void func(const char *name) {
  char filename[128];
  sprintf(filename, "%.123s.txt", name);
}
```

```
void func(const char *name) {
  std::string filename = (std::string)name + ".txt";
}
```


18

strcpy()

```
void func(const char* str) {
  char str2[128];
  strcpy(str2, str);
  /* … */
}
```

19

strcpy()

```
void func(const char* str) {
  char str2[128];
  strncpy(str2, str, 128);
  str2[127] = '\0';
  /* … */
}
```

MANUALLY PUT TERMINATOR!

20

C and C++ Secure Coding

FORMAT STRINGS

21

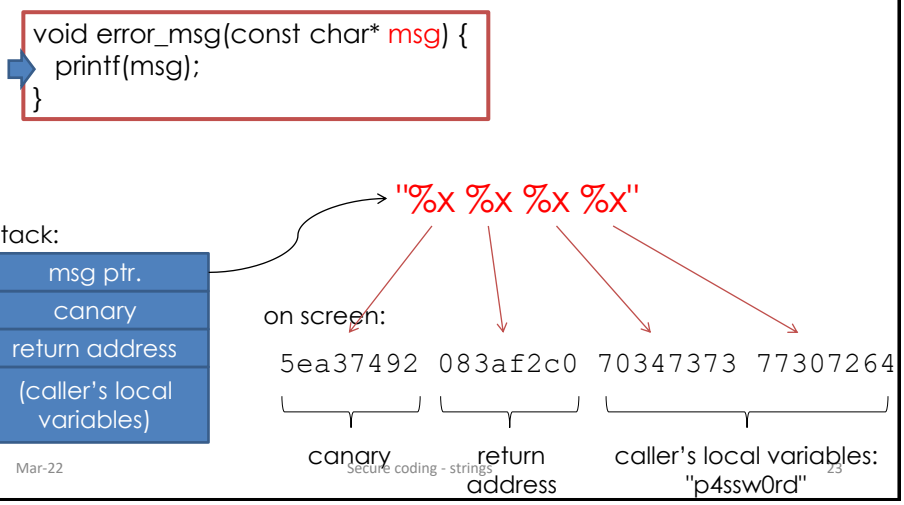
Catch the bug!

UNIVERSITÀ DI PISA

```
void error_msg(const char* msg) {
   printf(msg);
}
```

22

Tainted Format String



23

Tainted Format String

- Other advanced techniques allow an attacker to write on arbitrary memory locations → Integrity vulnerability!

24

Tainted Format String

UNIVERSITÀ DI PISA

```
void error_msg(const char* msg) {
   printf("%s", msg);
}
```

Mar-22 Secure coding - strings 25

25

Tainted Format String

UNIVERSITÀ DI PISA

```
void func(const char* str) {
  if (strlen(str) > 20) { /* Handle error */ }
  char buf[100];
  sprintf(buf, str);
}
```

BUFFER OVERFLOW!

```
void func(const char* str) {
  if (strlen(str) > 20) { /* Handle error */ }
  char buf[100];
  sprintf(buf, "%s", str);
}
```

Mar-22 Secure coding - strings

26