INTERNET OF THINGS MASTER

SOFTWARE DEPENDABILITY

University of Salerno

Computer Science

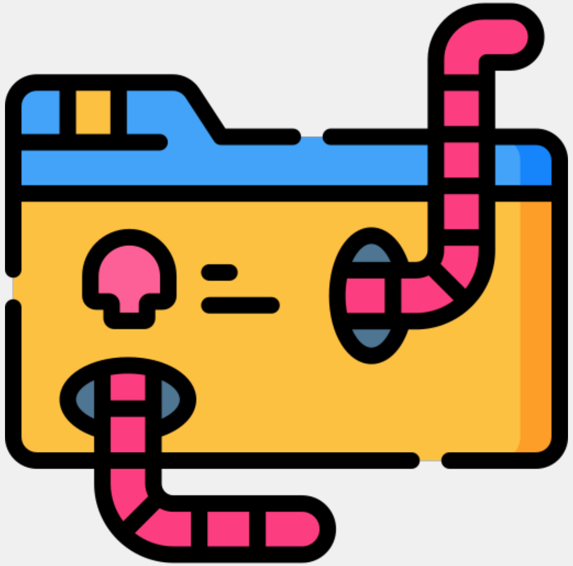
Vincenzo Gallicchio, 0522500642

Nicola Volpe, 052250654

SOFTWARE VULNERABILITIES

Software vulnerabilities are one of the main causes of security incidents in computer systems

They arise from deficiencies in the design of computer programs or mistakes in their implementation.



COMMON APPROACHES

1 **Manual Source Code Analysis**

this method involves reading the source code of a program and looking for security problems

2 **Fuzz Testing & RunTime Checking**

This method works by feeding invalid or malformed data to a program and monitoring its responses.
Inserting additional checks into a program to ensure that its behavior conforms to a certain set of restrictions

3 **Static Source Code Analysis (Automatic tools)**

Using a tool for code Analysis

VULNERABILITIES – BUFFER OVERFLOWS

Classic string buffer overflow

The classic buffer overflow is a result of misuse of string manipulation functions in the standard C library.

The programmer uses out of range memory zones!

```
char dst[256];
char* s = read_string ();
strcpy (dst, s);
```

If the length of the user string is greater than 256, the strcpy() function will write data past then end of the dst[ ] array.

VULNERABILITIES – BUFFER OVERFLOWS

Memcpy buffer overflow

The memcpy buffer overflow is a result of no input validation performed before its use, for the memcpy() function, as third parameter.

The programmer uses out of range memory zones!

```
char src[10], dst[10];
char n = read_int ();
memcpy (dst, src, n);
```

If the value of n is bigger than the size of the destination buffer, the memcpy() function will overwrite data past the end of the buffer.

VULNERABILITIES – BUFFER OVERFLOWS

Out of bounds array access

In this case no input validation is performed before it is used as an array index. The programmer uses out of range memory zones!

```
char s[256];
int n = read_int ();
s[n] = '\0';
```

By choosing a specific value of n, the attacker can access any memory location on the system. This can give the attacker the ability to read or write an arbitrary memory location.

VULNERABILITIES – BUFFER OVERFLOWS

Scanf buffer overflow

A function of the scanf family is used to read user input into a fixed size buffer. If the “%s” format specifier is used, scanf will read user input until either whitespace or the end of file is reached.

```
char s[256];
scanf(“%s”, s);
```

By supplying more data than the buffer can hold, an attacker can overwrite data past the end of the array.

VULNERABILITIES – FORMAT STRING BUGS

Format string bug

Data received in input is passed as a format string argument to one of the output formatting functions in the standard C library.
Some of these directives allow for writing to a memory location specified in the format string.

```
char* s = read_string ();
printf(s);
```

An attack could use format specifiers (“%n”) to write to memory locations!

VULNERABILITIES – INTEGER OVERFLOWS

Signed to unsigned conversion bug

Sign conversion bugs occur when a signed integer is converted to an unsigned integer. In many cases a small negative number, when converted to an unsigned integer, will become a very large positive number.

```
char src[10], dst[10];
int n = read_int();
if( n <= sizeof(dst))
    memcpy(dst, src, n);
```

The memcpy() function could overwrite data past the end of the destination folder!

VULNERABILITIES – INTEGER OVERFLOWS

Arithmetic overflow when allocating an array

Arithmetic overflows occur when a value larger than the maximum integer size is stored in an integer variable.

```
int n = read_int();
void *ptr = malloc(sizeof(struct s) * n);
```

By choosing a specific large value of n, the attacker can cause an arithmetic overflow and allocate 0 bytes for the array. Any subsequent use of the array will access data which has not been allocated!

VULNERABILITIES – MISCELLANOUS

There are vulnerabilities which arise from very complicated data dependencies and cannot be classified in one of the previous categories.

For example, the vulnerability in the ProFTPD server is caused by code that modifies the size of a buffer allocated by another function. The function proceeds to write to the buffer using its old size and overwrites unallocated memory.

CONCLUSION

All vulnerabilities in these classes are a result of an execution path with the following three characteristics:

- Data is read from an untrusted source.
- Untrusted data is insufficiently validated.
- Untrusted data is used in a potentially vulnerable function or a language construct.

A system that does not allow user input of any kind would be free of software vulnerabilities, because it would always perform as designed. An attacker would have no way to influence the execution of the program. Moreover, most computer programs are designed to process user data. Since user interaction is necessary in these programs, they must be regarded as potentially vulnerable.

INTERNET OF THINGS MASTER

SOFTWARE DEPENDABILITY

University of Salerno

Computer Science

Vincenzo Gallicchio, 0522500642

Nicola Volpe, 052250654