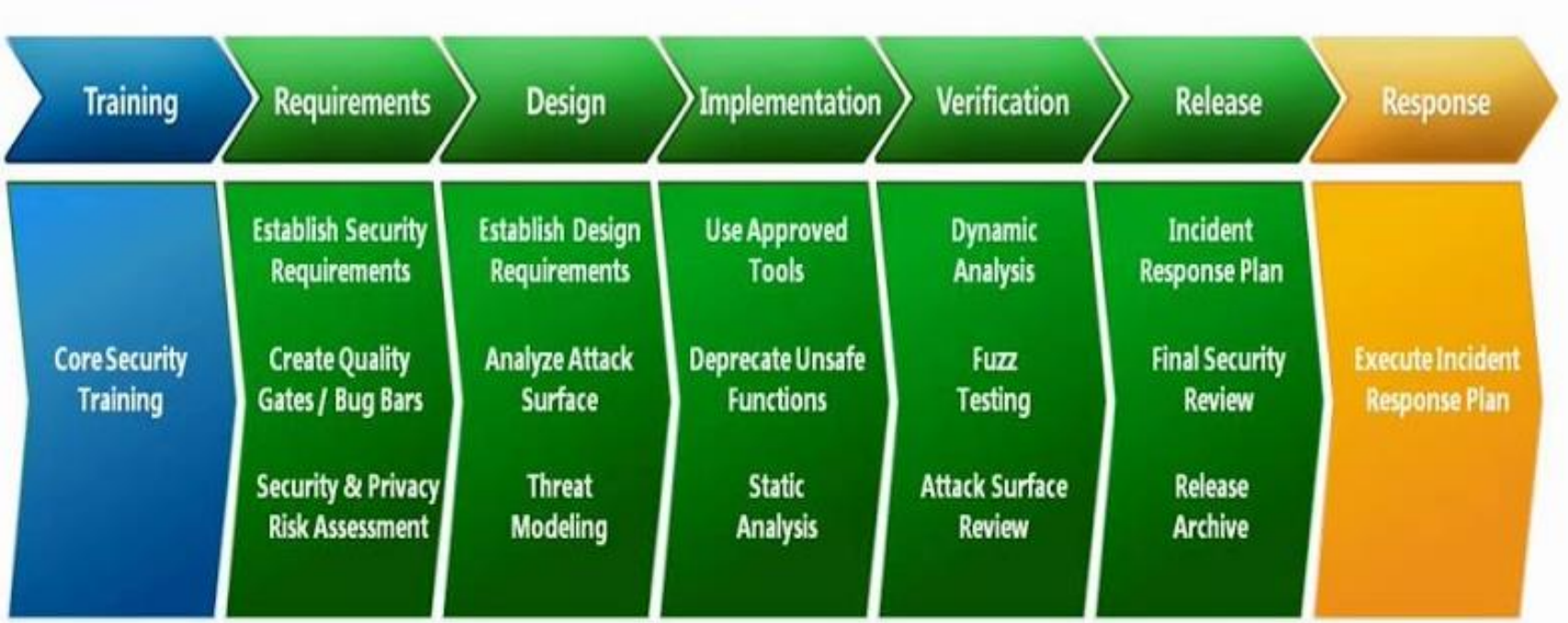
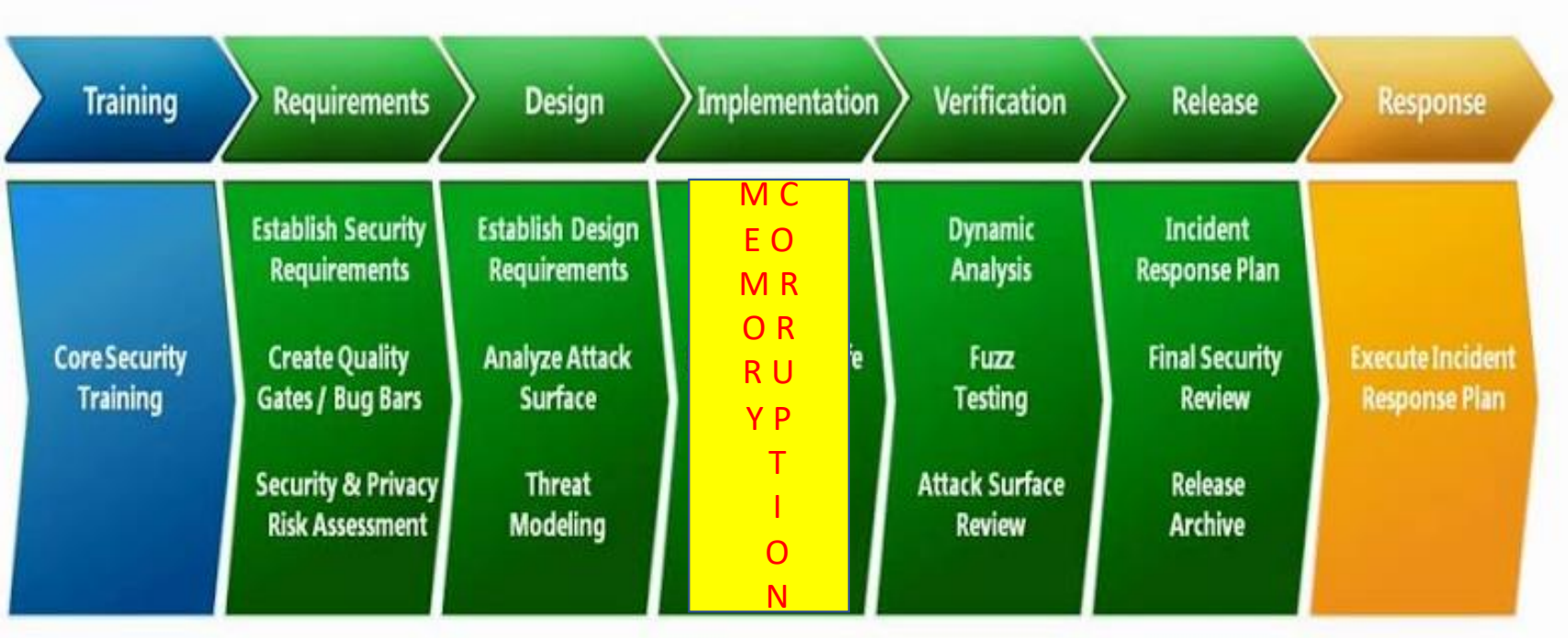

Security
SW LIFE CICLE

SECURITY IN THE SW LIFE CYCLE



SECURITY IN THE SW LIFE CYCLE
MEMORY CORRUPTION



The issues

usare la toolchain dei linguaggi di programmazione per vedere se ci sono problemi di memory corruption prima di mandare il programma in esecuzione, è ciò che avviene nel backend dei compilatori

How do memory corruption flaws work?

What can be the impact?

How can we spot such problems in programming languages

- **Static analysis**
- **Dynamic analysis**

*

Ghat can the platform do about it?

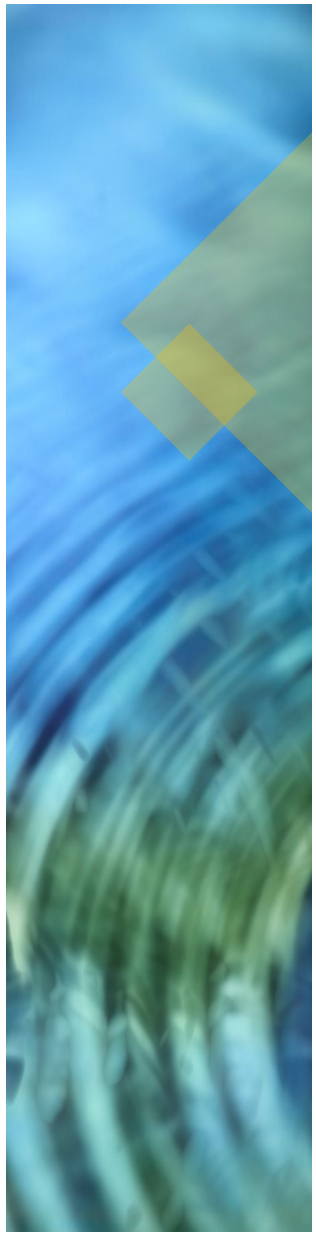
- **the loanguage tootchian ..**

What can the programmer do about it?

*

nel run time del linguaggio di affrontare le problematiche

si possono usare delle nuove piattaforme di toolchain per linguaggi di programmazione che affrontano il problema dall'inizio, cioè dalla progettazione del linguaggio di programmazione fino all'implementazione tenendo conto delle problematiche



The essence of the problem

Suppose in a C program you have declared an array of length 4

char buffer[4];

What happens if the statement below is executed?

buffer[4] = 'a';

Whn this statement is executed: ANYTHING can happen

può succedere qualsiasi cosa in quanto linguaggi come il C non affrontano in modo corretto

Anything can happen

Suppose in a C program you have declared an array of length 4

char buffer[4];

What happens if the statement below is executed?

buffer[4] = 'a';

If the attacker can control the value 'a'
then anything that the attacker wants may happen

If you are lucky, you only get a SEGMENTATION FAULT:
you'll know that something went wrong

If you are unlucky, there is remote code execution (RCE) and you won't know

Anything can happen

Suppose in a C program you have declared an array of length 4

char buffer[4];

What happens if the statement below is executed?

buffer[4] = 'a';

The compiler could remove the statement above, ie. Replace the statement wi
Skip statement (do nothing)
This would be correct compilation by the C standard because anything includes

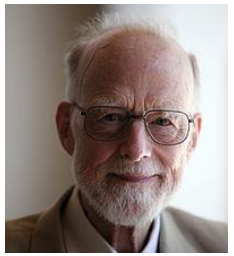
This may be unexpected, but compilers actually do this (as part of optimisation

sostanzialmente questo è quello che succede e che viene fatto dai compilatori come parte dell'ottimizzazione

Language-based solution to this problem

- Check array bounds at runtime
  - Algol 60 proposed this back in 1960!
- C and C++ have not adopted this solution
  - For efficiency
  - People often choose performance over security
- **Buffer overflows have been the no 1 security problem in software.**
- **Perl, Python, Java, C#, PHP, Javascript, and Visual Basic do check array bounds**

Tony Hoare on design principles of ALGOL 60



In his Turing Award lecture in 1980

> "The first principle was *security*: ... every subscript was checked at run time against both the upper and the lower declared bounds of the array. Many years later we asked our customers whether they wished an option to switch off these checks in the interests of efficiency. Unanimously, they urged us not to - they knew how frequently subscript errors occur on production runs where failure to detect them could be disastrous.
>
> I note with fear and horror that even in 1980, language designers and users have not learned this lesson. In any respectable branch of engineering, failure to observe such elementary precautions would have long been against the law."

[C.A.R. Hoare, The Emperor's Old Clothes, Communications of the ACM, 1980]

Applications written in programming languages like C or C++ are prone to *Memory Corruption* bugs .

The lack of memory safety (or type safety) in such languages enables attackers to exploit memory bugs by maliciously altering the program's behavior or even taking full control over the control-flow.

Today

- Discuss memory corruption attacks
- Identify different evaluate and compare proposed solutions for performance, compatibility, and robustness;
- Discuss why many proposed solutions are not adopted in practice and what the necessary criteria for a new solution are.

Buffer overflow

- The most common security problem in (machine code compiled from) C and C++
  - since the first Morris Worm in 1988
- Attacks are getting now cleverer, defeating ever better countermeasures

CVE example: https://cve.mitre.org/cgibin/cvekey.cgi?keyword=buffer

Common Vulnerabilities and Exposures

CVE List▾ CNAs▾ WGs▾ Board▾ About▾ News & Blog▾

NVD
Go to for:
CVSS Scores
CPE Info

Search CVE List | Downloads | Data Feeds | Update a CVE Record | Request CVE IDs

TOTAL CVE Records: 149102

HOME > CVE > SEARCH RESULTS

Search Results

There are **13368** CVE Records that match your search.

Name	Description
CVE-2021-3382	Stack buffer overflow vulnerability in gitea 1.9.0 through 1.13.1 allows remote attackers to cause a denial of service (crash) via vectors related to a file path.
CVE-2021-3345	_gcry_md_block_write in cipher/hash-common.c in Libgcrypt version 1.9.0 has a heap-based buffer overflow when the digest final function sets a large count value. It is recommended to upgrade to 1.9.1 or later.
CVE-2021-3304	Sagemcom F@ST 3686 v2 3.495 devices have a buffer overflow via a long sessionKey to the goform/login URI.
CVE-2021-3272	jp2_decode in jp2/jp2_dec.c in libjasper in JasPer 2.0.24 has a heap-based buffer over-read when there is an invalid relationship between the number of channels and the number of image components.
CVE-2021-3182	** UNSUPPORTED WHEN ASSIGNED ** D-Link DCS-5220 devices have a buffer overflow. NOTE: This vulnerability only affects products that are no longer supported by the maintainer.
CVE-2021-3177	Python 3.x through 3.9.1 has a buffer overflow in PyCArg_repr in _ctypes/callproc.c, which may lead to remote code execution in certain Python applications that accept floating-point numbers as untrusted input, as demonstrated by a 1e300 argument to c_double.from_param. This occurs because sprintf is used unsafely.
CVE-2021-3156	Sudo before 1.9.5p2 has a Heap-based Buffer Overflow, allowing privilege escalation to root via "sudoedit -s" and a command-line argument that ends with a single backslash character.
CVE-2021-26825	An integer overflow issue exists in Godot Engine up to v3.2 that can be triggered when loading specially crafted.TGA image files. The vulnerability exists in ImageLoaderTGA::load_image() function at line: const size_t buffer_size = (tga_header.image_width * tga_header.image_height) * pixel_size; The bug leads to Dynamic stack buffer overflow. Depending on the context of the application, attack vector can be local or remote, and can lead to code execution and/or system

Other memory corruption problems

Errors with pointers and with dynamic memory (the heap)

- C(++) programs use pointers
- C(++) programs uses dynamic memory, ie. malloc & free

In C/C++, the programmer is responsible for memory management, and this is very error-prone

In technical term: C and C++ do not offer memory-safety

Example

```
int main()  {
 int* p1, p2;
 int n = 30;
 p1 = &n;
 p2 = &n; // error Why?
}
```

Questo codice dichiara un puntatore a un intero p1 e un intero p2. Quando vado a referenziare p2 e voglio ottenere un puntatore al valore di n ottengo un errore.

Example

```
int main()  {
 int* p1, p2;  //It declares an integer pointer p1 and an integer p2.
 int n = 30;
 p1 = &n;
 p2 = &n; // error Why?
}
```

Example

```
int main()  {
 int* p1, p2;
 int n = 30;
 p1 = &n;
 p2 = &n; // error Why?
}
```

FIX

Use the following declaration to declare two pointers of the same type:

```
int *p1, *p2;
```

Alternatively, use a typedef –

```
typedef int* Pint;
```

and then, use this type when declaraing pointers:

```
Pint p1, p2;
```

Example

```
int main() {
  char* userSecurityQuestion = (char*)malloc(strlen("First Pet?") + 1);
   strcpy_s(userSecurityQuestion, strlen("First Pet?") + 1, "First Pet?");
//...
// Done with processing security question - stored in secured db etc.

   free(userSecurityQuestion);
}
```

mando al gestore della memoria il blocco di memoria che conteneva la risposta alla domanda di sicurezza. Facendo la free non è che vado a modificare il valore, il valore è ancora quello vecchio dunque un attaccante quello che potrebbe fare è andare a vedere il contenuto. Dunque quello che andrebbe fatto prima di liberare la memoria è fare un operazione di modifica del contenuto per evitare che il blocco di memoria che viene restituito al gestore della memoria sull'heap non contenga informazioni importanti

When an application terminates, most operating systems do not zero out or erase the heap memory that was in use by your application. The memory blocks used by your application can be allocated to another program, which can use the contents of non-zeroed out memory blocks.

Example

```
int main() {
  char* userSecurityQuestion = (char*)malloc(strlen("First Pet?") + 1);
   strcpy_s(userSecurityQuestion, strlen("First Pet?") + 1, "First Pet?");
//...
// Done with processing security question - stored in secured db etc.
   memset(userSecurityQuestion, 0, sizeof(userSecurityQuestion));
   free(userSecurityQuestion);
}
```

it's always a good idea to erase that memory block contents before returning the memory to the heap via free().

Se uno avesse una toolchain che prima di restituire fa l'operazione di modifica sarebbe molto più strutturato l'approccio al problema .Inoltre se fosse il sistema a gestire queste operazioni senza lasciare al programmatore l'onere di effettuare questo tipo di operazione di free sarebbe più sicuro.