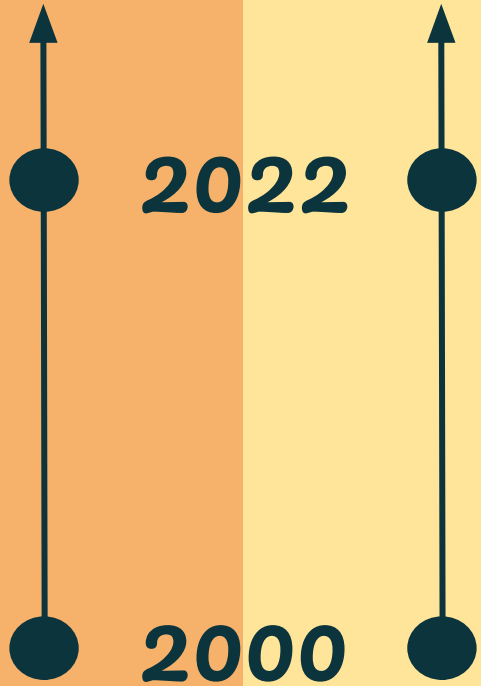

THE COLOR CHILDREN
PROTECT NATURE
TORTUGA
@PATI_GALLARDO
Turtle Sec


TurtleSec
Living in the future
@PATI_GALLARDO
2



Systems Programming

Binary Exploitation

Boomers: Y2K

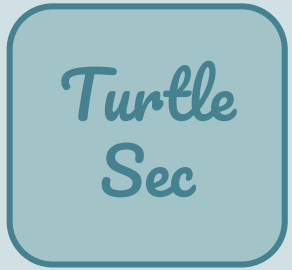
Zoomers: Taylor Swift was 11

Classic Vulnerabilities

ACCU 2022

Patricia Aas

@pati_gallardo



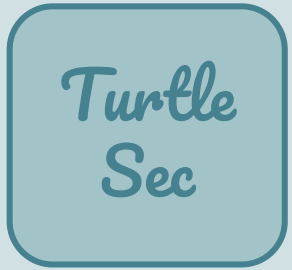
Patricia Aas - Trainer & Consultant

C++ Programmer, Application Security
Currently : **TurtleSec**

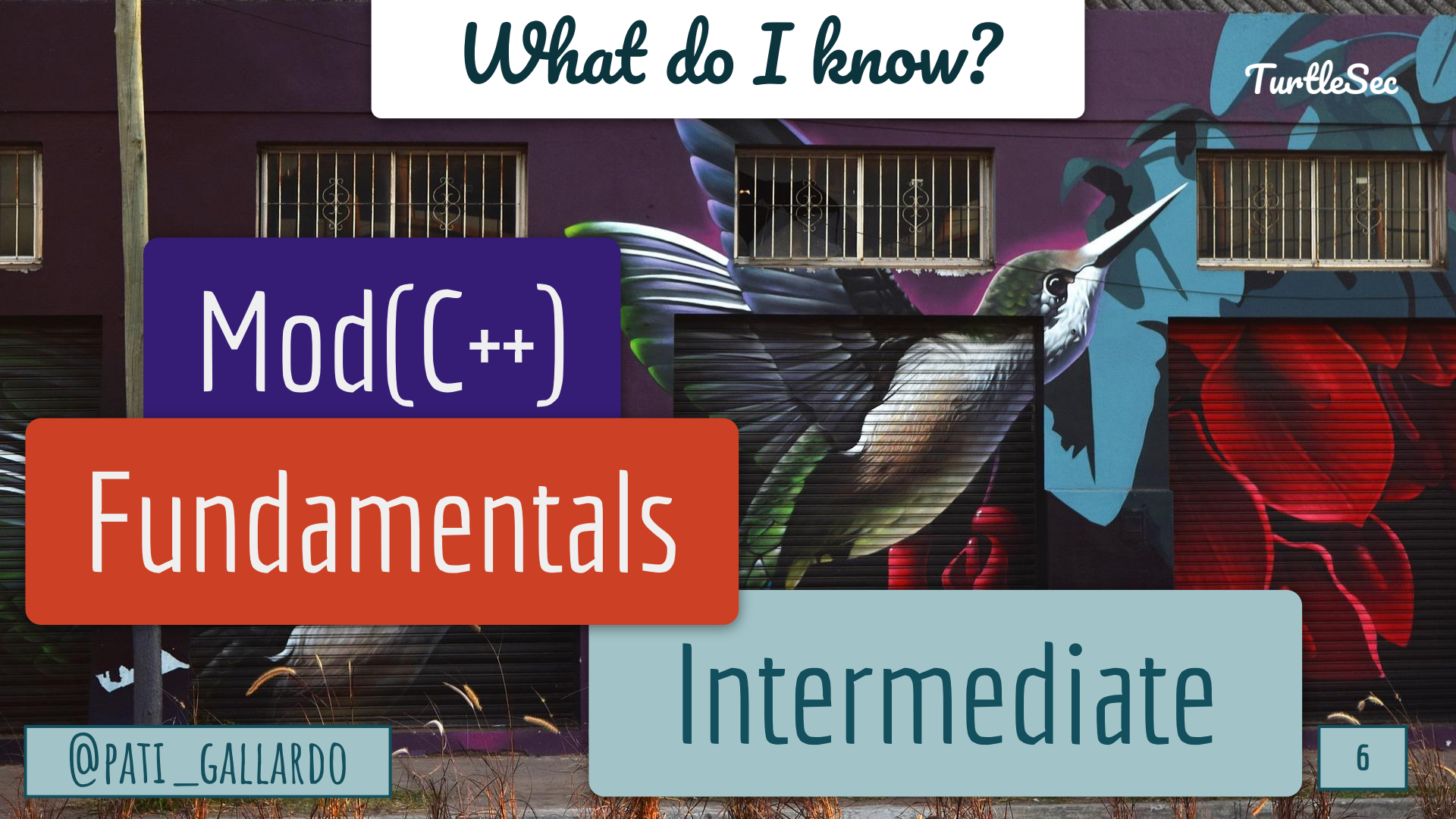
Previously : Vivaldi, Cisco Systems, Knowit, Opera Software
Master in Computer Science

Pronouns: she/they

@PATI_GALLARDO



What do I know?

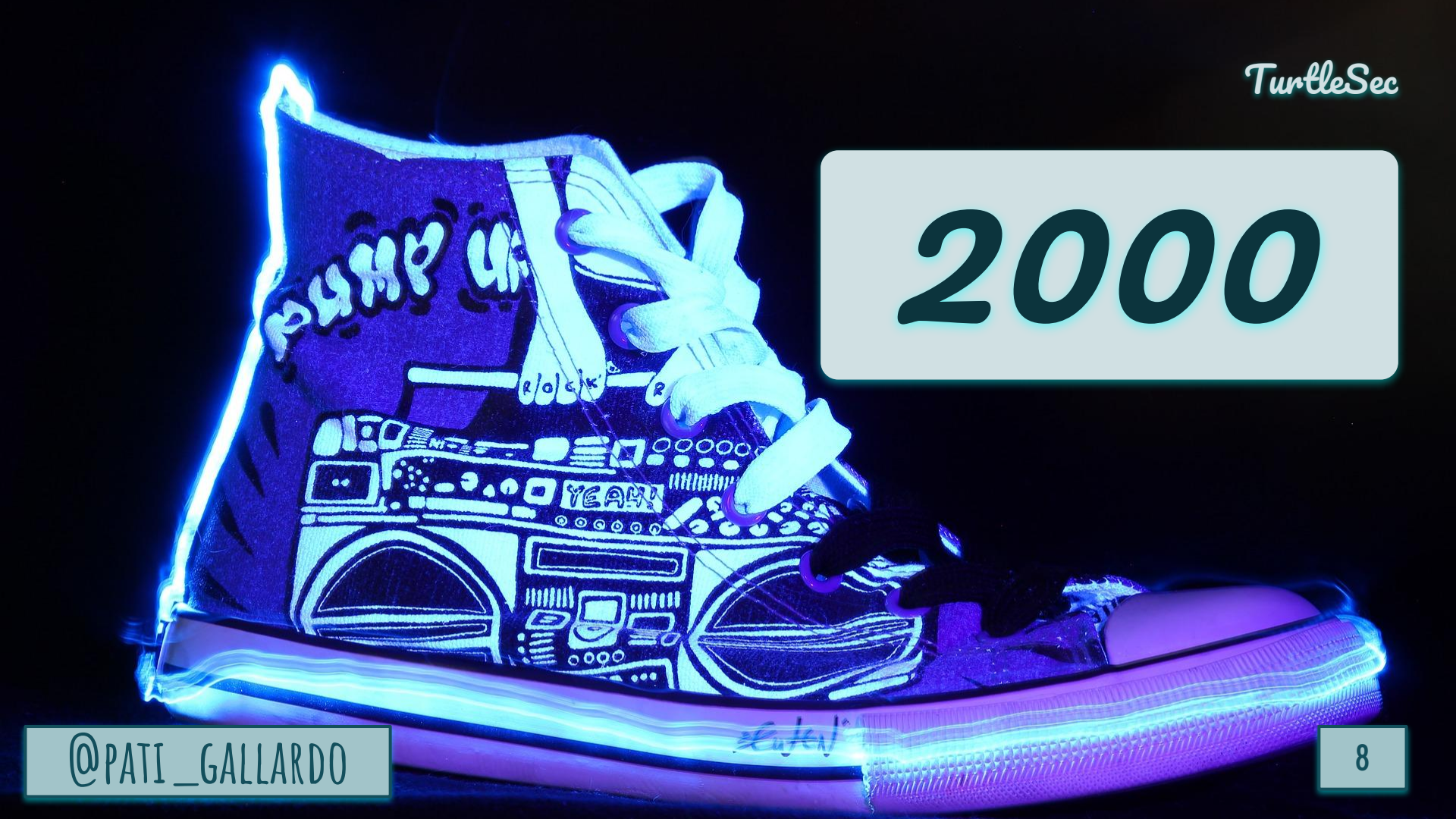


What do I know?

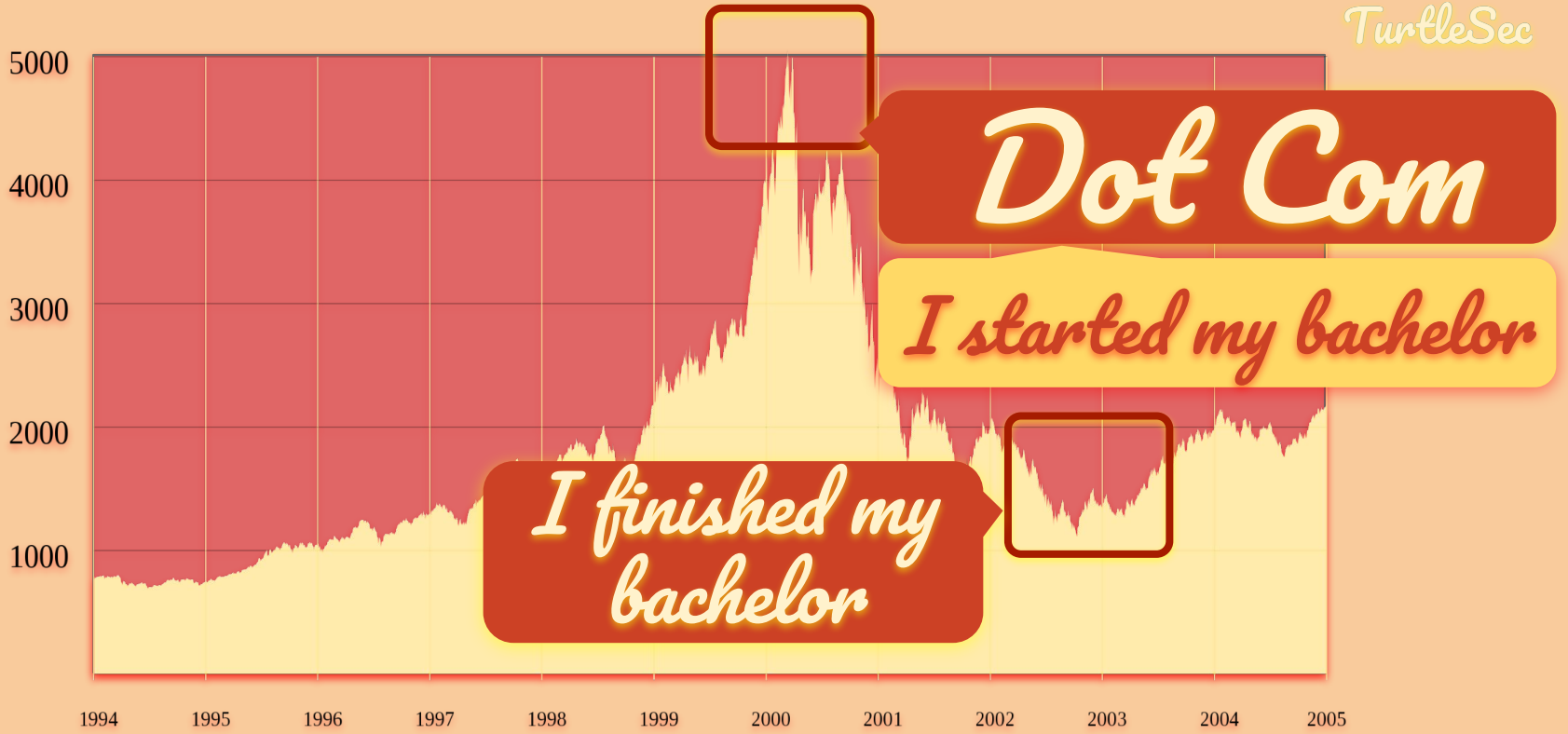
TurtleSec

(In)Secure C++

@PATI_GALLARDO



2000


TurtleSec
5000
4000
3000
2000
1000
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
Dot Com
I started my bachelor
I finished my bachelor

2000

2000 : 22 years ago

TurtleSec

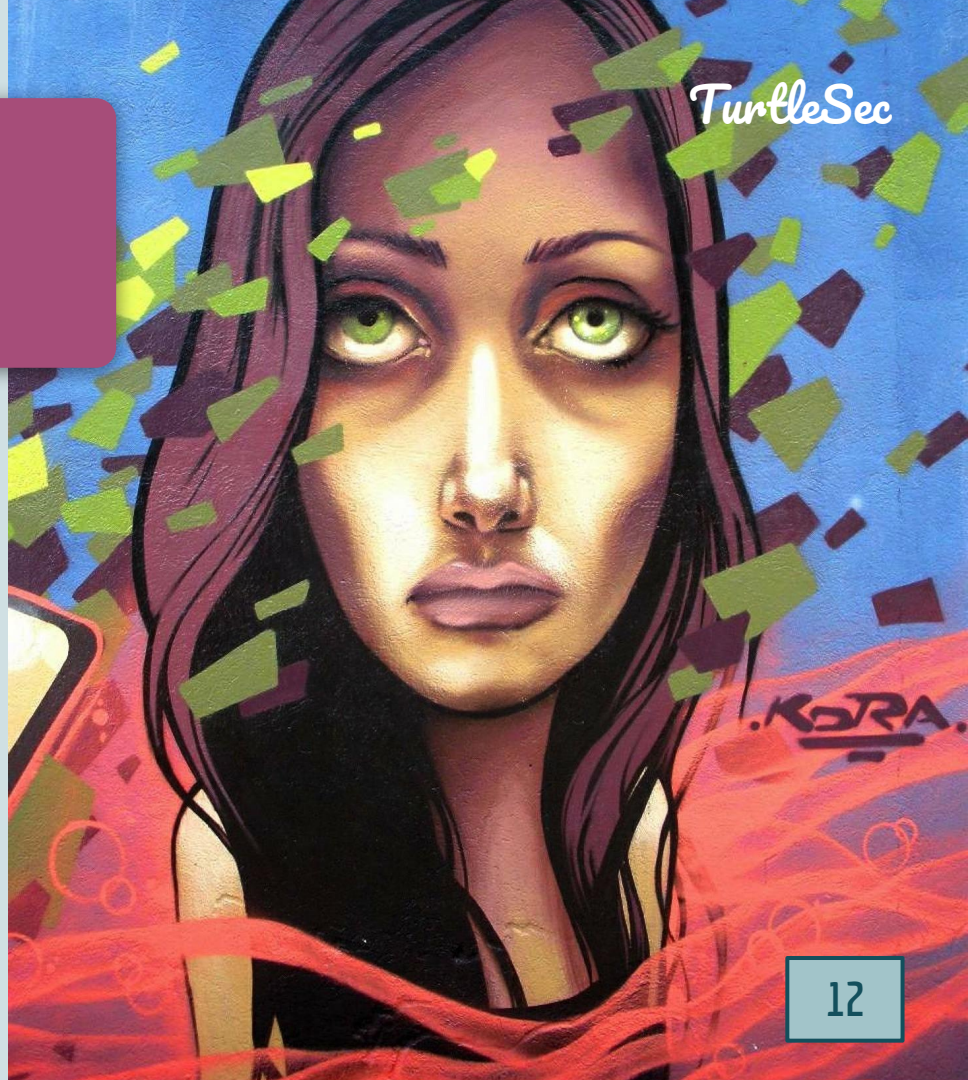
Say My Name - Destiny's Child

Bye Bye Bye - *NSYNC

In July 2000
Solar Designer
(Alexander Peslyak)
introduced the first
Generic Heap
Exploitation Technique

Doug Lea's malloc

The idea was to create a portable exploit that worked against many applications



Unlink Vulnerability Resources

TurtleSec

- *JPEG COM Marker Processing Vulnerability (CVE-2000-0655)*, Solar Designer, https://www.openwall.com/articles/JPEG-COM-Marker-Vulnerability
- *Vudo malloc tricks*, MaXX, *2001-08-11* Phrack Magazine, http://phrack.org/issues/57/8.html
- *Once upon a free()...*, anonymous, *2001-08-11* Phrack Magazine, http://phrack.org/issues/57/9.html
- *The Heap: Once upon a free() - bin 0x17*, LiveOverflow, https://youtu.be/gL45bjQvZSU
- *The Heap: dlmalloc unlink() exploit - bin 0x18*, LiveOverflow, https://youtu.be/HWhzH--89UQ
- *Alexander Peslyak (Solar Designer)*, https://en.wikipedia.org/wiki/Solar_Designer

Unlink Vulnerability

TurtleSec

@PATI_GALLARDO

Chocolate-Doom

Source port of the original Doom game from the early 90s

@PATI_GALLARDO

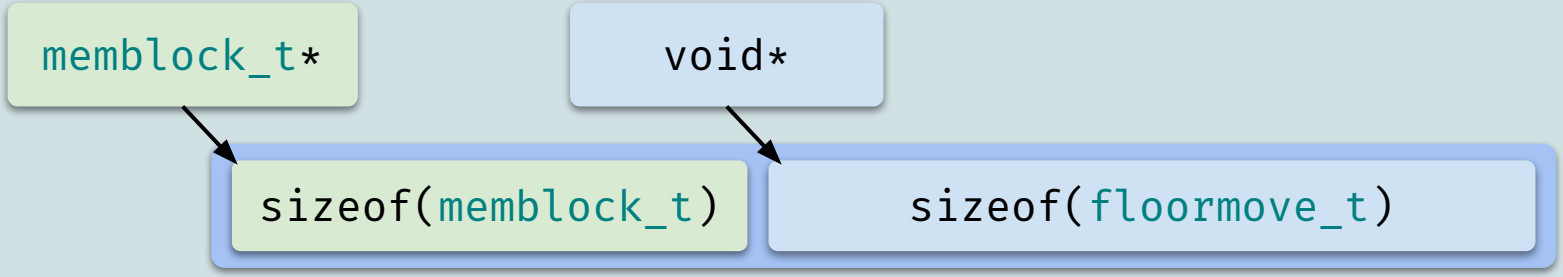
Z_Malloc

Allocator for Doom has a metadata section used to manage the memory

@PATI_GALLARDO

Z_Malloc : Doom allocations

TurtleSec




```
1.  floormove_t * floor =
2.    (floormove_t *) Z_Malloc(sizeof(floormove_t), PU_LEVSPEC, NULL);
```

tag

user

memblock_t

TurtleSec

z_native is an implementation of **Z_Malloc**

src/z_native.cpp

```
1.  struct memblock_t {
2.   int               id; // = ZONEID
3.   int               tag;
4.   int               size;
5.   void **           user;
6.   memblock_t * prev;
7.   memblock_t * next;
8.  };
```

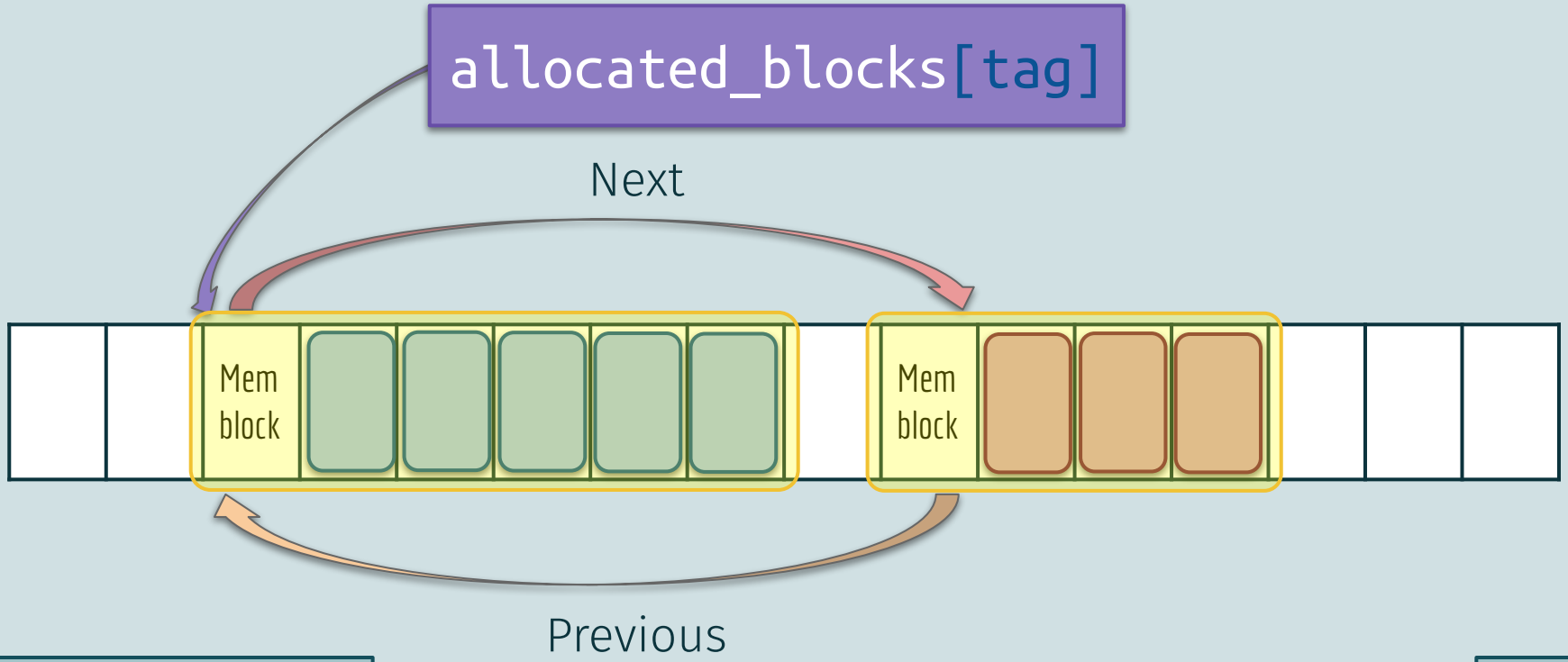
tag

user

Doubly linked list

Metadata stored in the heap

TurtleSec



src/z_native.cpp

```
1.  void Z_Free(void * ptr) {
2.   auto * byte_ptr = static_cast<uint8_t *>(ptr);
3.   auto * block = reinterpret_cast<memblock_t *>(byte_ptr - sizeof(memblock_t));
4.
5.   if (block->id != ZONEID) {
6.     I_Error("Z_Free: freed a pointer without ZONEID");
7.   }
8.
9.   if (block->tag != PU_FREE && block->user != nullptr) {
10.    // clear the user's mark
11.
12.    *block->user = nullptr;
13.  }
14.
15.  Z_RemoveBlock(block);
16.
17.  // Free back to system
18.
19.  free(block);
20. }
```

Metadata on allocation stored adjacent to the allocated heap memory

Before freeing the memory, remove the block from internal data structures

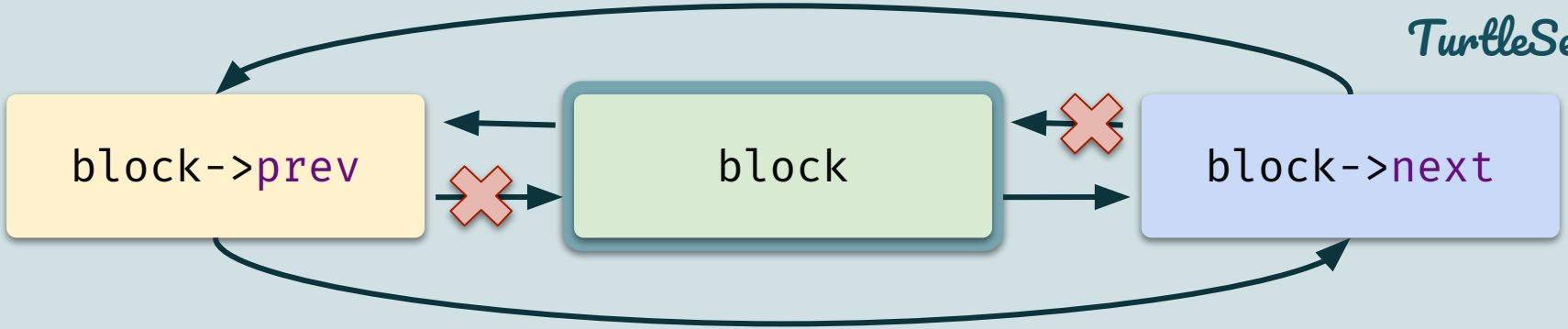
TurtleSec

src/z_native.cpp

```
1.  static void Z_RemoveBlock(memblock_t * block) {
2.   // Unlink from list
3.
4.   if (block->prev == nullptr) {
5.     // Start of list
6.
7.     allocated_blocks[block->tag] = block->next;
8.   } else {
9.     block->prev->next = block->next;
10.  }
11.
12.  if (block->next != nullptr) {
13.    block->next->prev = block->prev;
14.  }
15. }
16.
```

Classic unlinking from a doubly linked list

TurtleSec



src/z_native.cpp

```
1.  static void Z_RemoveBlock(memblock_t * block) {
2.   if (block->prev == nullptr) {
3.     allocated_blocks[block->tag] = block->next;
4.   } else {
5.     block->prev->next = block->next;
6.   }
7.   if (block->next != nullptr) {
8.     block->next->prev = block->prev;
9.   }
10.  }
```

Insight

If we can control both sides of an allocation we can create a Write-What-Where primitive

If we control **block->prev**
we control the where this write will happen
(adjusted for the offset of **next**)

If we control **block->next**
we control what to write there

src/z_native.cpp

```
1.  static void Z_RemoveBlock(memblock_t * block) {
2.   if (     where      == nullptr) {
3.     allocated_blocks[block->tag] = block->next;
4.   } else {
5.     where     ->next =     what     ;
6.   }
7.   if (block->next != nullptr) {
8.     block->next->prev = block->prev;
9.   }
10.  }
```

Write-What-Where

Proof of Concept

Corrupt the memblock_t metadata before freeing the memory

```
1.  void * guard  = Z_Malloc(10, PU_LEVEL, nullptr);
2.  void * ptr    = Z_Malloc(10, PU_LEVEL, nullptr);
3.  void * guard2 = Z_Malloc(10, PU_LEVEL, nullptr);
4.
5.  auto * byte_ptr    = (uint8_t *) ptr;
6.  auto * header      = (memblock_t *) (byte_ptr - sizeof(memblock_t));
7.
8.  long *  where     = nullptr;
9.  long ** where_ptr = &where;
10. long    what      = 0x42424242;
11. long *  what_ptr  = &what;
12.
13. auto distance = (uint8_t*)(&(header->next)) - (uint8_t*) header;
14. uint8_t * byte_where_ptr          = (uint8_t*) where_ptr;
15. uint8_t * adjusted_byte_where_ptr = byte_where_ptr - distance;
16.
17. header->prev = (memblock_t *) adjusted_byte_where_ptr;
18. header->next = (memblock_t *) what_ptr;
19.
20. assert(where == nullptr);
21. Z_Free(ptr);
22. assert(where != nullptr);
23. assert(*where == 0x42424242);
```

Allocate memory

Get **memblock***

Prepare **where**

Prepare **what**

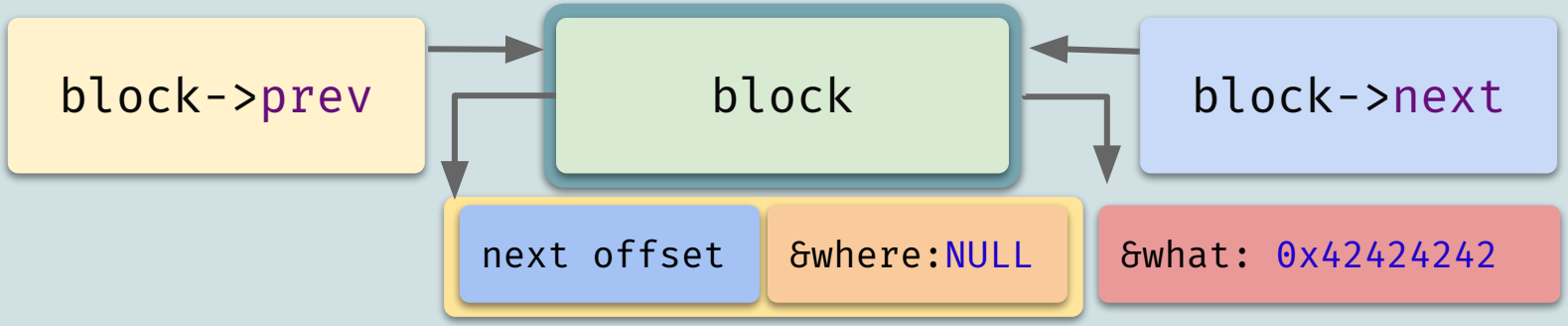
Adjust **what** for distance to **next**

Free memory - unlink happens

where has been set to **what**

TurtleSec

@PATI_GALLARDO



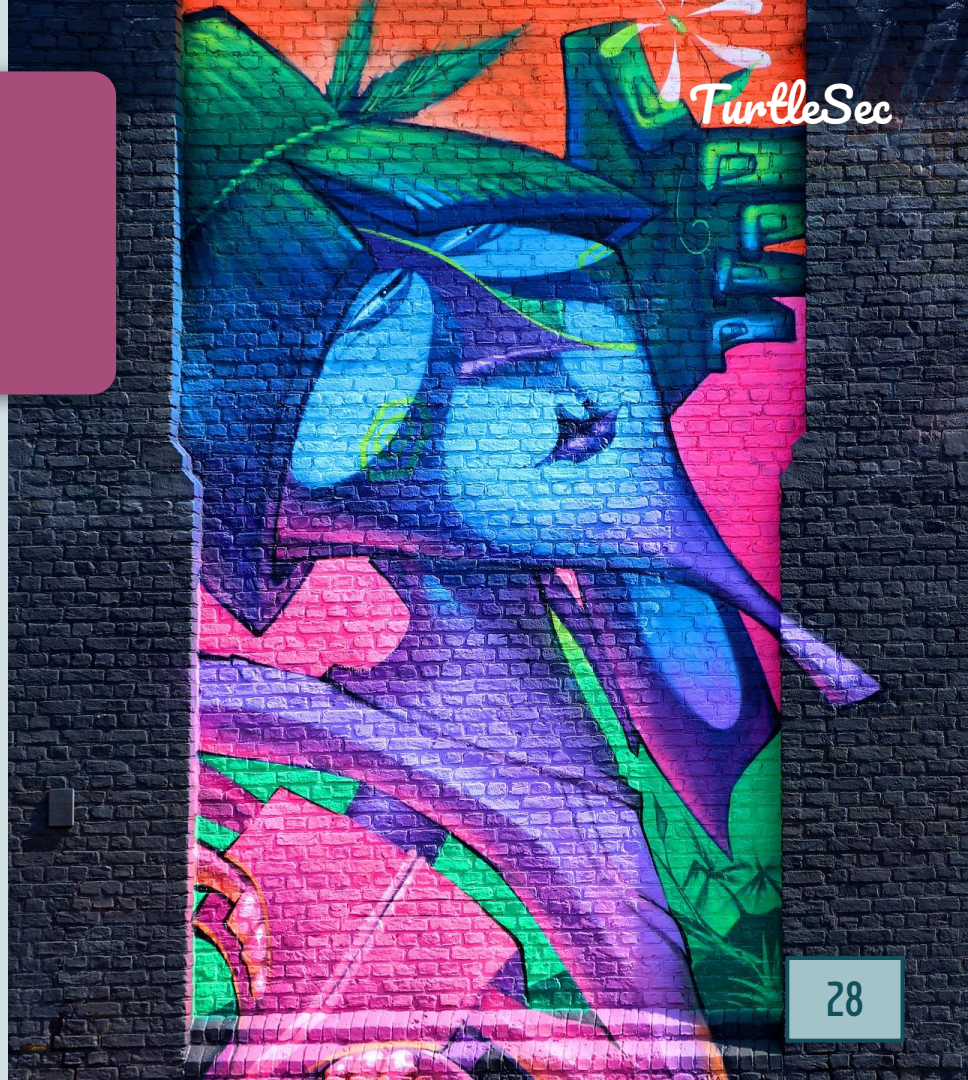
src/z_native.cpp

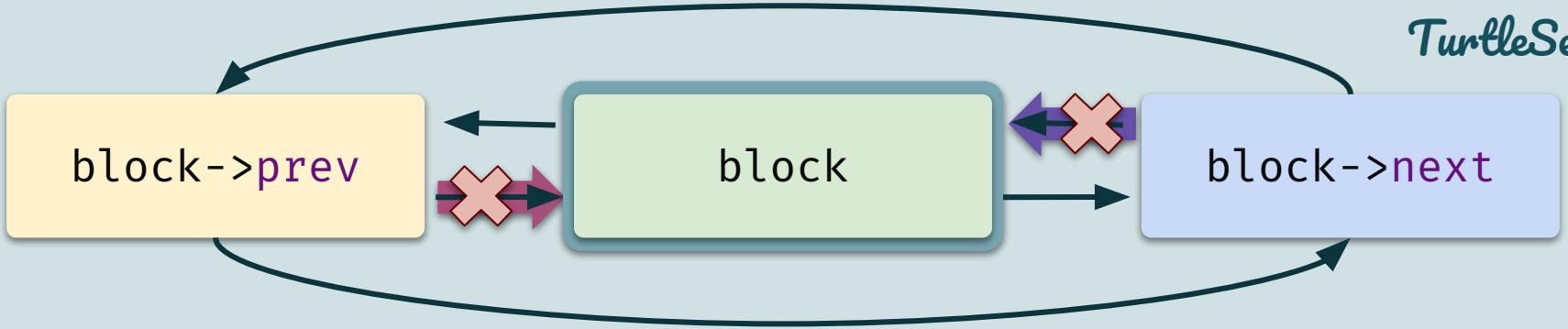
```
1.  static void Z_RemoveBlock(memblock_t * block) {
2.   if (block->prev == nullptr) {
3.     allocated_blocks[block->tag] = block->next;
4.   } else {
5.     block->prev->next = block->next;
6.   }
7.   if (block->next != nullptr) {
8.     block->next->prev = block->prev;
9.   }
10.  }
```

```
block->prev->next =
block->prev + next offset
```

Traditional mitigation

Check the pointers before unlinking





src/z_native.cpp

```
1.  // ...
2.  if (block->prev->next != block)
3.    exit(1);
4.  block->prev->next = block->next;
5.  // ...
6.  if (block->next->prev != block)
7.    exit(1);
8.  block->next->prev = block->prev;
9.  // ...
```

How to exploit

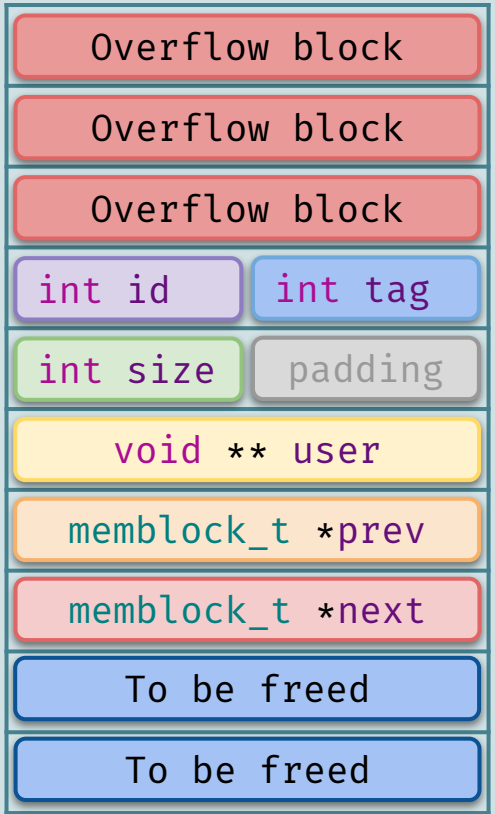
Using a heap buffer overflow

Heap Grooming
to overwrite adjacent memory

TurtleSec

```
1.  struct memblock_t {
2.    int             id;
3.    int             tag;
4.    int             size;
5.    void **         user;
6.    memblock_t * prev;
7.    memblock_t * next;
8.  };
```

@PATI_GALLARDO

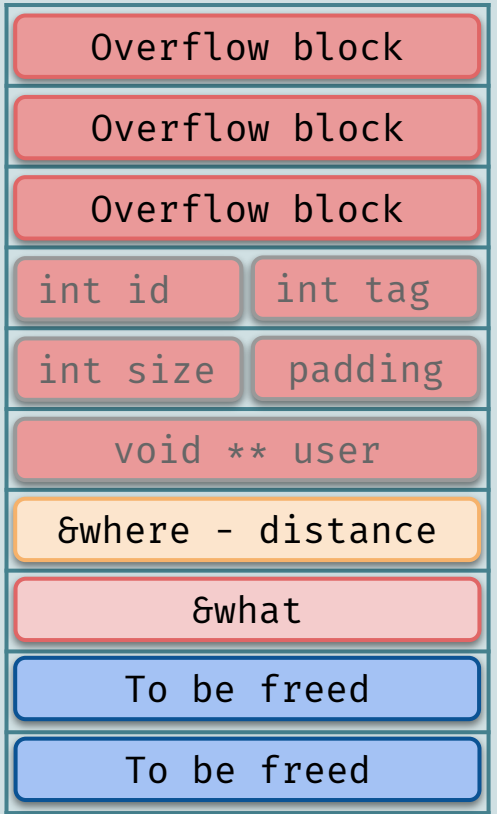


Heap Grooming
to overwrite adjacent memory

TurtleSec

```
1.  struct memblock_t {
2.    int             id;
3.    int             tag;
4.    int             size;
5.    void **         user;
6.    memblock_t * prev;
7.    memblock_t * next;
8.  };
```

@PATI_GALLARDO



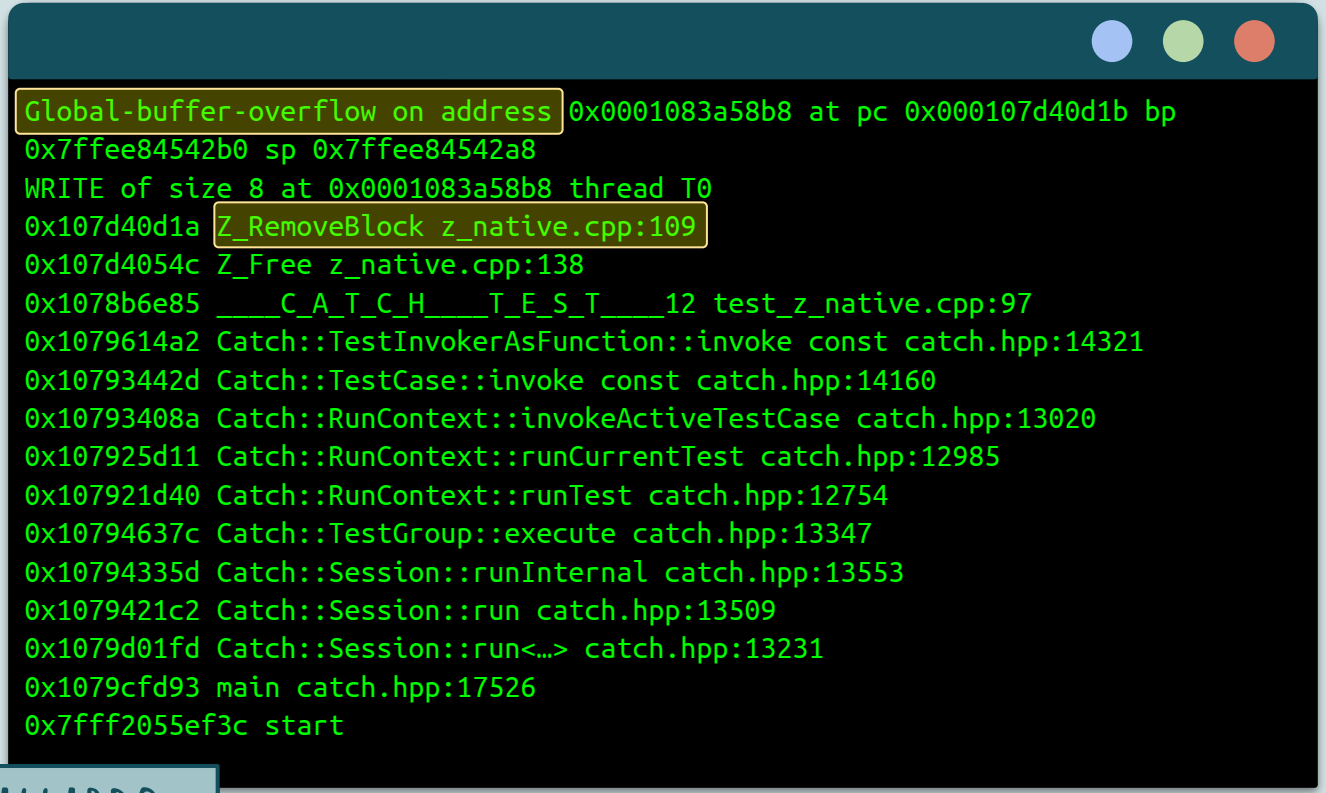
How to find them

Hard to find without in-code checks
This is valid memory that is being corrupted.

@PATI_GALLARDO

Test case fails in ASan

TurtleSec

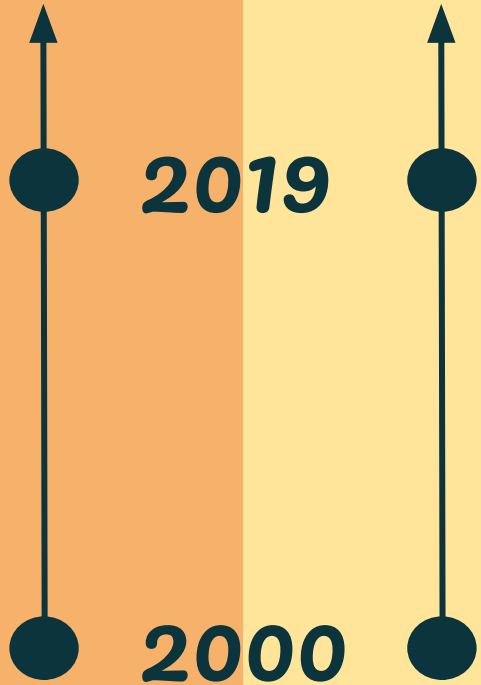


Doom Vulnerability Resources

TurtleSec

- *PR*: https://github.com/chocolate-doom/chocolate-doom/pull/1454
- *PoC* https://gist.github.com/patricia-gallardo/e8aef21a397b8c928a3aae9e4ae8445f
- *Issue*: https://github.com/chocolate-doom/chocolate-doom/issues/1453

Systems Programming



Binary Exploitation

Bad Binder: Android In-The-Wild Exploit

CVE-2019-2215

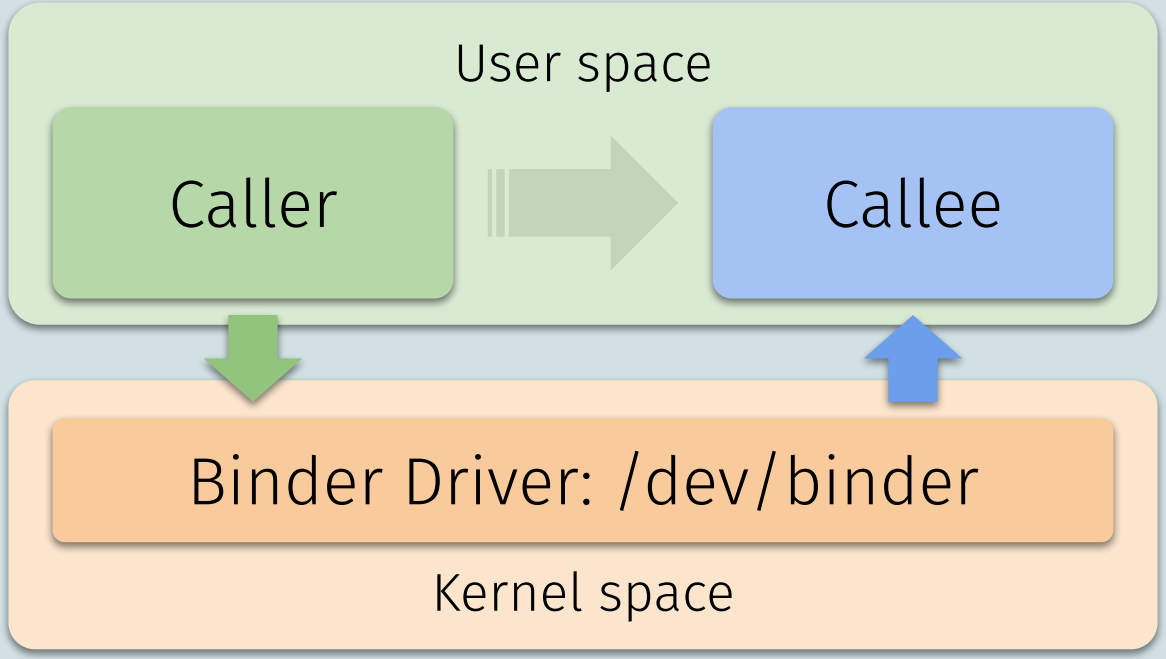
@PATI_GALLARDO

CVE-2019-2215

"A use-after-free in binder.c
allows
an elevation of privilege
from an application
to the Linux Kernel."

Binder: Androids IPC mechanism

TurtleSec



Threat Actor: NSO Group

TurtleSec

The *Bad Binder* Android exploit was attributed to *NSO Group*.
When it was reported it was being used in the wild.

NSO Group is an Israeli technology firm.
They have a product called *Pegasus* that enables remote surveillance of smartphones.

Information available

Arbitrary kernel read/write primitive

`CONFIG_DEBUG_LIST` breaks the primitive

lib/list_debug.c

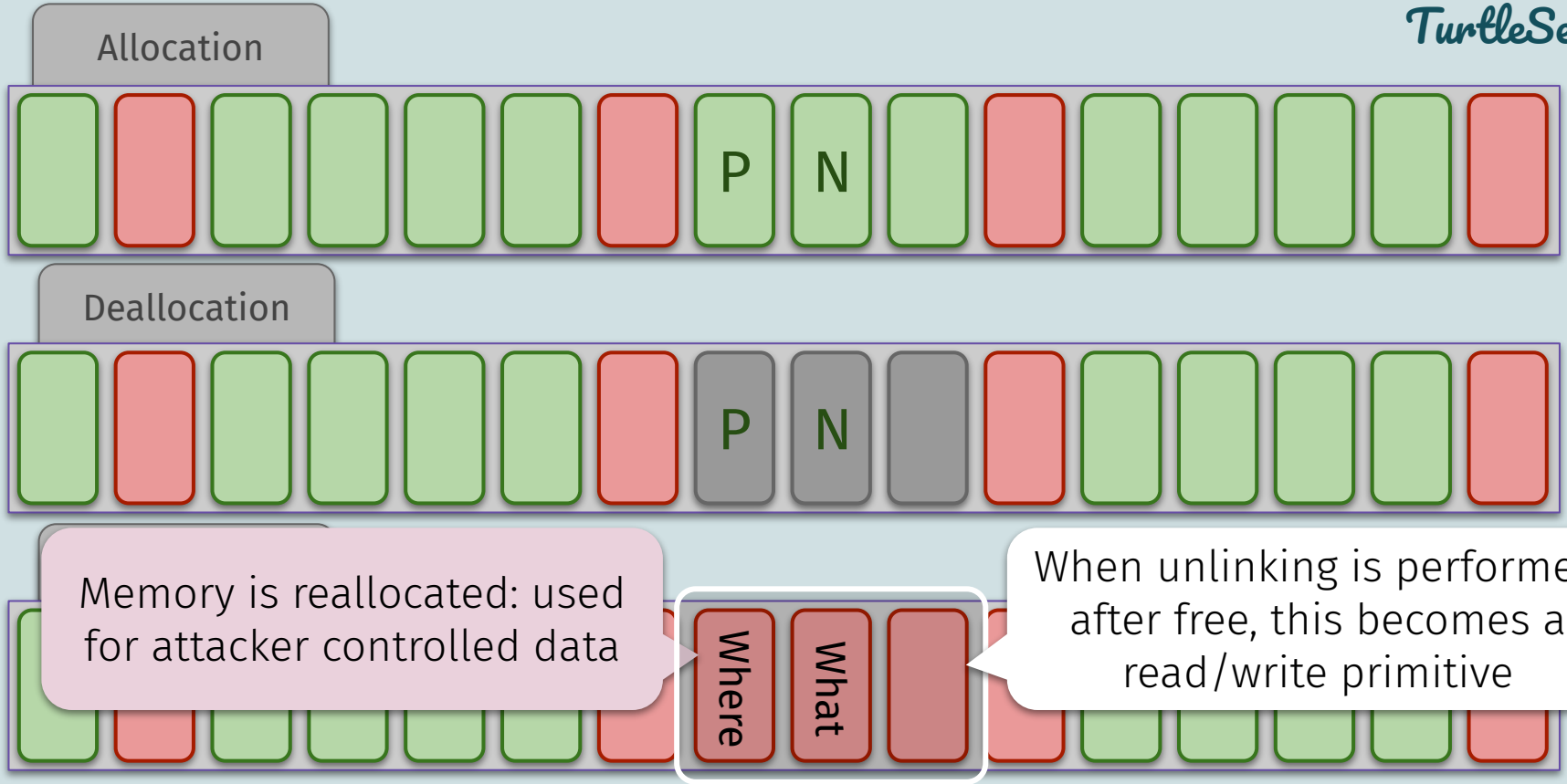
```
1.  void __list_del_entry(struct list_head *entry) {
2.   struct list_head *prev, *next;
3.   prev = entry->prev;
4.   next = entry->next;
5.
6.   if (WARN(next == LIST_POISON1,
7.            "list_del corruption, %p->next is LIST_POISON1 (%p)\n",
8.            entry, LIST_POISON1) ||
9.       WARN(prev == LIST_POISON2,
10.           "list_del corruption, %p->prev is LIST_POISON2 (%p)\n",
11.           entry, LIST_POISON2) ||
12.      WARN(prev->next != entry,
13.           "list_del corruption. prev->next should be %p, "
14.           "but was %p\n", entry, prev->next) ||
15.      WARN(next->prev != entry,
16.           "list_del corruption. next->prev should be %p, "
17.           "but was %p\n", entry, next->prev)) {
18.     BUG_ON(PANIC_CORRUPTION);
19.     return;
20.   }
21.   __list_del(prev, next);
22.  }
```

This might look familiar
This is the standard unlink vuln mitigation

CONFIG_DEBUG_LIST
breaks the primitive
by enabling this check

How to exploit

Use After Free



Memory is reallocated: used for attacker controlled data

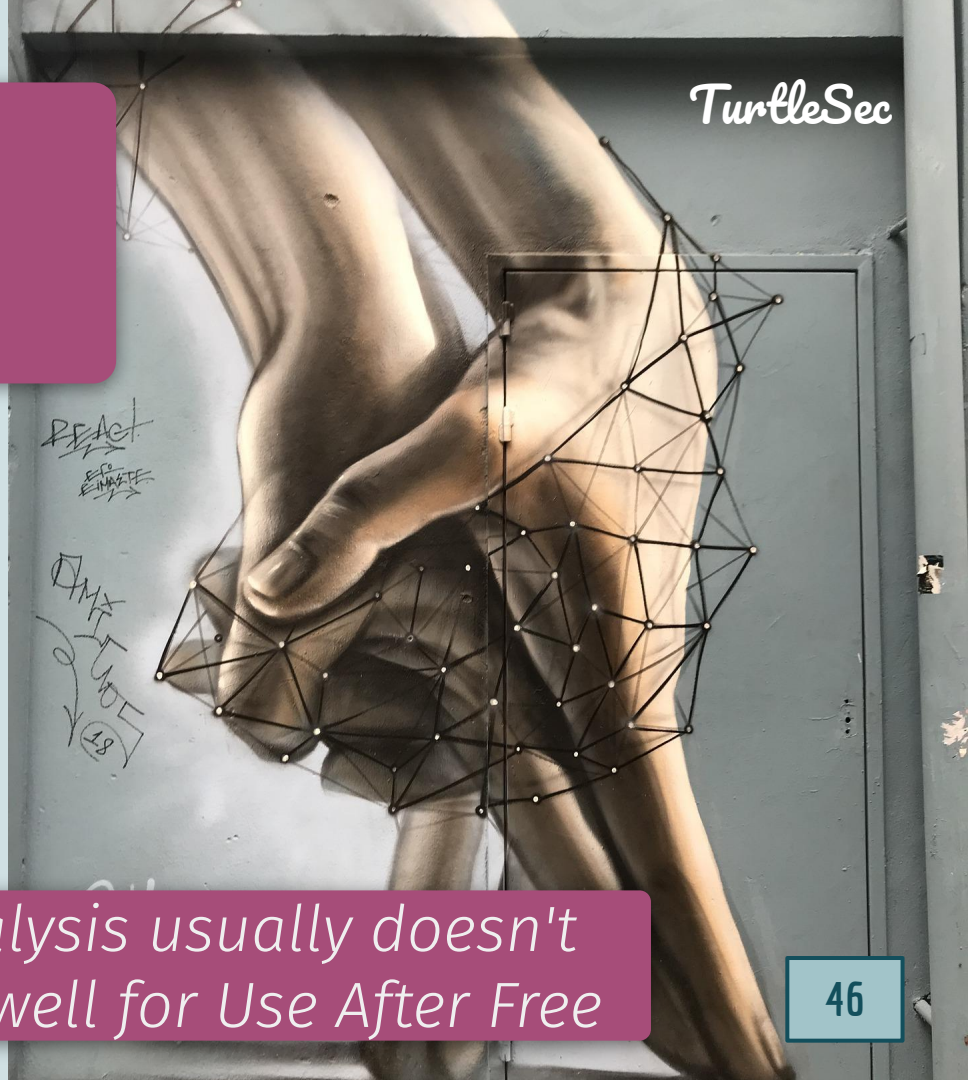
When unlinking is performed after free, this becomes a read/write primitive

Heap: Use After Free

The unlinking is done in privileged code
therefore this becomes:

Use-after-free
leading to
arbitrary kernel read/write primitive

How to find them



Address Sanitizer

Static Analysis usually doesn't work very well for Use After Free

Tools: Use After Free

TurtleSec

textscreen/txt_window.cpp

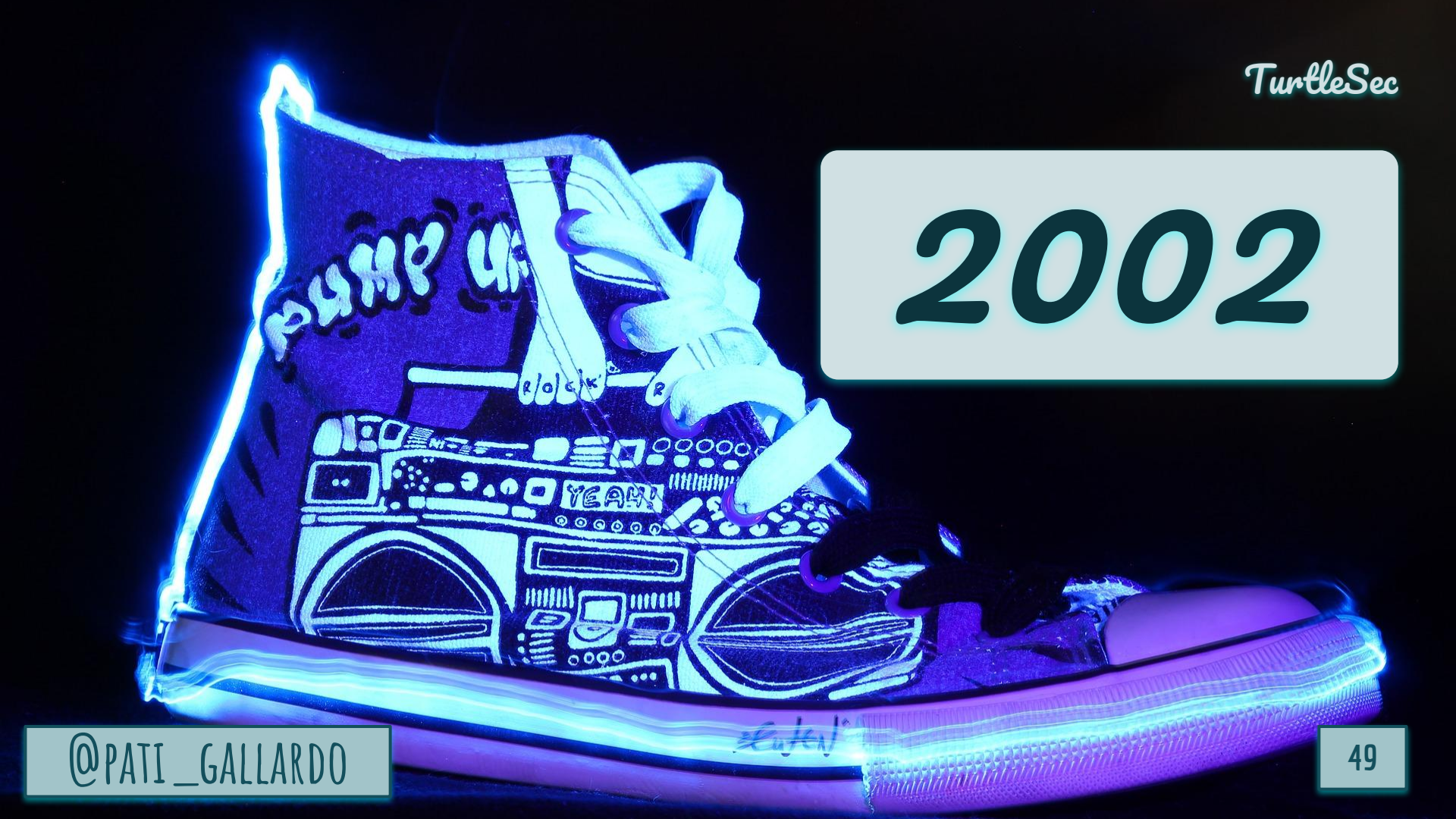
```
1.  void TXT_OpenURL(cstring_view url) {
2.   size_t cmd_len = url.size() + 30;
3.   char * cmd     = static_cast<char *>(malloc(cmd_len));
4.
5.   // ...
6.
7.   int retval = system(cmd);
8.   free(cmd);
9.   if (retval != 0) {
10.    fmt::fprintf(stderr,
11.                 "error executing '%s'; return code %d\n",
12.                 cmd, retval);
13.  }
14. }
```

"Local variable 'cmd' may point to deallocated memory"
Clang-Tidy: "Use of memory after it is freed"

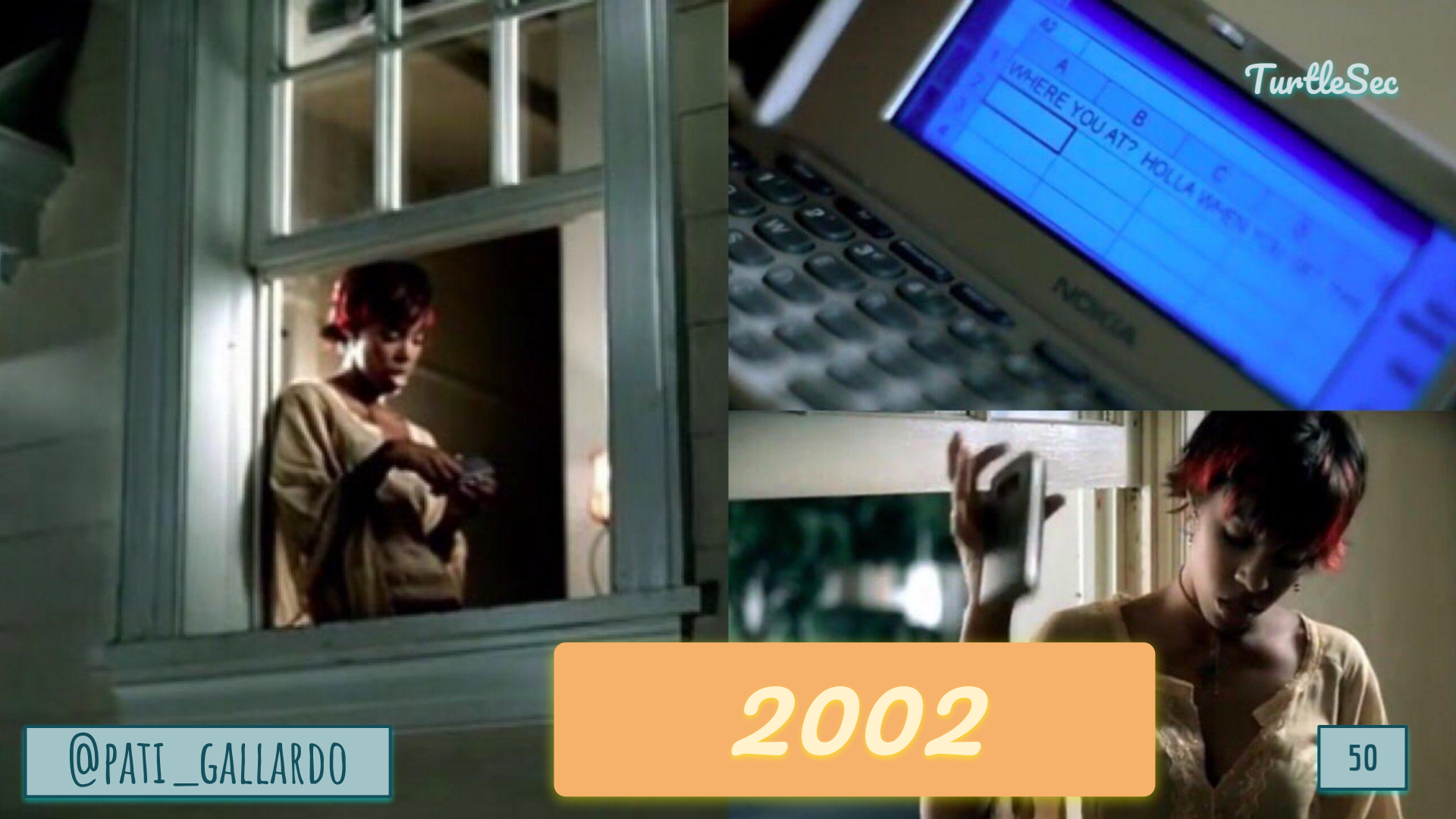
CVE-2019-2215 Resources

TurtleSec

- *Bad Binder: Finding an Android In The Wild (video)*, Maddie Stone, https://youtu.be/TAwQ4ezgEIo
- *Bad Binder: Finding an Android In The Wild (blog post)*, Maddie Stone, https://googleprojectzero.blogspot.com/2019/11/bad-binder-android-in-wild-exploit.html
- *CVE-2019-2215*, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-2215
- *Issue 1942: Android: Use-After-Free in Binder driver*, https://bugs.chromium.org/p/project-zero/issues/detail?id=1942



2002


TurtleSec
WHERE YOU AT? HOLLA
NOKIA
2002
@PATI_GALLARDO
50

2002 : 20 years ago

TurtleSec

Hot In Herre

Dilemma ft. Kelly Rowland

Integer Overflows Resources

TurtleSec

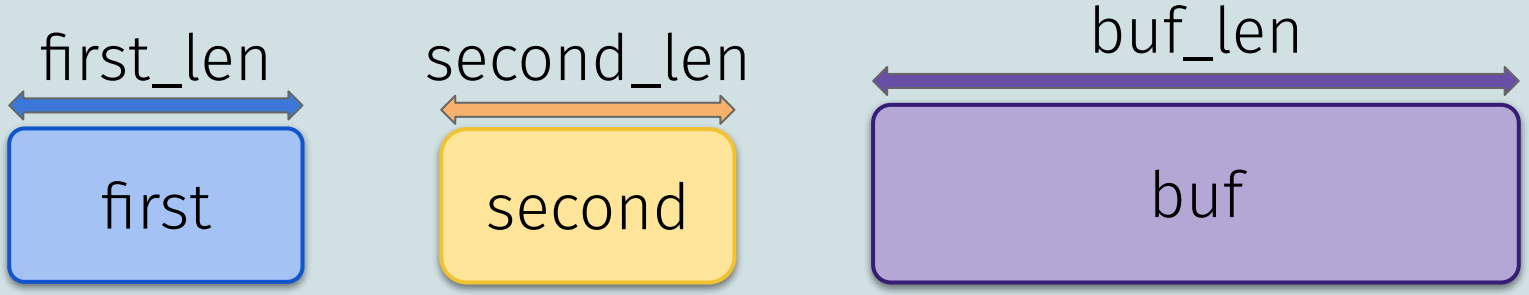
- *Basic Integer Overflows*, blexim, 2002-12-28 Phrack Magazine, http://phrack.org/issues/60/10.html

TurtleSec

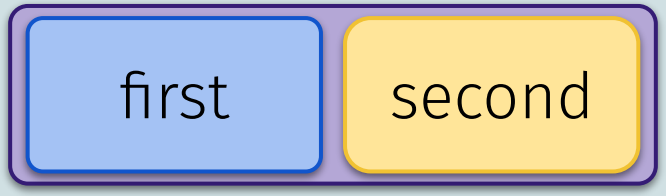
Signed Integer Overflow
Unsigned Int Wraparound

Copying buffers

TurtleSec



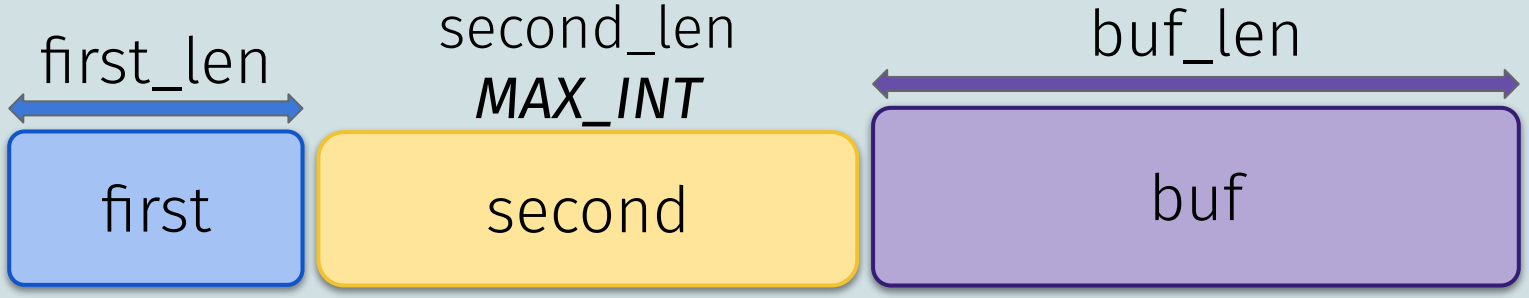
Is it safe to copy first and second into buf?

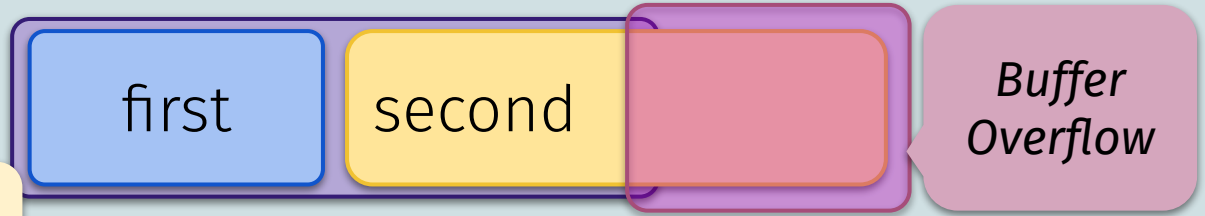



```
1.  if(first_len + second_len < buf_len)
2.    copy(first, second, buf);
```

Exploitation: Buffer Overflow

TurtleSec



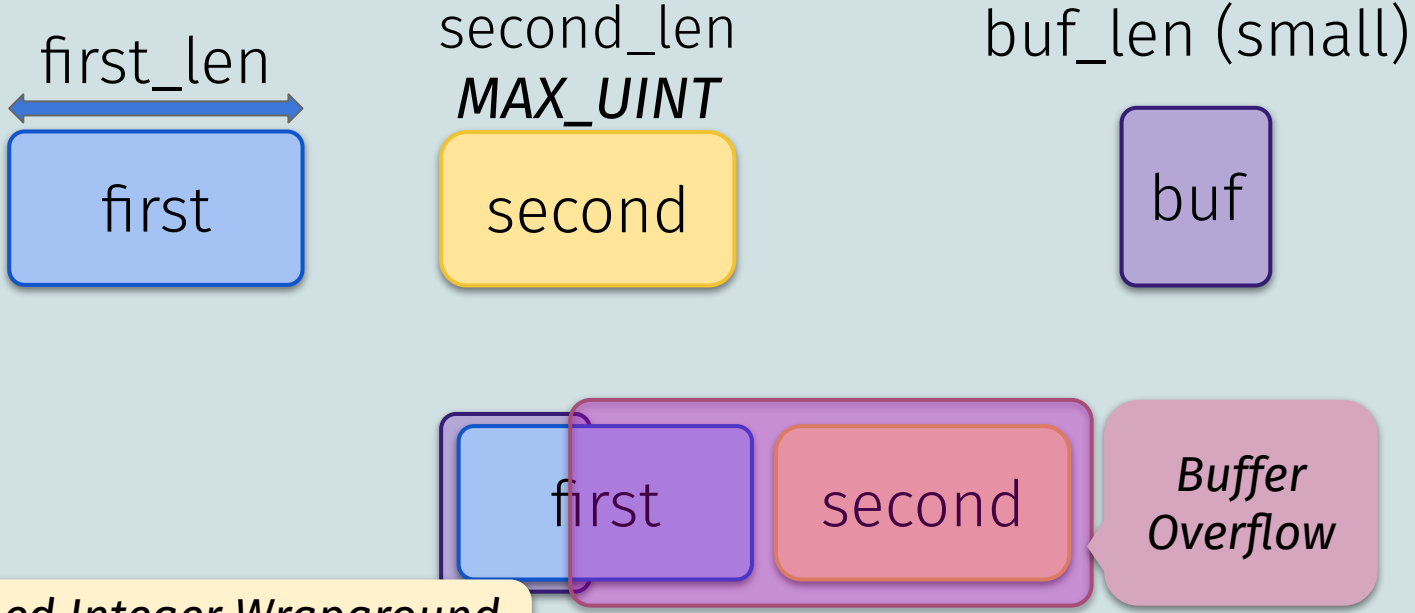


Signed Integer Overflow
Result is negative

```
1.  if(first_len + second_len < buf_len)
2.    copy(first, second, buf);
```

Exploitation: Buffer Overflow

TurtleSec

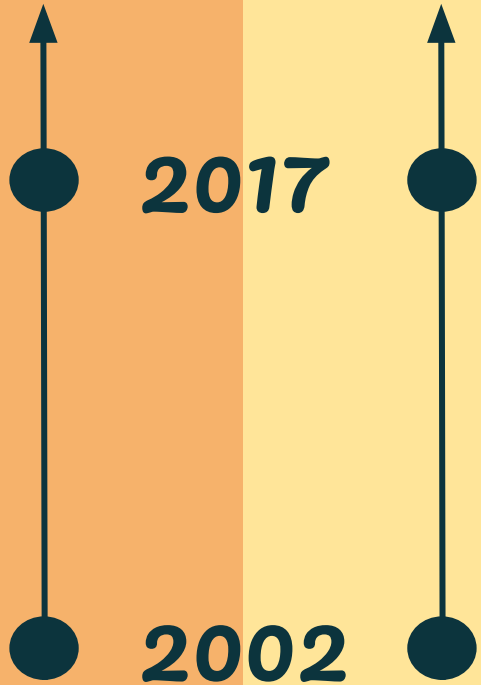


Unsigned Integer Wraparound
Result is small

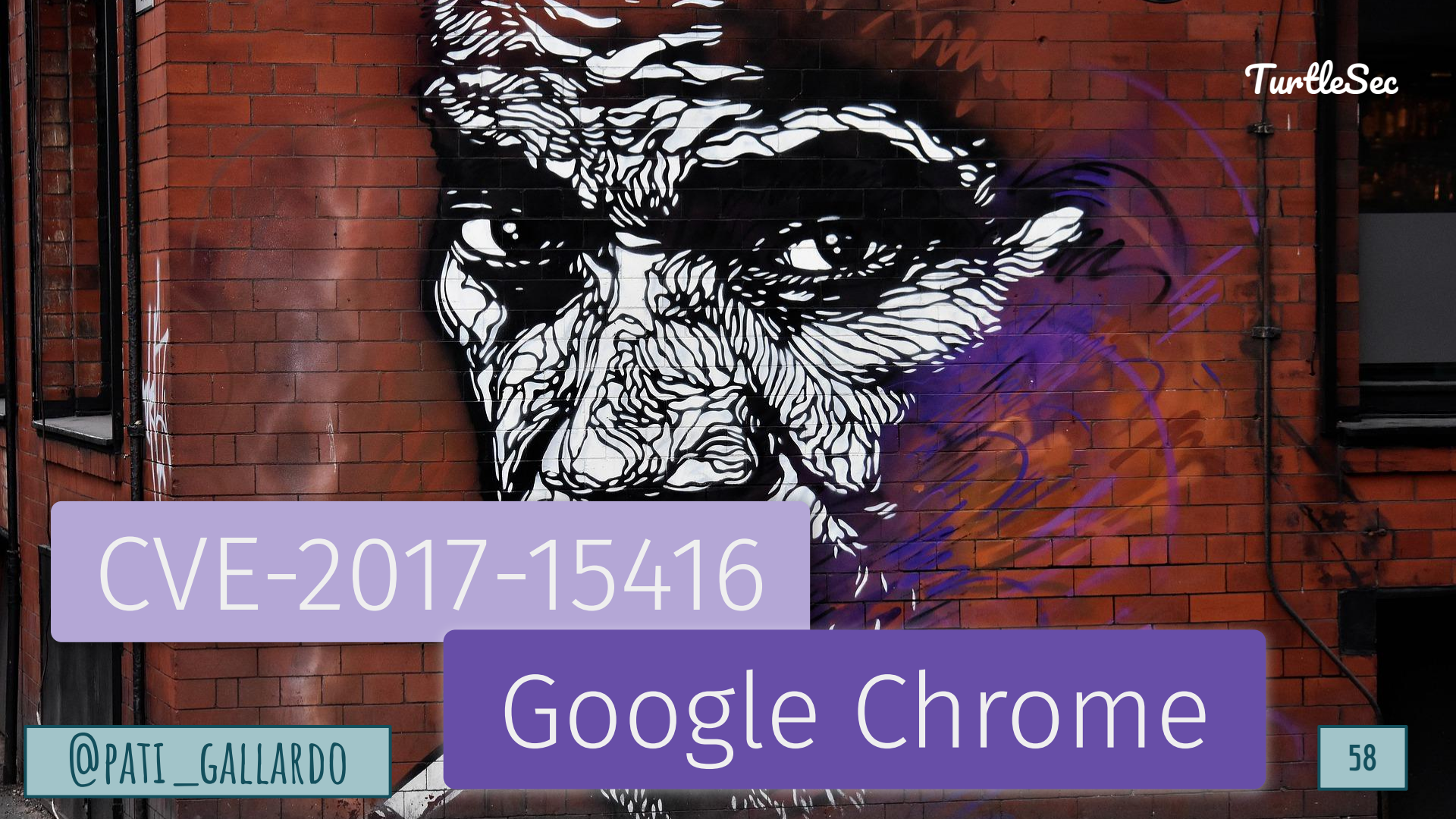
```
1.  buf_len = first_len + second_len;
2.  buf = allocate(buf_len);
3.  copy(first, second, buf);
```

@PATI_GALLARDO

Systems Programming



Binary Exploitation


TurtleSec
CVE-2017-15416
Google Chrome
@PATI_GALLARDO
58

CVE-2017-15416

"Heap buffer overflow in Blob API in Google Chrome [...] allowed a remote attacker to potentially exploit heap corruption"

@PATI_GALLARDO

Example: CVE-2017-15416

TurtleSec

Heap buffer overflow in Blob API in Google Chrome

chromium/storage/browser/blob/blob_storage_context.cc

```
1.    // Validate our reference has good offset & length.
2.  - if (input_element.offset() + length > ref_entry->total_size()) {
3.  + uint64_t end_byte;
4.  + if (!base::CheckAdd(input_element.offset(), length)
5.  +           .AssignIfValid(&end_byte) ||
6.  +      end_byte > ref_entry->total_size()) {
7.        status = BlobStatus::ERR_INVALID_CONSTRUCTION_ARGUMENTS;
8.        return;
9.      }
```

If add is safe

Assign to end_byte

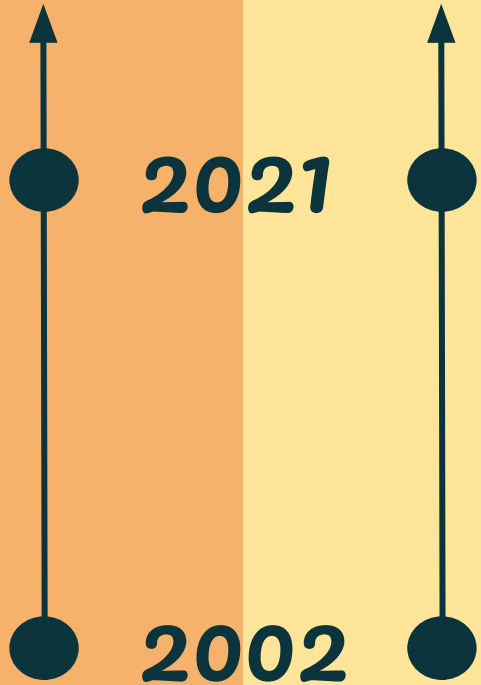
Check against ref_entry total_size

CVE-2017-15416 Resources

TurtleSec

- *CVE-2017-15416*, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-15416
- *CVE-2017-15416 (fix)*, https://chromium.googlesource.com/chromium/src.git/+/11bd4bc92f3fe704631e3e6ad1dd1a4351641f7c%5E%21/
- *Popping Calc with Hardware Vulnerabilities CVE-2017-15416 (exploitation)*, Stephen Roettger, https://youtu.be/ugZzQvXUTIk

Systems Programming



Binary Exploitation

TurtleSec

Apple iOS, iPadOS and macOS

CVE-2021-30860

@PATI_GALLARDO

CVE-2021-30860

"An integer overflow was addressed [...] Processing a maliciously crafted PDF may lead to arbitrary code execution."

32 bit uint

Increment with attacker controlled data

Allocate a buffer too small based on wrapped uint

Overflow too small buffer

```
Guint numSyms;
numSyms = 0;
for (i = 0; i < nRefSegs; ++i) {
 if ((seg = findSegment(refSegs[i]))) {
   if (seg->getType() == jbig2SegSymbolDict) {
     numSyms += ((JBIG2SymbolDict *)seg)->getSize();
   } else if (seg->getType() == jbig2SegCodeTable) {
     codeTables->append(seg);
   }
 } else {
   error(errSyntaxError, getPos(),
         "Invalid segment reference in JBIG2 text region");
   delete codeTables;
   return;
 }
}
 // ...
 // get the symbol bitmaps
syms = (JBIG2Bitmap **)gmallocn(numSyms, sizeof(JBIG2Bitmap *));
kk = 0;
for (i = 0; i < nRefSegs; ++i) {
 if ((seg = findSegment(refSegs[i]))) {
   if (seg->getType() == jbig2SegSymbolDict) {
     symbolDict = (JBIG2SymbolDict *)seg;
     for (k = 0; k < symbolDict->getSize(); ++k) {
       syms[kk++] = symbolDict->getBitmap(k);
     }
   }
 }
}
```

@PATI_GALLARDO

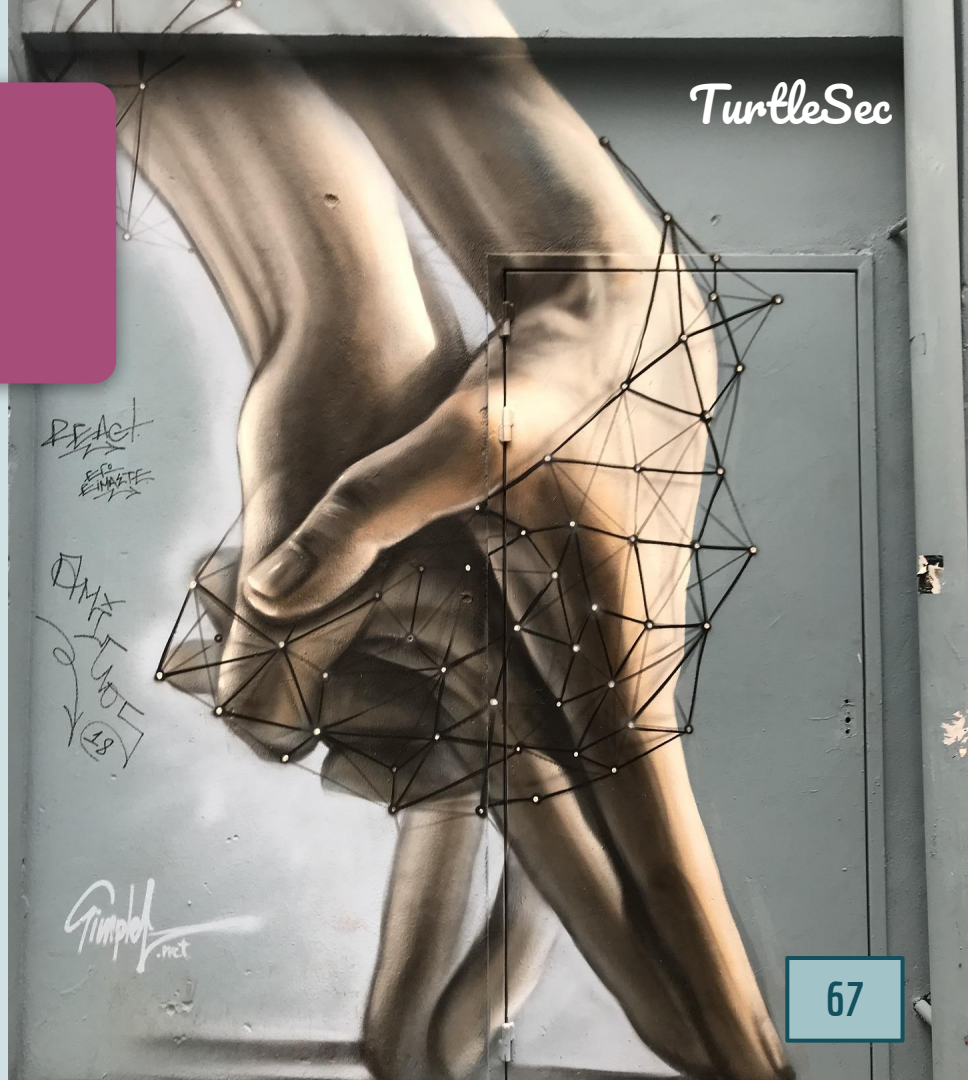
CVE-2021-30860 Resources

TurtleSec

- *A deep dive into an NSO zero-click iMessage exploit: Remote Code Execution*, Project Zero team at Google, https://googleprojectzero.blogspot.com/2021/12/a-deep-dive-into-nso-zero-click.html
- *FORCEDENTRY*, https://en.wikipedia.org/wiki/FORCEDENTRY
- *CVE-2021-30860*, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-30860
- *FORCEDENTRY: Sandbox Escape*, Ian Beer & Samuel Groß, https://googleprojectzero.blogspot.com/2022/03/forcedentry-sandbox-escape.html

How to find them

UB Sanitizer
Integer Sanitizer



@PATI_GALLARDO

C++20 Safe Integer Comparisons

TurtleSec

C++20

```cpp
#include <utility>

int main()
{
 static_assert( sizeof(int) == 4 );

 static_assert( -1 > 1U );
 static_assert( 0xFFFFFFFFU > 1U );
 static_assert( 0xFFFFFFFFU == static_cast<unsigned>(-1) );

 static_assert( std::cmp_less( -1, 1U ) );
 static_assert( std::cmp_less_equal( -1, 1U ) );
 static_assert( ! std::cmp_greater( -1, 1U ) );
 static_assert( ! std::cmp_greater_equal( -1, 1U ) );

 static_assert( -1 == 0xFFFFFFFFU );
 static_assert( std::cmp_not_equal( -1, 0xFFFFFFFFU ) );
}
```

Example Code from cppreference.com

2010

2010 : 12 years ago

TurtleSec

Rihanna - Rude Boy

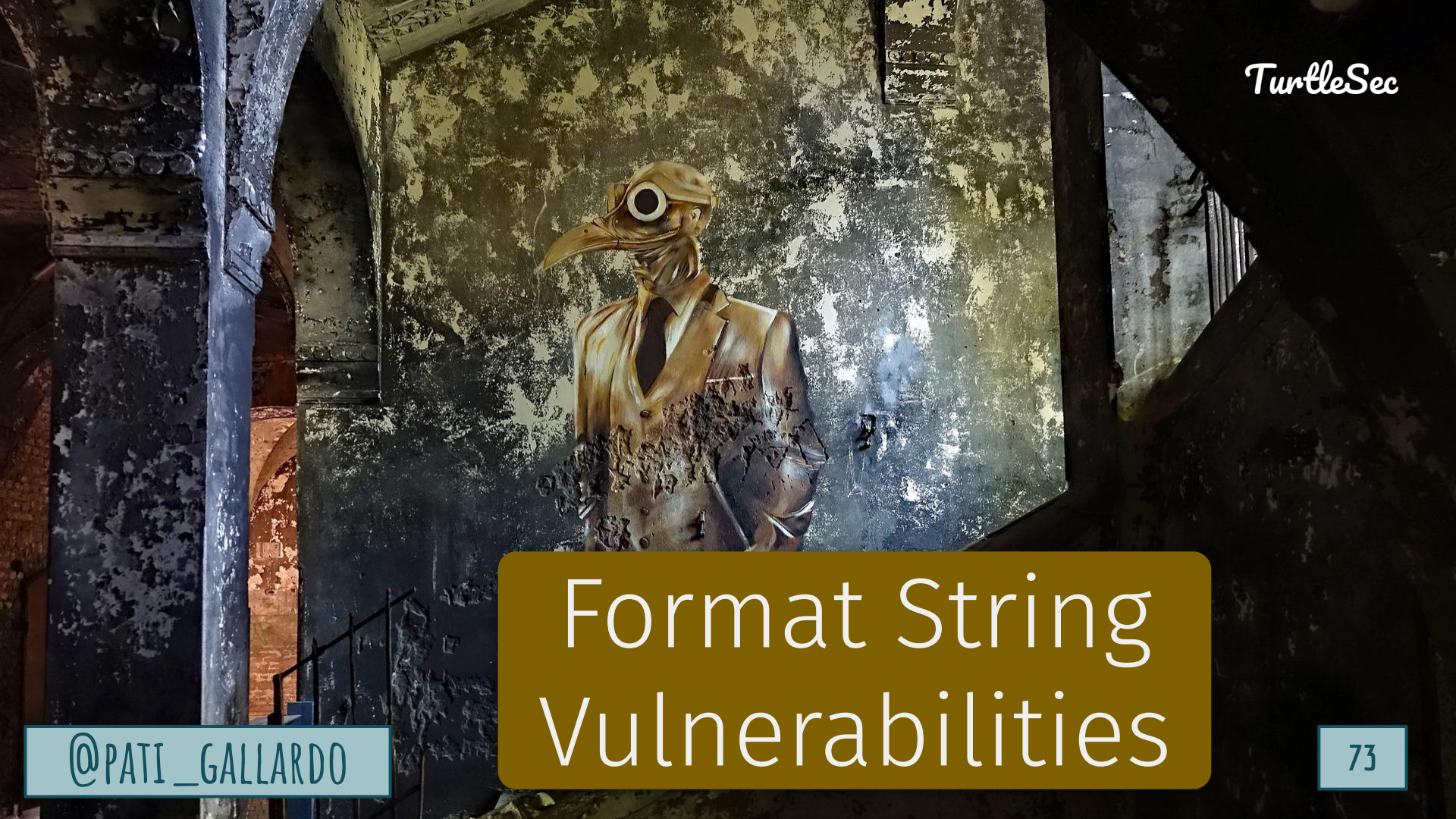
Lady Gaga - Bad Romance


TurtleSec
2010
@PATI_GALLARDO
71

Format String Vulnerability Resources

TurtleSec

- *A Eulogy for Format Strings*, Captain Planet, *2010-11-17* Phrack Magazine, http://phrack.org/issues/67/9.html
- *Advances in format string exploitation*, riq & gera, *2002-07-28* Phrack Magazine, http://phrack.org/issues/59/7.html



TurtleSec

Format String Vulnerabilities

@PATI_GALLARDO


TurtleSec
A couple of Lesser Known
Format String Features
@PATI_GALLARDO
74

Field width

TurtleSec

field_width.c

```c
int main(void) {
 printf("% 17d\n", 10);
 printf("% *d\n", 18, 10);
 printf("%2$ *1$d\n", 19, 10); // Direct Access
}
```

```
$ clang -o field_width field_width.c
$ ./field_width
               17 10
               18 10
               19 10
```

Chars written

TurtleSec

chars_written_1.c

```c
int main(void) {
 int num = 0;
 printf("abcdef%n\n", &num);
 printf("%d\n", num);
}
```

```
$ clang -o chars_written chars_written_1.c
$ ./chars_written
abcdef
6
```

Chars written

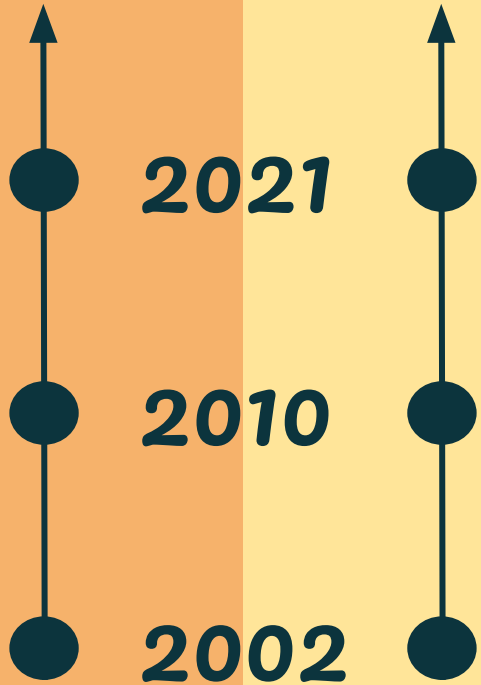
TurtleSec

chars_written_2.c

```c
int main(void) {
 int num = 0;
 printf("%42d%n\n", 1, &num);  // Field width
 printf("%d\n", num);
}
```

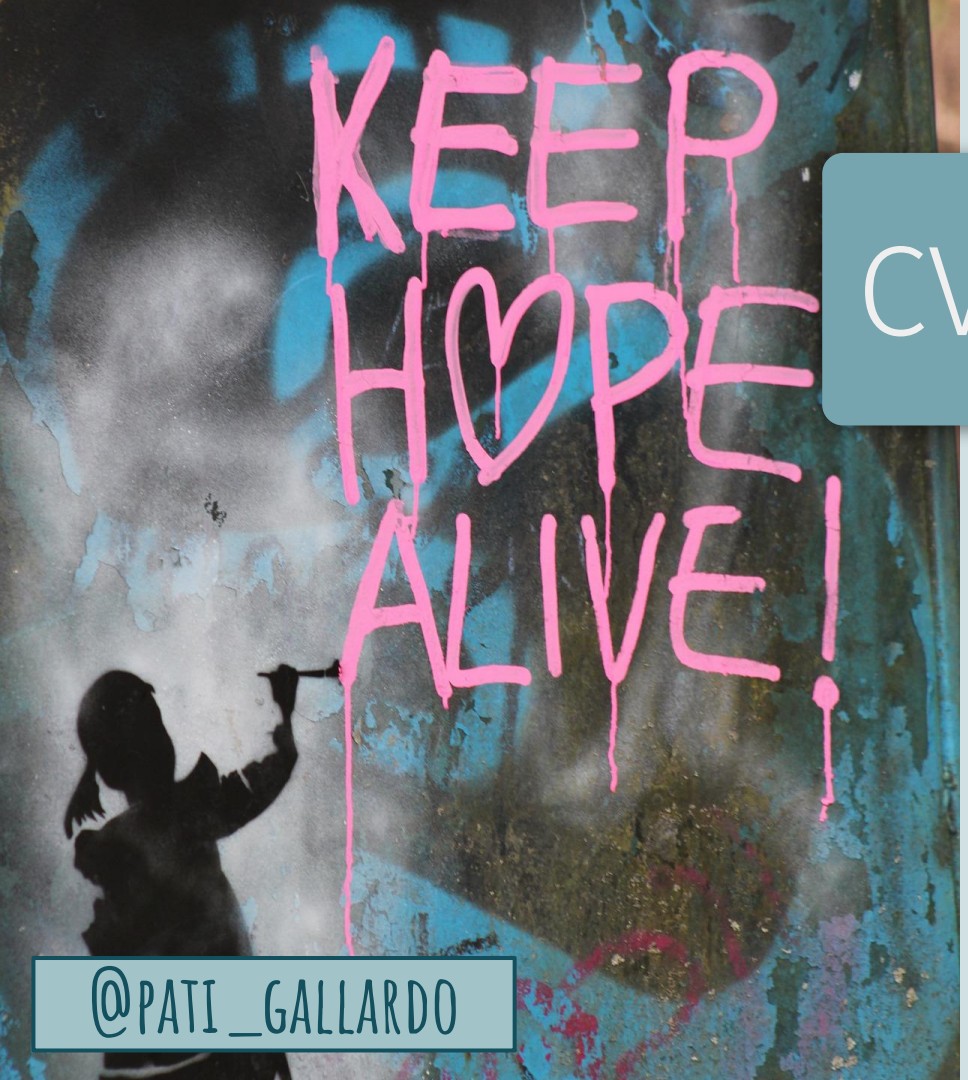
```
$ clang -o chars_written chars_written_2.c
$ ./chars_written
                                        1
42
```

Systems Programming



Binary Exploitation


TurtleSec
Apple iOS
CVE-2021-30800
@PATI_GALLARDO
79

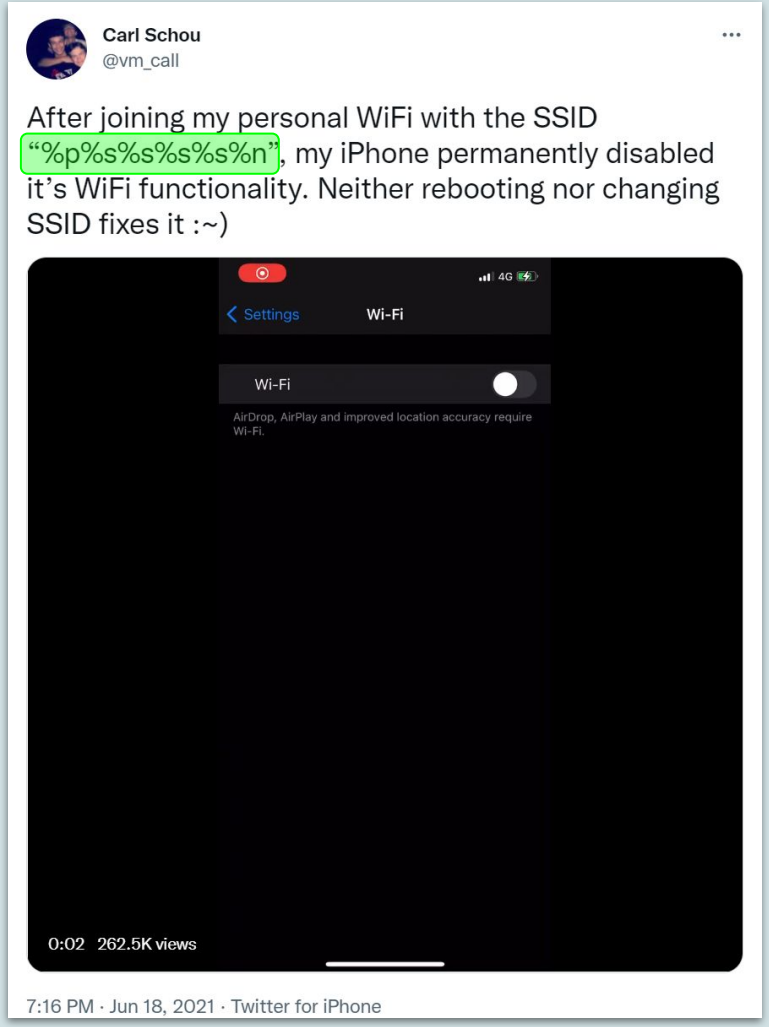


TurtleSec

CVE-2021-30800

"Joining a malicious Wi-Fi network may result in a denial of service or arbitrary code execution."

CVE-2021-30800

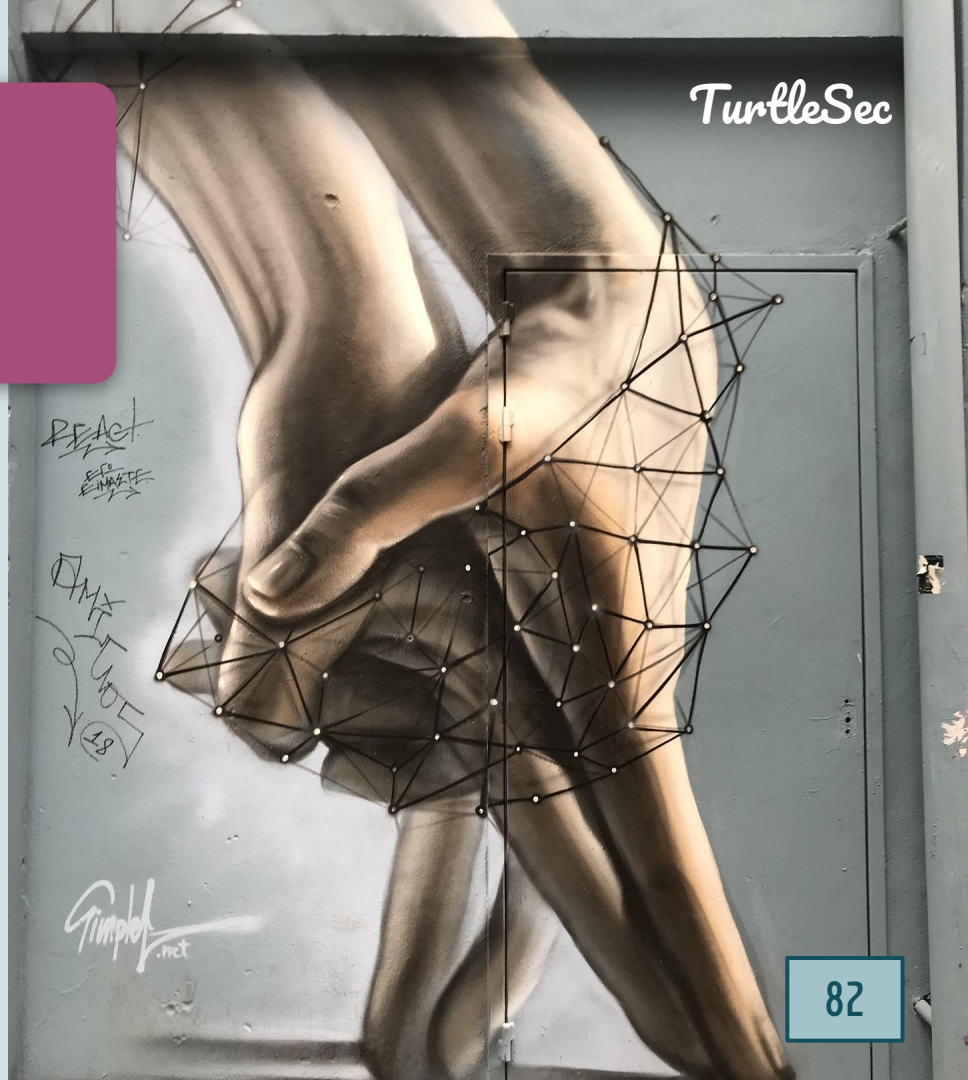


TurtleSec

@PATI_GALLARDO

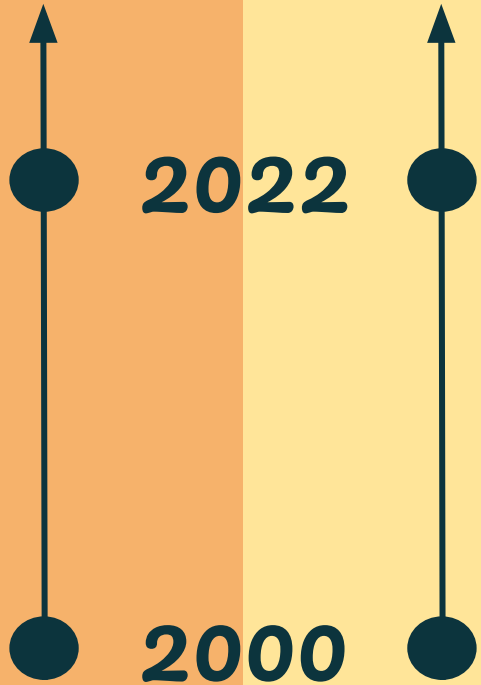
How to find them

Address Sanitizer
GCC & Clang:
-Wformat=2



@PATI_GALLARDO

Systems Programming



Binary Exploitation

Living in the future

@PATI_GALLARDO

TurtleSec

Cross community learning

@PATI_GALLARDO

Questions?

Photos from pixabay.com and Wikipedia

Patricia Aas, TurtleSec

@PATI_GALLARDO

TurtleSec


THE COLOR CHILDREN
PROTECT NATURE
TORTUGA
@PATI_GALLARDO
Turtle Sec