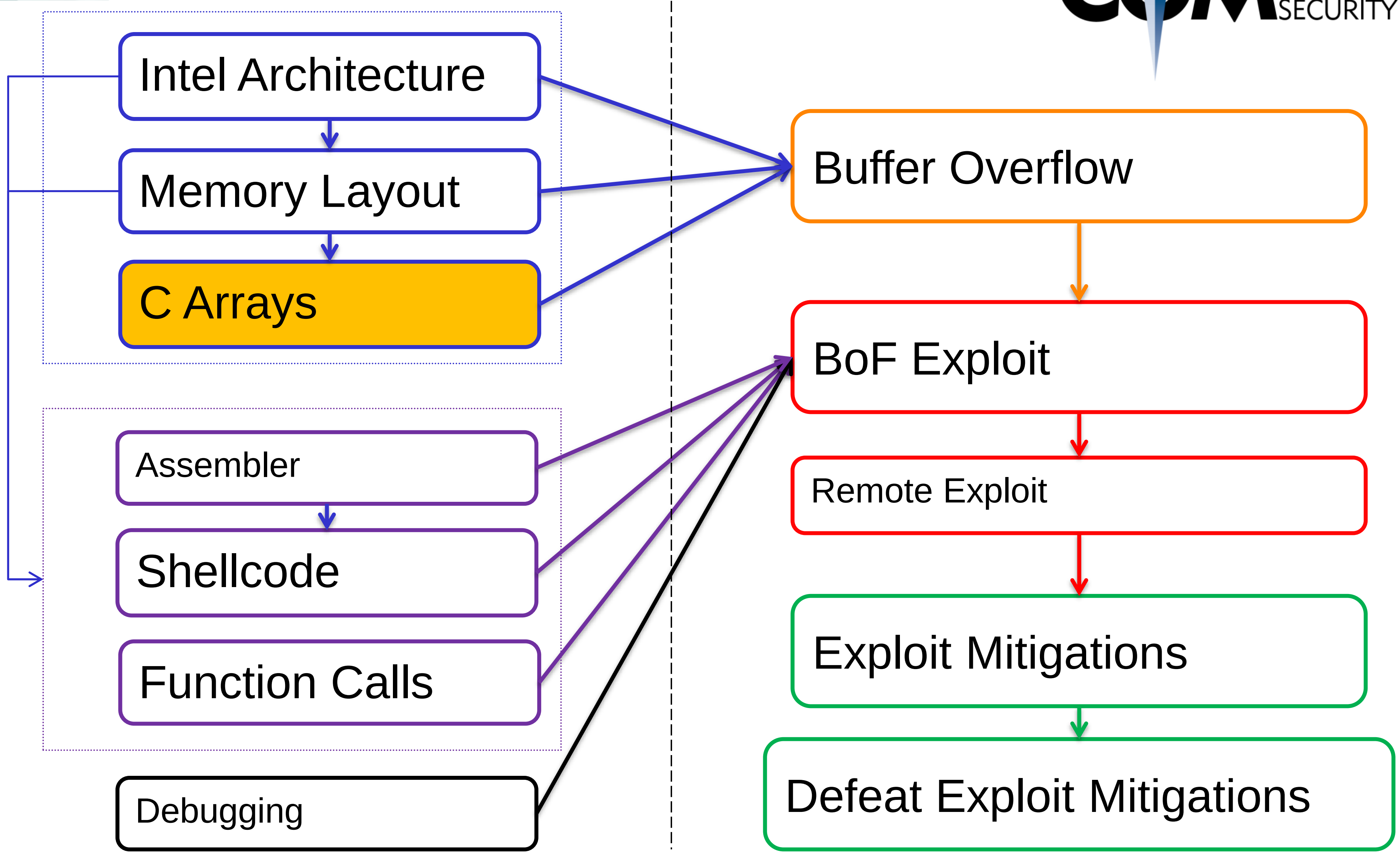
C Arrays and Pointers

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

Content


Intel Architecture
Memory Layout
C Arrays
Assembler
Shellcode
Function Calls
Debugging
Buffer Overflow
BoF Exploit
Remote Exploit
Exploit Mitigations
Defeat Exploit Mitigations

C Arrays & Pointers

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

C Arrays & Pointers

Valid C code:

```
int array[5] = {1, 2, 3, 4, 5};

array[0] = 0;
array[4] = 0;
```

C Arrays & Pointers

Valid C code:

```
int array[5] = {1, 2, 3, 4, 5};

array[0] = 0;
array[4] = 0;

array[5] = 0;
array[-1] = 0;
array[100] = 0;
printf("%i", array[1024]);
```

"Valid"!

C Arrays & Pointers

Valid C code:

```
int array[5] = {1, 2, 3, 4, 5};

int *a = array;
a += 100;
*a = 0;
```

```
array      = a           = 0x1000
array[2]   = a + 2 * 4   = 0x1008
array[100] = a + 2 * 100 = 0x10C8


(int is 32 bit = 4 bytes)
```

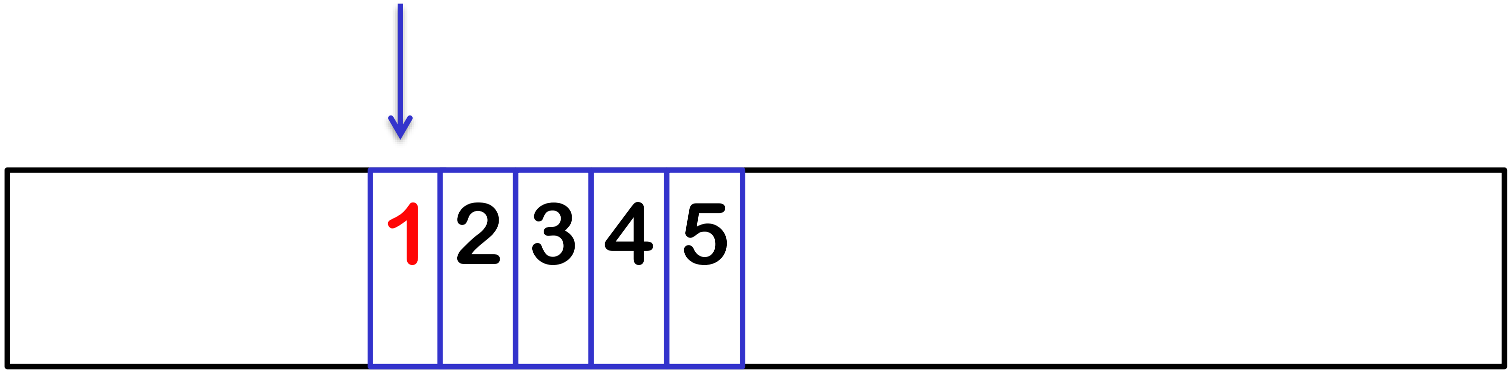
C Arrays & Pointers

Valid C code:

```
    int array[5] = {1, 2, 3, 4, 5};

    int *a = array;
```

***array = *a = 1**

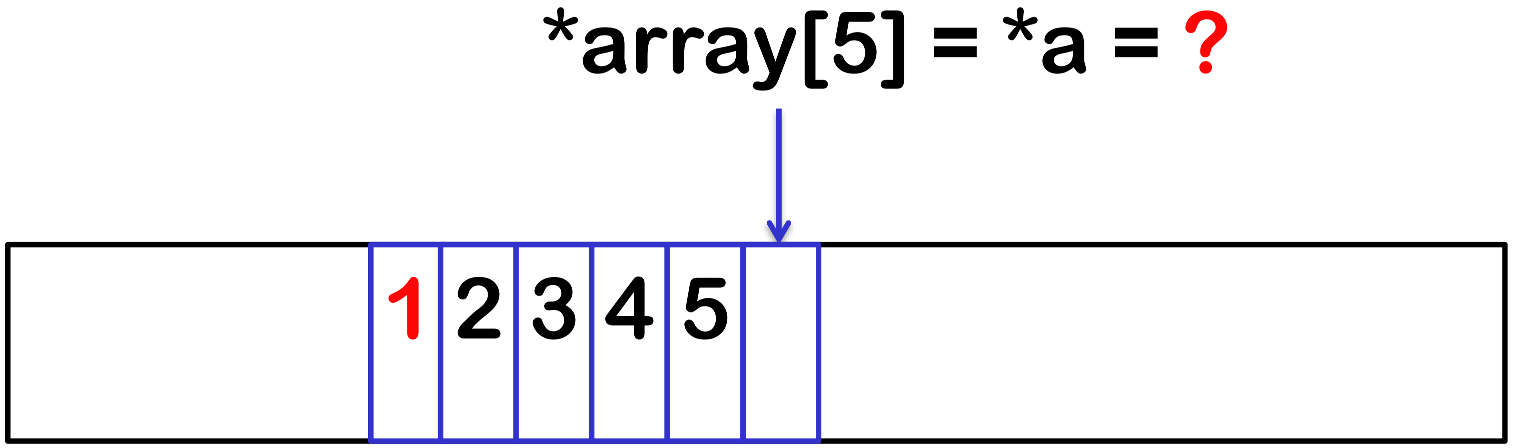


C Arrays & Pointers

Valid C code:

```
    int array[5] = {1, 2, 3, 4, 5};

    int *a = array[5];
```



C Arrays & Pointers

Other c code:

```
int a = 42;
int *b = &a;

printf(“%i”, a);     // 42
printf(“%i”, *b);    // 42

b++;

printf(“%i”, *b);    // ??
```

C Arrays & Pointers

Other c code:

```
int a = 42;
int *b = &a;

printf("%i", a);      // 42
printf("%i", &a);     // 0x1000
printf("%i", b);      // 0x1000
printf("%i", *b);     // 42

b++;

printf("%i", b);      // 0x1004
printf("%i", *b);     // ??
```

C Arrays & Pointers

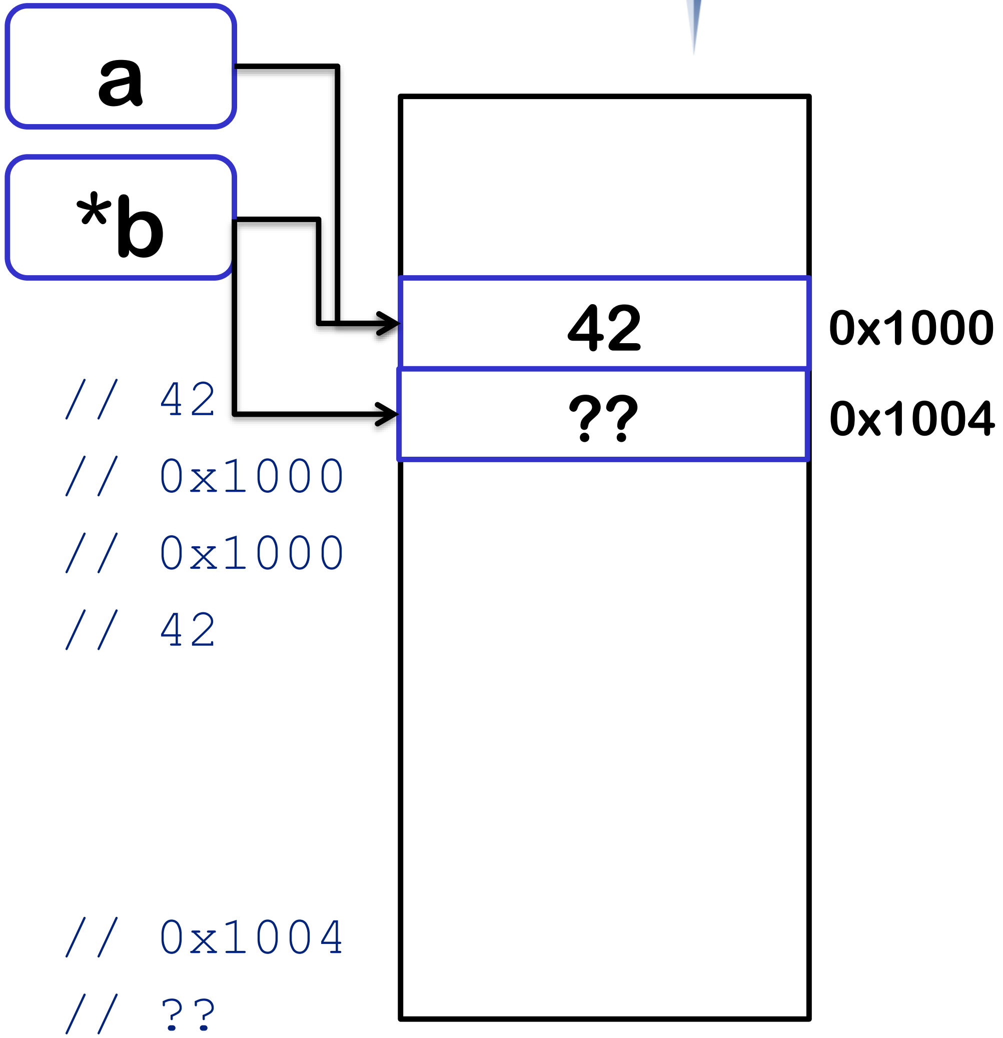
Other c code:

```
int a = 42;
int *b = &a;

printf("%i", a);      // 42
printf("%i", &a);     // 0x1000
printf("%i", b);      // 0x1000
printf("%i", *b);     // 42

b++;

printf("%i", b);      // 0x1004
printf("%i", *b);     // ??
```



strcpy()

What is a common vulnerability?

```
strcpy(destination, source);
strcpy(d, “Hallo”);
```

What is a common vulnerability?

```
strcpy(destination, source);
strcpy(d, “Hallo”);
```

How much does strcpy() actually copy?

- Until source “ends”
- Where is the end?
- 0 byte \x00

```
“Hallo\x00”
```

Exploitation Basics

strcpy() does not care about destination size

At all

```
char destination[8];
char source[16] = "1234567890123456"

strcpy(destination, source);
```

Exploitation Basics

strcpy() does not care about destination size

At all, because:

```
char destination[8];
char *d = &destination;
char source[16] = “1234567890123456”

strcpy(d, source);
```

Non-Arrays in C

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

Non-Arrays

C has:

- Basic Types (int, float)
- Enumerated Types
- Void Type (void)
- Derived Types

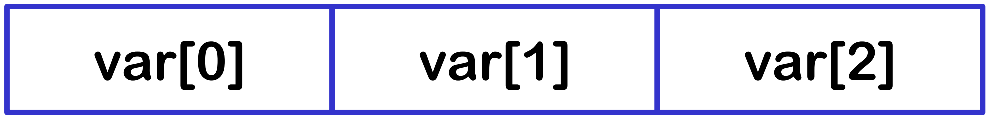
Derived types:

- Pointers
- Arrays
- Structure
- Union
- Function

Non-Arrays

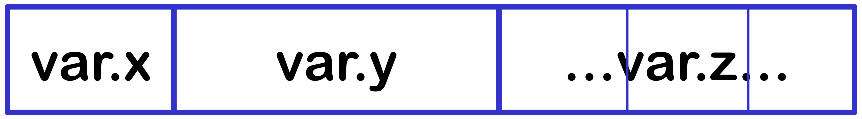
Arrays: Multiple elements of the **same type** behind each other

```
XXX var[3]:
```



Structs: Multiple elements of **different types** behind each other

```
struct var {
  short x;
  long y;
  char z[3];
}
```



Enum is a special case of integer

Union is a special case of struct

Non-arrays

Remember:

Basic types are stored **in memory**, and can be loaded into **registers**

- Pointers are a bit special basic type (they can be dereferenced), but are otherwise identical

Derived types are stored **in memory**, and **contain basic types**

- They cannot be loaded into a register, only some of their content can

Both are stored somewhere in memory, and therefore have an **address**.

Basic types are **modified in registers**

- Load from memory to register, modify, store into memory

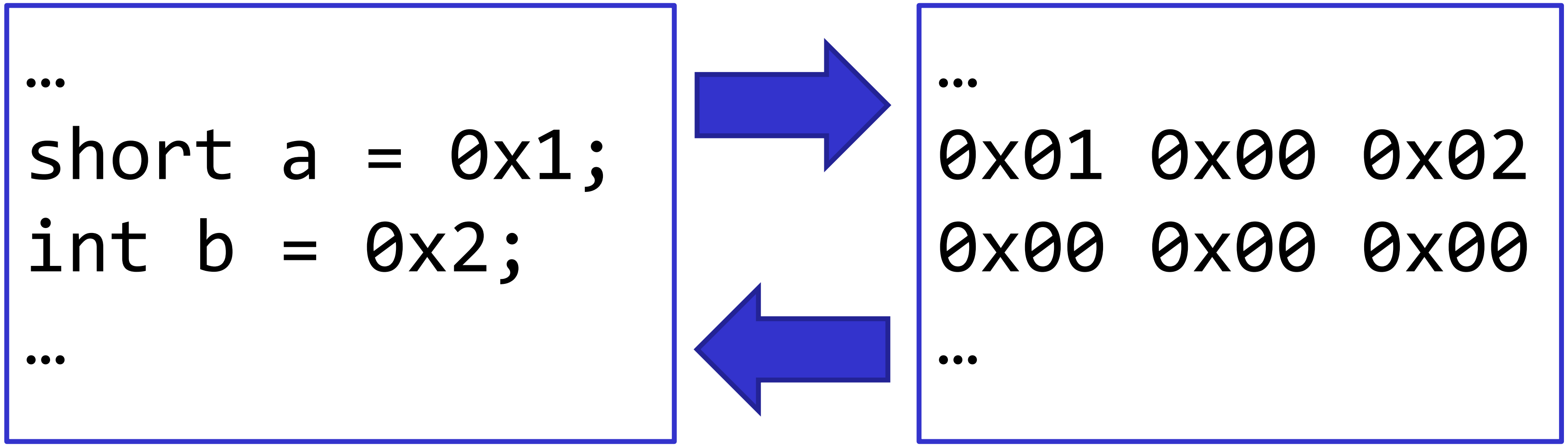
Non-arrays

Developers:

- The memory holds some variables of mine, which hold my data

Hackers:

- The memory contains data, which is associated with some variables



```
…
short a = 0x1;
int b = 0x2;
…
```

```
…
0x01 0x00 0x02
0x00 0x00 0x00
…
```

Conclusion

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel	+41 55 214 41 60
Fax	+41 55 214 41 61

team@csnc.ch
www.csnc.ch

Exploitation Basics

Recap:

- C does not care about buffer boundaries
- strcpy() does not care about size of destination buffer (only 0-byte in source buffer)
- One buffer can overflow into another buffer
- Local variables/buffers are adjoin to each other
- Pointer can point to any memory address