School of Computing
FACULTY OF ENGINEERING & PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

COMP3911 Secure Computing

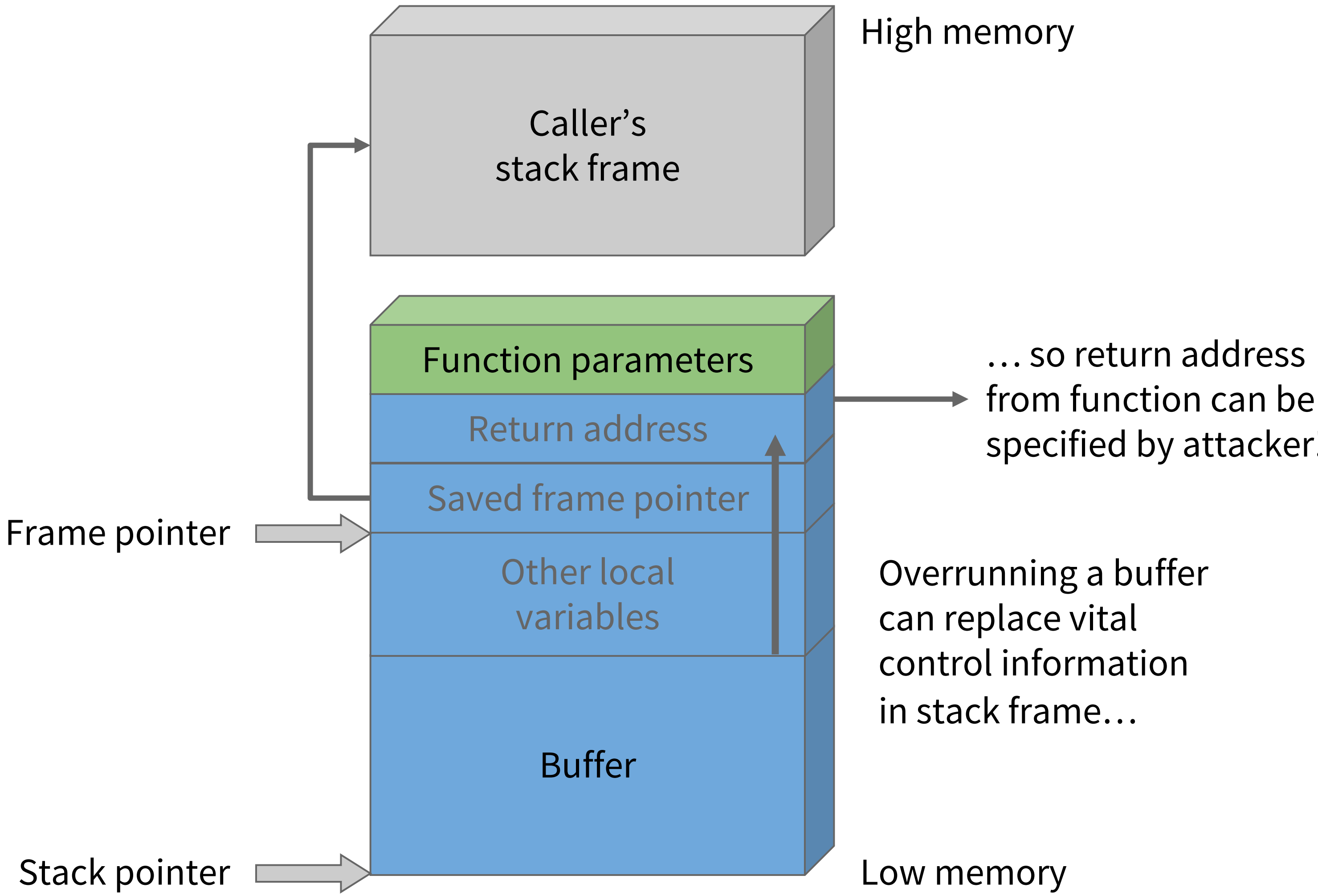
14: Other Low-Level Vulnerabilities

Nick Efford

https://comp3911.info

Last Time: Buffer Overruns

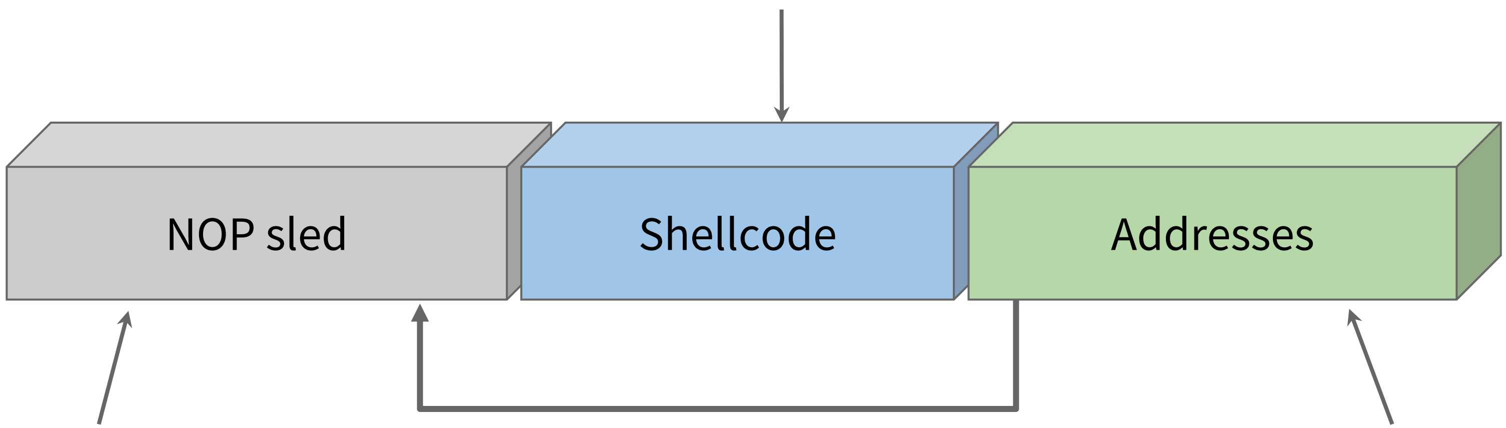
UNIVERSITY OF LEEDS



Last Time: Buffer Overruns

UNIVERSITY OF LEEDS

malicious machine code - typically attempting to run a shell on the attacked machine



NOPs do nothing, thus providing a safe ‘landing zone’ for the return from the function

address needs to lie within the ‘NOP sled’ at start of the payload

address to jump to, repeated to maximise chance of overwriting return address on stack

Last Time: Buffer Overruns

UNIVERSITY OF LEEDS

- Under the right circumstances, small stack frames or very small (even single-byte) overruns can be exploited
- Heap is also vulnerable, for same reasons (no bounds checking + mixing of user data and control info)
- Special compiler options can add bounds-checking code or stack protectors to executables…
- … and hardware & OS can also help
    - Flagging stack & heap as non-executable
    - Address space layout randomisation

Objectives

UNIVERSITY OF LEEDS

- To explore how C's `printf` functions can be abused (**Sin 6** of *24 Deadly Sins*)
- To consider vulnerabilities in the way that C++ supports dynamic binding of method calls
- To understand the risks of careless integer arithmetic (**Sin 7** of *24 Deadly Sins*)
- To discuss mitigations of these issues

Format String Bugs

UNIVERSITY OF LEEDS

- Affect `*printf` functions from C standard library
- Widely known about since 1999
- Source of many vulnerabilities in the past
- Easily spotted by source code auditing tools
- Most compilers will now warn you if there is a risk…

Format String Bugs

UNIVERSITY OF LEEDS

Correct way to print a string of user input:

```
printf("%s", user_input);
```

format string, containing a **format directive**

value to be formatted

Incorrect way to print a string of input:

```
printf(user_input);
```

what happens if `user_input` contains formatting directives?

Information Disclosure Threat

- In the absence of suitable arguments, `printf` takes the values needed by the format string **from the stack**
- So if the attacker can provide the format string as input and can see the output from `printf`, they have a way of probing stack contents
- … or even printing value at an arbitrary location

Code Execution Threat

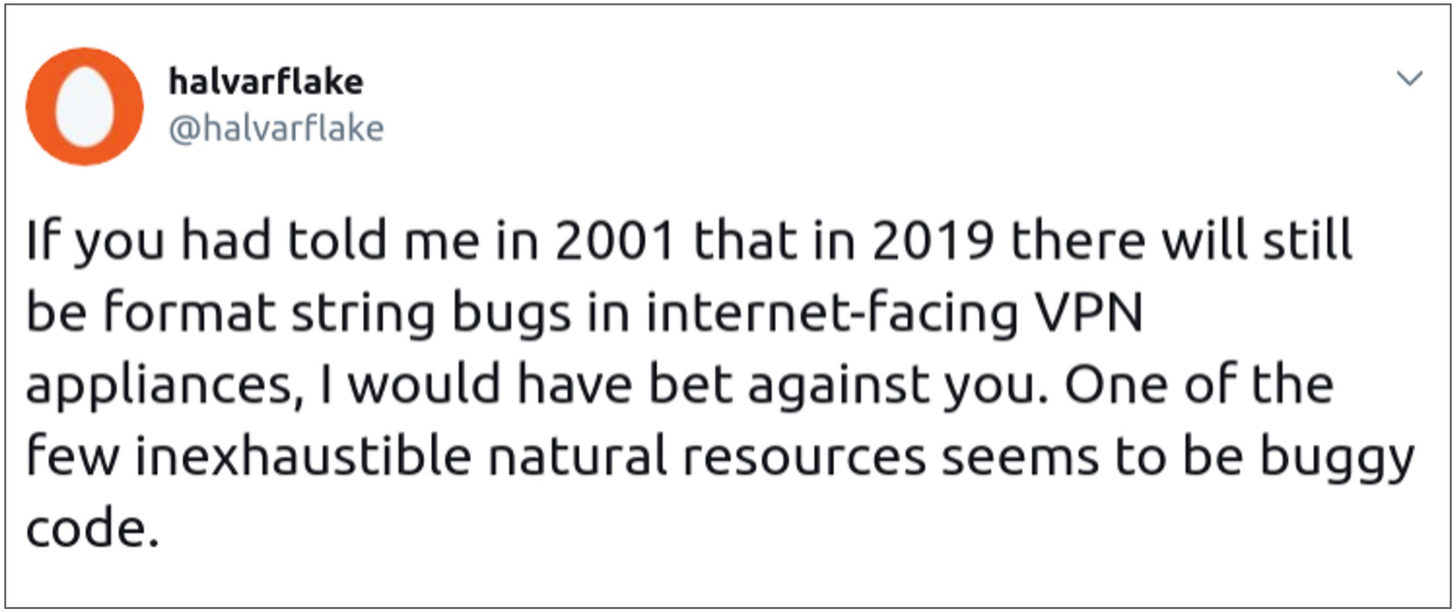
UNIVERSITY OF LEEDS

- `%n` format directive writes the number of characters formatted so far to a given memory address
- If no address is supplied as an argument to `printf`, it is taken from the stack
- Format string can be constructed by attacker so as to inject an address onto the stack <u>and</u> generate the value to be stored at that address
- … which means it is possible to overwrite function return addresses, function pointers, etc – pointing them to shellcode

Palo Alto VPN Bug

[Attacking SSL VPN… with Uber as Case Study!]

(Orange Tsai & Meh Chang, July 2019)


halvarflake
@halvarflake

If you had told me in 2001 that in 2019 there will still be format string bugs in internet-facing VPN appliances, I would have bet against you. One of the few inexhaustible natural resources seems to be buggy code.

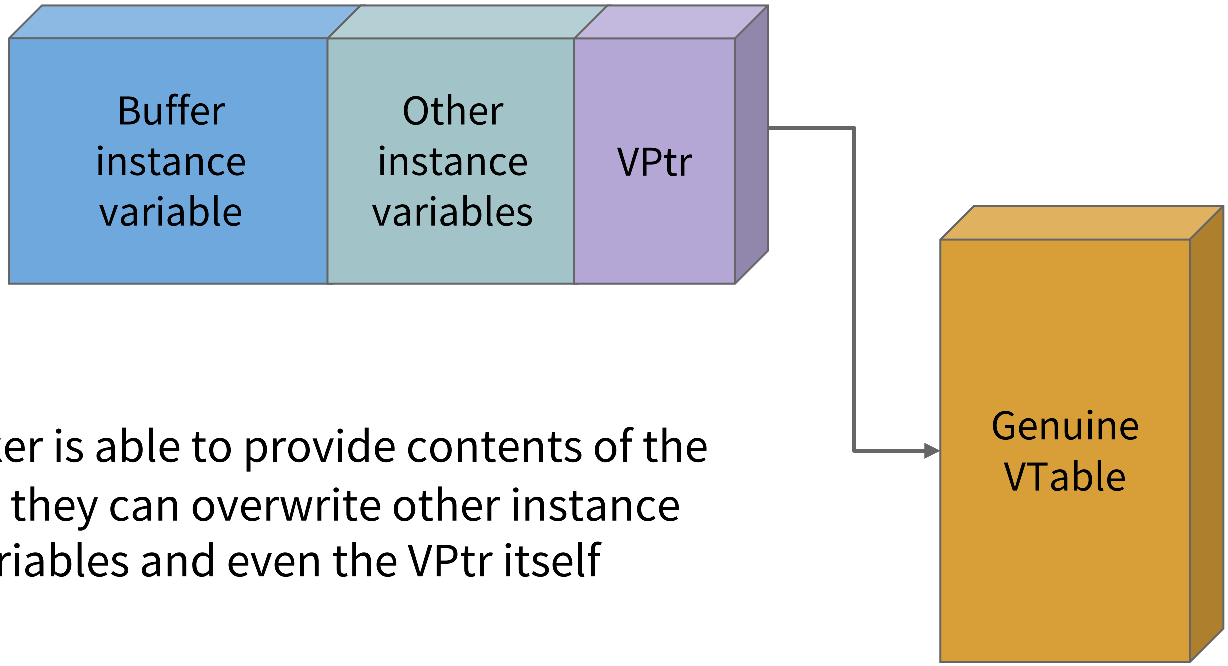
C++ VTable Attacks

UNIVERSITY OF LEEDS

- How does dynamic binding work in C++?
  - For each class that contains virtual methods, we need an array of function pointers: the **VTable**
  - Each instance of that class containers a pointer to the VTable: the **VPtr**
- VPtr can be overwritten…
- … giving attacker control over what happens when those methods are invoked on an object

C++ VTable Attacks

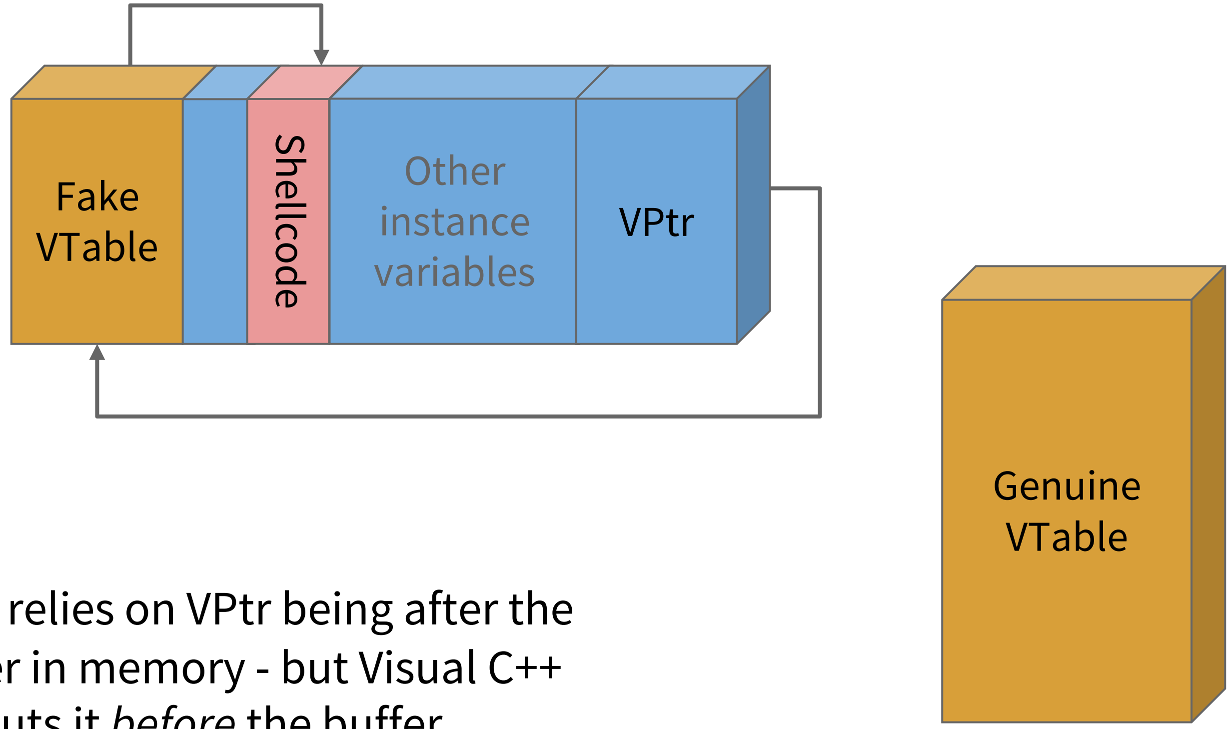
UNIVERSITY OF LEEDS



If attacker is able to provide contents of the buffer, they can overwrite other instance variables and even the VPtr itself

C++ VTable Attacks

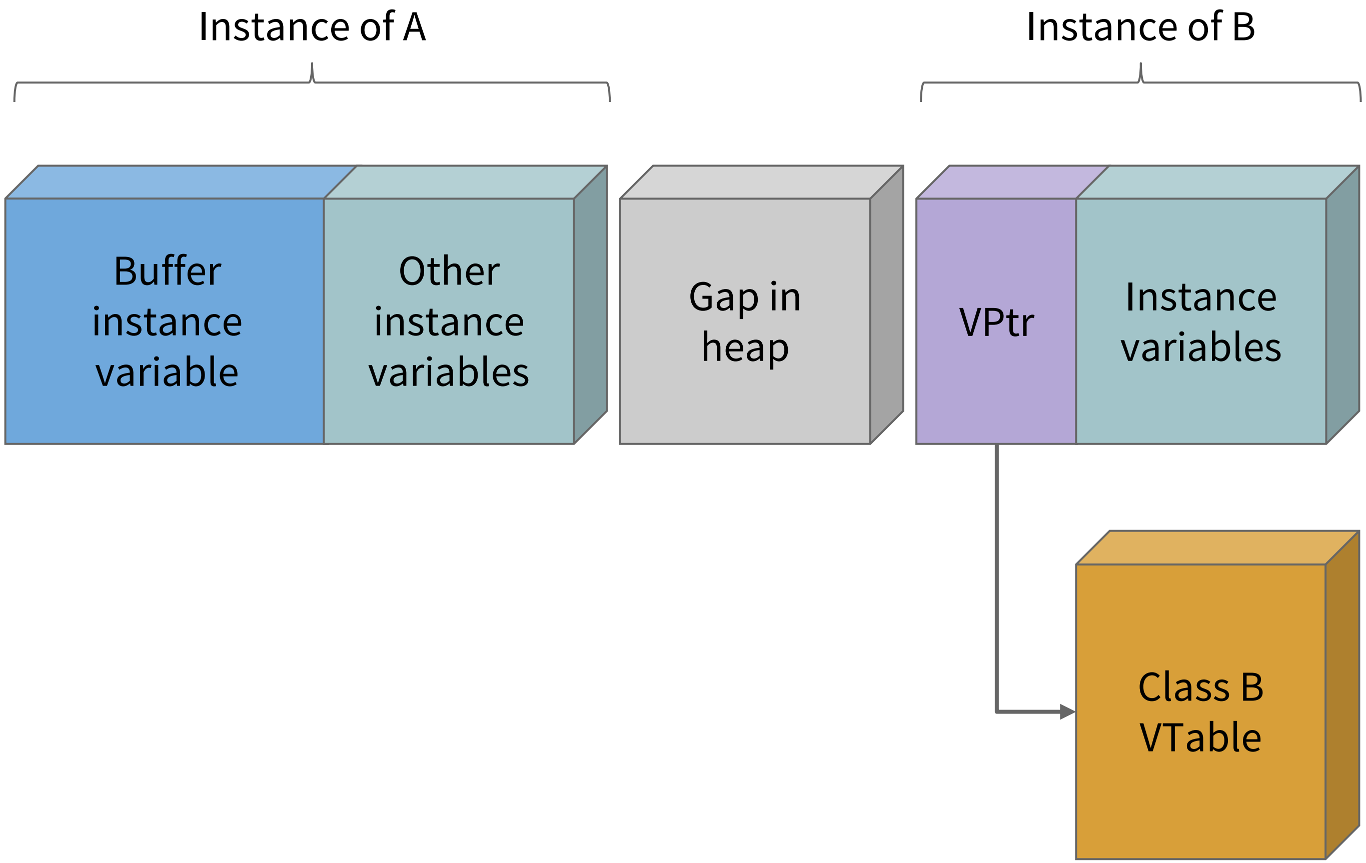
UNIVERSITY OF LEEDS



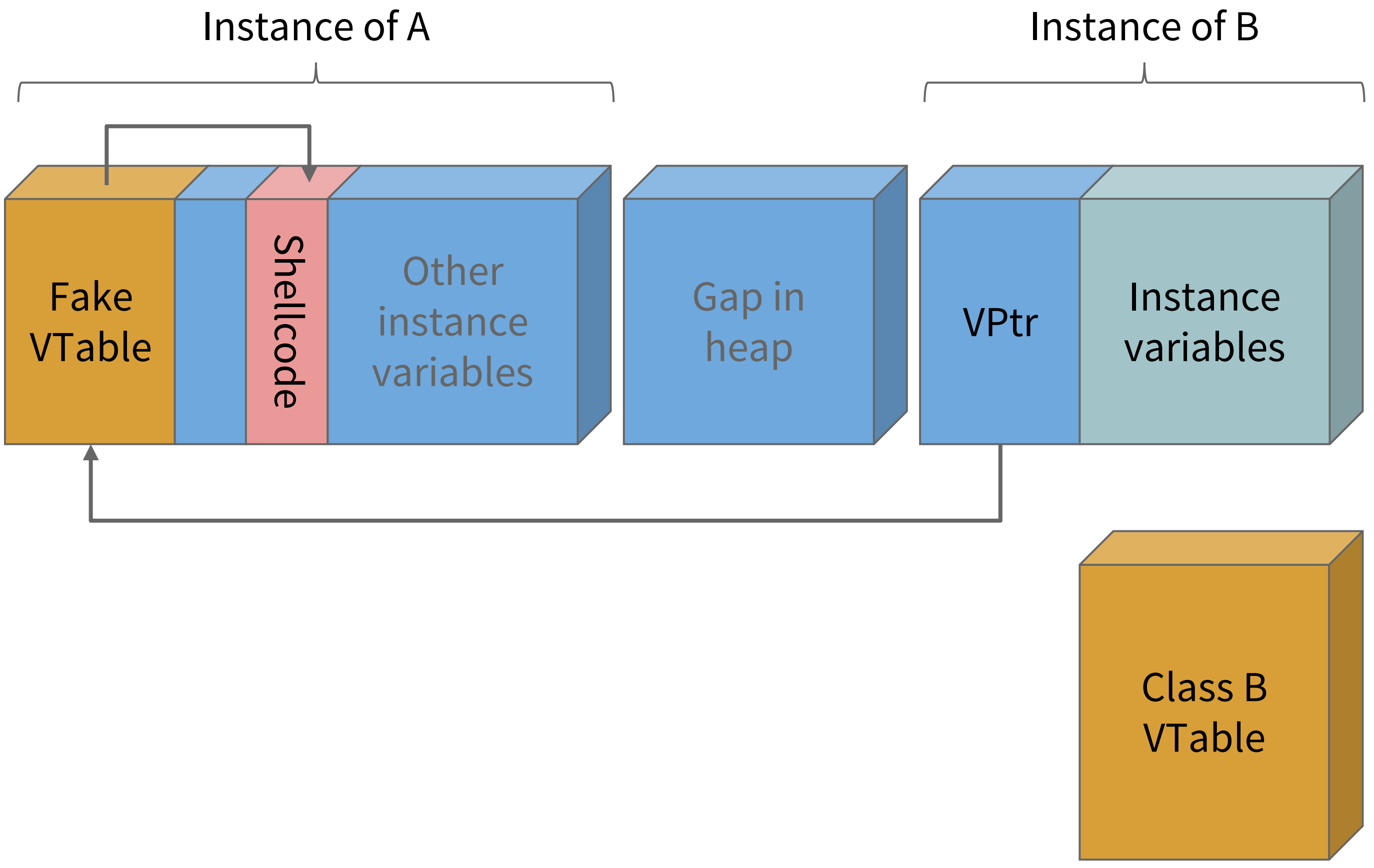
Note: relies on VPtr being after the buffer in memory - but Visual C++ puts it *before* the buffer…

Defeating Visual C++

UNIVERSITY OF LEEDS



Defeating Visual C++



Quick Quiz

UNIVERSITY OF LEEDS

1. $300 \times 300 = ?$

1. $-15{,}000 - 25{,}000 = ?$

1. $32{,}767 + 1 = ?$

Example

UNIVERSITY OF LEEDS

```
double* allocData(size_t n)
{
  double* data = new double[n];
  if (data == NULL) {
    throw ApplicationException("out of memory");
  }
  return data;
}
```

`size_t` is an alias for `unsigned int`, which means that the value of `n` can be in the range 0 to $2^{32}-1$

If we try to increase such a value beyond that upper limit, the calculation 'wraps around' to the lower end of the range…

Example

UNIVERSITY OF LEEDS

If a `double` value occupies 8 bytes,

and if `new` computes `n*sizeof(double)` in order to allocate memory,

1. How many bytes will be allocated when n = 536,870,911 ?

1. What about when n = 536,870,913 ?

Another Example

UNIVERSITY OF LEEDS

```
bool concatData(float* buf1, size_t length1,
                float* buf2, size_t length2)
{
  float tmp[256];

  size_t total = length1 + length2;
  if (total > 256)
    return false;

  int i = 0;
  for (; i < length1; ++i)
    tmp[i] = buf1[i];

  for (; i < total; ++i)
    tmp[i] = buf2[i-length1];
  ...
}
```

`length1` = 320
`length2` = 4,294,967,232
⇒ `total` = 256

`tmp` overrun by **64 bytes**

Does This Happen Often?

UNIVERSITY OF LEEDS

Software	Identifier	Month
OpenSSL 1.0.2x, 1.1.1i	CVE-2021-23840	January
iOS 14, iPadOS 14, macOS Big Sur…	CVE-2020-27911	October
Linux kernel 4.4 through 5.7.1	CVE-2020-13974	June
SQLite 3.32	CVE-2020-13434	May
iOS 13.5, iPadOS 13.5…	CVE-2020-9875	March
libssh2	CVE-2019-13115	June
Linux kernel (TCP code)	CVE-2019-11477	April
Chrome download manager	CVE-2019-5829	January

(small selection, found via https://cve.mitre.org)

Avoiding Over/Underflow Issues

UNIVERSITY OF LEEDS

- Enable all relevant compiler warnings
- Use unsigned types where possible
- Examine *very* carefully any code that calculates array indices or buffer lengths
- Perform comparisons properly – e.g., by checking explicitly for wrap-around:

```
if (a+b >= a && a+b < MAX_VALUE) { ... }
```

- Use dedicated functions, designed to perform arithmetic safely, throughout your code

Example: Fix for CVE-2016-9387 (Integer Overflow in JasPer JPEG library)

http://bit.ly/ioflowfix

6 src/libjasper/jpc/jpc_dec.c

```diff
@@ -1195,6 +1195,7 @@ static int jpc_dec_process_siz(jpc_dec_t *dec, jpc_ms_t *ms)
 	int htileno;
 	int vtileno;
 	jpc_dec_cmpt_t *cmpt;
+	size_t size;
 
 	dec->xstart = siz->xoff;
 	dec->ystart = siz->yoff;
@@ -1231,7 +1232,10 @@ static int jpc_dec_process_siz(jpc_dec_t *dec, jpc_ms_t *ms)
 
 	dec->numhtiles = JPC_CEILDIV(dec->xend - dec->tilexoff, dec->tilewidth);
 	dec->numvtiles = JPC_CEILDIV(dec->yend - dec->tileyoff, dec->tileheight);
-	dec->numtiles = dec->numhtiles * dec->numvtiles;
+	if (!jas_safe_size_mul(dec->numhtiles, dec->numvtiles, &size)) {
+		return -1;
+	}
+	dec->numtiles = size;
 	JAS_DBGLOG(10, ("numtiles = %d; numhtiles = %d; numvtiles = %d;\n",
 	  dec->numtiles, dec->numhtiles, dec->numvtiles));
 	if (!(dec->tiles = jas_alloc2(dec->numtiles, sizeof(jpc_dec_tile_t)))) {
```

General Strategies

UNIVERSITY OF LEEDS

- Enable all relevant protections provided by hardware and the OS (e.g., NX bit & ASLR)
- Change our development practices
  - Language choice
  - Appropriate use of compiler options
  - Proper use of standard library
  - Special ‘safe’ libraries
  - Do integer arithmetic correctly!

Library Usage

UNIVERSITY OF LEEDS

- When writing C++, use its standard library, not C's!
  - `std::string` instead of `char*`, etc
- Use C standard library cautiously
  - Never, ever use `gets` to read a string!
  - Avoid `strcpy`, `strcat`, etc
    - Bounds-limited versions `strncpy`, `strncat` are a bit safer but can still be abused…
  - Take care with `printf` – never allow a user-supplied string to be a format specifier

Summary

UNIVERSITY OF LEEDS

We have seen that

- C's `*printf` functions place far too much trust in format strings, allowing unrestricted manipulation of memory
- Buffer overruns can affect the VTables of C++ classes
- Problems with fixed-precision integer arithmetic can also trigger buffer overruns
- There are various defences against low-level issues – e.g., using hardware & OS features, compiler options, safer languages or libraries, or just being more careful

Follow-Up / Further Reading

UNIVERSITY OF LEEDS

- Sins 6 & 7 of *24 Deadly Sins of Software Security*
  - See also Sin 8 for discussion of other C++ problems
- Examine US-CERT bulletins or the SecurityFocus BugTraq archives for examples of low-level vulnerabilities