

## (In)Security in C++

### Secure Coding Practices in C++

*(In)Security in C++* teaches C++ developers fundamental concepts from Exploit Development, and uses these concepts to demonstrate common vulnerabilities in C++ codebases. This background is used to help the students to view their code from an attacker's perspective. They develop a sense of what common vulnerable constructs in C++ look like, and also which tools can help them find different types of vulnerabilities in their existing code bases.

---

#### Attendees can expect to gain

- A basic understanding of the mindset of an exploit developer
- A good overview of tooling that can be used to find vulnerable constructs
- An understanding of the key things to look for in code reviews
- A sound overview of Secure Coding Practices in C++

#### Secure Coding Practices

The Secure Coding Practices taught are largely based on the C++ Core Guidelines, the Common Weakness Enumeration (CWE) and the SEI CERT Coding Standards for C++.

#### Duration and Format

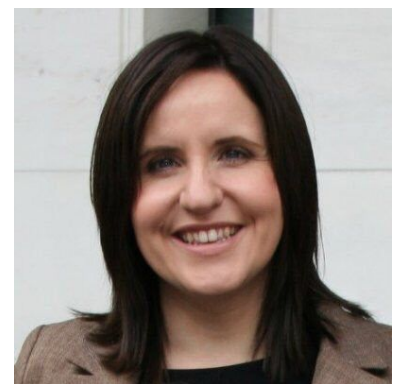
Attendees are assumed to be proficient C++ programmers. The course is over 4 days and consists of lectures and corresponding exercises, either done by the students on their computer, or mob-style on the trainer's computer.

---

#### Trainer

Patricia Aas is a frequent speaker at conferences including CppCon, ACCU, C++OnSea, NDC Security and NDC Oslo, on subjects ranging from Sandboxing in Chromium to Vulnerabilities in C++. She has taught a range of subjects in Computer Science at the University of Oslo.

Patricia has a masters degree in Computer Science and nearly 15 years professional experience as a programmer, most of that time programming in C++. During that time she has worked in codebases with a high focus on security: two browsers (Opera and Vivaldi) and embedded Cisco telepresence systems.



---

#### Tools and Techniques

- Exploitation: Stack Buffer Overflow Exploit, Return Oriented Programming and Format String Exploit
- Vulnerability Detection: Static Analysis, Warnings, Sanitizers and Fuzzers
- Platform Mitigation: Stack Canaries, Address Space Layout Randomization (ASLR), Non-executable memory

# (In)Security in C++

## Secure Coding Practices in C++

### Modules

Introduction Bug vs Vulnerability	Undefined Behavior and Compiler Optimizations	A Simple Exploit
Platform Mitigation and Tooling	Exploitation Thinking	Vulnerable Code Memory I
Vulnerable Code Memory II	Vulnerable Code Numbers	Insecure Coding I Language
Insecure Coding II Practices	Secure Coding Prefer C++ to C	Secure Coding Resource Management
Secure Coding Pitfalls	Secure Coding Functionality	Peer Review Things to consider

### Vulnerabilities Covered

Stack Buffer Overflow (CWE-121)	Heap Buffer Overflow (CWE-122)
Buffer Underflow (CWE-124)	Use After Free (CWE-416)
Double Free (CWE-415)	Unsigned Integer Wraparound (CWE-190)
Signed Integer Overflow (CWE-190)	Numeric Truncation (CWE-197)
Incorrect Type Conversion (CWE-704)	Uncontrolled Format String (CWE-134)