Check Point
SOFTWARE TECHNOLOGIES LTD

DEFENSIVE PROGRAMING

Core C++ Meetup

Noam Weiss | Architect

WELCOME TO THE FUTURE OF
CYBER SECURITY
POWERED BY CHECK POINT INFINITY

CLOUD • MOBILE • THREAT PREVENTION

Agenda

- What is Defensive Programing
- Some Defensive Programing Theory
- Case Studies

What is Defensive Programing

- Does it includes input validation?
  - How about input sanitation?

- How does it relates to contracts?

- Which of these actions is defensive:
  - Assertion
  - Logging
  - Sanitation
  - Exception

What is Defensive Programing

For Wikipedia:

"

Defensive programming is a form of defensive design intended to ensure the continuing function of a piece of software under unforeseen circumstances.

Overly defensive programming, however, may safeguard against errors that will never be encountered, thus incurring runtime and maintenance costs. There is also the risk that the code traps or prevents too many exceptions, potentially resulting in unnoticed, incorrect results.

"

The Problem

- Congratulations: Your code works!
  - It’s fast! It’s clean! It’s great!
  - Assuming everything behaves correctly…

- So you start hardening your code…
  - Which raise some question:
    - What you should be hardening against?
    - How you should be hardening?
  - Your code doesn’t look so great anymore…

Why does it all go wrong?

- That’s not how design should work.
- Separating “functionality” from “security”.
- Not having a clear policy as to error handling:
  - When
  - Where
  - How

How to think about Defensive Programing

- Strategic:
  - Error handling policy.
    - Input validation policy.
    - Exception policy.
  - Contract policy (wide\narrow).

- Tactical:
  - API (public\private, memory ownership, and so on).
  - Maintaining object consistency.
  - Etc.

Two aspects of Defensive Programming

- Protect against the client misuse
  - **Nobody reads the manual**
  - Murphy’s input law
  - Simplicity

- Protect the client
  - **Nobody checks if an operation succeeded**
  - Invariants
  - Simplicity

(Some of) What C++ can do for us

- Public vs. Private methods
- `const`
- Passing arguments using `&` instead of `*`
- RAII

- Veridic Templates vs. Macros
- Smart Pointers
  - `std::unique_ptr`
  - `std::shared_ptr`
  - `std::function`
- `noexcept`
- `static_assert`

CASE STUDY 1

Cereal

- C++11 Library for serialization
  - https://uscilab.github.io/cereal/

- Supports many formats
  - Binary
  - XML
  - JSON

- Throws exceptions on errors

The problem

- We want to read a field of a JSON input
  - It is legitimate for the field not to be present in the JSON
- If the field isn't in the JSON, cereal will throw an exception
- After caching the exception we want to read the next field
- But we can't…

Library code (simplified)

Check Point
SOFTWARE TECHNOLOGIES LTD

```
struct JSONInputArchive
{
    void startNode() {
        if( itsNextName ) search();
        …
    }

    void search() {
        …
        if( !found ) throw Exception("Parsing failed");
        itsNextName = nullptr;
    }
}
```

Partial list of functions involved

```
JSONInputArchive::search

JSONInputArchive::Iterator::search

JSONInputArchive::startNode

JSONInputArchive::setNextName

template <class T> JSONInputArchive::loadValue

template <class T> prologue(JSONInputArchive &, NameValuePair<T> const &)
```

The “solution”

```
void
serialize(JSONInputArchive &ar)
{
    try {
        …
    } catch (Exception const &e) {
        ar.setNextName(nullptr);
        …
    }
}
```

Insights

- An ounce of prevention is worth a pound of cure

Insights

Check Point
SOFTWARE TECHNOLOGIES LTD

- Beware of Exceptions and object consistency

- Exceptions are not a replacement for error handling policy

Insights

- Consider using wrappers as a protective layer

CASE STUDY 2

The problem

- We received bytes on the wire and we want to parse them into a structure called Packet
  - `Packet(const char *bytes, uint number_of_bytes);`
- However, the parsing may failed
  - Either due to network problems or malicious attack
- So how do we handle such cases?
  - Constructors can’t return an error value

Option 1: Handle the problem internally

- On error the constructor should set the Packet to indicate that an error has occurred

- Cons:
  - People will keep forgetting to check the error status
  - Anybody who receive a Packet will be suspicious of it
  - Not clear who should actually handle the error

- Generally, don’t do this

Option 2: Use initialization function

- First create the Packet instance, than use another method to initialize the packet.

- Pros:
  - It is clear who should address the error

- Cons:
  - You could have an uninitialized Packet (breaks RAII)
  - There’s an overhead in cases where the input is known to be valid
  - It’s easy to ignore the return value from the initialization

Option 3: Throw an exception

- The constructor should throw an exception on error – let someone else deal with it

- Pros:
  - All packets are always valid

- Cons:
  - No clear owner as to who should actually handle the error
  - People don’t expect constructors to throw
  - There are some delicate points about throwing from a constructor

Option 4: Use a factory method

- Have a static method that returns either the constructed Packet or an error

- Pros:
  - Packets are always valid
  - Clear owner of issues

- Cons:
  - Less standard approach

Option 4: Use a factory method

```
class Packet
{
public:
    Packet(const Packet &);
    static PacketWrapper genPacket(const char *, unit);

private:
    Packet(const char *, unit);
};
```

Option 4: Use a factory method

Check Point
SOFTWARE TECHNOLOGIES LTD

```
auto possible_packet = Packet::genPacket(input.data(), input.size());
if (!possible_packet.ok()) {
    // Error handling
}
Packet incoming_packet = possible_packet.unwrap();




Packet outgoing_packet =
    Packet::genPacket(output.data(), output.size()).unwrap();
```

Insights

- Prefer API that clearly indicates that a problem is possible, and whose responsibility it is to handle it.

- Prefer to always keep your object initialized and consistent

CASE STUDY 3

The problem

- We want to output an object into a stream (std::cout)
- But the print method of the class may fail
- How do we know the state of the stream if such failure occurs?

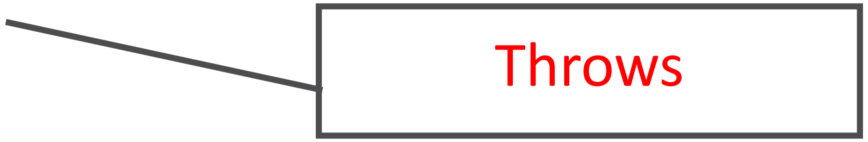
Code Sample

```
class PrintableObject
{
public:
        void print(std::ostream &) const;
};

std::ostream &
operator<<(std::ostream &os, const PrintableObject &obj)
{
    obj.print(os);
    return os;
}
```

Throws

Code Sample

```
class PrintableObject
{
public:
      void print(std::ostream &) const;
};

template <typename AnyPrintableObject>
std::ostream &
operator<<(std::ostream &os, const AnyPrintableObject &obj)
{
    obj.print(os);
    return os;
}
```



WELCOME TO THE FUTURE OF CYBER SECURITY

Strict Contracts – Defend internally

- Have a well defined behavior for print so that it either succeed or leave the stream unchanged

- Pros:
  - clean code
  - Minimal performance impact

- Cons:
  - Easier said than done
  - Put a lot of responsibility on the class developer (relevant especially for templates)

Defend Externally

```
std::ostream &
operator<<(std::ostream &os, const PrintableObject &obj)
{
    std::stringstream temp_output;
    obj.print(temp_output);

    os << temp_output.str();
    return os;
}
```

But even that doesn’t really solves the problem

```
try {

    std::cout << "My first PrintableObject: " << obj1
        << ", my second PrintableObject: " << obj2 << std::endl;

} catch (PrintableObjectException &exception)
{
}
```

Kicking the ball

- Don’t let the function fail, instead have a default action done
  - Print to the stream “<<<Error>>>”

- Pros:
  - For the rest of the system, it looks like nothing happened

- Cons:
  - Can mask real problems in the code
  - Can cause problems if another code expect the “real” output
  - Not clear what the default action should be

Insight

Check Point
SOFTWARE TECHNOLOGIES LTD

- Sometimes there are no perfect solutions

- Be wary of APIs that “mustn’t fail”

- Contracts are efficient but hard to enforce
- Safeguards are easy to enforce but inefficient

Summary

- Error Handling should be part of the design
- Most problems can be avoided by ensuring object consistency

Check Point
SOFTWARE TECHNOLOGIES LTD

THANK YOU

WELCOME TO THE FUTURE OF
CYBER SECURITY

POWERED BY CHECK POINT INFINITY

CLOUD • MOBILE • THREAT PREVENTION