Solid C++ code by example

olve.maudal@cisco.com

Sometimes you see code that is perfectly OK according to the definition of the language, but which is flawed because it breaks too many established idioms and conventions. On the other hand, a solid piece of code is something that looks like it is written by an experienced person who cares about professionalism in programming.

This will be an interactive discussion about good vs bad C++ code. We will discuss simple C++ idioms and coding conventions, but we will also touch upon best practices when working with C++ in large codebases with lots of developers with mixed skills.

A 90 minute presentation at the ACCU conference
Oxford, April 13-16 2011

Disclaimer I: the is basically a rerun of the talk I did at ACCU 2010, 80-90% of the material is the same. However, the audience is not the same, so discussions might provide 80-90% new insight...

Disclaimer II: there are some issues that we do not address properly here (at least 3). They are left as an exercise for the reader. Email me if you find them, or if you want to know more!

~/myprog/foo.hpp

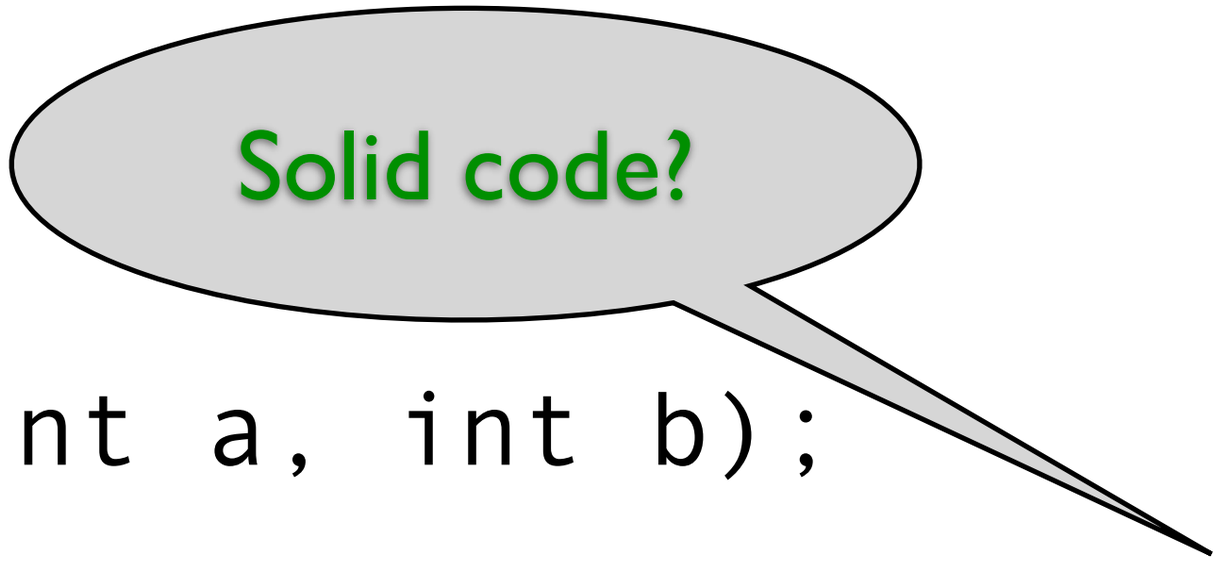
```
namespace bar {
  class Foo {
    int _value;
    int my_magic(int a, int b);
  public:
    Foo(int seed);
    int calc(int number = 7);
    int getValue() {
      return _value;
    }
    void print(char* prefix);
  };
}
```

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
    int _value;
    int my_magic(int a, int b);
  public:
    Foo(int seed);
    int calc(int number = 7);
    int getValue() {
      return _value;
    }
    void print(char* prefix);
  };
}
```

~/myprog/foo.hpp

```
namespace bar {
   class Foo {
      int _value;
      int my_magic(int a, int b);
   public:
      Foo(int seed);
      int calc(int number = 7);
      int getValue() {
         return _value;
      }
      void print(char* prefix);
   };
}
```

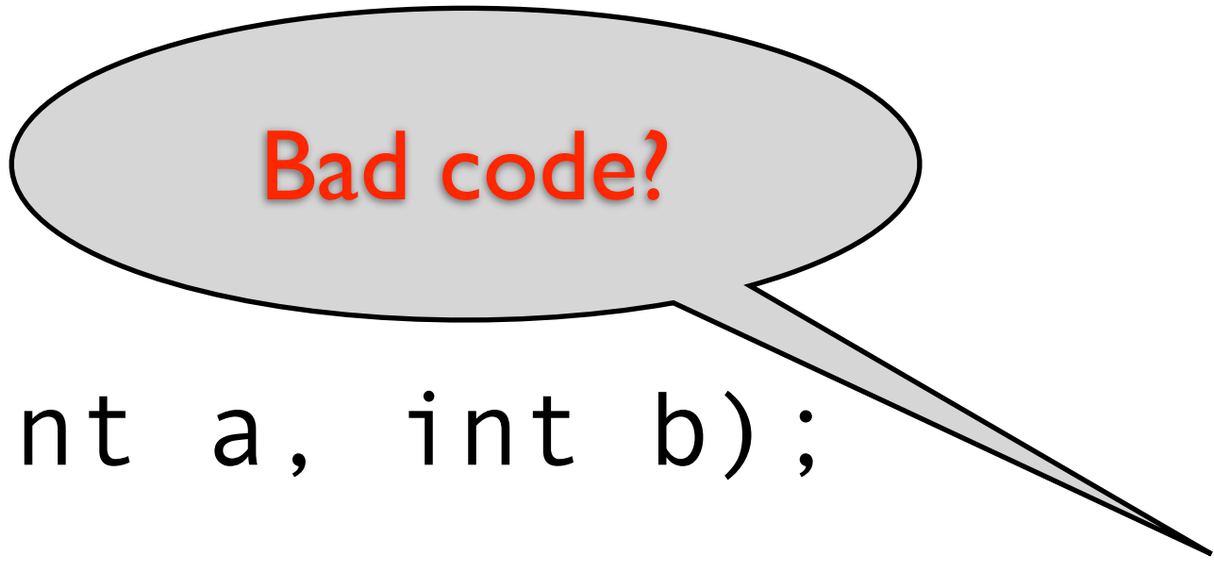


~/myprog/foo.hpp

```
namespace bar {
  class Foo {
    int _value;
    int my_magic(int a, int b);
  public:
    Foo(int seed);
    int calc(int number = 7);
    int getValue() {
      return _value;
    }
    void print(char* prefix);
  };
}
```

~/myprog/foo.hpp

```
namespace bar {
   class Foo {
      int _value;
      int my_magic(int a, int b);
   public:
      Foo(int seed);
      int calc(int number = 7);
      int getValue() {
         return _value;
      }
      void print(char* prefix);
   };
}
```



~/myprog/foo.hpp

```
namespace bar {
  class Foo {
    int _value;
    int my_magic(int a, int b);
  public:
    Foo(int seed);
    int calc(int number = 7);
    int getValue() {
      return _value;
    }
    void print(char* prefix);
  };
}
```

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
    int _value;
    int my_magic(int a, int b);
  public:
    Foo(int seed);
    int calc(int number = 7);
    int getValue() {
      return _value;
    }
    void print(char* prefix);
  };
}
```

PAL - a Primitive Authentication Library in C++ for educational purposes

http://github.com/olvemaudal/pal

(from the README file)

Here is the main "use story":

As a client, when prompted for ntlm authentication by the server, I want a tool/library that can help me to create the initial ntlm **request** (type 1 message) that I can send to the server to receive a **challenge** (type 2 message) that needs to be solved by applying my username and password to create an ntlm **response** that I can send to the server. (phew...)

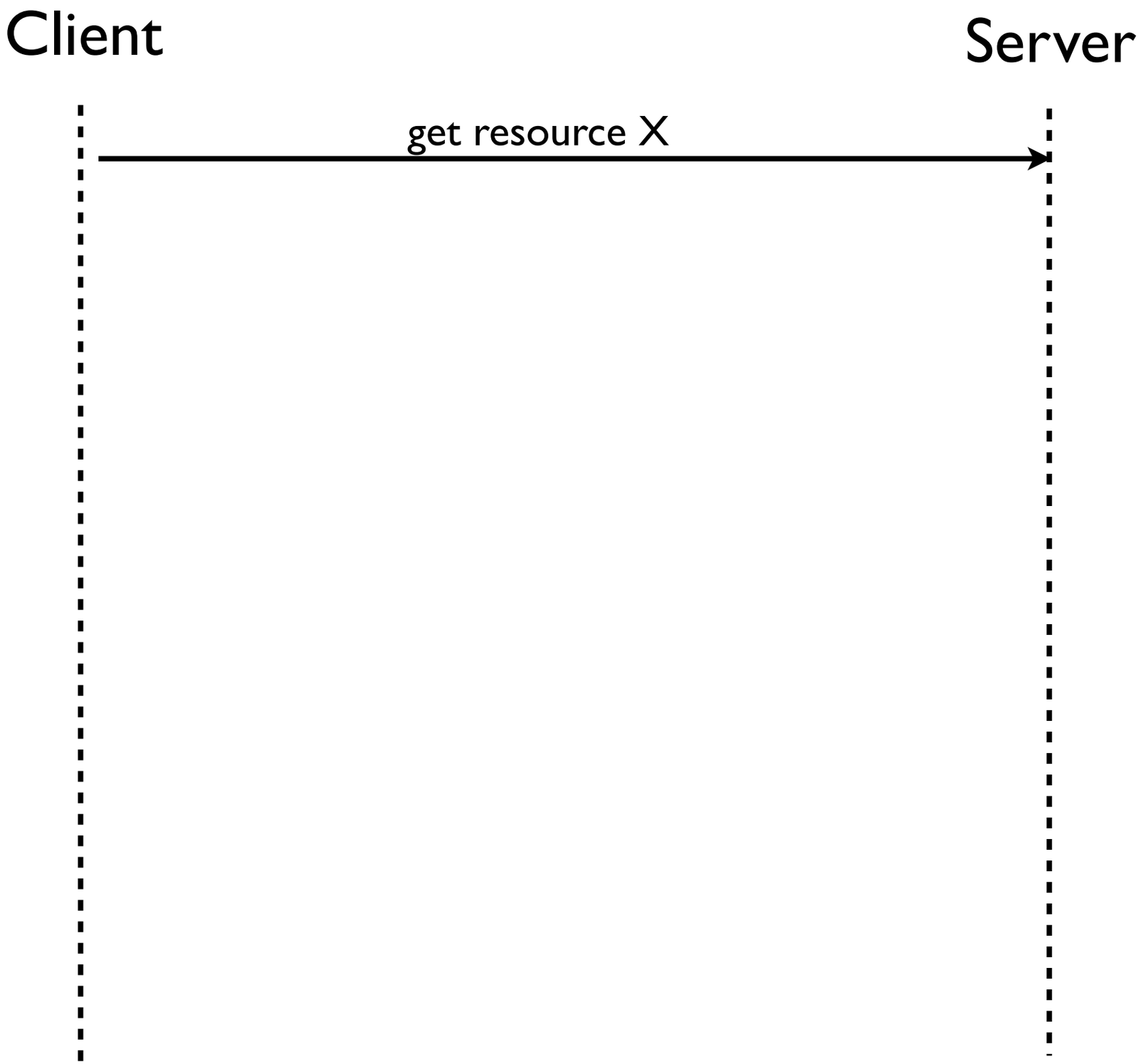
Here are some (imaginary) additional requirements:
- target typically an embedded system
- it should be possible and convenient to create a <100kb client using the PAL library
- must use C++ (due to PHB decision)
- due to compiler support, tr1 and boost can not be used
- initally the library will only be used to communicate HTTP with a Windows 2003 Server SP2, currently no need to support other servers

The NTLM Protocol (simplified)

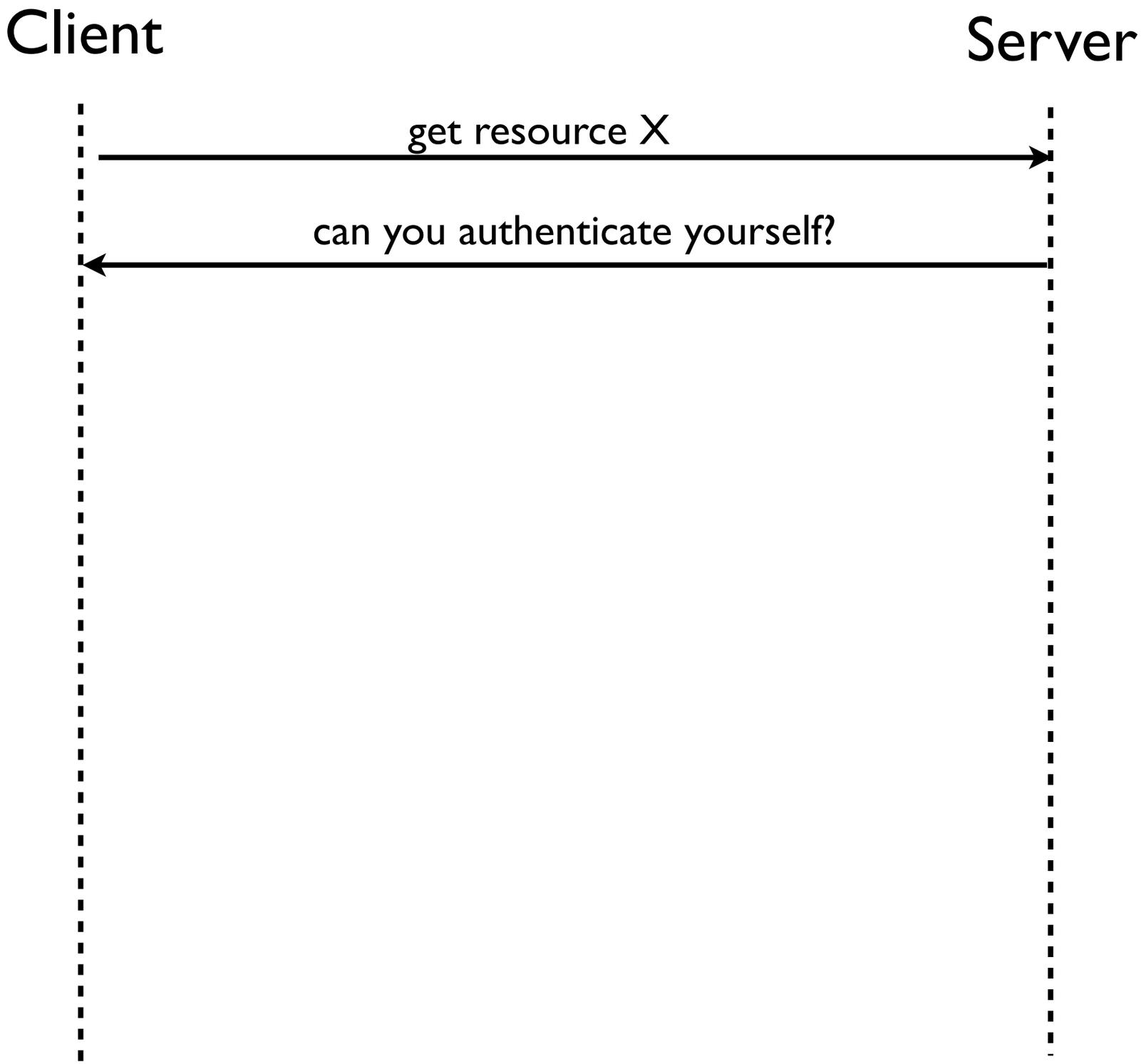
Client

Server

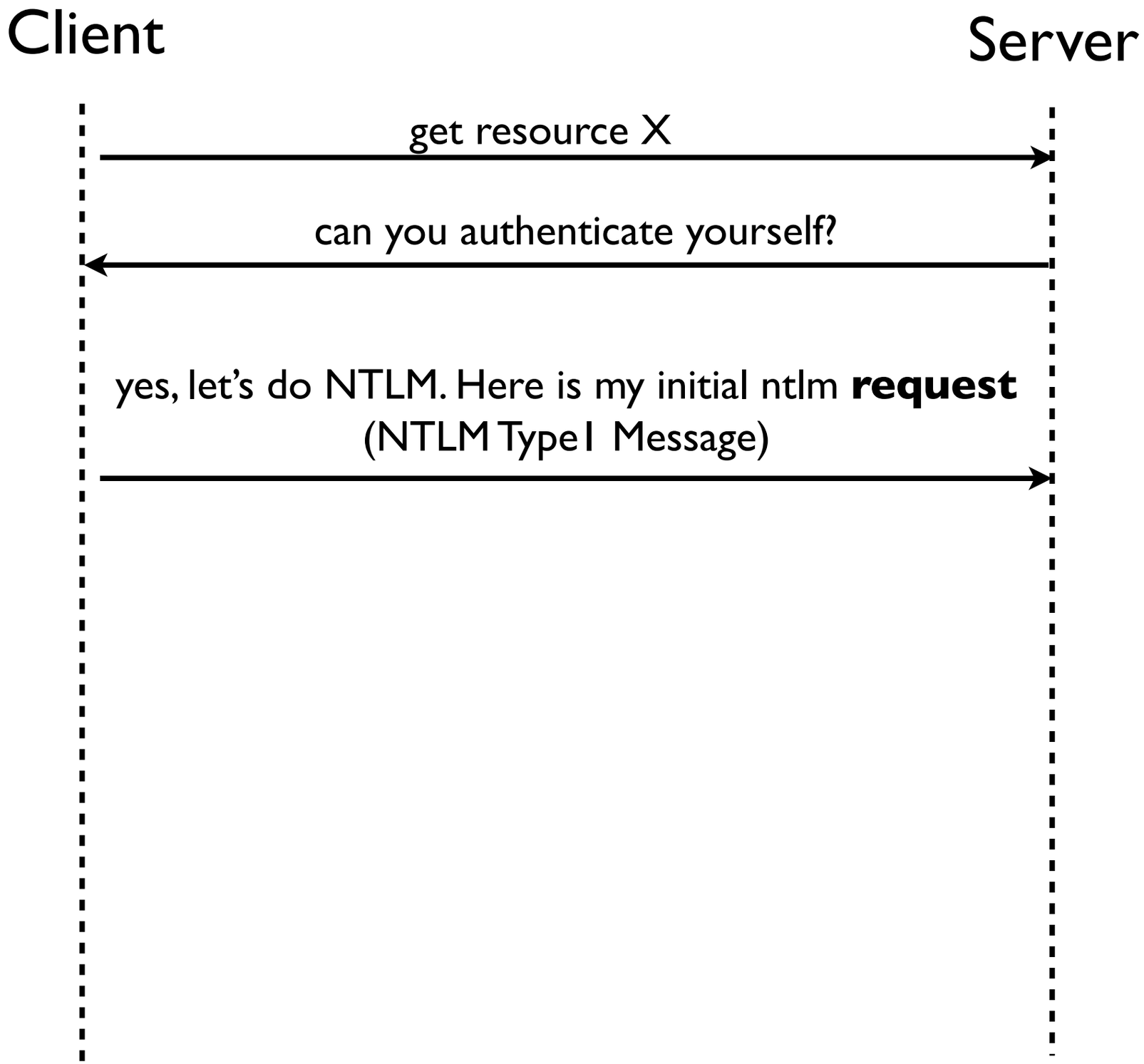
The NTLM Protocol (simplified)



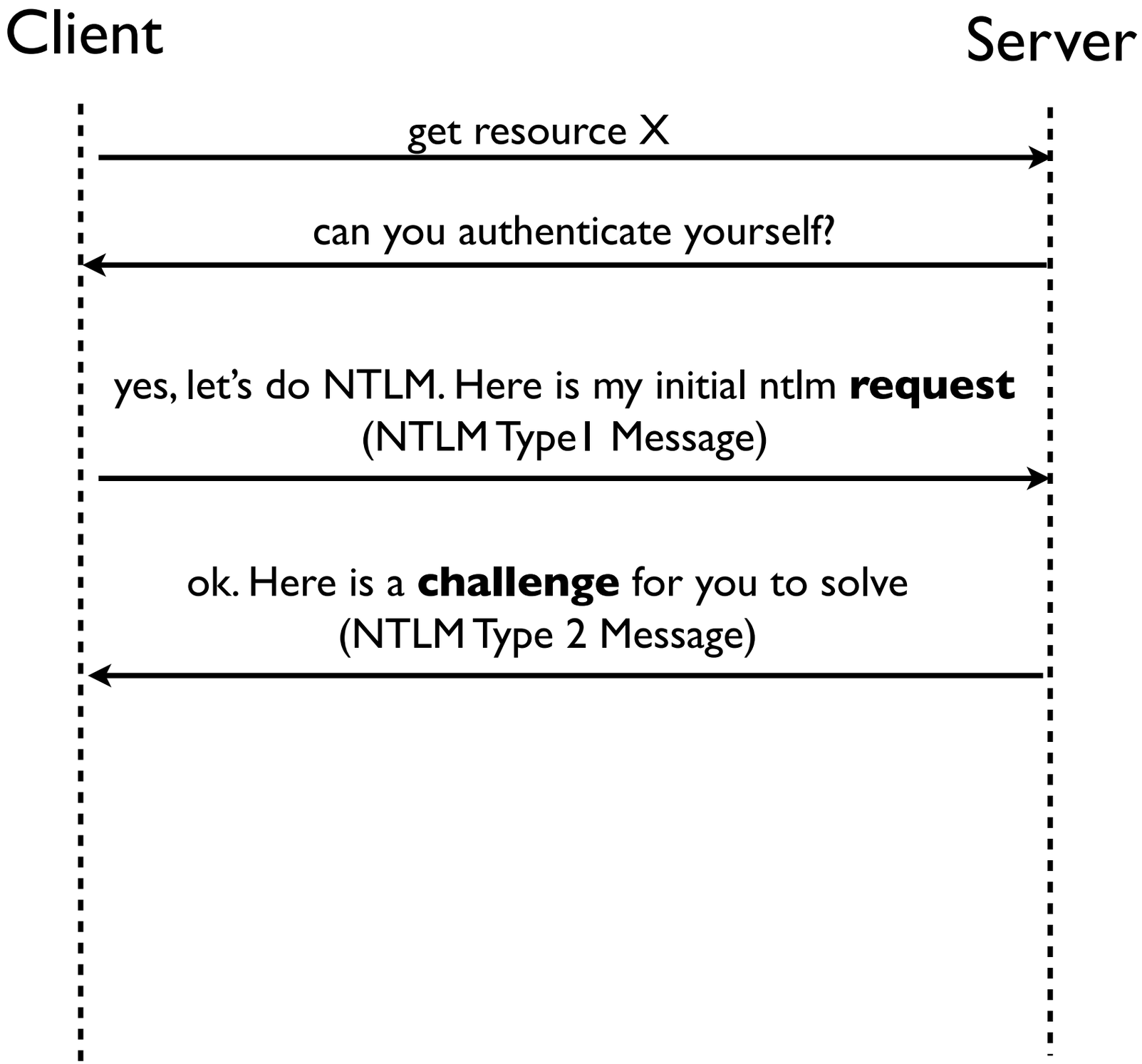
The NTLM Protocol (simplified)



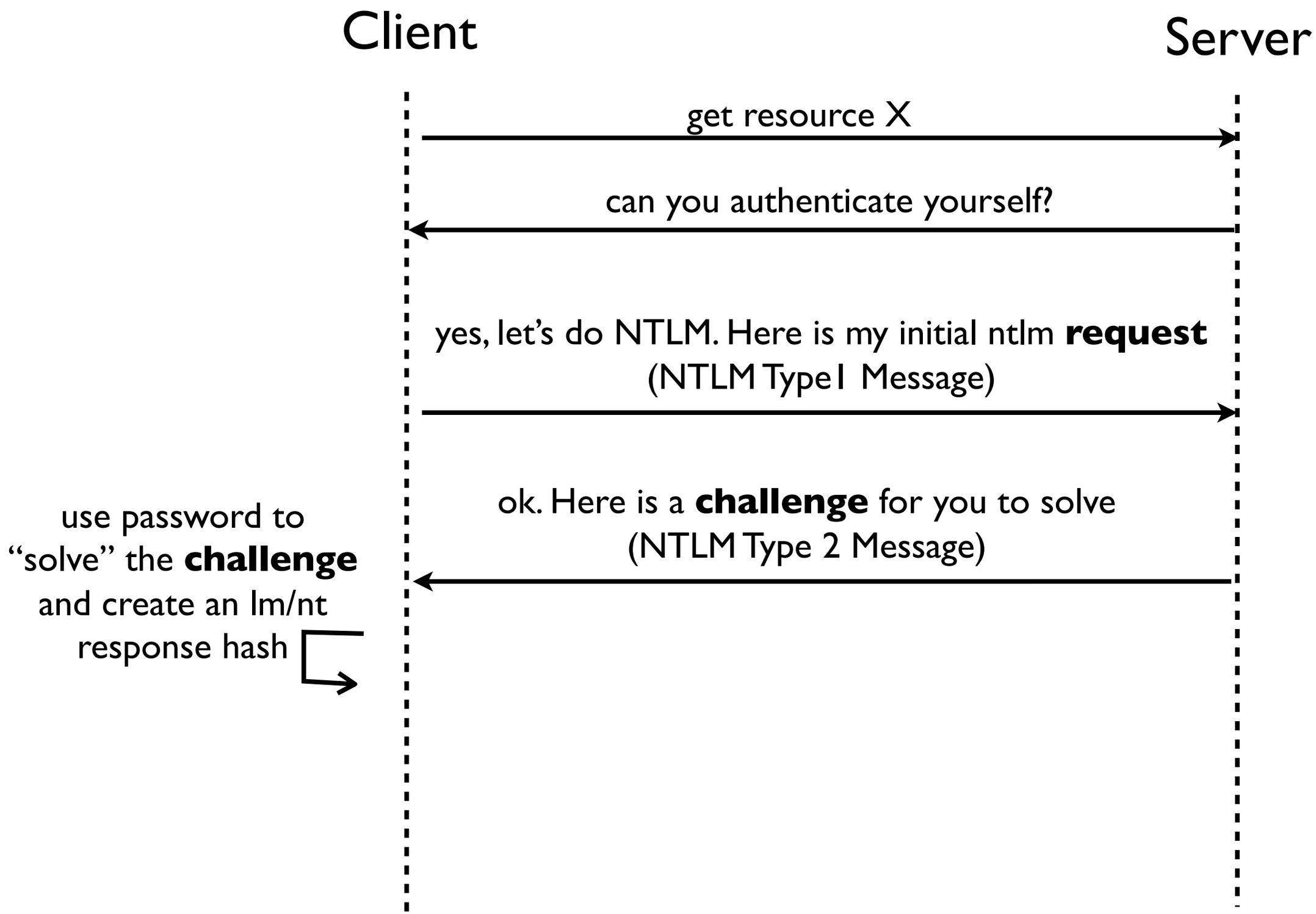
The NTLM Protocol (simplified)



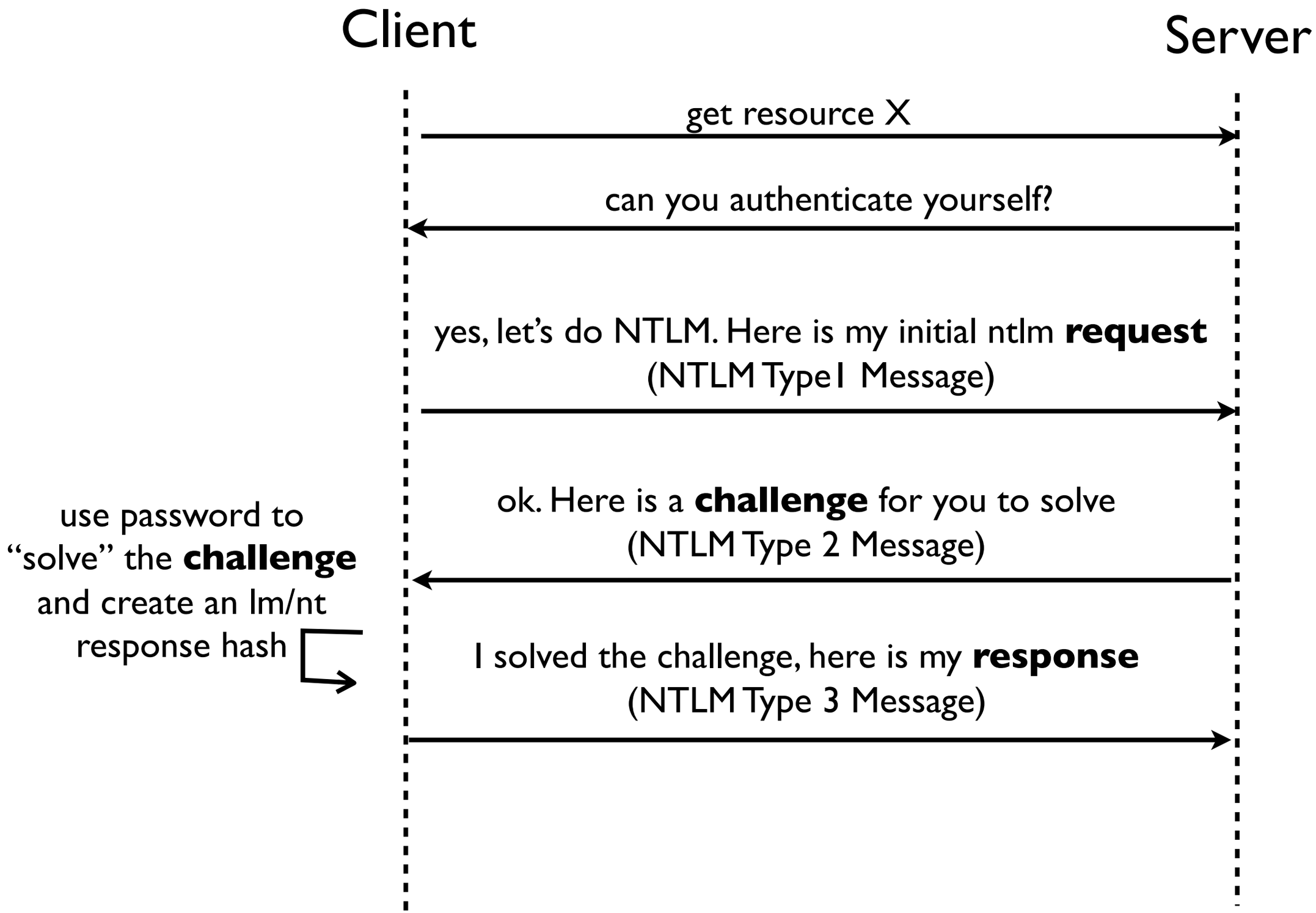
The NTLM Protocol (simplified)



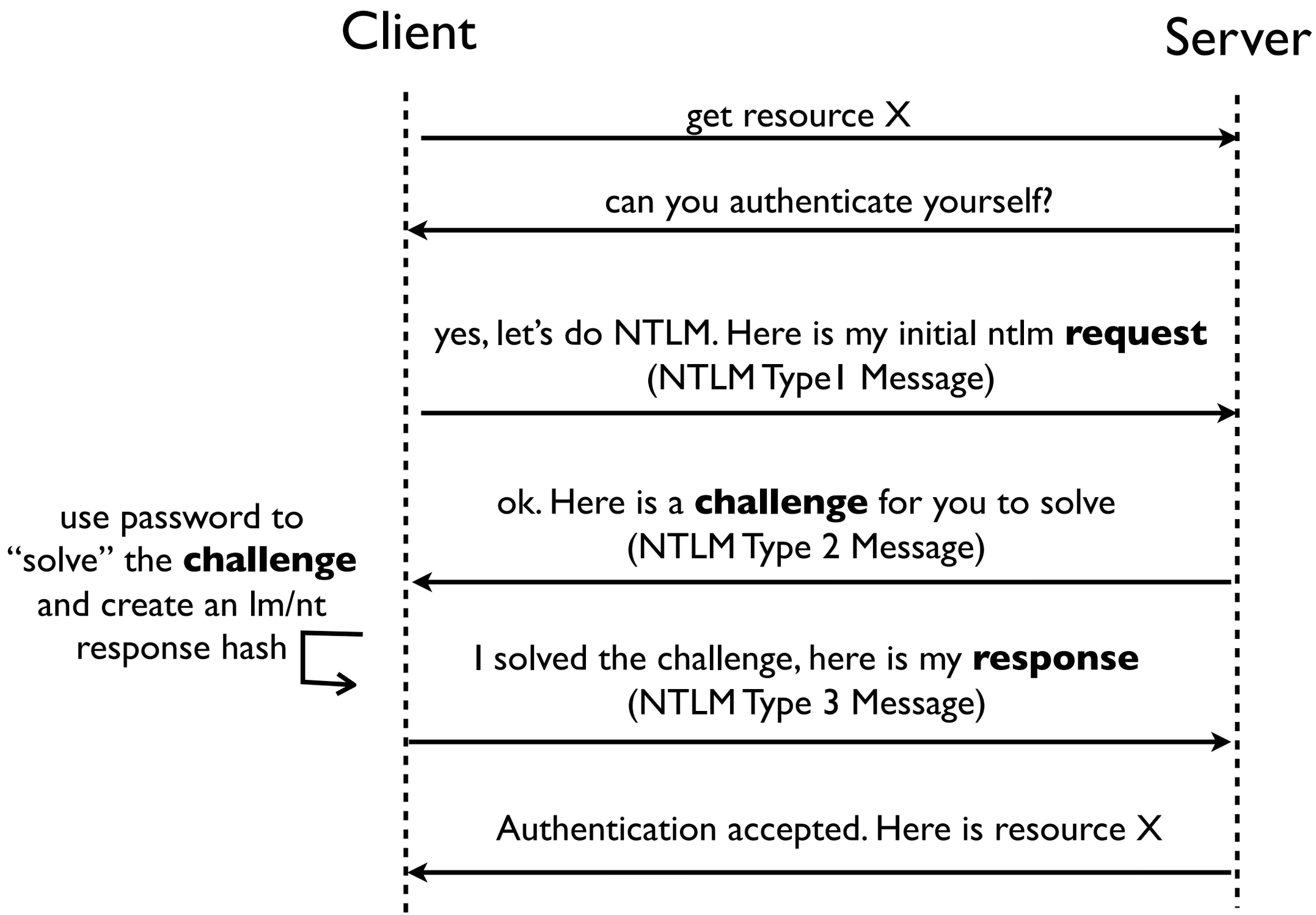
The NTLM Protocol (simplified)



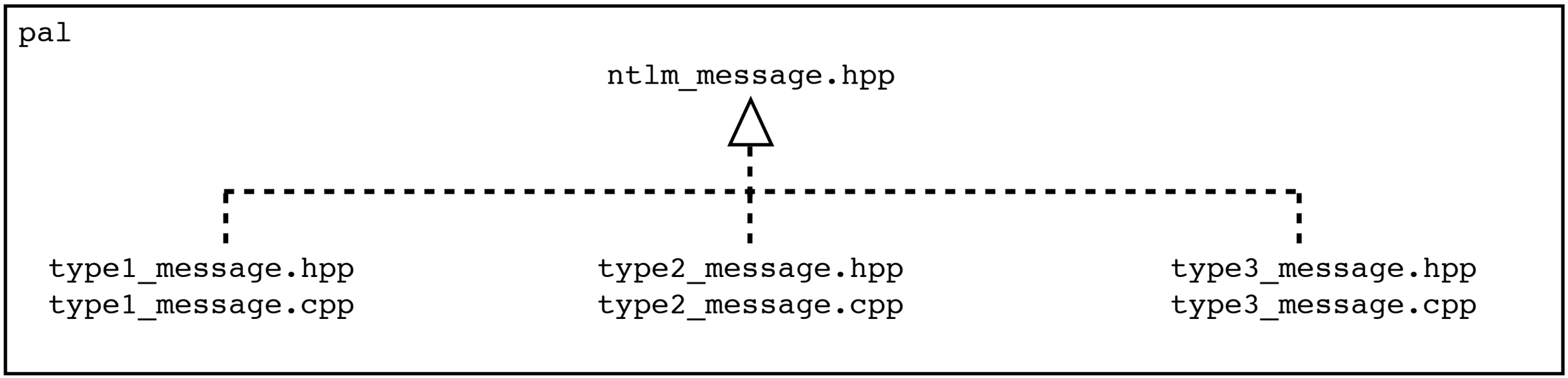
The NTLM Protocol (simplified)



The NTLM Protocol (simplified)



Code under discussion

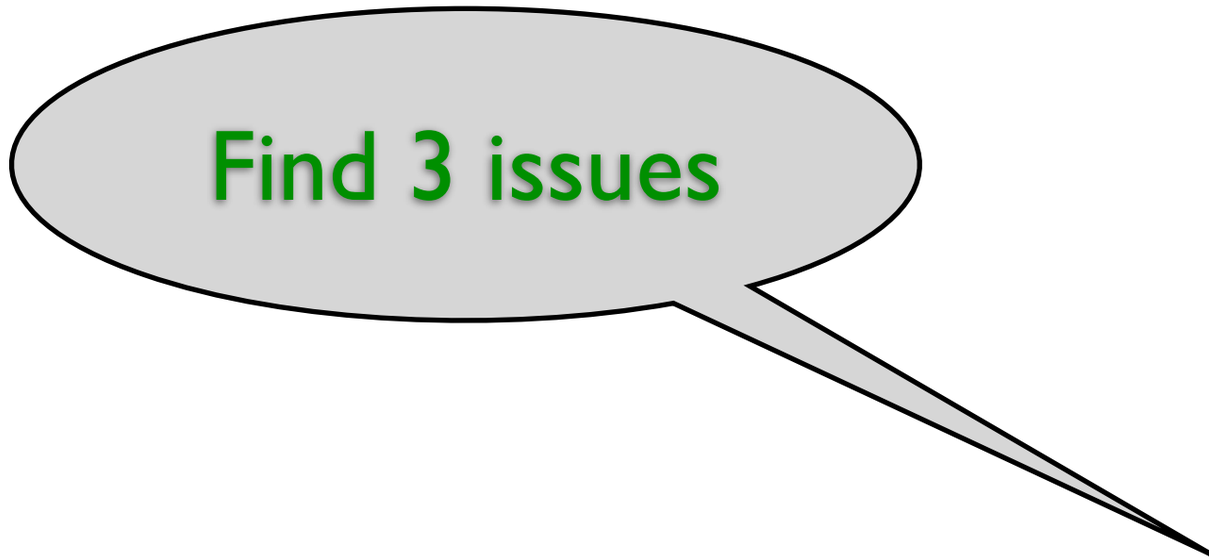


***Note**: the code under discussion compiles just fine and does exactly what it is supposed to do... but can you suggest improvements?*

Study code....

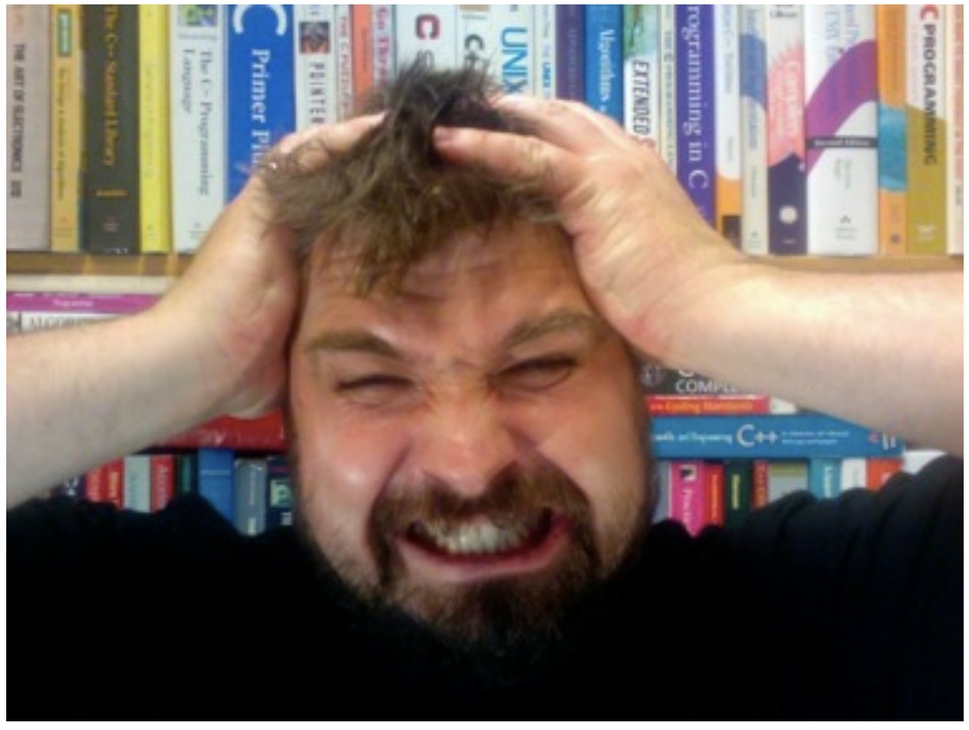
ntlm_message.hpp

```
#include <vector>

namespace pal {

    class ntlm_message {
    public:
        virtual std::vector<uint8_t> as_bytes() const = 0;
    };

}
```



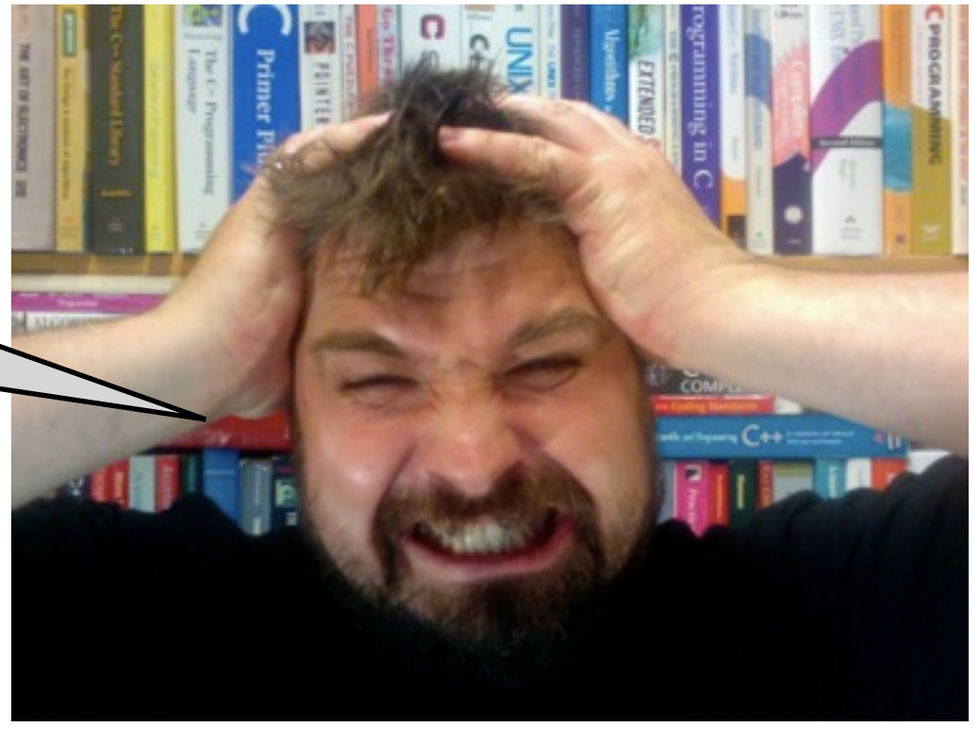

```
#include <vector>

namespace pal {

    class ntlm_message {
    public:
        virtual std::vector<uint8_t> as_bytes() const = 0;
    };

}
```

```
#include <vector>

namespace pal {

    class ntlm_message {
    public:
        virtual std::vector<uint8_t> as_bytes() const = 0;
    };

}
```



```
#include <vector>

namespace pal {

    class ntlm_message {
    public:
        virtual std::vector<uint8_t> as_bytes() const = 0;
    };

}
```

WTF? This class must have a virtual destructor!



```
#include <vector>

namespace pal {

    class ntlm_message {
    public:
        virtual std::vector<uint8_t> as_bytes() const = 0;
    };

}
```

```
#include <vector>

namespace pal {

    class ntlm_message {
    public:
        virtual std::vector<uint8_t> as_bytes() const = 0;
    };

}
```

missing header guard

```
#include <vector>

namespace pal {

    class ntlm_message {
    public:
        virtual std::vector<uint8_t> as_bytes() const = 0;
    };

}
```

```
#include <vector>

namespace pal {

    class ntlm_message {
    public:
        virtual std::vector<uint8_t> as_bytes() const = 0;
    };

}
```

should include <stdint.h>

```
#include <vector>

namespace pal {

    class ntlm_message {
    public:
        virtual std::vector<uint8_t> as_bytes() const = 0;
    };

}
```

```
#ifndef PAL_NTLM_MESSAGE_HPP_INCLUDED
#define PAL_NTLM_MESSAGE_HPP_INCLUDED

#include <vector>

namespace pal {

    class ntlm_message {
    public:
        virtual std::vector<uint8_t> as_bytes() const = 0;
    };

}

#endif
```

```
#ifndef PAL_NTLM_MESSAGE_HPP_INCLUDED
#define PAL_NTLM_MESSAGE_HPP_INCLUDED

#include <vector>

namespace pal {

    class ntlm_message {
    public:

        virtual std::vector<uint8_t> as_bytes() const = 0;
    };

}

#endif
```

```
#ifndef PAL_NTLM_MESSAGE_HPP_INCLUDED
#define PAL_NTLM_MESSAGE_HPP_INCLUDED

#include <vector>

namespace pal {

    class ntlm_message {
    public:
        virtual ~ntlm_message() {}
        virtual std::vector<uint8_t> as_bytes() const = 0;
    };

}

#endif
```

```
#ifndef PAL_NTLM_MESSAGE_HPP_INCLUDED
#define PAL_NTLM_MESSAGE_HPP_INCLUDED


#include <vector>

namespace pal {

    class ntlm_message {
    public:
        virtual ~ntlm_message() {}
        virtual std::vector<uint8_t> as_bytes() const = 0;
    };

}

#endif
```

```
#ifndef PAL_NTLM_MESSAGE_HPP_INCLUDED
#define PAL_NTLM_MESSAGE_HPP_INCLUDED

#include <stdint.h>
#include <vector>

namespace pal {

    class ntlm_message {
    public:
        virtual ~ntlm_message() {}
        virtual std::vector<uint8_t> as_bytes() const = 0;
    };

}

#endif
```

```
#ifndef PAL_NTLM_MESSAGE_HPP_INCLUDED
#define PAL_NTLM_MESSAGE_HPP_INCLUDED

#include <stdint.h>
#include <vector>

namespace pal {

    class ntlm_message {
    public:
        virtual ~ntlm_message() {}
        virtual std::vector<uint8_t> as_bytes() const = 0;
    };

}

#endif
```

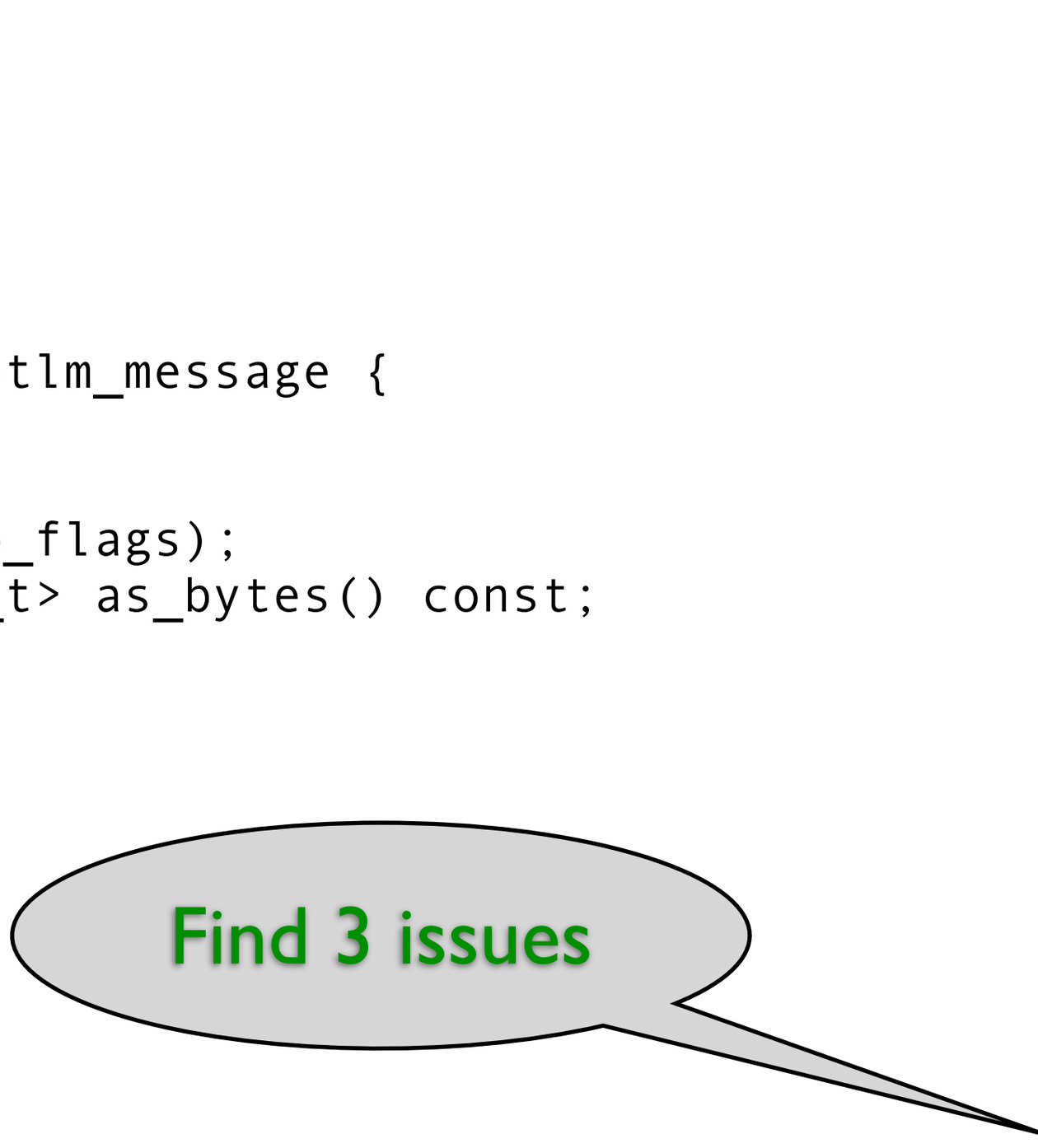
```
#ifndef PAL_NTLM_MESSAGE_HPP_INCLUDED
#define PAL_NTLM_MESSAGE_HPP_INCLUDED

#include <stdint.h>
#include <vector>

namespace pal {

    class ntlm_message {
    public:
        virtual ~ntlm_message() {}
        virtual std::vector<uint8_t> as_bytes() const = 0;
    };

}

#endif
```

This is Solid C++!

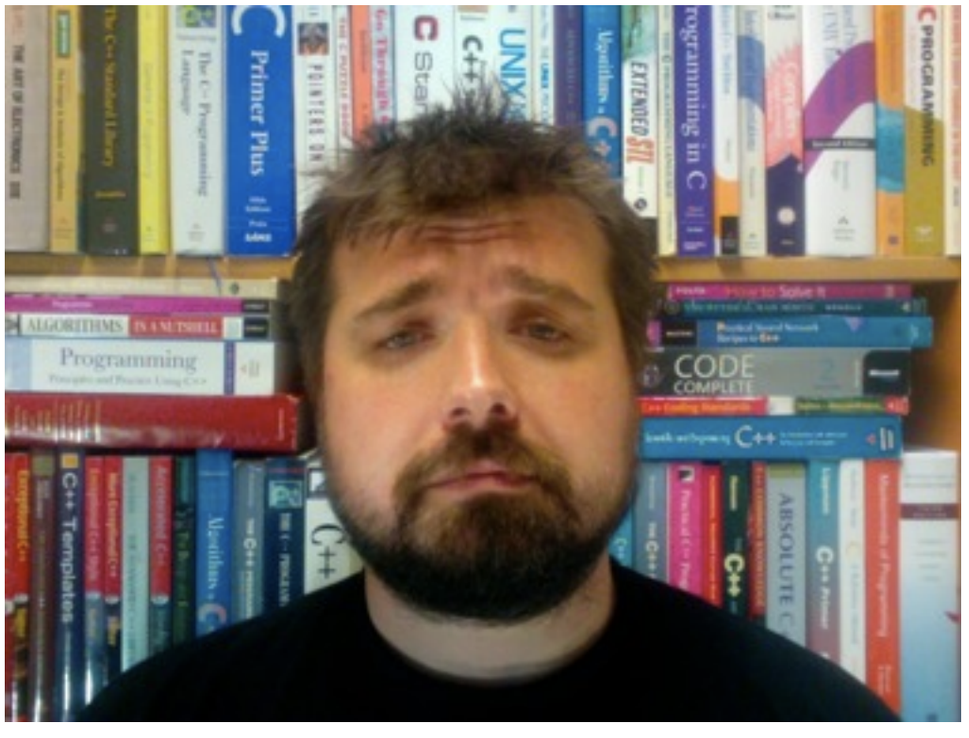
type1_message.hpp

```
#ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
#define PAL_TYPE1_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <vector>

namespace pal {

    class type1_message : public ntlm_message {
        uint32_t ssp_flags_;
    public:
        type1_message(uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
    };

}

#endif
```

```
#ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
#define PAL_TYPE1_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <vector>

namespace pal {

    class type1_message : public ntlm_message {
        uint32_t ssp_flags_;
    public:
        type1_message(uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
    };

}

#endif
```



```
#ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
#define PAL_TYPE1_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <vector>

namespace pal {

    class type1_message : public ntlm_message {
        uint32_t ssp_flags_;
    public:
        type1_message(uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
    };

}

#endif
```



```
#ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
#define PAL_TYPE1_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <vector>

namespace pal {

    class type1_message : public ntlm_message {
        uint32_t ssp_flags_;
    public:
        type1_message(uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
    };

}

#endif
```

```
#ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
#define PAL_TYPE1_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <vector>

namespace pal {

    class type1_message : public ntlm_message {
        uint32_t ssp_flags_;
    public:
        type1_message(uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
    };

}

#endif
```

```
#ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
#define PAL_TYPE1_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <vector>

namespace pal {

    class type1_message : public ntlm_message {
        uint32_t ssp_flags_;
    public:
        type1_message(uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
    };

}

#endif
```

```
#ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
#define PAL_TYPE1_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <vector>

namespace pal {

    class type1_message : public ntlm_message {
        uint32_t ssp_flags_;
    public:
        type1_message(uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
    };

}

#endif
```

```
#ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
#define PAL_TYPE1_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <vector>

namespace pal {

    class type1_message : public ntlm_message {
        uint32_t ssp_flags_;
    public:
        type1_message(uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
    };

}

#endif
```

single argument constructors should nearly always have the `explicit` specifier

```
#ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
#define PAL_TYPE1_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <vector>

namespace pal {

    class type1_message : public ntlm_message {
        uint32_t ssp_flags_;
    public:
        type1_message(uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
    };

}

#endif
```

```
#ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
#define PAL_TYPE1_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type1_message : public ntlm_message {
        uint32_t ssp_flags_;
    public:
        type1_message(uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
    };

}

#endif
```

```
#ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
#define PAL_TYPE1_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type1_message : public ntlm_message {

        uint32_t ssp_flags_;
    public:
        type1_message(uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
    };

}

#endif
```

```
#ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
#define PAL_TYPE1_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type1_message : public ntlm_message {
    private:
        uint32_t ssp_flags_;
    public:
        type1_message(uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
    };

}

#endif
```

```
#ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
#define PAL_TYPE1_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type1_message : public ntlm_message {
    private:
        uint32_t ssp_flags_;
    public:
        type1_message(uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
    };

}

#endif
```

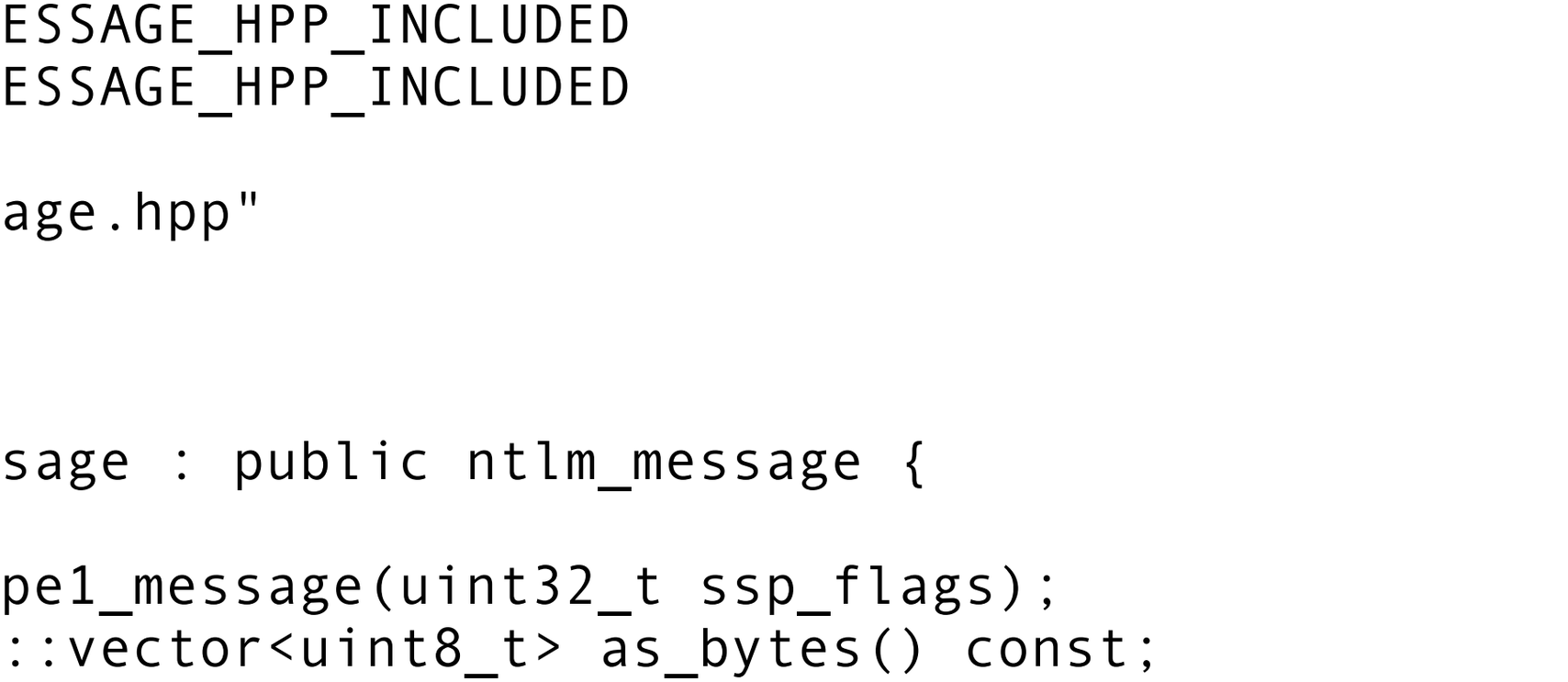
```
#ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
#define PAL_TYPE1_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type1_message : public ntlm_message {
    public:
        type1_message(uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
    private:
        uint32_t ssp_flags_;
    };

}

#endif
```

```
#ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
#define PAL_TYPE1_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type1_message : public ntlm_message {
    public:
        type1_message(uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
    private:
        uint32_t ssp_flags_;
    };

}

#endif
```

```
#ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
#define PAL_TYPE1_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type1_message : public ntlm_message {
    public:
        explicit type1_message(uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
    private:
        uint32_t ssp_flags_;
    };

}

#endif
```

```
#ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
#define PAL_TYPE1_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type1_message : public ntlm_message {
    public:
        explicit type1_message(uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
    private:
        uint32_t ssp_flags_;
    };

}

#endif
```

```
#ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
#define PAL_TYPE1_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type1_message : public ntlm_message {
    public:
        explicit type1_message(uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
    private:
        uint32_t ssp_flags_;
    };

}

#endif
```



type1_message.cpp

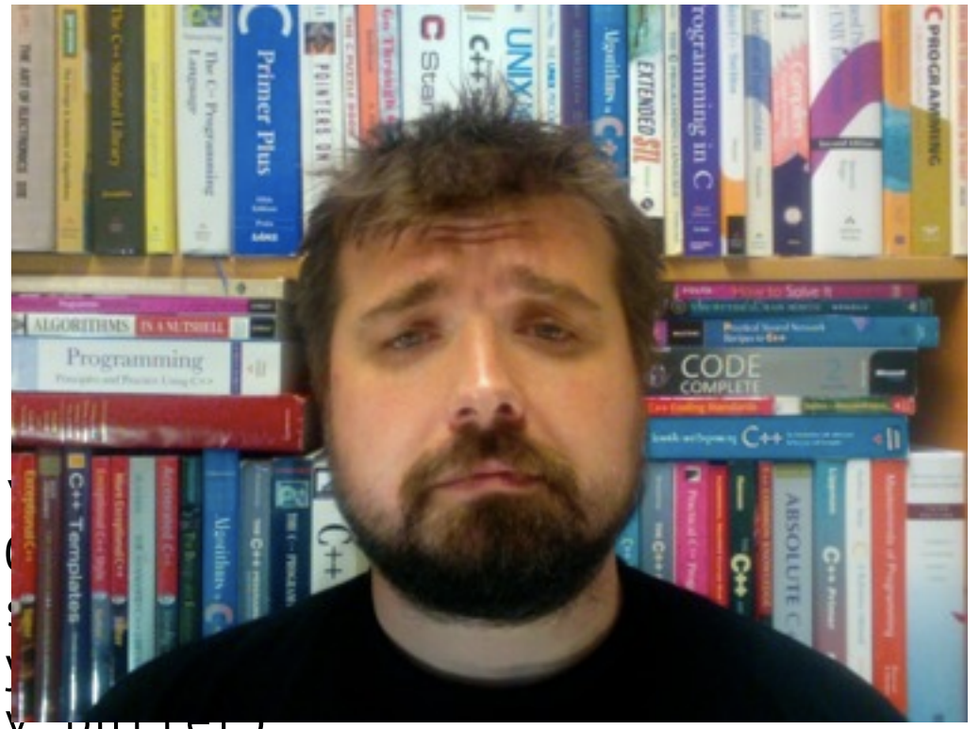
```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                 "NTLMSSP\0"
 *    8  NTLM Message Type                 {0x01,0x00,0x00,0x00}
 *   12  Flags                             uint32 as little endian
 *  (16) Supplied Domain (optional)        (security buffer)
 *  (24) Supplied Workstation (optional)   (security buffer)
 *  (32) (start of datablock) if required
 */

using namespace std;

pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const int ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                 "NTLMSSP\0"
 *    8  NTLM Message Type                 {0x01,0x00,0x00,0x00}
 *   12  Flags                             uint32 as little endian
 *  (16) Supplied Domain (optional)        (security buffer)
 *  (24) Supplied Workstation (optional)   (security buffer)
 *  (32) (start of datablock) if required
 */
```

Find at least 3 issues here

```
using namespace std;

pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const int ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return vector<uint8_t>(message, message + sizeof message);
}
```

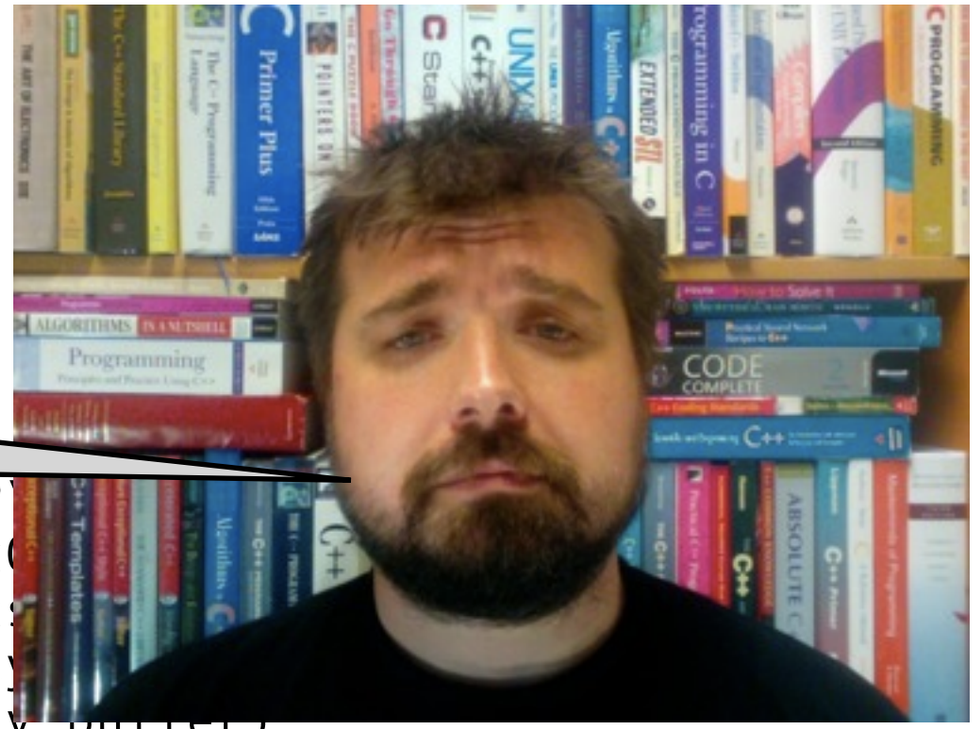
```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                          "NTLMSSP
 *    8  NTLM Message Type                          {0x01,0x
 *   12  Flags                                      uint32 a
 * (16) Supplied Domain (optional)                  (security
 * (24) Supplied Workstation (optional)             (security buffer)
 * (32) (start of datablock) if required
 */
```




```
using namespace std;

pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const int ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return vector<uint8_t>(message, message + sizeof message);
}
```

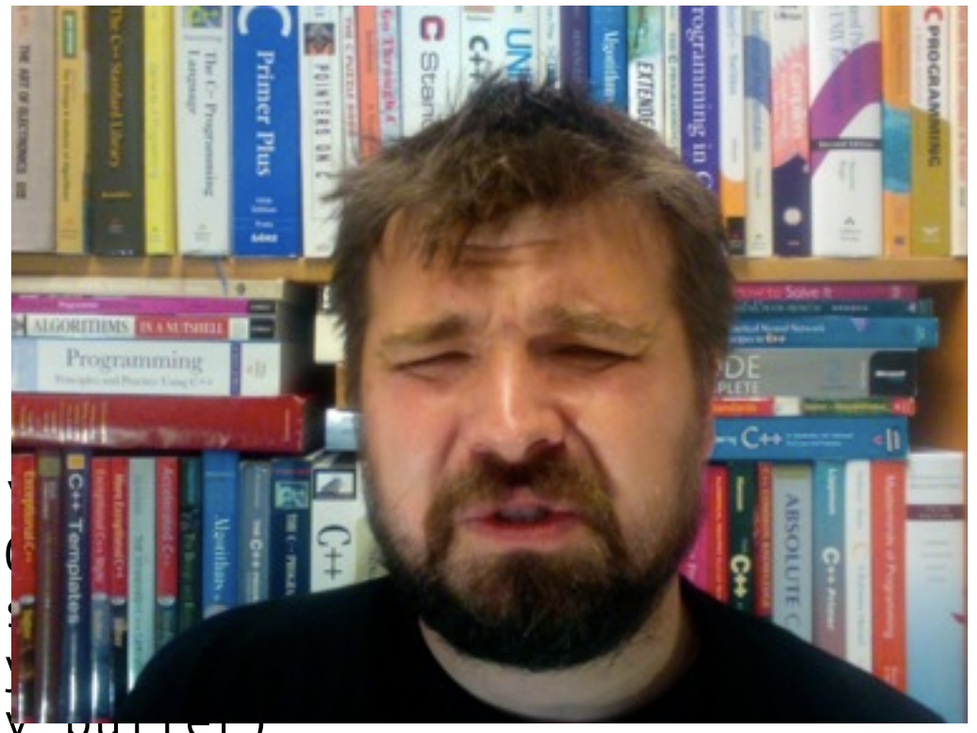
```
#include "type1_message.hpp"

#include "tools.hpp"
```

focus on readability;
importing a namespace might save some keystrokes, but often it does not increase readability




```
m.html

NTLMSSP
0x01,0x0
uint32 a
(security
(security buffer)

 * (24) Supplied Workstation (optional)
 * (32) (start of datablock) if required
 */

using namespace std;

pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const int ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                          "NTLMSSP
 *    8  NTLM Message Type                          {0x01,0x
 *   12  Flags                                      uint32 a
 * (16) Supplied Domain (optional)                  (security
 * (24) Supplied Workstation (optional)             (security buffer)
 * (32) (start of datablock) if required
 */
```



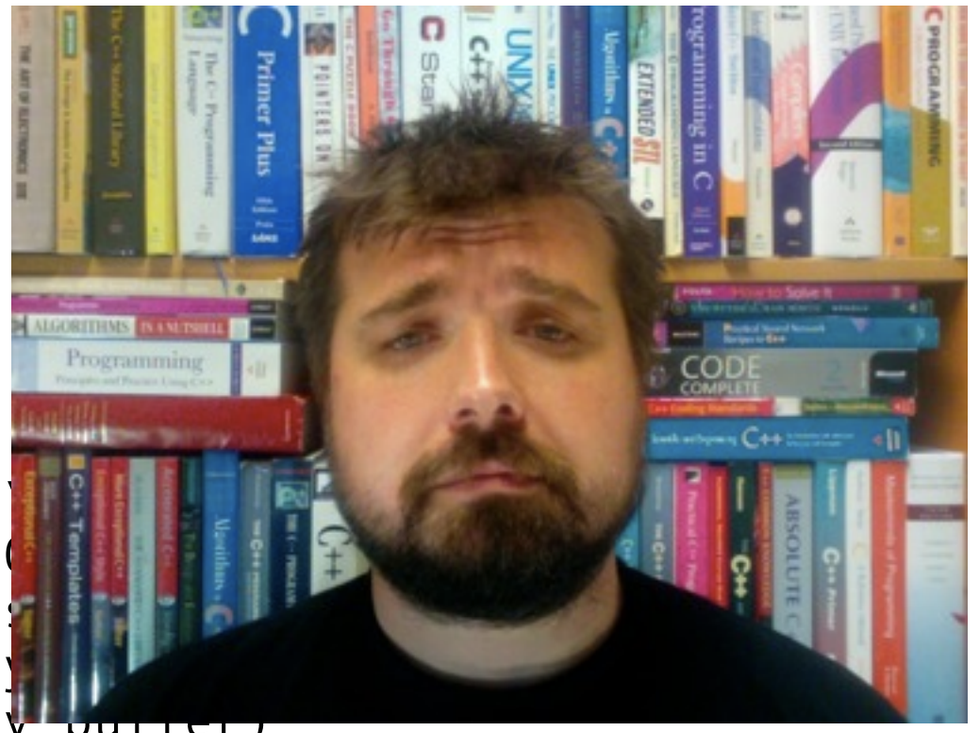

```
using namespace std;

pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const int ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "to

/*
 * See http://davenport
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                "NTLMSSP
 *    8  NTLM Message Type                {0x01,0x0
 *   12  Flags                            uint32 as
 * (16) Supplied Domain (optional)        (security
 * (24) Supplied Workstation (optional)   (security buffer)
 * (32) (start of datablock) if required
 */
```

use the initializer list

```
using namespace std;

pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const int ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                    "NTLMSSP
 *    8  NTLM Message Type                    {0x01,0x
 *   12  Flags                                uint32 a
 * (16) Supplied Domain (optional)            (security
 * (24) Supplied Workstation (optional)       (security buffer)
 * (32) (start of datablock) if required
 */
```




```
using namespace std;

pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const int ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tool

/*
 * See http://
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                "NTLMSSP
 *    8  NTLM Message Type                {0x01,0x0
 *   12  Flags                            uint32 as
 * (16) Supplied Domain (optional)        (security
 * (24) Supplied Workstation (optional)   (security buffer)
 * (32) (start of datablock) if required
 */
```

use std::size_t when appropriate

```
using namespace std;

pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const int ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return vector<uint8_t>(message, message + sizeof message);
}
```

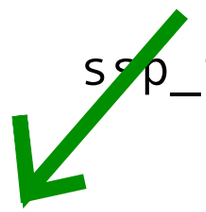
```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                 "NTLMSSP\0"
 *    8  NTLM Message Type                 {0x01,0x00,0x00,0x00}
 *   12  Flags                             uint32 as little endian
 *  (16) Supplied Domain (optional)        (security buffer)
 *  (24) Supplied Workstation (optional)   (security buffer)
 *  (32) (start of datablock) if required
 */

using namespace std;

pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const int ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                 "NTLMSSP\0"
 *    8  NTLM Message Type                 {0x01,0x00,0x00,0x00}
 *   12  Flags                             uint32 as little endian
 *  (16) Supplied Domain (optional)        (security buffer)
 *  (24) Supplied Workstation (optional)   (security buffer)
 *  (32) (start of datablock) if required
 */

using namespace std;

pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const int ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                 "NTLMSSP\0"
 *    8  NTLM Message Type                 {0x01,0x00,0x00,0x00}
 *   12  Flags                             uint32 as little endian
 * (16) Supplied Domain (optional)        (security buffer)
 * (24) Supplied Workstation (optional)   (security buffer)
 * (32) (start of datablock) if required
 */
```

```
pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const int ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                 "NTLMSSP\0"
 *    8  NTLM Message Type                 {0x01,0x00,0x00,0x00}
 *   12  Flags                             uint32 as little endian
 *  (16) Supplied Domain (optional)        (security buffer)
 *  (24) Supplied Workstation (optional)   (security buffer)
 *  (32) (start of datablock) if required
 */
```



```
pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const int ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                 "NTLMSSP\0"
 *    8  NTLM Message Type                 {0x01,0x00,0x00,0x00}
 *   12  Flags                             uint32 as little endian
 *  (16) Supplied Domain (optional)        (security buffer)
 *  (24) Supplied Workstation (optional)   (security buffer)
 *  (32) (start of datablock) if required
 */


pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

std::vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const int ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                 "NTLMSSP\0"
 *    8  NTLM Message Type                 {0x01,0x00,0x00,0x00}
 *   12  Flags                             uint32 as little endian
 * (16) Supplied Domain (optional)        (security buffer)
 * (24) Supplied Workstation (optional)   (security buffer)
 * (32) (start of datablock) if required
 */


pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

std::vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const int ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                 "NTLMSSP\0"
 *    8  NTLM Message Type                 {0x01,0x00,0x00,0x00}
 *   12  Flags                             uint32 as little endian
 * (16) Supplied Domain (optional)        (security buffer)
 * (24) Supplied Workstation (optional)   (security buffer)
 * (32) (start of datablock) if required
 */


pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

std::vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const int ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return std::vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                 "NTLMSSP\0"
 *    8  NTLM Message Type                 {0x01,0x00,0x00,0x00}
 *   12  Flags                             uint32 as little endian
 * (16) Supplied Domain (optional)        (security buffer)
 * (24) Supplied Workstation (optional)   (security buffer)
 * (32) (start of datablock) if required
 */


pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

std::vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const int ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return std::vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                 "NTLMSSP\0"
 *    8  NTLM Message Type                 {0x01,0x00,0x00,0x00}
 *   12  Flags                             uint32 as little endian
 * (16) Supplied Domain (optional)        (security buffer)
 * (24) Supplied Workstation (optional)   (security buffer)
 * (32) (start of datablock) if required
 */


pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

std::vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const int ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return std::vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                 "NTLMSSP\0"
 *    8  NTLM Message Type                 {0x01,0x00,0x00,0x00}
 *   12  Flags                             uint32 as little endian
 * (16) Supplied Domain (optional)        (security buffer)
 * (24) Supplied Workstation (optional)   (security buffer)
 * (32) (start of datablock) if required
 */


pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

std::vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const std::size_t ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return std::vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                 "NTLMSSP\0"
 *    8  NTLM Message Type                 {0x01,0x00,0x00,0x00}
 *   12  Flags                             uint32 as little endian
 *  (16) Supplied Domain (optional)        (security buffer)
 *  (24) Supplied Workstation (optional)   (security buffer)
 *  (32) (start of datablock) if required
 */


pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

std::vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const std::size_t ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return std::vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                 "NTLMSSP\0"
 *    8  NTLM Message Type                 {0x01,0x00,0x00,0x00}
 *   12  Flags                             uint32 as little endian
 * (16) Supplied Domain (optional)        (security buffer)
 * (24) Supplied Workstation (optional)   (security buffer)
 * (32) (start of datablock) if required
 */


pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

std::vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const std::size_t ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return std::vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                 "NTLMSSP\0"
 *    8  NTLM Message Type                 {0x01,0x00,0x00,0x00}
 *   12  Flags                             uint32 as little endian
 * (16) Supplied Domain (optional)        (security buffer)
 * (24) Supplied Workstation (optional)   (security buffer)
 * (32) (start of datablock) if required
 */


pal::type1_message::type1_message(uint32_t ssp_flags)
    : ssp_flags_(ssp_flags)
{
}

std::vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const std::size_t ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return std::vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                 "NTLMSSP\0"
 *    8  NTLM Message Type                 {0x01,0x00,0x00,0x00}
 *   12  Flags                             uint32 as little endian
 *  (16) Supplied Domain (optional)        (security buffer)
 *  (24) Supplied Workstation (optional)   (security buffer)
 *  (32) (start of datablock) if required
 */


pal::type1_message::type1_message(uint32_t ssp_flags)
    : ssp_flags_(ssp_flags)
{
}

std::vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const std::size_t ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return std::vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                 "NTLMSSP\0"
 *    8  NTLM Message Type                 {0x01,0x00,0x00,0x00}
 *   12  Flags                             uint32 as little endian
 * (16) Supplied Domain (optional)        (security buffer)
 * (24) Supplied Workstation (optional)   (security buffer)
 * (32) (start of datablock) if required
 */

pal::type1_message::type1_message(uint32_t ssp_flags)
    : ssp_flags_(ssp_flags)
{
}

std::vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const std::size_t ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return std::vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                 "NTLMSSP\0"
 *    8  NTLM Message Type                 {0x01,0x00,0x00,0x00}
 *   12  Flags                             uint32 as little endian
 *  (16) Supplied Domain (optional)        (security buffer)
 *  (24) Supplied Workstation (optional)   (security buffer)
 *  (32) (start of datablock) if required
 */
```

```
pal::type1_message::type1_message(uint32_t ssp_flags)
    : ssp_flags_(ssp_flags)
{
}

std::vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const std::size_t ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return std::vector<uint8_t>(message, message + sizeof message);
}
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenpo
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                    "NTLMSS
 *    8  NTLM Message Type                    {0x01,0x0
 *   12  Flags                                uint32 as
 * (16) Supplied Domain (optional)            (security
 * (24) Supplied Workstation (optional)       (security buffer)
 * (32) (start of datablock) if required
 */
```

```
pal::type1_message::type1_message(uint32_t ssp_flags)
    : ssp_flags_(ssp_flags)
{
}

std::vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const std::size_t ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return std::vector<uint8_t>(message, message + sizeof message);
}
```

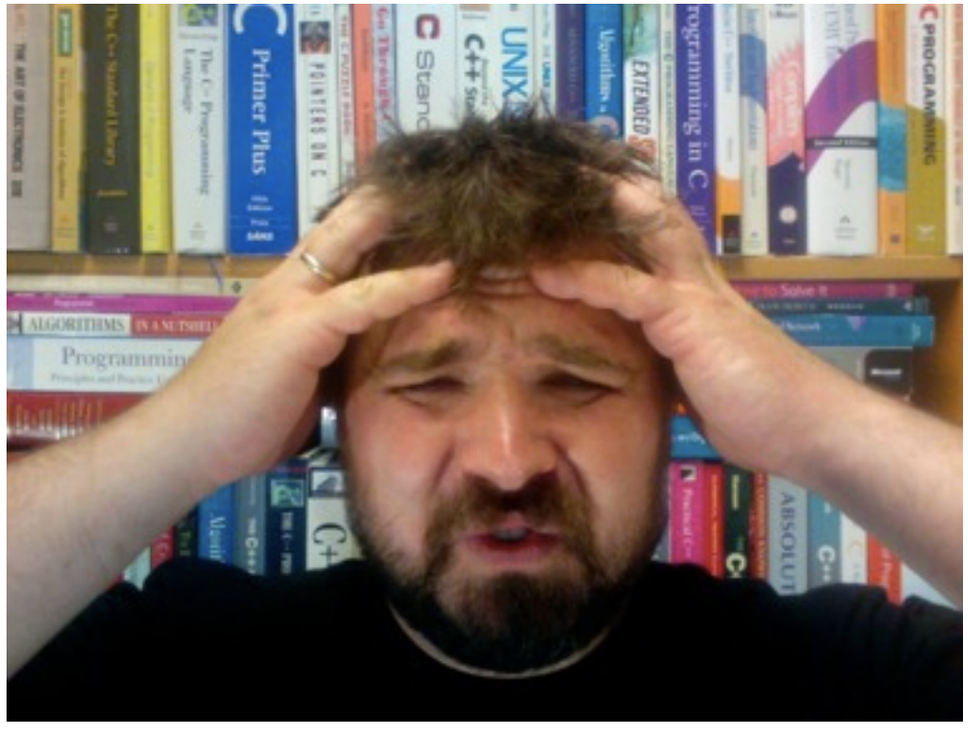
type2_message.hpp

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

using namespace std;

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(vector<uint8_t> buffer);
        virtual vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const vector<uint8_t> buffer_;
    };

}

#endif
```

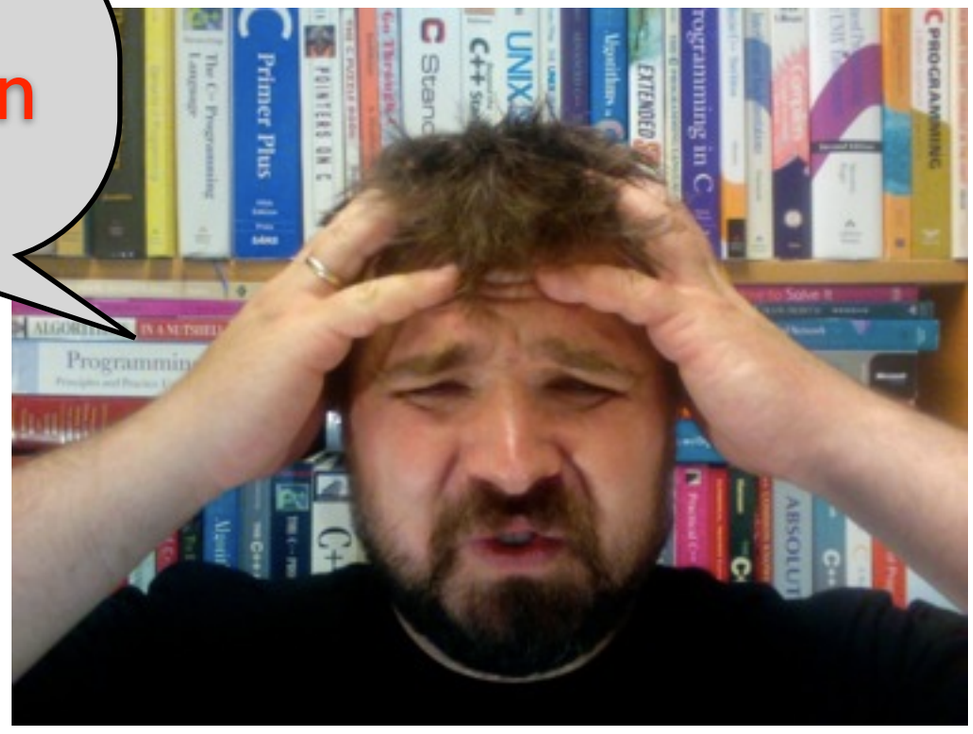
Find 3 issues

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

using namespace std;

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(vector<uint8_t> buffer);
        virtual vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const vector<uint8_t> buffer_;
    };

}

#endif
```

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

using namespace std;

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(vector<uint8_t> buffer);
        virtual vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const vector<uint8_t> buffer_;
    };

}

#endif
```



Ouch, do not import namespaces in header files!



```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

using namespace std;

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(vector<uint8_t> buffer);
        virtual vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const vector<uint8_t> buffer_;
    };

}

#endif
```

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

using namespace std;

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(vector<uint8_t> buffer);
        virtual vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const vector<uint8_t> buffer_;
    };

}

#endif
```

pass this vector as const &

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

using namespace std;

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(vector<uint8_t> buffer);
        virtual vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const vector<uint8_t> buffer_;
    };

}

#endif
```

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

using namespace std;

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(vector<uint8_t> buffer);
        virtual vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const vector<uint8_t> buffer_;
    };

}

#endif
```

this looks like query methods, they should probably be const

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

using namespace std;

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(vector<uint8_t> buffer);
        virtual vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const vector<uint8_t> buffer_;
    };

}

#endif
```

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

using namespace std;

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(vector<uint8_t> buffer);
        virtual vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const vector<uint8_t> buffer_;
    };

}

#endif
```

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(vector<uint8_t> buffer);
        virtual vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const vector<uint8_t> buffer_;
    };

}

#endif
```

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(vector<uint8_t> buffer);
        virtual vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const vector<uint8_t> buffer_;
    };

}

#endif
```

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(     vector<uint8_t> buffer);
        virtual      vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const       vector<uint8_t> buffer_;
    };

}

#endif
```

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(std::vector<uint8_t> buffer);
        virtual std::vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const std::vector<uint8_t> buffer_;
    };

}

#endif
```

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(std::vector<uint8_t> buffer);
        virtual std::vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const std::vector<uint8_t> buffer_;
    };

}

#endif
```

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(std::vector<uint8_t> buffer);
        virtual std::vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const std::vector<uint8_t> buffer_;
    };

}

#endif
```

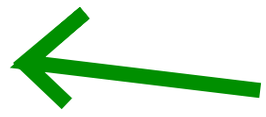
```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(std::vector<uint8_t> buffer);
        virtual std::vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const std::vector<uint8_t> buffer_;
    };

}

#endif
```

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(      std::vector<uint8_t>   buffer);
        virtual std::vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const std::vector<uint8_t> buffer_;
    };

}

#endif
```

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(const std::vector<uint8_t> & buffer);
        virtual std::vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const std::vector<uint8_t> buffer_;
    };

}

#endif
```

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(const std::vector<uint8_t> & buffer);
        virtual std::vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const std::vector<uint8_t> buffer_;
    };

}

#endif
```

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(const std::vector<uint8_t> & buffer);
        virtual std::vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags() const;
        uint64_t challenge() const;
    private:
        const std::vector<uint8_t> buffer_;
    };

}

#endif
```



```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(const std::vector<uint8_t> & buffer);
        virtual std::vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags() const;
        uint64_t challenge() const;
    private:
        const std::vector<uint8_t> buffer_;
    };

}

#endif
```

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(const std::vector<uint8_t> & buffer);
        virtual std::vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags() const;
        uint64_t challenge() const;
    private:
        const std::vector<uint8_t> buffer_;
    };

}

#endif
```

type2_message.cpp

... page 1 of 2

```
#include <cstddef>
#include <algorithm>
#include <stdexcept>

#include "tools.hpp"

#include "type2_message.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 2 Message
 *
 *   0  NTLMSSP Signature                {'N','T','L','M','S','S','P','\0'}
 *   8  NTLM Message Type                {0x02,0x00,0x00,0x00}
 *  12  Target Name                      (security buffer)
 *  20  Flags                            uint32 as little endian
 *  24  Challenge                        8 bytes / uint64 as little endian
 * (32) Context (optional)               8 bytes (2xlong)
 * (40) Target Information               (security buffer)
 * (48) (start of datablock)
 *      targetname
 *      targetinfo
 *          server (type=0x0100, len, data)
 *          domain (type=0x0200, len, data)
 *          dnsserver (type=0x0300, len, data)
 *          dnsdomain (type=0x0400, len, data)
 *          type5 (type=0x0500, len, data)   // unknown role
 *          <terminator> (type=0,len=0)
 */
```

...

... page 1 of 2

Find one issue

```
#include <cstddef>
#include <algorithm>
#include <stdexcept>

#include "tools.hpp"

#include "type2_message.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 2 Message
 *
 *   0  NTLMSSP Signature                {'N','T','L','M','S','S','P','\0'}
 *   8  NTLM Message Type                {0x02,0x00,0x00,0x00}
 *  12  Target Name                      (security buffer)
 *  20  Flags                            uint32 as little endian
 *  24  Challenge                        8 bytes / uint64 as little endian
 * (32) Context (optional)               8 bytes (2xlong)
 * (40) Target Information               (security buffer)
 * (48) (start of datablock)
 *      targetname
 *      targetinfo
 *          server (type=0x0100, len, data)
 *          domain (type=0x0200, len, data)
 *          dnsserver (type=0x0300, len, data)
 *          dnsdomain (type=0x0400, len, data)
 *          type5 (type=0x0500, len, data)   // unknown role
 *          <terminator> (type=0,len=0)
 */
```

...

... page 1 of 2

```
#include <cstddef>
#include <algorithm>
#include <stdexcept>

#include "tools.hpp"

#include "type2_message.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 2 Message
 *
 *   0  NTLMSSP Signature                {'N','T','L','M','S','S','P','\0'}
 *   8  NTLM Message Type                {0x02,0x00,0x00,0x00}
 *  12  Target Name                      (security buffer)
 *  20  Flags                            uint32 as little endian
 *  24  Challenge                        8 bytes / uint64 as little endian
 * (32) Context (optional)               8 bytes (2xlong)
 * (40) Target Information               (security buffer)
 * (48) (start of datablock)
 *      targetname
 *      targetinfo
 *          server (type=0x0100, len, data)
 *          domain (type=0x0200, len, data)
 *          dnsserver (type=0x0300, len, data)
 *          dnsdomain (type=0x0400, len, data)
 *          type5 (type=0x0500, len, data)   // unknown role
 *          <terminator> (type=0,len=0)
 */
```

...

... page 1 of 2

```
#include <cstddef>
#include <algorithm>
#include <stdexcept>

#include "tools.hpp"

#include "type2_message.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 2 Message
 *
 *   0  NTLMSSP Signature                {'N','T','L','M','S','S','P','\0'}
 *   8  NTLM Message Type                {0x02,0x00,0x00,0x00}
 *  12  Target Name                      (security buffer)
 *  20  Flags                            uint32 as little endian
 *  24  Challenge                        8 bytes / uint64 as little endian
 * (32) Context (optional)               8 bytes (2xlong)
 * (40) Target Information               (security buffer)
 * (48) (start of datablock)
 *      targetname
 *      targetinfo
 *          server (type=0x0100, len, data)
 *          domain (type=0x0200, len, data)
 *          dnsserver (type=0x0300, len, data)
 *          dnsdomain (type=0x0400, len, data)
 *          type5 (type=0x0500, len, data)   // unknown role
 *          <terminator> (type=0,len=0)
 */
```

...

... page 1 of 2

```
#include <cstddef>
#include <algorithm>
#include <stdexcept>

#include "tools.hpp"

#include "type2_message.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 2 Message
 *
 *   0  NTLMSSP Signature                {'N','T','L','M','S','S','P','\0'}
 *   8  NTLM Message Type                {0x02,0x00,0x00,0x00}
 *  12  Target Name                      (security buffer)
 *  20  Flags                            uint32 as little endian
 *  24  Challenge                        8 bytes / uint64 as little endian
 * (32) Context (optional)               8 bytes (2xlong)
 * (40) Target Information               (security buffer)
 * (48) (start of datablock)
 *      targetname
 *      targetinfo
 *          server (type=0x0100, len, data)
 *          domain (type=0x0200, len, data)
 *          dnsserver (type=0x0300, len, data)
 *          dnsdomain (type=0x0400, len, data)
 *          type5 (type=0x0500, len, data)   // unknown role
 *          <terminator> (type=0,len=0)
 */
```

...

... page 1 of 2

```
#include <cstddef>
#include <algorithm>
#include <stdexcept>

#include "tools.hpp"

#include "type2_message.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 2 Message
 *
 *   0  NTLMSSP Signature                {'N','T','L','M','S','S','P','\0'}
 *   8  NTLM Message Type                {0x02,0x00,0x00,0x00}
 *  12  Target Name                      (security buffer)
 *  20  Flags                            uint32 as little endian
 *  24  Challenge                        8 bytes / uint64 as little endian
 * (32) Context (optional)               8 bytes (2xlong)
 * (40) Target Information               (security buffer)
 * (48) (start of datablock)
 *      targetname
 *      targetinfo
 *          server (type=0x0100, len, data)
 *          domain (type=0x0200, len, data)
 *          dnsserver (type=0x0300, len, data)
 *          dnsdomain (type=0x0400, len, data)
 *          type5 (type=0x0500, len, data)   // unknown role
 *          <terminator> (type=0,len=0)
 */
```

...

... page 1 of 2

```
#include "type2_message.hpp"

#include "tools.hpp"

#include <cstddef>
#include <algorithm>
#include <stdexcept>

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 2 Message
 *
 *   0  NTLMSSP Signature                {'N','T','L','M','S','S','P','\0'}
 *   8  NTLM Message Type                {0x02,0x00,0x00,0x00}
 *  12  Target Name                      (security buffer)
 *  20  Flags                            uint32 as little endian
 *  24  Challenge                        8 bytes / uint64 as little endian
 * (32) Context (optional)               8 bytes (2xlong)
 * (40) Target Information               (security buffer)
 * (48) (start of datablock)
 *      targetname
 *      targetinfo
 *          server (type=0x0100, len, data)
 *          domain (type=0x0200, len, data)
 *          dnsserver (type=0x0300, len, data)
 *          dnsdomain (type=0x0400, len, data)
 *          type5 (type=0x0500, len, data)   // unknown role
 *          <terminator> (type=0,len=0)
 */
```

...

... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof(prefix), buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() const
{
    return pal::read_uint32_from_little_endian(&buffer_[20]);
}

uint64_t pal::type2_message::challenge() const
{
    return pal::read_uint64_from_little_endian(&buffer_[24]);
}

std::vector<uint8_t> pal::type2_message::as_bytes() const
{
    return (buffer_);
}
```

... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof(prefix), buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() const
{
    return pal::read_uint32_from_little_endian(&buffer_[20]);
}

uint64_t pal::type2_message::challenge() const
{
    return pal::read_uint64_from_little_endian(&buffer_[24]);
}

std::vector<uint8_t> pal::type2_message::as_bytes() const
{
    return (buffer_);
}
```

Any issues here?

... page 2 of 2

```
pal::type2_message::type2_message(const std::vecto
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 m
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof(prefix
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() const
{
    return pal::read_uint32_from_little_endian(&buffer_[20]);
}

uint64_t pal::type2_message::challenge() const
{
    return pal::read_uint64_from_little_endian(&buffer_[24]);
}

std::vector<uint8_t> pal::type2_message::as_bytes() const
{
    return (buffer_);
}
```

... page 2 of 2

magic numbers

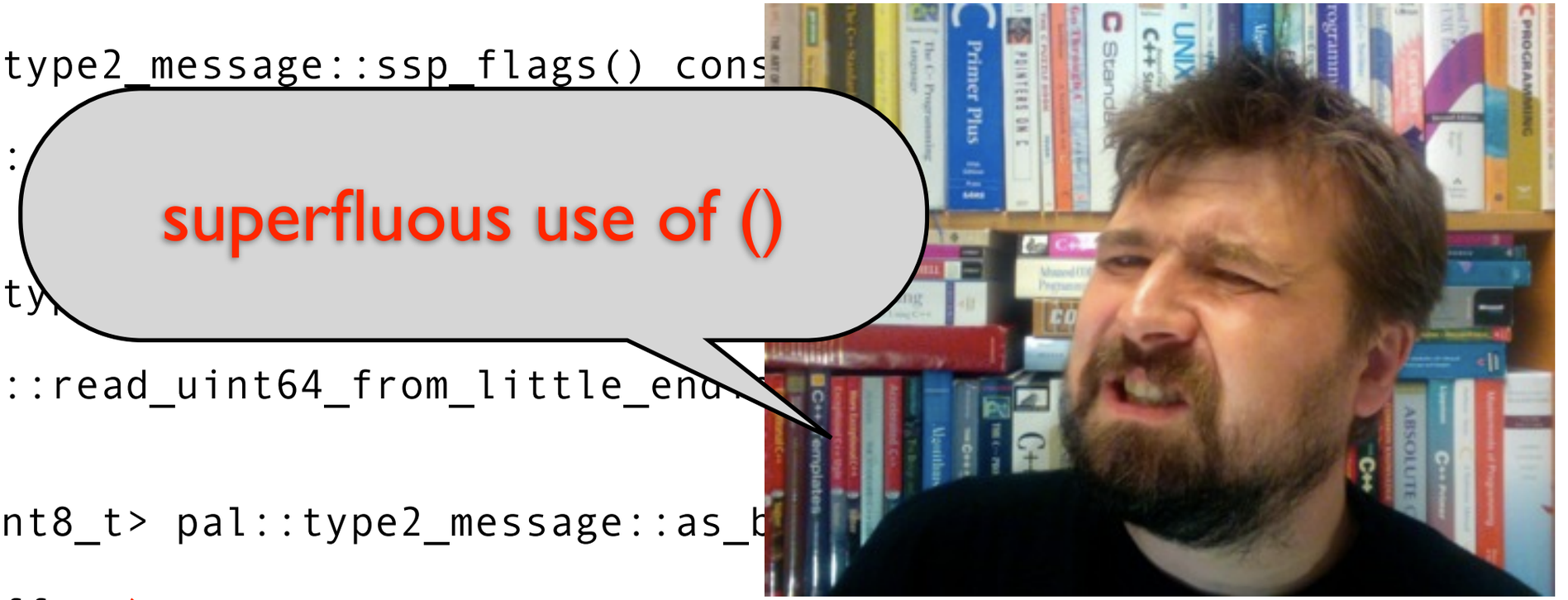
```
pal::type2_message::typ
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 m
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof(prefix
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() const
{
    return pal::read_uint32_from_little_endian(&buffer_[20]);
}

uint64_t pal::type2_message::challenge() const
{
    return pal::read_uint64_from_little_endian(&buffer_[24]);
}

std::vector<uint8_t> pal::type2_message::as_bytes() const
{
    return (buffer_);
}
```

... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof(prefix), buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() cons
{
    return pal::read_uint32_from_little_endia
}

uint64_t pal::type2_message::challenge() cons
{
    return pal::read_uint64_from_little_endia
}

std::vector<uint8_t> pal::type2_message::as_b
{
    return (buffer_);
}
```

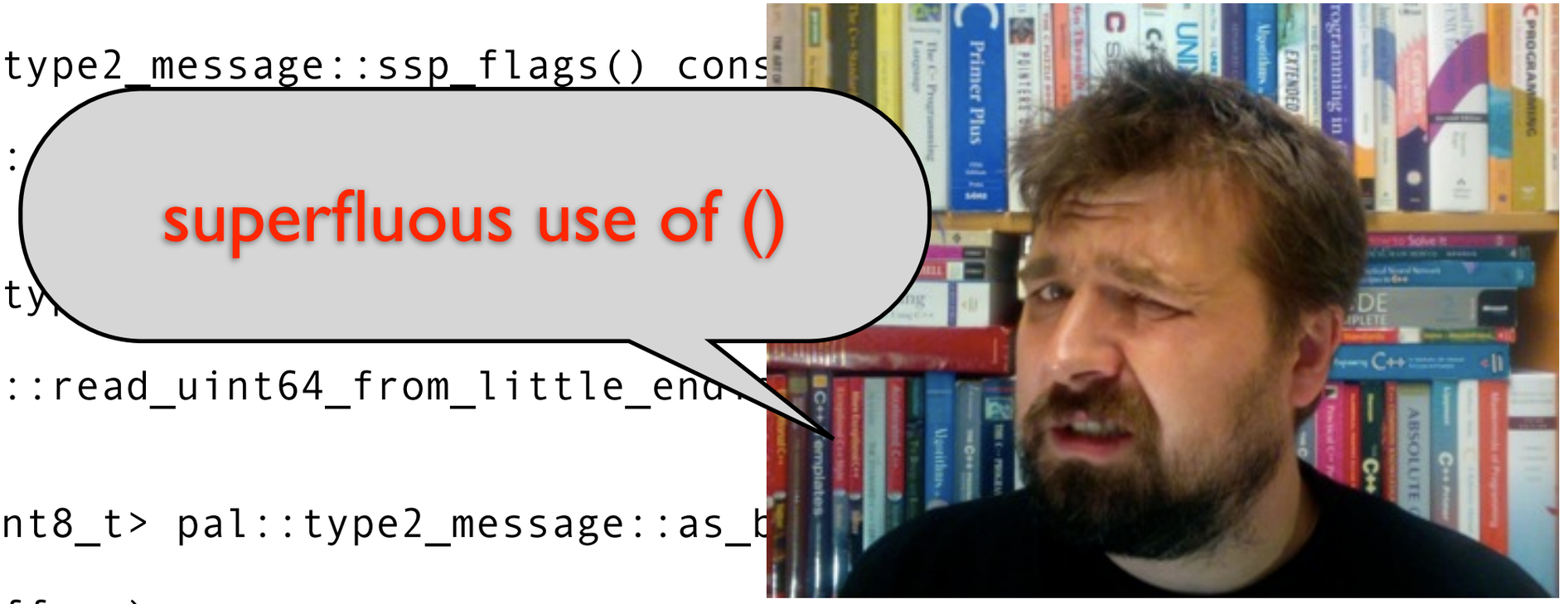
... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof(prefix), buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() cons
{
    return pal:
}

uint64_t pal::ty
{
    return pal::read_uint64_from_little_end
}

std::vector<uint8_t> pal::type2_message::as_b
{
    return (buffer_);
}
```



... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof(prefix), buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() cons
{
    return pal:
}

uint64_t pal::ty
{
    return pal::read_uint64_from_little_end
}

std::vector<uint8_t> pal::type2_message::as_b
{
    return (buffer_);
}
```



... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof((prefix)), buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() cons
{
    return
}

uint64_t pa
{
    return pal::read_uint64_from_little_endia
}

std::vector<uint8_t> pal::type2_message::as_b
{
    return (((buffer_)));
}
```

... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = (32);
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof((prefix)), buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() cons
{
    return
}

uint64_t pa
{
    return pal::read_uint64_from_little_endia
}

std::vector<uint8_t> pal::type2_message::as_b
{
    return (((buffer_)));
}
```

... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof(prefix), buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() const
{
    return pal::read_uint32_from_little_endian(&buffer_[20]);
}

uint64_t pal::type2_message::challenge() const
{
    return pal::read_uint64_from_little_endian(&buffer_[24]);
}

std::vector<uint8_t> pal::type2_message::as_bytes() const
{
    return (buffer_);
}
```

... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof(prefix), buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() const
{

    return pal::read_uint32_from_little_endian(&buffer_[20]);
}

uint64_t pal::type2_message::challenge() const
{
    return pal::read_uint64_from_little_endian(&buffer_[24]);
}

std::vector<uint8_t> pal::type2_message::as_bytes() const
{
    return (buffer_);
}
```

... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof(prefix), buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() const
{

    return pal::read_uint32_from_little_endian(&buffer_[20]);
}

uint64_t pal::type2_message::challenge() const
{

    return pal::read_uint64_from_little_endian(&buffer_[24]);
}

std::vector<uint8_t> pal::type2_message::as_bytes() const
{
    return (buffer_);
}
```

... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof(prefix), buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() const
{

    return pal::read_uint32_from_little_endian(&buffer_[20]);
}

uint64_t pal::type2_message::challenge() const
{

    return pal::read_uint64_from_little_endian(&buffer_[24]);
}

std::vector<uint8_t> pal::type2_message::as_bytes() const
{
    return (buffer_);
}
```

... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof(prefix), buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() const
{
    const std::size_t ssp_flags_offset = 20;
    return pal::read_uint32_from_little_endian(&buffer_[20]);
}

uint64_t pal::type2_message::challenge() const
{
    const std::size_t challenge_offset = 24;
    return pal::read_uint64_from_little_endian(&buffer_[24]);
}

std::vector<uint8_t> pal::type2_message::as_bytes() const
{
    return (buffer_);
}
```

... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof(prefix), buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() const
{
    const std::size_t ssp_flags_offset = 20;
    return pal::read_uint32_from_little_endian(&buffer_[20]);
}

uint64_t pal::type2_message::challenge() const
{
    const std::size_t challenge_offset = 24;
    return pal::read_uint64_from_little_endian(&buffer_[24]);
}

std::vector<uint8_t> pal::type2_message::as_bytes() const
{
    return (buffer_);
}
```

... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof(prefix), buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() const
{
    const std::size_t ssp_flags_offset = 20;
    return pal::read_uint32_from_little_endian(&buffer_[ssp_flags_offset]);
}

uint64_t pal::type2_message::challenge() const
{
    const std::size_t challenge_offset = 24;
    return pal::read_uint64_from_little_endian(&buffer_[24]);
}

std::vector<uint8_t> pal::type2_message::as_bytes() const
{
    return (buffer_);
}
```

... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof(prefix), buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}
```

```
uint32_t pal::type2_message::ssp_flags() const
{
    const std::size_t ssp_flags_offset = 20;
    return pal::read_uint32_from_little_endian(&buffer_[ssp_flags_offset]);
}

uint64_t pal::type2_message::challenge() const
{
    const std::size_t challenge_offset = 24;
    return pal::read_uint64_from_little_endian(&buffer_[challenge_offset]);
}

std::vector<uint8_t> pal::type2_message::as_bytes() const
{
    return (buffer_);
}
```

… page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof(prefix), buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}
```

```
uint32_t pal::type2_message::ssp_flags() const
{
    const std::size_t ssp_flags_offset = 20;
    return pal::read_uint32_from_little_endian(&buffer_[ssp_flags_offset]);
}

uint64_t pal::type2_message::challenge() const
{
    const std::size_t challenge_offset = 24;
    return pal::read_uint64_from_little_endian(&buffer_[challenge_offset]);
}

std::vector<uint8_t> pal::type2_message::as_bytes() const
{
    return (buffer_);
}
```

... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof prefix , buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}
```

```
uint32_t pal::type2_message::ssp_flags() const
{
    const std::size_t ssp_flags_offset = 20;
    return pal::read_uint32_from_little_endian(&buffer_[ssp_flags_offset]);
}

uint64_t pal::type2_message::challenge() const
{
    const std::size_t challenge_offset = 24;
    return pal::read_uint64_from_little_endian(&buffer_[challenge_offset]);
}

std::vector<uint8_t> pal::type2_message::as_bytes() const
{
    return (buffer_);
}
```

... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof prefix , buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() const
{
    const std::size_t ssp_flags_offset = 20;
    return pal::read_uint32_from_little_endian(&buffer_[ssp_flags_offset]);
}

uint64_t pal::type2_message::challenge() const
{
    const std::size_t challenge_offset = 24;
    return pal::read_uint64_from_little_endian(&buffer_[challenge_offset]);
}

std::vector<uint8_t> pal::type2_message::as_bytes() const
{
    return (buffer_);
}
```

... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof prefix , buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() const
{
    const std::size_t ssp_flags_offset = 20;
    return pal::read_uint32_from_little_endian(&buffer_[ssp_flags_offset]);
}

uint64_t pal::type2_message::challenge() const
{
    const std::size_t challenge_offset = 24;
    return pal::read_uint64_from_little_endian(&buffer_[challenge_offset]);
}

std::vector<uint8_t> pal::type2_message::as_bytes() const
{
    return  buffer_ ;
}
```

... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof prefix, buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() const
{
    const std::size_t ssp_flags_offset = 20;
    return pal::read_uint32_from_little_endian(&buffer_[ssp_flags_offset]);
}

uint64_t pal::type2_message::challenge() const
{
    const std::size_t challenge_offset = 24;
    return pal::read_uint64_from_little_endian(&buffer_[challenge_offset]);
}

std::vector<uint8_t> pal::type2_message::as_bytes() const
{
    return buffer_;
}
```

... page 2 of 2

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof prefix, buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() const
{
    const std::size_t ssp_flags_offset = 20;
    return pal::read_uint32_from_little_endian(&buffer_[ssp_flags_offset]);
}

uint64_t pal::type2_message::challenge() const
{
    const std::size_t challenge_offset = 24;
    return pal::read_uint64_from_little_endian(&buffer_[challenge_offset]);
}

std::vector<uint8_t> pal::type2_message::as_bytes() const
{
    return buffer_;
}
```

type3_message.hpp

```
#ifndef PAL_TYPE3_MESSAGE_HPP_INCLUDED
#define PAL_TYPE3_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <string>
#include <iostream>

namespace pal {

    class type3_message : public ntlm_message {
    public:
        explicit type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & user,
            uint32_t ssp_flags = 0x202);
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

Find 3 issues

```
#ifndef PAL_TYPE3_MESSAGE_HPP_INCLUDED
#define PAL_TYPE3_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <string>
#include <iostream>

namespace pal {

    class type3_message : public ntlm_message {
    public:
        explicit type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & user,
            uint32_t ssp_flags = 0x202);
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

```
#ifndef PAL_TYPE3_MESSAGE_HPP_INCLUDED
#define PAL_TYPE3_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <string>
#include <iostream>

namespace pal {

    class type3_message : public ntlm_message {
    public:
        explicit type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & user,
            uint32_t ssp_flags = 0x202);
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

```
#ifndef PAL_TYPE3_MESS
#define PAL_TYPE3_MES

#include "ntlm_message

#include <string>
#include <iostream>

namespace pal {

    class type3_message : public ntlm_message {
    public:
        explicit type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & user,
            uint32_t ssp_flags = 0x202);
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

include <iosfwd> instead

```
#ifndef PAL_TYPE3_MESSAGE_HPP_INCLUDED
#define PAL_TYPE3_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <string>
#include <iostream>

namespace pal {

    class type3_message : public ntlm_message {
    public:
        explicit type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & domain,
            const std::string & user,
            uint32_t ssp_flags = 0x202);
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

```
#ifndef PAL_TYPE3_MESS
#define PAL_TYPE3_MES

#include "ntlm_message

#include <string>
#include <iostream>

namespace pal {

    class type3_message : public ntlm_message {
    public:
        explicit type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & domain,
            const std::string & user,
            uint32_t ssp_flags = 0x202);
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

this default argument is probably not needed

```
#ifndef PAL_TYPE3_MESSAGE_HPP_INCLUDED
#define PAL_TYPE3_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <string>
#include <iostream>

namespace pal {

    class type3_message : public ntlm_message {
    public:
        explicit type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & user,
            uint32_t ssp_flags = 0x202);
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

```
#ifndef PAL_TYPE3_MESSAGE_HPP
#define PAL_TYPE3_MESSAGE_HPP

#include "ntlm_message.hpp"

#include <string>
#include <iostream>

namespace pal {

    class type3_message : public ntlm_message {
    public:
        explicit type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & user,
            uint32_t ssp_flags = 0x202);
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

useless explicit specifier

```
#ifndef PAL_TYPE3_MESSAGE_HPP_INCLUDED
#define PAL_TYPE3_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <string>
#include <iostream>

namespace pal {

    class type3_message : public ntlm_message {
    public:
        explicit type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & user,
            uint32_t ssp_flags = 0x202);
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

```
#ifndef PAL_TYPE3_MESSAGE_HPP_INCLUDED
#define PAL_TYPE3_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <string>
#include <iostream>

namespace pal {

    class type3_message : public ntlm_message {
    public:
        explicit type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & user,
            uint32_t ssp_flags = 0x202);
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

```
#ifndef PAL_TYPE3_MESSAGE_HPP_INCLUDED
#define PAL_TYPE3_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <string>
#include <iosfwd>

namespace pal {

    class type3_message : public ntlm_message {
    public:
        explicit type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & user,
            uint32_t ssp_flags = 0x202);
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

```
#ifndef PAL_TYPE3_MESSAGE_HPP_INCLUDED
#define PAL_TYPE3_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <string>
#include <iosfwd>

namespace pal {

    class type3_message : public ntlm_message {
    public:
        explicit type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & user,
            uint32_t ssp_flags = 0x202);
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

```
#ifndef PAL_TYPE3_MESSAGE_HPP_INCLUDED
#define PAL_TYPE3_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <string>
#include <iosfwd>

namespace pal {

    class type3_message : public ntlm_message {
    public:
        explicit type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & user,
            uint32_t ssp_flags = 0x202);
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

```
#ifndef PAL_TYPE3_MESSAGE_HPP_INCLUDED
#define PAL_TYPE3_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <string>
#include <iosfwd>

namespace pal {

    class type3_message : public ntlm_message {
    public:
                type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & user,
                uint32_t ssp_flags = 0x202);
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

```
#ifndef PAL_TYPE3_MESSAGE_HPP_INCLUDED
#define PAL_TYPE3_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <string>
#include <iosfwd>

namespace pal {

    class type3_message : public ntlm_message {
    public:
                type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & user,
            uint32_t ssp_flags = 0x202);
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

```
#ifndef PAL_TYPE3_MESSAGE_HPP_INCLUDED
#define PAL_TYPE3_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <string>
#include <iosfwd>

namespace pal {

    class type3_message : public ntlm_message {
    public:
                type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & user,
            uint32_t ssp_flags        );
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

```
#ifndef PAL_TYPE3_MESSAGE_HPP_INCLUDED
#define PAL_TYPE3_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <string>
#include <iosfwd>

namespace pal {

    class type3_message : public ntlm_message {
    public:
        type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & user,
            uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

```
#ifndef PAL_TYPE3_MESSAGE_HPP_INCLUDED
#define PAL_TYPE3_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <string>
#include <iosfwd>

namespace pal {

    class type3_message : public ntlm_message {
    public:
        type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & user,
            uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

```
#ifndef PAL_TYPE3_MESSAGE_HPP_INCLUDED
#define PAL_TYPE3_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <string>
#include <iosfwd>

namespace pal {

    class type3_message : public ntlm_message {
    public:
        type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & user,
            uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

type3_message.cpp

...

```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),
    user_(user),
    ssp_flags_(ssp_flags),
    session_key_(session_key_size)
{
    if (lm_response_.size() != lm_response_size)
        throw new std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw new std::invalid_argument("invalid size of nt_response");
}
```

...

...

Find 3 issues

```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),
    user_(user),
    ssp_flags_(ssp_flags),
    session_key_(session_key_size)
{
    if (lm_response_.size() != lm_response_size)
        throw new std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw new std::invalid_argument("invalid size of nt_response");
}
```

...

...

```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),
    user_(user),
    ssp_flags_(ssp_flags),
    session_key_(session_key_size)
{
    if (lm_response_.size() != lm_response_size)
        throw new std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw new std::invalid_argument("invalid size of nt_response");
}
```

...

...

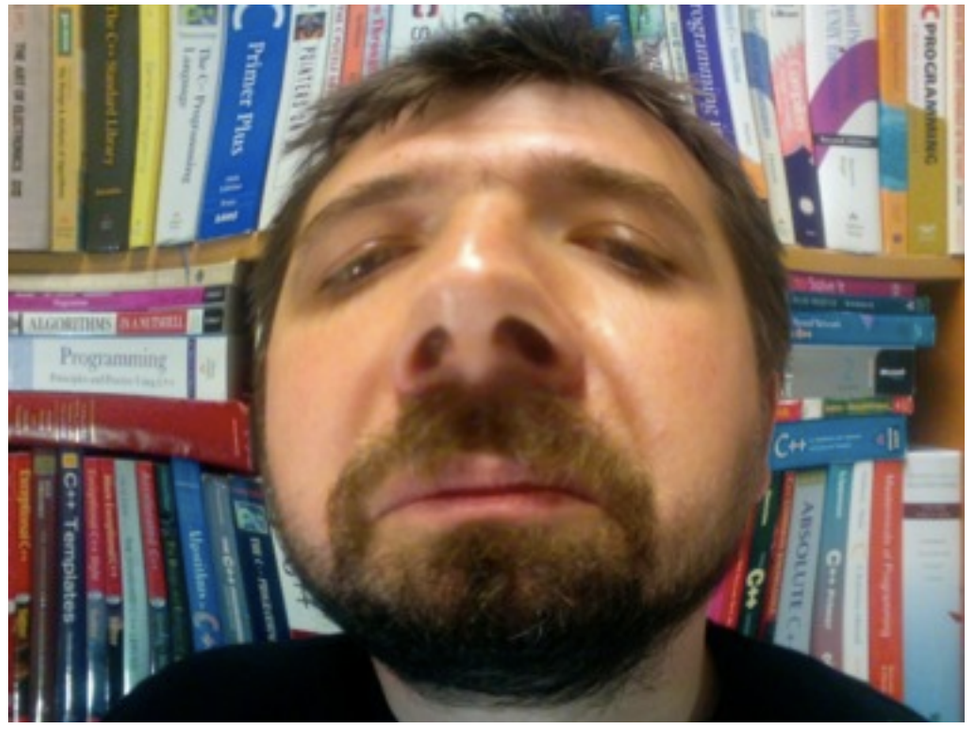
```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),
    user_(user),
    ssp_flags_(ssp_flags),
    session_key_(session_key_size)
{
    if (lm_response_.size() != lm_response_size)
        throw new std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw new std::invalid_argument("invalid size of nt_response");
}
```

...

...

```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),
    user_(user),
    ssp_flags_(ssp_flags),
    session_key_(session_key_size)
{
    if (lm_response_.size() != lm_response_size)
        throw new std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw new std::invalid_argument("invalid size of nt_response");
}
```

...



...

Does not match initialization order...

```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),
    user_(user),
    ssp_flags_(ssp_flags),
    session_key_(session_key_size)
{
    if (lm_response_.size() != lm_response_size)
        throw new std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw new std::invalid_argument("invalid size of nt_response");
}
```

...

...

```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),
    user_(user),
    ssp_flags_(ssp_flags),
    session_key_(session_key_size)
{
    if (lm_response_.size() != lm_response_size)
        throw new std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw new std::invalid_argument("invalid size of nt_response");
}
```

...

...

... but this indicates a deeper problem, the user is not compiling with -Wall and -Wextra, or even worse... they do not care about warnings!

```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),
    user_(user),
    ssp_flags_(ssp_flags),
    session_key_(session_key_size)
{
    if (lm_response_.size() != lm_response_size)
        throw new std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw new std::invalid_argument("invalid size of nt_response");
}
```

...

...

```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),
    user_(user),
    ssp_flags_(ssp_flags),
    session_key_(session_key_size)
{
    if (lm_response_.size() != lm_response_size)
        throw new std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw new std::invalid_argument("invalid size of nt_response");
}
```

...

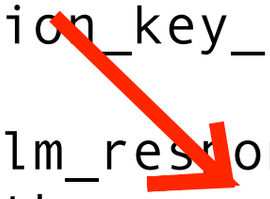
...

```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),
    user_(user),
    ssp_flags_(ssp_flags),
    session_key_(session_key_size)
{
    if (lm_response_.size() != lm_response_size)
        throw new std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw new std::invalid_argument("invalid size of nt_response");
}
```

...

...

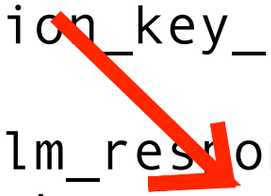
```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),
    user_(user),
    ssp_flags_(ssp_flags),
    session_key_(session_key_size)
{
    if (lm_response_.size() != lm_response_size)
        throw new std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw new std::invalid_argument("invalid size of nt_response");
}
```



...

...

```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),
    user_(user),
    ssp_flags_(ssp_flags),
    session_key_(session_key_size)
{
    if (lm_response_.size() != lm_response_size)
        throw     std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw     std::invalid_argument("invalid size of nt_response");
}
```



...

...

```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),
    user_(user),
    ssp_flags_(ssp_flags),
    session_key_(session_key_size)
{
    if (lm_response_.size() != lm_response_size)
        throw std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw std::invalid_argument("invalid size of nt_response");
}
```

...

...

```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),
    user_(user),
    ssp_flags_(ssp_flags),
    session_key_(session_key_size)
{
    if (lm_response_.size() != lm_response_size)
        throw std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw std::invalid_argument("invalid size of nt_response");
}
```

...

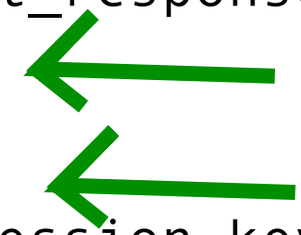
...

```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),
    user_(user),
    session_key_(session_key_size),
    ssp_flags_(ssp_flags)
{
    if (lm_response_.size() != lm_response_size)
        throw std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw std::invalid_argument("invalid size of nt_response");
}
```

...

...

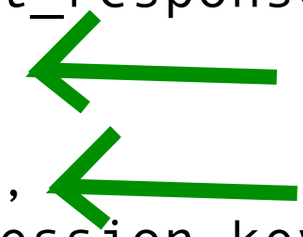
```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),

    user_(user),

    session_key_(session_key_size),
    ssp_flags_(ssp_flags)
{
    if (lm_response_.size() != lm_response_size)
        throw std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw std::invalid_argument("invalid size of nt_response");
}
```



...

...

```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),
    domain_(),
    user_(user),
    workstation_(),
    session_key_(session_key_size),
    ssp_flags_(ssp_flags)
{
    if (lm_response_.size() != lm_response_size)
        throw std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw std::invalid_argument("invalid size of nt_response");
}
```



...

...

```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),
    domain_(),
    user_(user),
    workstation_(),
    session_key_(session_key_size),
    ssp_flags_(ssp_flags)
{
    if (lm_response_.size() != lm_response_size)
        throw std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw std::invalid_argument("invalid size of nt_response");
}
```

...

...

```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),
    domain_(),
    user_(user),
    workstation_(),
    session_key_(session_key_size),
    ssp_flags_(ssp_flags)
{
    if (lm_response_.size() != lm_response_size)
        throw std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw std::invalid_argument("invalid size of nt_response");
}
```

...

...

```
void pal::type3_message::debug_print(std::ostream & out) const
{
    out << "### type3_message:" << '\n'
        << pal::as_hex_dump(as_bytes())
        << "lm_response = " << pal::as_hex_string(lm_response_)
        << "\nnt_response = " << pal::as_hex_string(nt_response_)
        << "\ndomain = " << domain_
        << "\nuser = " << user_
        << "\nworkstation = " << workstation_
        << "\nsession_key = " << pal::as_hex_string(session_key_)
        << std::hex << std::setw(8) << std::setfill('0')
        << "\nssp_flags = " << ssp_flags_;
}
```

...

Any issues here?

...

```
void pal::type3_message::debug_print(std::ostream & out) const
{
    out << "### type3_message:" << '\n'
        << pal::as_hex_dump(as_bytes())
        << "lm_response = " << pal::as_hex_string(lm_response_)
        << "\nnt_response = " << pal::as_hex_string(nt_response_)
        << "\ndomain = " << domain_
        << "\nuser = " << user_
        << "\nworkstation = " << workstation_
        << "\nsession_key = " << pal::as_hex_string(session_key_)
        << std::hex << std::setw(8) << std::setfill('0')
        << "\nssp_flags = " << ssp_flags_;
}
```

...

...

```
void pal::type3_message::debug_print(std::ostream & out) const
{
    out << "### type3_message:" << '\n'
        << pal::as_hex_dump(as_bytes())
        << "lm_response = " << pal::as_hex_string(lm_response_)
        << "\nnt_response = " << pal::as_hex_string(nt_response_)
        << "\ndomain = " << domain_
        << "\nuser = " << user_
        << "\nworkstation = " << workstation_
        << "\nsession_key = " << pal::as_hex_string(session_key_)
        << std::hex << std::setw(8) << std::setfill('0')
        << "\nssp_flags = " << ssp_flags_;
}
```

...

DO NOT MESS WITH BORROWED THINGS!

...

```
void pal::type3_message::debug_print(std::ostream & out) const
{
    out << "### type3_message:" << '\n'
        << pal::as_hex_dump(as_bytes())
        << "lm_response = " << pal::as_hex_string(lm_response_)
        << "\nnt_response = " << pal::as_hex_string(nt_response_)
        << "\ndomain = " << domain_
        << "\nuser = " << user_
        << "\nworkstation = " << workstation_
        << "\nsession_key = " << pal::as_hex_string(session_key_)
        << std::hex << std::setw(8) << std::setfill('0')
        << "\nssp_flags = " << ssp_flags_;
}
```

...

...

```
void pal::type3_message::debug_print(std::ostream & out) const
{
    out << "### type3_message:" << '\n'
        << pal::as_hex_dump(as_bytes())
        << "lm_response = " << pal::as_hex_string(lm_response_)
        << "\nnt_response = " << pal::as_hex_string(nt_response_)
        << "\ndomain = " << domain_
        << "\nuser = " << user_
        << "\nworkstation = " << workstation_
        << "\nsession_key = " << pal::as_hex_string(session_key_)
        << std::hex << std::setw(8) << std::setfill('0')
        << "\nssp_flags = " << ssp_flags_;
}
```

...

No easy way to fix this! Probably better to build into a local stringstream and return it.

```
...
void pal::type3_message::debug_print(std::ostream & out) const
{
    out << "### type3_message:" << '\n'
        << pal::as_hex_dump(as_bytes())
        << "lm_response = " << pal::as_hex_string(lm_response_)
        << "\nnt_response = " << pal::as_hex_string(nt_response_)
        << "\ndomain = " << domain_
        << "\nuser = " << user_
        << "\nworkstation = " << workstation_
        << "\nsession_key = " << pal::as_hex_string(session_key_)
        << std::hex << std::setw(8) << std::setfill('0')
        << "\nssp_flags = " << ssp_flags_;
}


...
```

...

```
void pal::type3_message::debug_print(std::ostream & out) const
{
    out << "### type3_message:" << '\n'
        << pal::as_hex_dump(as_bytes())
        << "lm_response = " << pal::as_hex_string(lm_response_)
        << "\nnt_response = " << pal::as_hex_string(nt_response_)
        << "\ndomain = " << domain_
        << "\nuser = " << user_
        << "\nworkstation = " << workstation_
        << "\nsession_key = " << pal::as_hex_string(session_key_)
        << std::hex << std::setw(8) << std::setfill('0')
        << "\nssp_flags = " << ssp_flags_;
}
```

...

...

```
std::string pal::type3_message::debug_print() const
{
    std::ostringstream buf;
    buf << "### type3_message:" << '\n'
        << pal::as_hex_dump(as_bytes())
        << "lm_response = " << pal::as_hex_string(lm_response_)
        << "\nnt_response = " << pal::as_hex_string(nt_response_)
        << "\ndomain = " << domain_
        << "\nuser = " << user_
        << "\nworkstation = " << workstation_
        << "\nsession_key = " << pal::as_hex_string(session_key_)
        << std::hex << std::setw(8) << std::setfill('0')
        << "\nssp_flags = " << ssp_flags_;
    return buf.str();
}
```

...

...

```
std::string pal::type3_message::debug_print() const
{
    std::ostringstream buf;
    buf << "### type3_message:" << '\n'
        << pal::as_hex_dump(as_bytes())
        << "lm_response = " << pal::as_hex_string(lm_response_)
        << "\nnt_response = " << pal::as_hex_string(nt_response_)
        << "\ndomain = " << domain_
        << "\nuser = " << user_
        << "\nworkstation = " << workstation_
        << "\nsession_key = " << pal::as_hex_string(session_key_)
        << std::hex << std::setw(8) << std::setfill('0')
        << "\nssp_flags = " << ssp_flags_;
    return buf.str();
}
```

...

Big improvement!

...

```
std::string pal::type3_message::debug_print() const
{
    std::ostringstream buf;
    buf << "### type3_message:" << '\n'
        << pal::as_hex_dump(as_bytes())
        << "lm_response = " << pal::as_hex_string(lm_response_)
        << "\nnt_response = " << pal::as_hex_string(nt_response_)
        << "\ndomain = " << domain_
        << "\nuser = " << user_
        << "\nworkstation = " << workstation_
        << "\nsession_key = " << pal::as_hex_string(session_key_)
        << std::hex << std::setw(8) << std::setfill('0')
        << "\nssp_flags = " << ssp_flags_;
    return buf.str();
}
```

...

Issues discussed here

Issues discussed here

1. base classes should have virtual destructors

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff
7. always focus on readability, you spend more time reading code than writing it

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff
7. always focus on readability, you spend more time reading code than writing it
8. importing a namespace in implementation files is usually not a good idea

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff
7. always focus on readability, you spend more time reading code than writing it
8. importing a namespace in implementation files is usually not a good idea
9. "never" import a namespace in a header file

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff
7. always focus on readability, you spend more time reading code than writing it
8. importing a namespace in implementation files is usually not a good idea
9. "never" import a namespace in a header file
10. initialize objects properly, use the initialization list

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff
7. always focus on readability, you spend more time reading code than writing it
8. importing a namespace in implementation files is usually not a good idea
9. "never" import a namespace in a header file
10. initialize objects properly, use the initialization list
11. prefer std::size_t when working with memory indexing and offsets

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff
7. always focus on readability, you spend more time reading code than writing it
8. importing a namespace in implementation files is usually not a good idea
9. "never" import a namespace in a header file
10. initialize objects properly, use the initialization list
11. prefer std::size_t when working with memory indexing and offsets
12. for non-trivial objects, prefer pass by const over pass by copy

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff
7. always focus on readability, you spend more time reading code than writing it
8. importing a namespace in implementation files is usually not a good idea
9. "never" import a namespace in a header file
10. initialize objects properly, use the initialization list
11. prefer std::size_t when working with memory indexing and offsets
12. for non-trivial objects, prefer pass by const over pass by copy
13. query methods should be specified as const

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff
7. always focus on readability, you spend more time reading code than writing it
8. importing a namespace in implementation files is usually not a good idea
9. "never" import a namespace in a header file
10. initialize objects properly, use the initialization list
11. prefer std::size_t when working with memory indexing and offsets
12. for non-trivial objects, prefer pass by const over pass by copy
13. query methods should be specified as const
14. order the include files like this; own, project/platform, standard

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff
7. always focus on readability, you spend more time reading code than writing it
8. importing a namespace in implementation files is usually not a good idea
9. "never" import a namespace in a header file
10. initialize objects properly, use the initialization list
11. prefer std::size_t when working with memory indexing and offsets
12. for non-trivial objects, prefer pass by const over pass by copy
13. query methods should be specified as const
14. order the include files like this; own, project/platform, standard
15. avoid magic numbers, use explaination variables

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff
7. always focus on readability, you spend more time reading code than writing it
8. importing a namespace in implementation files is usually not a good idea
9. "never" import a namespace in a header file
10. initialize objects properly, use the initialization list
11. prefer std::size_t when working with memory indexing and offsets
12. for non-trivial objects, prefer pass by const over pass by copy
13. query methods should be specified as const
14. order the include files like this; own, project/platform, standard
15. avoid magic numbers, use explaination variables
16. avoid superfluous use of ()

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff
7. always focus on readability, you spend more time reading code than writing it
8. importing a namespace in implementation files is usually not a good idea
9. "never" import a namespace in a header file
10. initialize objects properly, use the initialization list
11. prefer std::size_t when working with memory indexing and offsets
12. for non-trivial objects, prefer pass by const over pass by copy
13. query methods should be specified as const
14. order the include files like this; own, project/platform, standard
15. avoid magic numbers, use explaination variables
16. avoid superfluous use of ()
17. prefer forward declarations when you can

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff
7. always focus on readability, you spend more time reading code than writing it
8. importing a namespace in implementation files is usually not a good idea
9. "never" import a namespace in a header file
10. initialize objects properly, use the initialization list
11. prefer std::size_t when working with memory indexing and offsets
12. for non-trivial objects, prefer pass by const over pass by copy
13. query methods should be specified as const
14. order the include files like this; own, project/platform, standard
15. avoid magic numbers, use explaination variables
16. avoid superfluous use of ()
17. prefer forward declarations when you can
18. do not use explicit on multi argument constructors

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff
7. always focus on readability, you spend more time reading code than writing it
8. importing a namespace in implementation files is usually not a good idea
9. "never" import a namespace in a header file
10. initialize objects properly, use the initialization list
11. prefer std::size_t when working with memory indexing and offsets
12. for non-trivial objects, prefer pass by const over pass by copy
13. query methods should be specified as const
14. order the include files like this; own, project/platform, standard
15. avoid magic numbers, use explaination variables
16. avoid superfluous use of ()
17. prefer forward declarations when you can
18. do not use explicit on multi argument constructors
19. consider explainability, don't do things that needs elaborate explainations

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff
7. always focus on readability, you spend more time reading code than writing it
8. importing a namespace in implementation files is usually not a good idea
9. "never" import a namespace in a header file
10. initialize objects properly, use the initialization list
11. prefer std::size_t when working with memory indexing and offsets
12. for non-trivial objects, prefer pass by const over pass by copy
13. query methods should be specified as const
14. order the include files like this; own, project/platform, standard
15. avoid magic numbers, use explaination variables
16. avoid superfluous use of ()
17. prefer forward declarations when you can
18. do not use explicit on multi argument constructors
19. consider explainability, don't do things that needs elaborate explainations
20. avoid default arguments (they are often not used anyway)

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff
7. always focus on readability, you spend more time reading code than writing it
8. importing a namespace in implementation files is usually not a good idea
9. "never" import a namespace in a header file
10. initialize objects properly, use the initialization list
11. prefer std::size_t when working with memory indexing and offsets
12. for non-trivial objects, prefer pass by const over pass by copy
13. query methods should be specified as const
14. order the include files like this; own, project/platform, standard
15. avoid magic numbers, use explaination variables
16. avoid superfluous use of ()
17. prefer forward declarations when you can
18. do not use explicit on multi argument constructors
19. consider explainability, don't do things that needs elaborate explainations
20. avoid default arguments (they are often not used anyway)
21. do not throw pointers to exceptions (eg, not "throw new ...")

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff
7. always focus on readability, you spend more time reading code than writing it
8. importing a namespace in implementation files is usually not a good idea
9. "never" import a namespace in a header file
10. initialize objects properly, use the initialization list
11. prefer std::size_t when working with memory indexing and offsets
12. for non-trivial objects, prefer pass by const over pass by copy
13. query methods should be specified as const
14. order the include files like this; own, project/platform, standard
15. avoid magic numbers, use explaination variables
16. avoid superfluous use of ()
17. prefer forward declarations when you can
18. do not use explicit on multi argument constructors
19. consider explainability, don't do things that needs elaborate explainations
20. avoid default arguments (they are often not used anyway)
21. do not throw pointers to exceptions (eg, not "throw new ...")
22. treat warnings like errors (-Werror) and compile with high warning levels (-Wall -Wextra)

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff
7. always focus on readability, you spend more time reading code than writing it
8. importing a namespace in implementation files is usually not a good idea
9. "never" import a namespace in a header file
10. initialize objects properly, use the initialization list
11. prefer std::size_t when working with memory indexing and offsets
12. for non-trivial objects, prefer pass by const over pass by copy
13. query methods should be specified as const
14. order the include files like this; own, project/platform, standard
15. avoid magic numbers, use explaination variables
16. avoid superfluous use of ()
17. prefer forward declarations when you can
18. do not use explicit on multi argument constructors
19. consider explainability, don't do things that needs elaborate explainations
20. avoid default arguments (they are often not used anyway)
21. do not throw pointers to exceptions (eg, not "throw new ...")
22. treat warnings like errors (-Werror) and compile with high warning levels (-Wall -Wextra)
23. consider using -Weffc++ and -pedantic as well

Issues discussed here

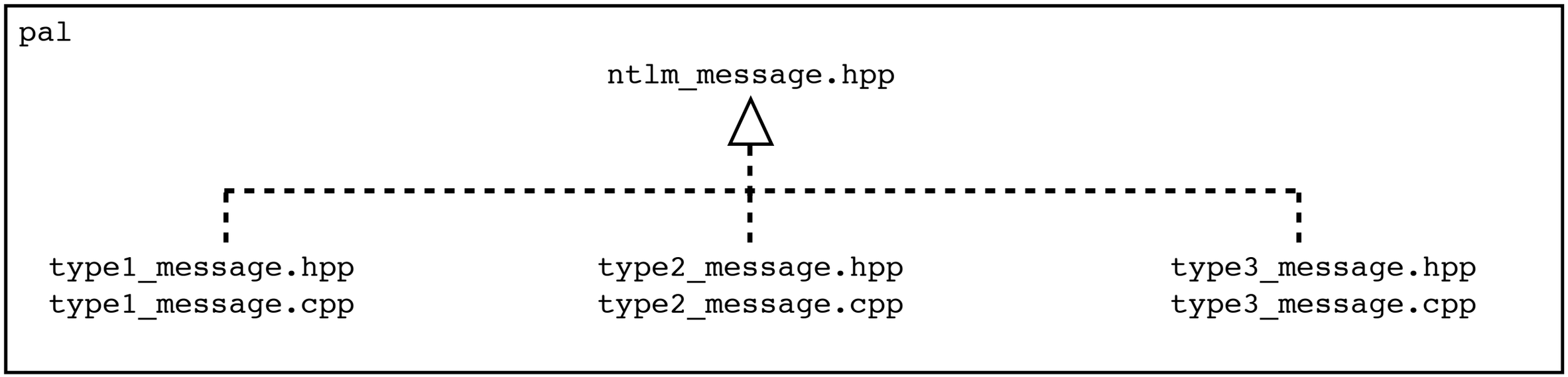
1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff
7. always focus on readability, you spend more time reading code than writing it
8. importing a namespace in implementation files is usually not a good idea
9. "never" import a namespace in a header file
10. initialize objects properly, use the initialization list
11. prefer std::size_t when working with memory indexing and offsets
12. for non-trivial objects, prefer pass by const over pass by copy
13. query methods should be specified as const
14. order the include files like this; own, project/platform, standard
15. avoid magic numbers, use explaination variables
16. avoid superfluous use of ()
17. prefer forward declarations when you can
18. do not use explicit on multi argument constructors
19. consider explainability, don't do things that needs elaborate explainations
20. avoid default arguments (they are often not used anyway)
21. do not throw pointers to exceptions (eg, not "throw new ...")
22. treat warnings like errors (-Werror) and compile with high warning levels (-Wall -Wextra)
23. consider using -Weffc++ and -pedantic as well
24. do not mess with borrowed things

Issues discussed here

1. base classes should have virtual destructors
2. header files should have header guards
3. make sure you include the proper header files, especially in header files
4. no need to include files included by the base class declaration
5. single argument constructors should be specified as explicit
6. focus on usage of class; public stuff first, then private stuff
7. always focus on readability, you spend more time reading code than writing it
8. importing a namespace in implementation files is usually not a good idea
9. "never" import a namespace in a header file
10. initialize objects properly, use the initialization list
11. prefer std::size_t when working with memory indexing and offsets
12. for non-trivial objects, prefer pass by const over pass by copy
13. query methods should be specified as const
14. order the include files like this; own, project/platform, standard
15. avoid magic numbers, use explaination variables
16. avoid superfluous use of ()
17. prefer forward declarations when you can
18. do not use explicit on multi argument constructors
19. consider explainability, don't do things that needs elaborate explainations
20. avoid default arguments (they are often not used anyway)
21. do not throw pointers to exceptions (eg, not "throw new ...")
22. treat warnings like errors (-Werror) and compile with high warning levels (-Wall -Wextra)
23. consider using -Weffc++ and -pedantic as well
24. do not mess with borrowed things
25. always consider the sideeffects of what you do

Handouts

Code under discussion



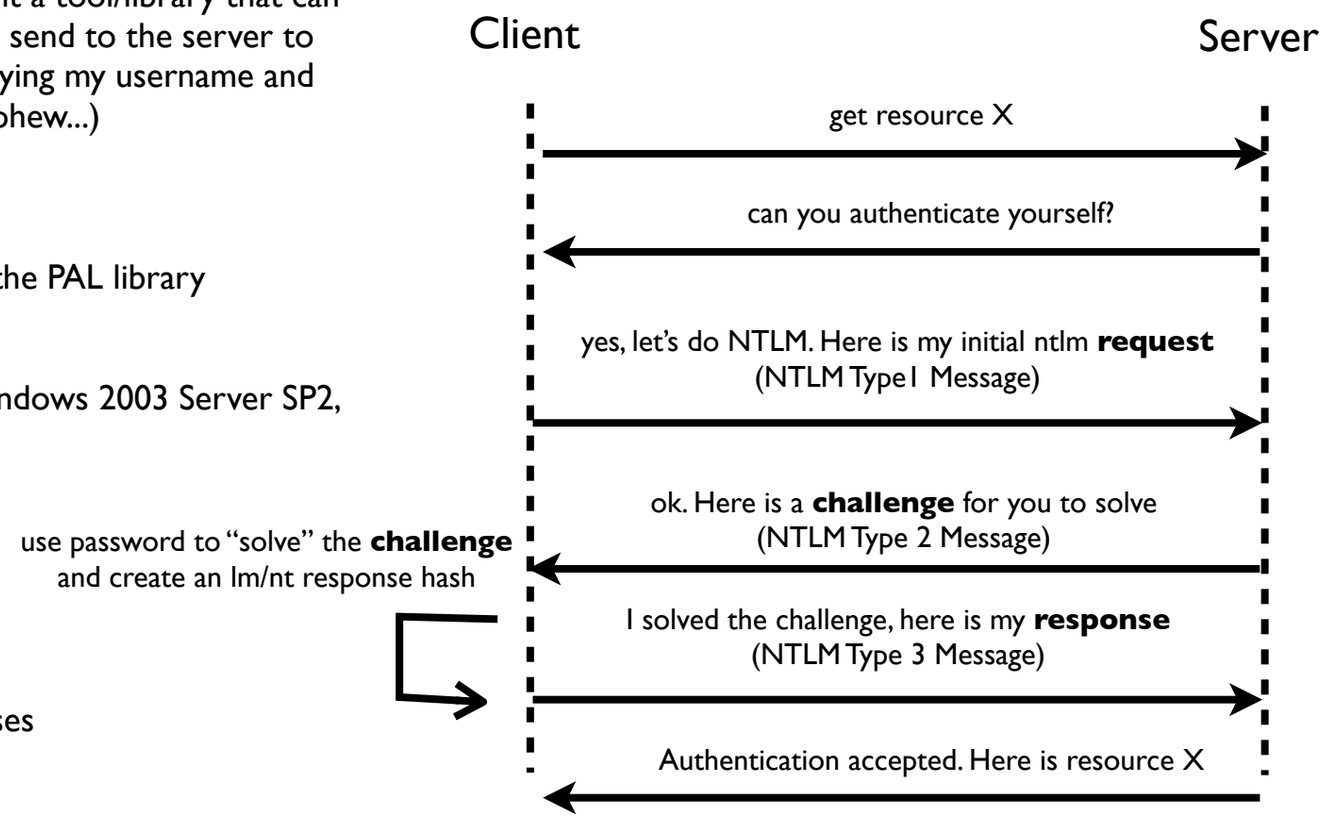
***Note**: the following code compiles just fine and does exactly what it is supposed to do... can you improve it?*

As a client, when prompted for ntlm authentication by the server, I want a tool/library that can help me to create the initial ntlm **request** (type 1 message) that I can send to the server to receive a **challenge** (type 2 message) that needs to be solved by applying my username and password to create an ntlm **response** that I can send to the server. (phew...)

Here are some (imaginary) additional requirements:
- target typically an embedded system
- it should be possible and convenient to create a <100kb client using the PAL library
- must use C++ (due to PHB decision)
- due to compiler support tr1 and boost can not be used
- initally the library will only be used to communicate HTTP with a Windows 2003 Server SP2, currently no need to support other servers

For more info:
PAL - a Primitive Authentication Library in C++ for educational purposes
http://github.com/olvemaudal/pal

The NTLM Protocol (simplified)



```
#include <vector>

namespace pal {

    class ntlm_message {
    public:
        virtual std::vector<uint8_t> as_bytes() const = 0;
    };

}
```

```
#ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
#define PAL_TYPE1_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <vector>

namespace pal {

    class type1_message : public ntlm_message {
        uint32_t ssp_flags_;
    public:
        type1_message(uint32_t ssp_flags);
        virtual std::vector<uint8_t> as_bytes() const;
    };

}

#endif
```

```
#include "type1_message.hpp"

#include "tools.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 1 Message
 *
 *    0  NTLMSSP Signature                 "NTLMSSP\0"
 *    8  NTLM Message Type                 {0x01,0x00,0x00,0x00}
 *   12  Flags                             uint32 as little endian
 * (16) Supplied Domain (optional)        (security buffer)
 * (24) Supplied Workstation (optional)   (security buffer)
 * (32) (start of datablock) if required
 */

using namespace std;

pal::type1_message::type1_message(uint32_t ssp_flags)
{
    ssp_flags_ = ssp_flags;
}

vector<uint8_t> pal::type1_message::as_bytes() const
{
    uint8_t message[16] = {
        'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
        0x01, 0x00, 0x00, 0x00, 0, 0, 0, 0
    };
    const int ssp_flags_offset = 12;
    pal::write_little_endian_from_uint32(&message[ssp_flags_offset], ssp_flags_);
    return vector<uint8_t>(message, message + sizeof message);
}
```

```
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

using namespace std;

namespace pal {

    class type2_message : public ntlm_message {
    public:
        explicit type2_message(vector<uint8_t> buffer);
        virtual vector<uint8_t> as_bytes() const;
        uint32_t ssp_flags();
        uint64_t challenge();
    private:
        const vector<uint8_t> buffer_;
    };

}

#endif
```

```
#include <cstddef>
#include <algorithm>
#include <stdexcept>

#include "tools.hpp"

#include "type2_message.hpp"

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 2 Message
 *
 *   0  NTLMSSP Signature                {'N','T','L','M','S','S','P','\0'}
 *   8  NTLM Message Type                {0x02,0x00,0x00,0x00}
 *  12  Target Name                      (security buffer)
 *  20  Flags                            uint32 as little endian
 *  24  Challenge                        8 bytes / uint64 as little endian
 * (32) Context (optional)               8 bytes (2xlong)
 * (40) Target Information               (security buffer)
 * (48) (start of datablock)
 *      targetname
 *      targetinfo
 *          server (type=0x0100, len, data)
 *          domain (type=0x0200, len, data)
 *          dnsserver (type=0x0300, len, data)
 *          dnsdomain (type=0x0400, len, data)
 *          type5 (type=0x0500, len, data)   // unknown role
 *          <terminator> (type=0,len=0)
 */
```

...

```
pal::type2_message::type2_message(const std::vector<uint8_t> & buffer)
    : buffer_(buffer)
{
    const std::size_t min_type2_buffer_size = 32;
    if (buffer.size() < min_type2_buffer_size)
        throw std::invalid_argument("not a type2 message, message too short");
    const uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x02,0x00,0x00,0x00
    };
    if (!std::equal(prefix, prefix + sizeof(prefix), buffer.begin()))
        throw std::invalid_argument("not a type2 message, invalid prefix");
}

uint32_t pal::type2_message::ssp_flags() const
{
    return pal::read_uint32_from_little_endian(&buffer_[20]);
}

uint64_t pal::type2_message::challenge() const
{
    return pal::read_uint64_from_little_endian(&buffer_[24]);
}

std::vector<uint8_t> pal::type2_message::as_bytes() const
{
    return (buffer_);
}
```

```
#ifndef PAL_TYPE3_MESSAGE_HPP_INCLUDED
#define PAL_TYPE3_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <string>
#include <iostream>

namespace pal {

    class type3_message : public ntlm_message {
    public:
        explicit type3_message(
            const std::vector<uint8_t> & lm_response,
            const std::vector<uint8_t> & nt_response,
            const std::string & user,
            uint32_t ssp_flags = 0x202);
        virtual std::vector<uint8_t> as_bytes() const;
        void debug_print(std::ostream & out) const;
    private:
        const std::vector<uint8_t> lm_response_;
        const std::vector<uint8_t> nt_response_;
        const std::string domain_;
        const std::string user_;
        const std::string workstation_;
        const std::vector<uint8_t> session_key_;
        const uint32_t ssp_flags_;
    };

}

#endif
```

...

```
pal::type3_message::type3_message(
    const std::vector<uint8_t> & lm_response,
    const std::vector<uint8_t> & nt_response,
    const std::string & user,
    uint32_t ssp_flags)
    :
    lm_response_(lm_response),
    nt_response_(nt_response),
    user_(user),
    ssp_flags_(ssp_flags),
    session_key_(session_key_size)
{
    if (lm_response_.size() != lm_response_size)
        throw new std::invalid_argument("invalid size of lm_response");
    if (nt_response_.size() != nt_response_size)
        throw new std::invalid_argument("invalid size of nt_response");
}
```

...

...

```
void pal::type3_message::debug_print(std::ostream & out) const
{
    out << "### type3_message:" << '\n'
        << pal::as_hex_dump(as_bytes())
        << "lm_response = " << pal::as_hex_string(lm_response_)
        << "\nnt_response = " << pal::as_hex_string(nt_response_)
        << "\ndomain = " << domain_
        << "\nuser = " << user_
        << "\nworkstation = " << workstation_
        << "\nsession_key = " << pal::as_hex_string(session_key_)
        << std::hex << std::setw(8) << std::setfill('0')
        << "\nssp_flags = " << ssp_flags_;
}
```

...