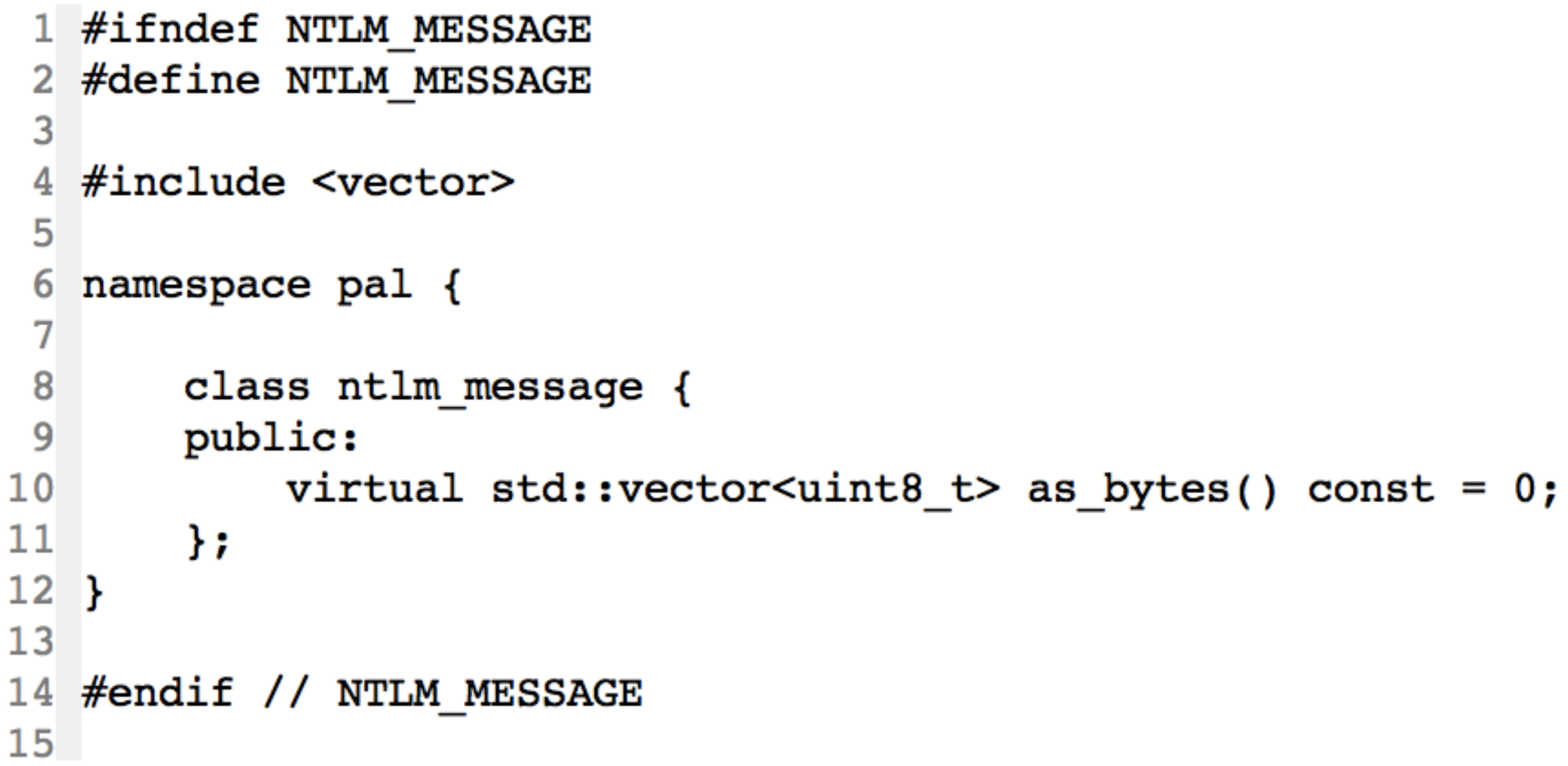
Solid C++ Source Code

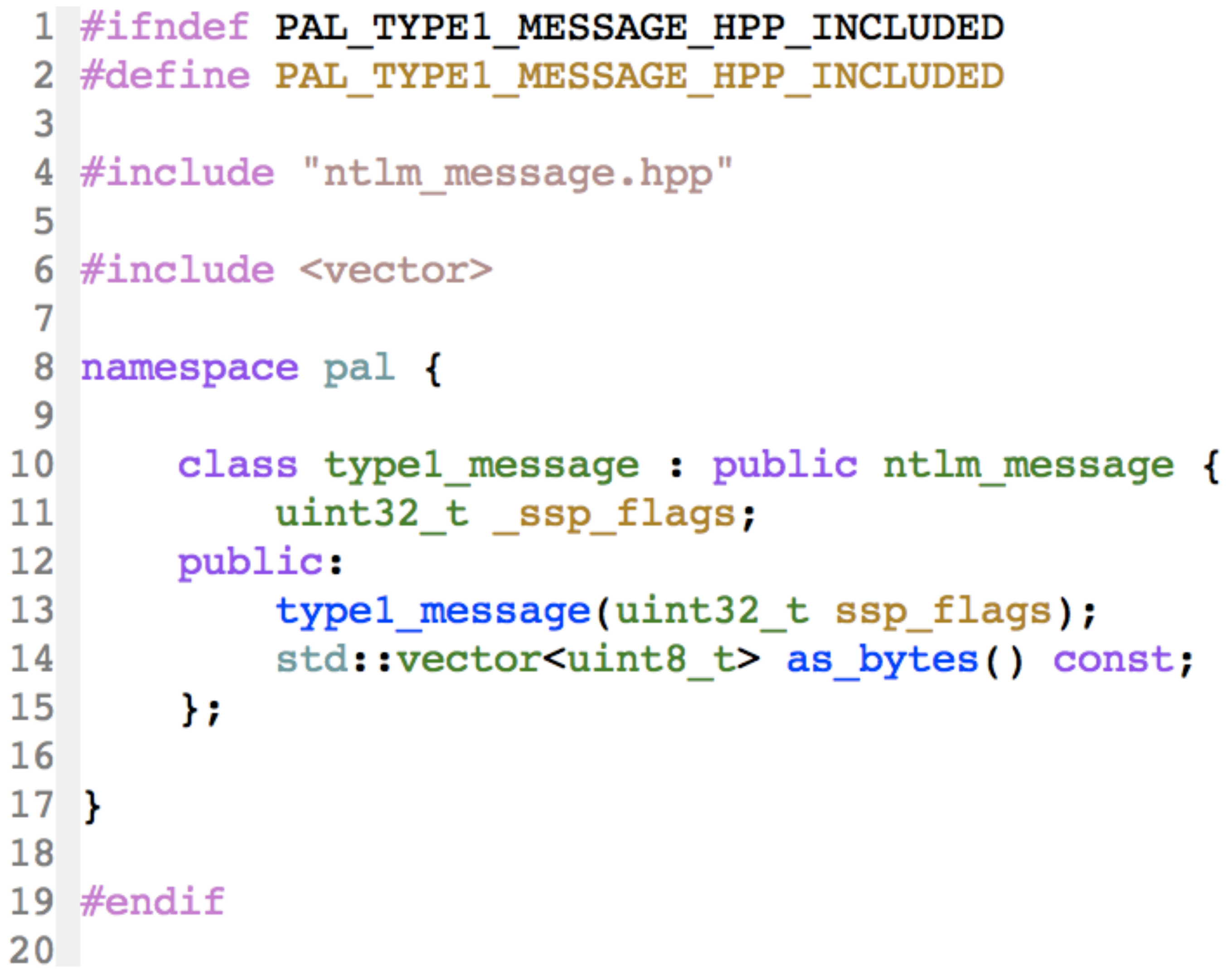
Please do **not** look ahead.
Wait for instructions.

ntlm_message.hpp



```cpp
#ifndef NTLM_MESSAGE
#define NTLM_MESSAGE

#include <vector>

namespace pal {

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

#endif // NTLM_MESSAGE
```

type1_message.hpp



```cpp
#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);
        std::vector<uint8_t> as_bytes() const;
    };

}

#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;

namespace pal {

    type1_message::type1_message(uint32_t ssp_flags)
    {
        _ssp_flags = ssp_flags;
    }

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

}
```

type2_message.hpp

```cpp
#ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
#define PAL_TYPE2_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#include <stdexcept>

using namespace std;

namespace pal {

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

}

#endif

```

type2_message.cpp

```cpp
#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','S','\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(std::vector<uint8_t> buffer)
     throw (std::invalid_argument)
     : buffer_(buffer)
{
     const 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 size_t ssp_flags_offset = 20;
     return pal::read_uint32_from_little_endian(&buffer_[ssp_flags_offset]);
}

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

```cpp
#ifndef PAL_TYPE3_MESSAGE_HPP_INCLUDED
#define PAL_TYPE3_MESSAGE_HPP_INCLUDED

#include "ntlm_message.hpp"

#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

```

type3_message.cpp (1/2)

```
#include "type3_message.hpp"

#include "tools.hpp"

#include <iomanip>
#include <algorithm>
#include <sstream>
#include <stdexcept>
#include <iomanip>
#include <iterator>

/*
 * See http://davenport.sourceforge.net/ntlm.html
 *
 * Type 3 Message
 *
 *   0  NTLMSSP Signature                    "NTLMSSP\0"
 *   8  NTLM Message Type                    {0x03,0x00,0x00,0x00}
 *  12  LM/LMv2 Response                     (security buffer)
 *  20  NTLM/NTLMv2 Response                 (security buffer)
 *  28  Domain Name                          (security buffer)
 *  36  User Name                            (security buffer)
 *  44  Workstation Name                     (security buffer)
 * (52) Session Key (optional)               (security buffer)
 * (60) Flags (optional)                     uint32 as little endian
 * (64) (start of datablock)
 *      domain name
 *      user name
 *      workstation name
 *      lm response data
 *      ntlm response data
 *
 * Security buffer: (works like a lookup into the data block)
 *   0  length      (uint16 as little endian)
 *   2  size        (uint16 as little endian)
 *   4  length      (uint32 as little endian)
 */

const std::size_t lm_response_sb_offset = 12;
const std::size_t nt_response_sb_offset = 20;
const std::size_t domain_sb_offset = 28;
const std::size_t user_sb_offset = 36;
const std::size_t workstation_sb_offset = 44;
const std::size_t session_key_sb_offset = 52;
const std::size_t ssp_flags_offset = 60;
const std::size_t data_block_offset = 64;

const std::size_t lm_response_size = 24;
const std::size_t nt_response_size = 24;
const std::size_t session_key_size = 16;

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");
}

```

…

type3_message.cpp (2/2)

…

```cpp
void append_data(
    std::vector<uint8_t> & to,
    std::size_t offset,
    const std::vector<uint8_t> & from)
{
    const std::size_t data_offset(to.end() - to.begin());
    std::copy(from.begin(), from.end(), std::back_inserter(to));
    pal::write_little_endian_from_uint16(&to[offset+0], from.size());
    pal::write_little_endian_from_uint16(&to[offset+2], from.size());
    pal::write_little_endian_from_uint32(&to[offset+4], data_offset);
}

std::vector<uint8_t> pal::type3_message::as_bytes() const
{
    uint8_t prefix[12] = {
        'N','T','L','M','S','S','P','\0',
        0x03,0x00,0x00,0x00
    };
    std::vector<uint8_t> buffer(prefix, prefix + sizeof(prefix));
    buffer.resize(data_block_offset);
    pal::write_little_endian_from_uint32(&buffer[ssp_flags_offset], ssp_flags_);

    append_data(buffer, lm_response_sb_offset, lm_response_);
    append_data(buffer, nt_response_sb_offset, nt_response_);
    append_data(buffer, domain_sb_offset, pal::as_bytes(domain_));
    append_data(buffer, user_sb_offset, pal::as_bytes(user_));
    append_data(buffer, workstation_sb_offset, pal::as_bytes(workstation_));
    append_data(buffer, session_key_sb_offset, session_key_);

    return buffer;
}
```

```cpp

void pal::type3_message::debug_print(std::ostream & out) const
{
    out << "### type3_message:" << std::endl
        << pal::as_hex_dump(as_bytes())
        << "lmReponse = " << pal::as_hex_string(lm_response_)
        << "\nntReponse = " << pal::as_hex_string(nt_response_)
        << "\ndomain = " << domain_
        << "\nuser = " << user_
        << "\nworkstation = " << workstation_
        << "\nsessionKey = " << pal::as_hex_string(session_key_)
        << std::hex << std::setw(8) << std::setfill('0')
        << "\nsspFlags = " << ssp_flags_ << std::endl;
}

```