

ntlm_message.hpp

```
1 #ifndef NTLM_MESSAGE
2 #define NTLM_MESSAGE
3
4 #include <vector>
5     include <stdint.h>
6 namespace pal {
7
8     class ntlm_message {
9     public:
10         virtual std::vector<uint8_t> as_bytes() const = 0;
11     };
12 }
13
14 #endif // NTLM_MESSAGE
15
```

bad name for an include guard

include <stdint.h>

virtual destructor

how useful is this?

seek tool support, eg use syntax highlighting

type1_message.hpp

```
1 #ifndef PAL_TYPE1_MESSAGE_HPP_INCLUDED
2 #define PAL_TYPE1_MESSAGE_HPP_INCLUDED
3
4 #include "ntlm_message.hpp"
5
6 #include <vector>
7
8 namespace pal {
9
10     class type1_message : public ntlm_message {
11         uint32_t _ssp_flags;
12     public:
13         type1_message(uint32_t ssp_flags);
14         std::vector<uint8_t> as_bytes() const;
15     };
16
17 }
18
19 #endif
20
```

not needed

do not _prefix

const?

explicit

repeat the virtual specifier

order of public/private

type1_message.cpp

```
1 #include "type1_message.hpp"
2
3 #include "tools.hpp"
4
5 /*
6  * See http://davenport.sourceforge.net/ntlm.html
7  *
8  * Type 1 Message
9  *
10 * 0 NTLMSSP Signature "NTLMSSP\0"
11 * 8 NTLM Message Type {0x01,0x00,0x00,0x00}
12 * 12 Flags uint32 as little endian
13 * (16) Supplied Domain (optional) (security buffer)
14 * (24) Supplied Workstation (optional) (security buffer)
15 * (32) (start of datablock) if required
16 */
17
18 using namespace std;
19
20 namespace pal {
21
22     type1_message::type1_message(uint32_t ssp_flags)
23     {
24         _ssp_flags = ssp_flags;
25     }
26
27     vector<uint8_t> type1_message::as_bytes() const
28     {
29         const int message_size = 16;
30         uint8_t message[message_size] = {
31             'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
32             0x01, 0x00, 0x00, 0x00
33         };
34         write_little_endian_from_uint32(&message[12], _ssp_flags);
35         return vector<uint8_t>(message, message + sizeof message);
36     }
37 }
38 }
```

does not improve readability

do not open a namespace here

use initializer list

should be a const member

is this useful?

use size_t

avoid magic numbers

type2_message.hpp

```
1 #ifndef PAL_TYPE2_MESSAGE_HPP_INCLUDED
2 #define PAL_TYPE2_MESSAGE_HPP_INCLUDED
3
4 #include "ntlm_message.hpp"
5
6 #include <stdexcept>
7
8 using namespace std; ← never do this
9
10 namespace pal {
11
12     class type2_message : public ntlm_message {
13     public:
14         explicit type2_message(std::vector<uint8_t> buffer)
15             throw (std::invalid_argument); ← "checked" exceptions?
16         virtual std::vector<uint8_t> as_bytes() const;
17         uint32_t ssp_flags(); ← const
18         uint64_t challenge(); ← const
19     private:
20         const std::vector<uint8_t> buffer_;
21     };
22
23 }
24
25 #endif
26
```

Annotations:

- never do this (points to `using namespace std;`)
- use const & (points to `as_bytes() const`)
- "checked" exceptions? (points to `throw (std::invalid_argument);`)
- const (points to `ssp_flags()` and `challenge()`)

type2_message.cpp (1/2)

include order is not optimal

```
1 #include <stdexcept>
2 #include "tools.hpp"
3 #include "type2_message.hpp"
4
5 /*
6  * See http://davenport.sourceforge.net/ntlm.html
7  *
8  * Type 2 Message
9  *
10 * 0 NTLMSSP Signature {'N','T','L','M','S','S','S','\0'}
11 * 8 NTLM Message Type {0x02,0x00,0x00,0x00}
12 * 12 Target Name (security buffer)
13 * 20 Flags uint32 as little endian
14 * 24 Challenge 8 bytes / uint64 as little endian
15 * (32) Context (optional) 8 bytes (2xlong)
16 * (40) Target Information (security buffer)
17 * (48) (start of datablock)
18 * targetname
19 * targetinfo
20 * server (type=0x0100, len, data)
21 * domain (type=0x0200, len, data)
22 * dnsserver (type=0x0300, len, data)
23 * dnsdomain (type=0x0400, len, data)
24 * type5 (type=0x0500, len, data) // unknown role
25 * <terminator> (type=0,len=0)
26 */
27
```

...

type2_message.cpp (2/2)

```
...
28 pal::type2_message::type2_message(std::vector<uint8_t> buffer)
29     throw (std::invalid_argument)
30     : buffer_(buffer)
31 {
32     const size_t min_type2_buffer_size = 32;
33     if (buffer.size() < min_type2_buffer_size)
34         throw std::invalid_argument("not a type2 message, message too short");
35     const uint8_t prefix[12] = { 'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
36                                 0x02, 0x00, 0x00, 0x00 };
37     if (!std::equal(prefix, prefix + sizeof prefix, buffer.begin()))
38         throw std::invalid_argument("not a type2 message, invalid prefix");
39 }
40
41 uint32_t pal::type2_message::ssp_flags()
42 {
43     const size_t ssp_flags_offset = 20;
44     return pal::read_uint32_from_little_endian(&buffer_[ssp_flags_offset]);
45 }
46
47 uint64_t pal::type2_message::challenge()
48 {
49     const size_t challenge_offset = 24;
50     return pal::read_uint64_from_little_endian(&buffer_[challenge_offset]);
51 }
52
53 std::vector<uint8_t> pal::type2_message::as_bytes() const
54 {
55     return (buffer_);
56 }
```

const ref

"checked" exceptions?

use robust layout

std::size_t

no need for parentheses here

type3_message.hpp

```
1 #ifndef PAL_TYPE3_MESSAGE_HPP_INCLUDED
2 #define PAL_TYPE3_MESSAGE_HPP_INCLUDED
3
4 #include "ntlm_message.hpp"
5
6 #include <iostream>
7
8 namespace pal {
9
10     class type3_message : public ntlm_message {
11     public:
12         explicit type3_message(
13             const std::vector<uint8_t>& lm_response,
14             const std::vector<uint8_t> &nt_response,
15             const std::string & user,
16             uint32_t ssp_flags = 0x202);
17         virtual std::vector<uint8_t> as_bytes() const;
18         void debug_print(std::ostream & out) const;
19     private:
20         const std::vector<uint8_t> lm_response_;
21         const std::vector<uint8_t> nt_response_;
22         const std::string domain_;
23         const std::string user_;
24         const std::string workstation_;
25         const std::vector<uint8_t> session_key_;
26         const uint32_t ssp_flags_;
27     };
28
29 }
30
31 #endif
32
```

use <iosfwd> instead

star wars

not needed

is it used?

this is fragile code layout

type3_message.cpp (1/4)

```
1 #include "type3_message.hpp"
2
3 #include "tools.hpp"
4
5 #include <iomanip>
6 #include <algorithm>
7 #include <sstream>
8 #include <stdexcept>
9 #include <iomanip>
10 #include <iterator>
11
12 /*
13  * See http://davenport.sourceforge.net/ntlm.html
14  *
15  * Type 3 Message
16  *
17  * 0 NTLMSSP Signature "NTLMSSP\0"
18  * 8 NTLM Message Type {0x03,0x00,0x00,0x00}
19  * 12 LM/LMv2 Response (security buffer)
20  * 20 NTLM/NTLMv2 Response (security buffer)
21  * 28 Domain Name (security buffer)
22  * 36 User Name (security buffer)
23  * 44 Workstation Name (security buffer)
24  * (52) Session Key (optional) (security buffer)
25  * (60) Flags (optional) uint32 as little endian
26  * (64) (start of datablock)
27  * domain name
28  * user name
29  * workstation name
30  * lm response data
31  * ntlm response data
32  *
33  * Security buffer: (works like a lookup into the data block)
34  * 0 length (uint16 as little endian)
35  * 2 size (uint16 as little endian)
36  * 4 length (uint32 as little endian)
37  */
38
```

consider alphabetical ordering


```

...
39 const std::size_t lm_response_sb_offset = 12;
40 const std::size_t nt_response_sb_offset = 20;
41 const std::size_t domain_sb_offset = 28;
42 const std::size_t user_sb_offset = 36;
43 const std::size_t workstation_sb_offset = 44;
44 const std::size_t session_key_sb_offset = 52;
45 const std::size_t ssp_flags_offset = 60;
46 const std::size_t data_block_offset = 64;
47
48 const std::size_t lm_response_size = 24;
49 const std::size_t nt_response_size = 24;
50 const std::size_t session_key_size = 16;
51
52 pal::type3_message::type3_message(
53     const std::vector<uint8_t> & lm_response,
54     const std::vector<uint8_t> & nt_response,
55     const std::string & user,
56     uint32_t ssp_flags)
57     :
58     lm_response_(lm_response),
59     nt_response_(nt_response),
60     user_(user),
61     ssp_flags_(ssp_flags),
62     session_key_(session_key_size)
63 {
64     if(lm_response_.size() != lm_response_size)
65         throw new std::invalid_argument("invalid size of lm_response");
66     if(nt_response_.size() != nt_response_size)
67         throw new std::invalid_argument("invalid size of nt_response");
68 }
69
...

```

anonymous namespace

what about initializing domain_ and workstation_?

wrong order of initialization

space after keywords

new?

type3_message.cpp (3/4)

```
...
70 void append_data(
71     std::vector<uint8_t> & to,
72     std::size_t offset,
73     const std::vector<uint8_t> & from)
74 {
75     const std::size_t data_offset(to.end() - to.begin());
76     std::copy(from.begin(), from.end(), std::back_inserter(to));
77     pal::write_little_endian_from_uint16(&to[offset+0], from.size());
78     pal::write_little_endian_from_uint16(&to[offset+2], from.size());
79     pal::write_little_endian_from_uint32(&to[offset+4], data_offset);
80 }
81
82 std::vector<uint8_t> pal::type3_message::as_bytes() const
83 {
84     uint8_t prefix[12] = {
85         'N', 'T', 'L', 'M', 'S', 'S', 'P', '\0',
86         0x03, 0x00, 0x00, 0x00
87     };
88     std::vector<uint8_t> buffer(prefix, prefix + sizeof(prefix));
89     buffer.resize(data_block_offset);
90     pal::write_little_endian_from_uint32(&buffer[ssp_flags_offset], ssp_flags_);
91
92     append_data(buffer, lm_response_sb_offset, lm_response_);
93     append_data(buffer, nt_response_sb_offset, nt_response_);
94     append_data(buffer, domain_sb_offset, pal::as_bytes(domain_));
95     append_data(buffer, user_sb_offset, pal::as_bytes(user_));
96     append_data(buffer, workstation_sb_offset, pal::as_bytes(workstation_));
97     append_data(buffer, session_key_sb_offset, session_key_);
98
99     return buffer;
100 }
...
```

anonymous namespace

use assignment syntax instead

remove parenthesis

type3_message.cpp (4/4)

...

```
101
102 void pal::type3_message::debug_print(std::ostream & out) const
103 {
104     out << "### type3_message:" << std::endl
105         << pal::as_hex_dump(as_bytes())
106         << "\nReponse = " << pal::as_hex_string(lm_response_)
107         << "\nntReponse = " << pal::as_hex_string(nt_response_)
108         << "\ndomain = " << domain_
109         << "\nuser = " << user_
110         << "\nworkstation = " << workstation_
111         << "\nsessionKey = " << pal::as_hex_string(session_key_)
112         << std::hex << std::setw(8) << std::setfill('0')
113         << "\nsspFlags = " << ssp_flags_ << std::endl;
114 }
115
```

perhaps return a string instead?

do you want to flush the stream?

don't mess with borrowed things