OpenWBEM audit report

June 24, 2005

Package:	OpenWBEM
Version:	3.1.0
Product:	SLES9
Date:	2005-06-22
Reason:	security relevant package
Scope:	auth, SSL, HTTP, XML parsing, UTF8 decoding code
Reviewer:	Sebastian Krahmer
Report-Version:	1
Bug-ID:	85842
References:	-
	-

1 Introduction

Since OpenWBEM is shipped with SLES9 and since it runs at a critical position (admin interface) a security review is necessary. Taking the future use of this package into account a review is even more needed.

2 Audit environment

- SUSE Linux 9.3 x86_64
- openwbem-3.1.0-4.1
- novell-life-1.0.0-3.1

3 Methodology

The OpenWBEM RPM package has been installed and tested for basic functionallity such as start/restart or syslog functionallity. Important parts of the source code have been reviewed for buffer overflows, integer wrap arounds, DoS conditions, temp races etc. The results of this review are presented in the following section.

4 Results

The reviewed classes and modules are listed below. If there are concerns with regard to security, the code piece is shown and the problem is described. Additionally, one sections describes which general problems might exist due to design or conception of the whole product.

4.1 Reviewed Classes/Modules

- ByteBuf
- BaseStreamBuffer
- Char16
- DigestAuthentication
- IOIFCStreamBuffer
- GetPass
- HTTPChunkedIStreamBuffer

```
int
HTTPChunkedIStreamBuffer::buffer_from_device(char* c, int n)
{
        if (m_isEOF)
        {
                return -1;
        }
        int tmpInLen = 0;
        int offset = 0;
        int lastRead = 0;
        while (offset < n && m_istr.good())
        {
                if (m_inLen == -1)
                {
                        m_istr >> std::hex >> m_inLen >> std::dec; //AUD: m_inLen can be negative
        ...
                // min of (n - offset) and (m_inLen - m_inPos)
                tmpInLen = ((n - offset) < (m_inLen - m_inPos)) ? (n - offset)
                        : (m_inLen - m_inPos);
                m_istr.read(c + offset, tmpInLen);
                lastRead = m_istr.gcount();
                offset += lastRead;
                m_inPos += lastRead;

        ...
```

m_inLen is signed and can therefore be negative. The min-check can therefore lead to unexpected results. Depending on the type of *streamsize* (second parameter to *istream::read()*) this could lead to a overflow or a DoS condition since the loop will never be left. Since this is the HTTP chunked decoding, chances are good that this can happen prior authentication. Better check for *m_inLen* <0. Having *int* as type of size parameter *n* is also unclean.

- HTTPDeflateIStreamBuffer
- HTTPLenLimitIStreamBuffer
  The *m_length* member variable is of type *Int64* and is filled with the value of the Content-Length HTTP header tag. The *HTTPLengthLimitStreamBuffer::buffer_from_device()* method however expects *int* as size parameter and does the checks and assignments across *int* and *Int64* boundaries. This is not 64 bit clean.

```
int
HTTPLengthLimitStreamBuffer::buffer_from_device(char* c, int n)
{
        if (m_isEnd)
        {
                return -1;
        }
        // min of n and (length - pos) AUD: 64 Bit unclean

        int tmpInLen = (n < (m_length - m_pos)) ? n : (m_length - m_pos);
        m_istr.read(c, tmpInLen);
        int lastRead = m_istr.gcount();
        m_pos += lastRead;
        if (m_pos == m_length)
        {
                m_isEnd = true;
        }
        return lastRead;
}
```

  A *m_length* value of 0xff1122330000ffff for example will pass the check since the value is negative, thus its not bigger than *n* and assigned to the 32Bit type *tmpInLen* which yields a value of 0x0000ffff then. This will easily overflow the *c* buffer.

- HTTPServer
- HTTPSvrConnection
- LocalAuthentication

  This class is using the weak *Random* class to generate the challenges.
  Some parts of the code are unclear, in particular the declaration of the *struct passwd* variables which never get used.

- Logger
- LogAppender
- OW_HTTPServer.cpp
- OW_HTTPUtils.cpp
- OW_XMLParserDOM.cpp

- OW_XMLQualifier.cpp
- OW_UTF8Utils.cpp

  I am not sure whether the UTF8 decoding works unquie i.e. whether it would accept longer sequences than the shortest possible one. This could clash with filters which might be applied in future versions.

- RandomNumber
  The random number generator is seeded with a value taken from */dev/random*. The real randomness is then taken from calls to *random()*. Since */dev/random* has bigger entropy it should be used directly to obtain the random values instead of the weak libc based *random()* function.

- SSLCtxMgr
  The SSL program code is weird and quite unclear. OpenWBEM integrates the abillity to authenticate clients via X509 certificates. However major parts of the SSL code are commented out. Important function calls to load the trusted CA or to check the certificates themself are missing. A commented out call to *SSL_CTX_load_verify_locations()* is one such example.

- SSLSocketImpl
- String

```
size_t
String::lastIndexOf(const char* arg, size_t fromIndex) const
{
        if (fromIndex == npos || fromIndex >= length())
        {
                if (static_cast<int>(fromIndex = length()-1) < 0)
                {
                        return npos;
                }
        }

        int arglen = (arg) ? ::strlen(arg) : 0; // AUD: return value of strlen() is size_t
...
```

  The conversion from *size_t* into *int* may cause problems.

- StringBuffer
- SyslogAppender
- TmpFile
- TmpFileImpl
- URL
- XMLNode
- XMLNodeImpl
- XMLOperationGeneric

- XMLParserCore
- XMLPullParser
- XMLUnescape()
  Checks like

```
long lval = strtol( thisTokStart + 3, NULL, 16 );
if (lval > CHAR_MAX) //AUD
```

  miss the fact that the value could be negative and therefore passes the XML code validation check. This happens two times in this function.

During the audit the main view was on the classes and modules which might be used before authentication. The big amount of LOC and the complexity did not allow a complete review of the code. Therefore only parts of the code have been revied which were classifi ed as critical.

4.2 General problems

The complexity of OpenWBEM is a fundamental problem since it is security critical software. Due to the large code base it is not possible to say it is 'secure'. Chances are good that security problems in OpenWBEM may arise in future which have not been uncovered yet. Small changes in one part may cause big security problems in other parts of the code and in the life cycle of such a product, changes are very likely. General problems in the design of OpenWBEM which are not actually security holes but may open some in future:

- The complete string, buffer and array handling has been re-implemented instead of using existing STL or libc implementations. Even functiosn such as *strtoll()* or *strchr()* have been re-implemented. For security reasons it is not acceptable to re-invent the wheel each time. Better build on trusted, reliable, reviewd and tested libraries.
- Even though OpenWBEM offers various authentication mechanisms, its not fi ne grained. If PAM is used for authentication every user with an account on the machine can access the OpenWBEM and admninistrate the machine. The *owcimond* runs as root user the whole time. [1]

5 Peculiarities

If the SSL code is 'fi xed' or at least cleaned up, there is an outstanding review of the SSL code of the client browser (which is not part of OpenWBEM) since much of the SSL security has to be implemented in the client (certifi cate checking).
With XML-RPC systems, one has to ensure that remote clients can only call functions which are meant to be called by XML-RPC. It should not be possible to call internal functions or functions like *system()*.

6 Exploits

No exploits have been written.

7 Status

Code review of the described parts complete. Awaiting comments from the developers and maintainers.

References

[1] Bugzilla:

`https://bugzilla.novell.com/show_bug.cgi?id=65423`

[2] Bugzilla:

`https://bugzilla.novell.com/show_bug.cgi?id=85842`