Scanning HacmeBank Web-Services

F1 'How-To Series'

Ounce Labs, Inc.
100 Fifth Avenue
Waltham, MA
(781) 290-5333

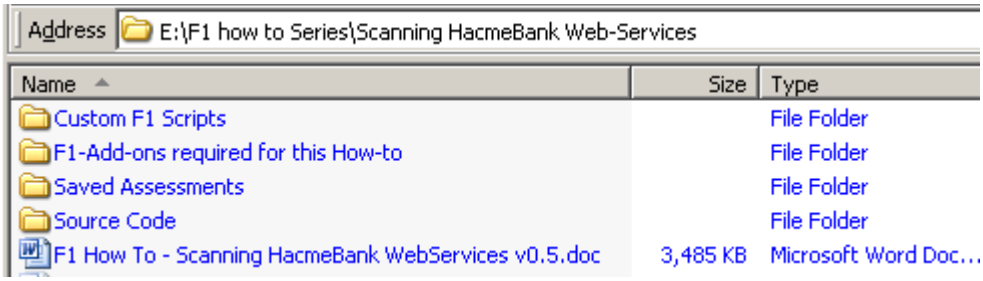
OVERVIEW AND OBJECTIVES

INTRODUCTION

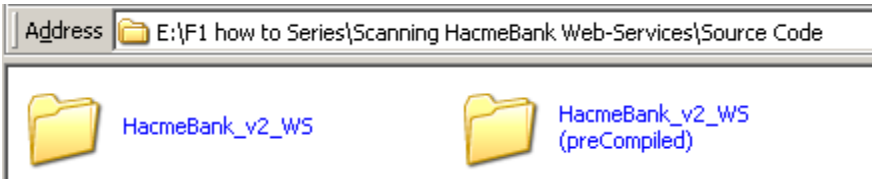
This document contains detailed technical information on how to use Ounce's technology (CORE + F1) to analyse the HacmeBank's Web-services.

SUPPORT MATERIALS

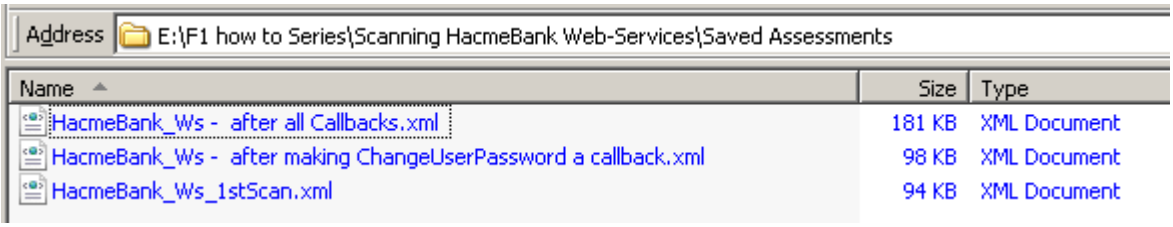
The *Scanning HacmeBank Web-Services.zip* file contains all materials required for this how to



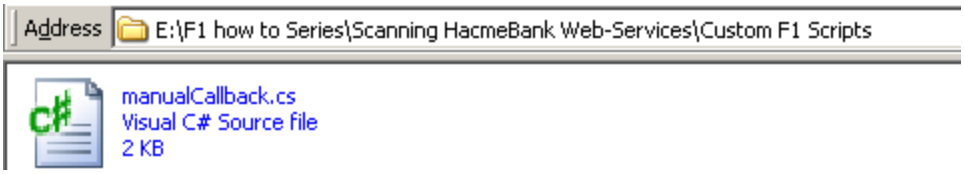
The Source Code of the HacmeBank WebServices and the respective precompiled directory:



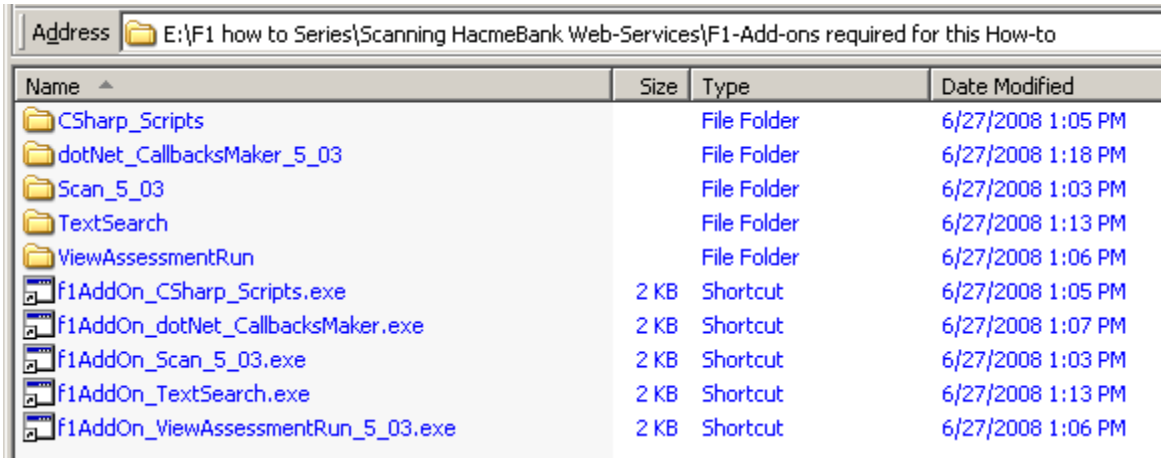
The assessment files that will be created



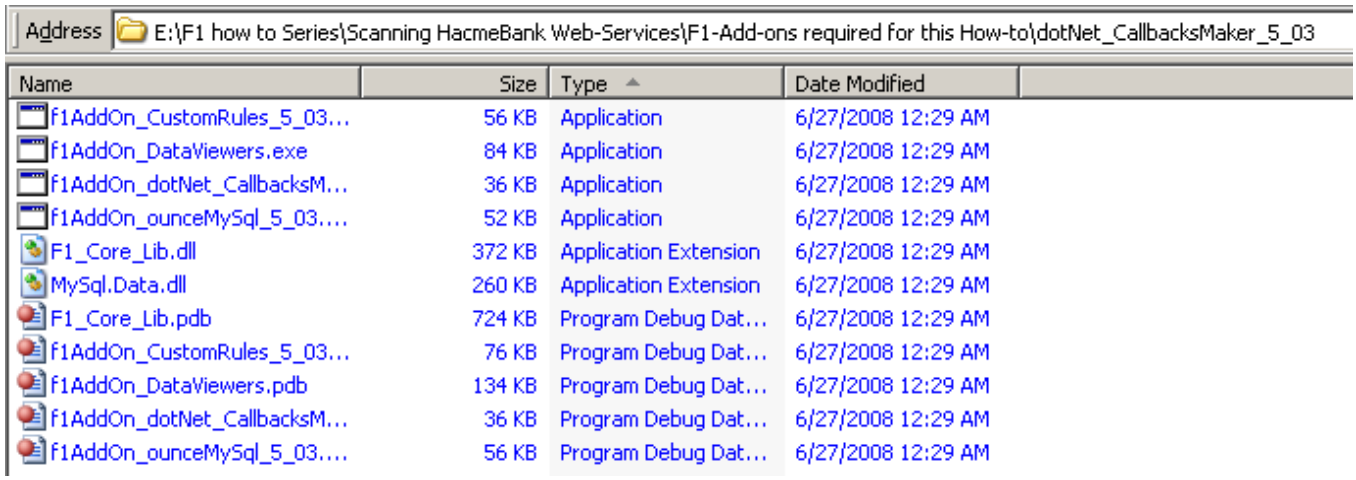
The script used to manually create Callbacks



The F1 Add-ons required:

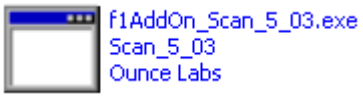


Each F1 Add-on is packaged as a unit (i.e. with all dependencies included) and is designed to be executed independently:

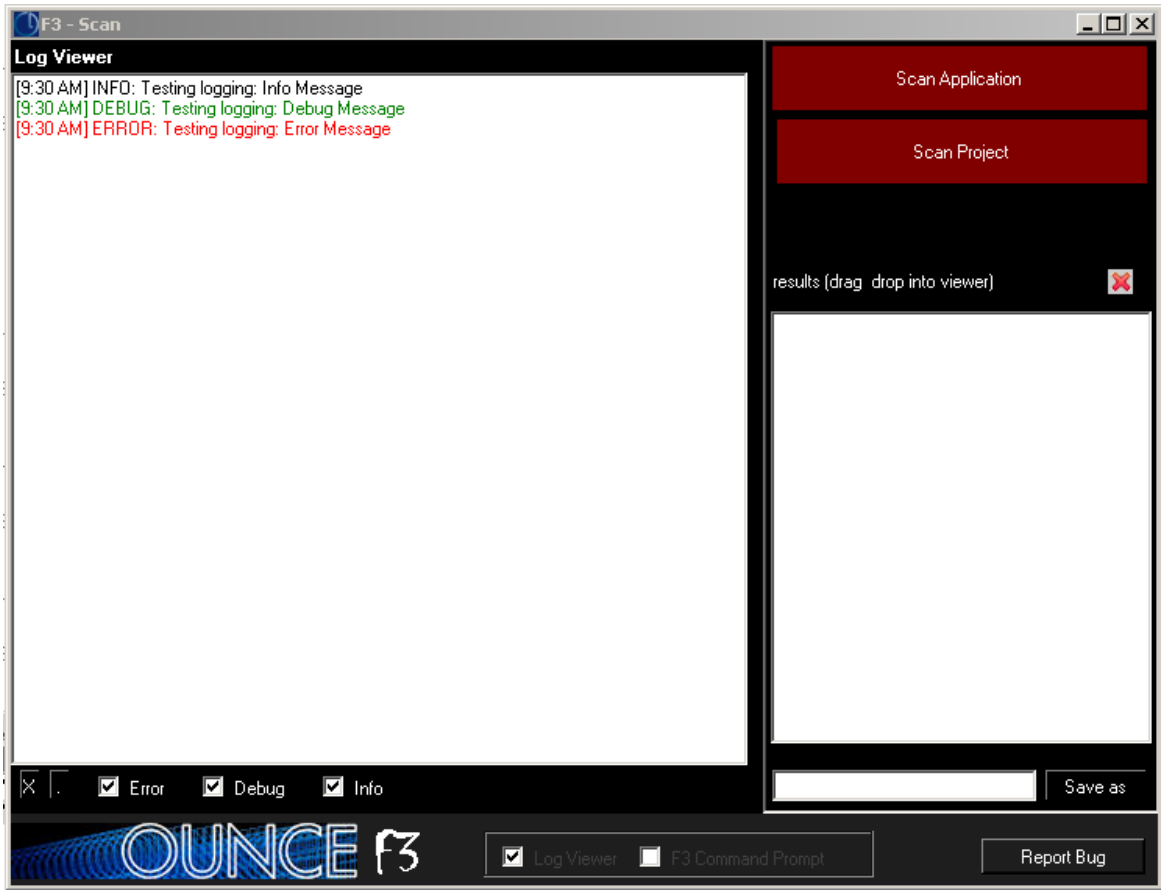


SCANNING WEBSITE

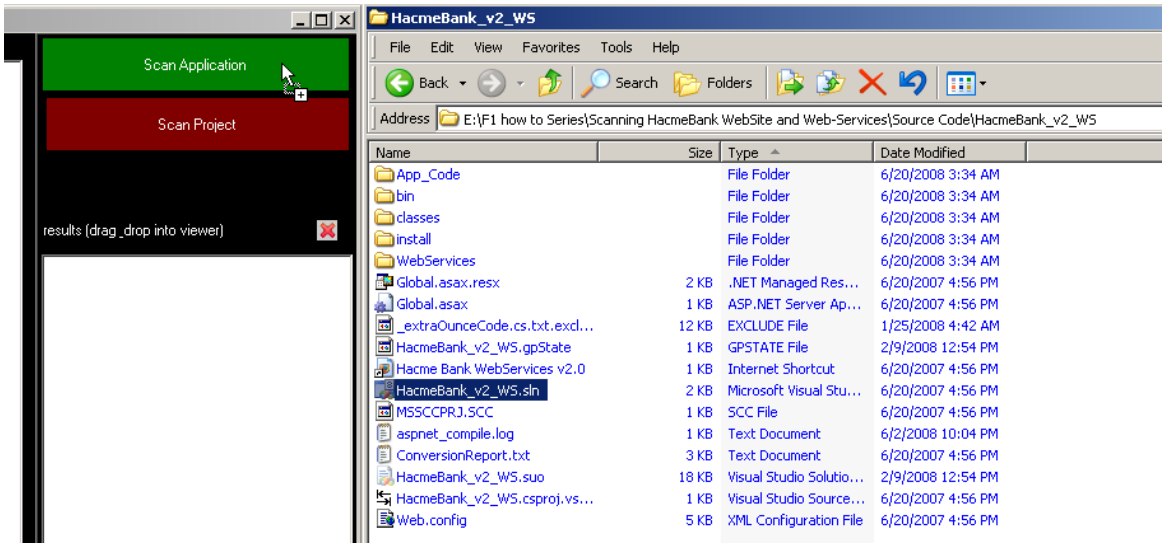
Open f1AddOn_Scan_5.03.exe



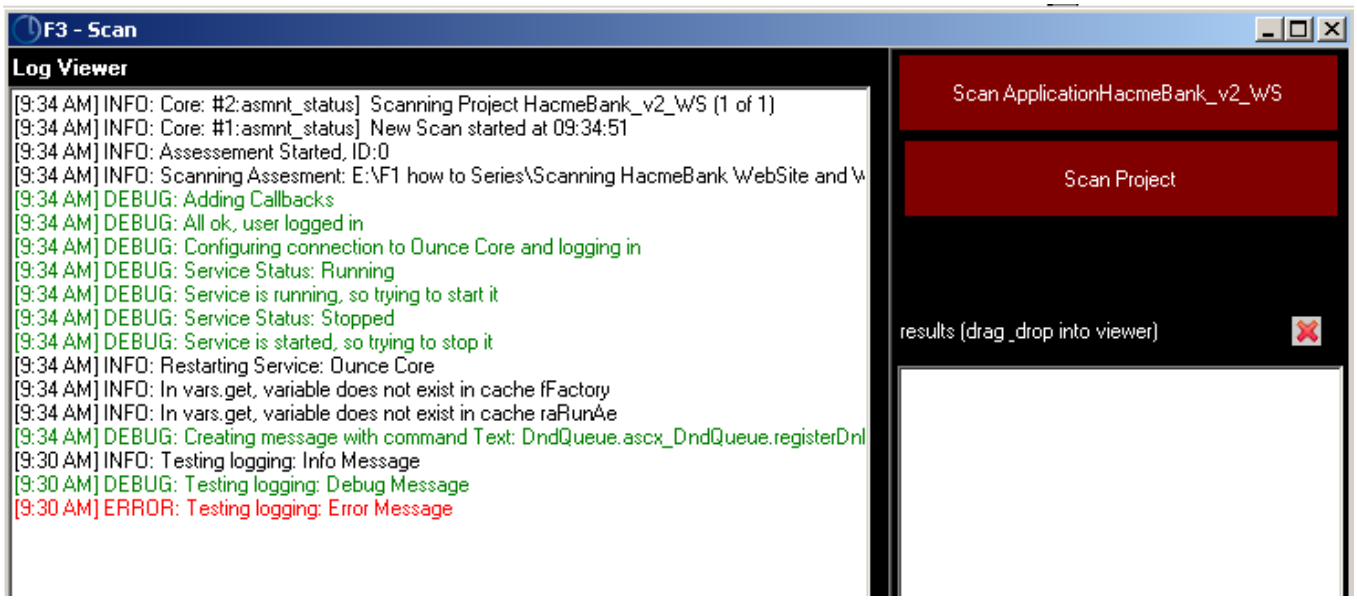
And that will open the scan interface in ‘F3 Mode’



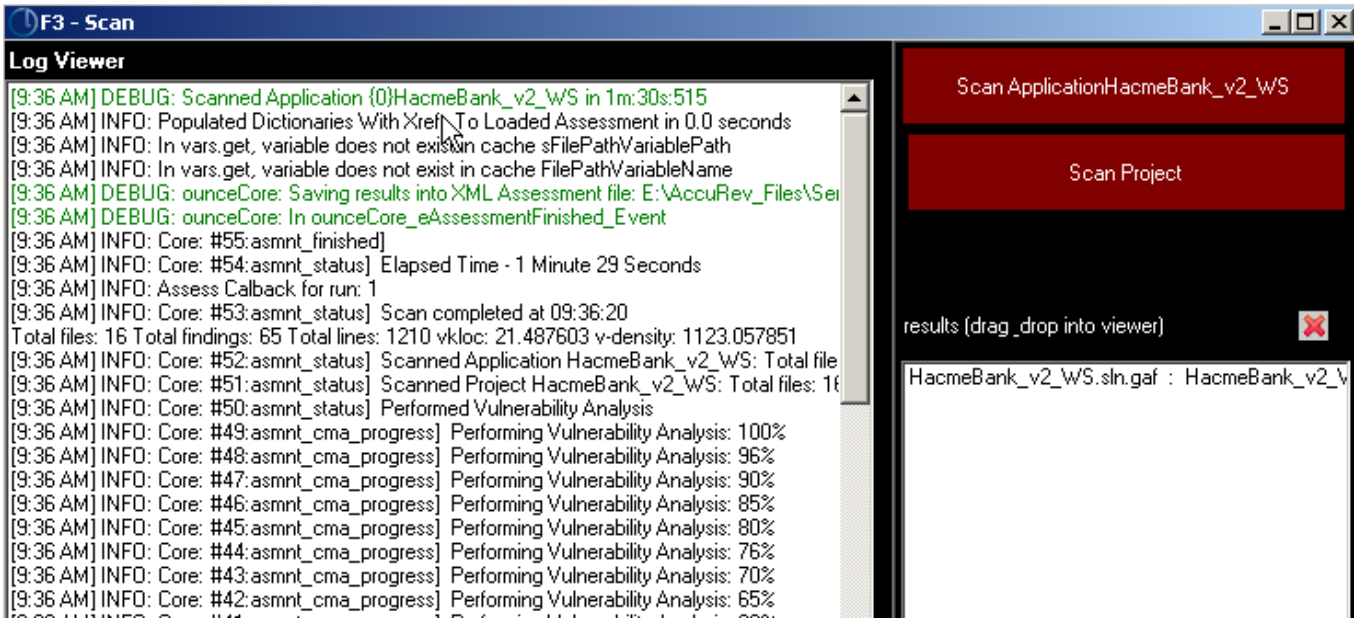
Drag & Drop the Solution file into the ‘Scan Application’ Box:



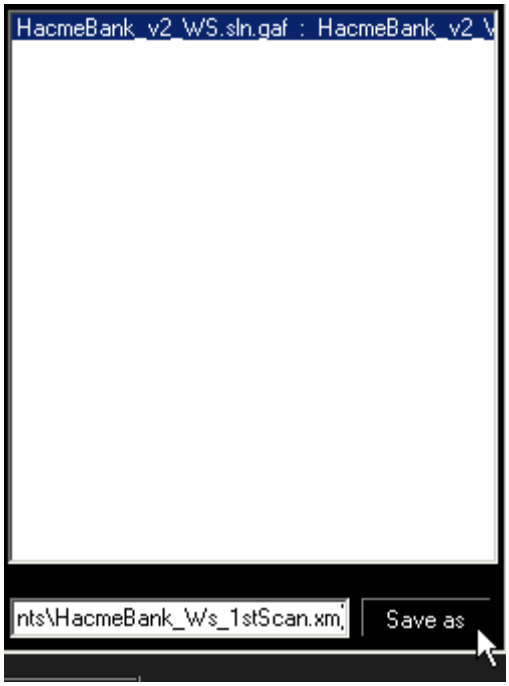
This will trigger the scan

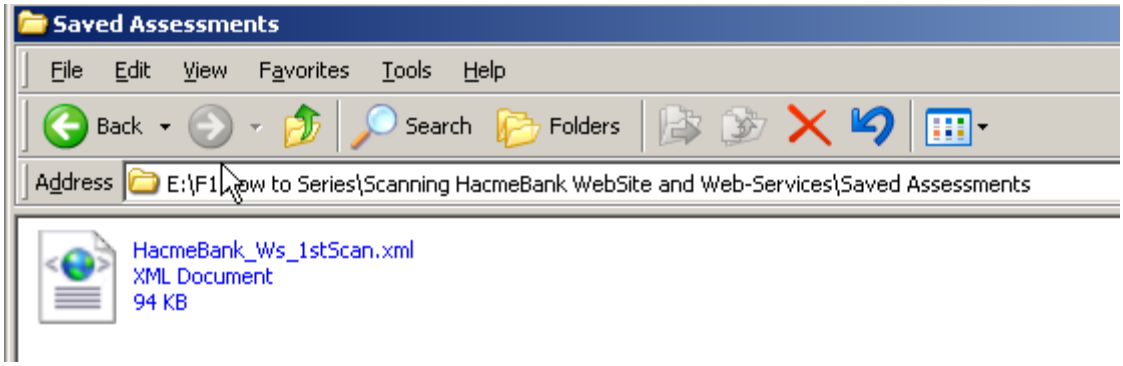


When the scan is completed:



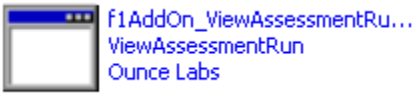
To save the assessment, select the entry in the results list box, enter the path and filename and click on 'Save As'

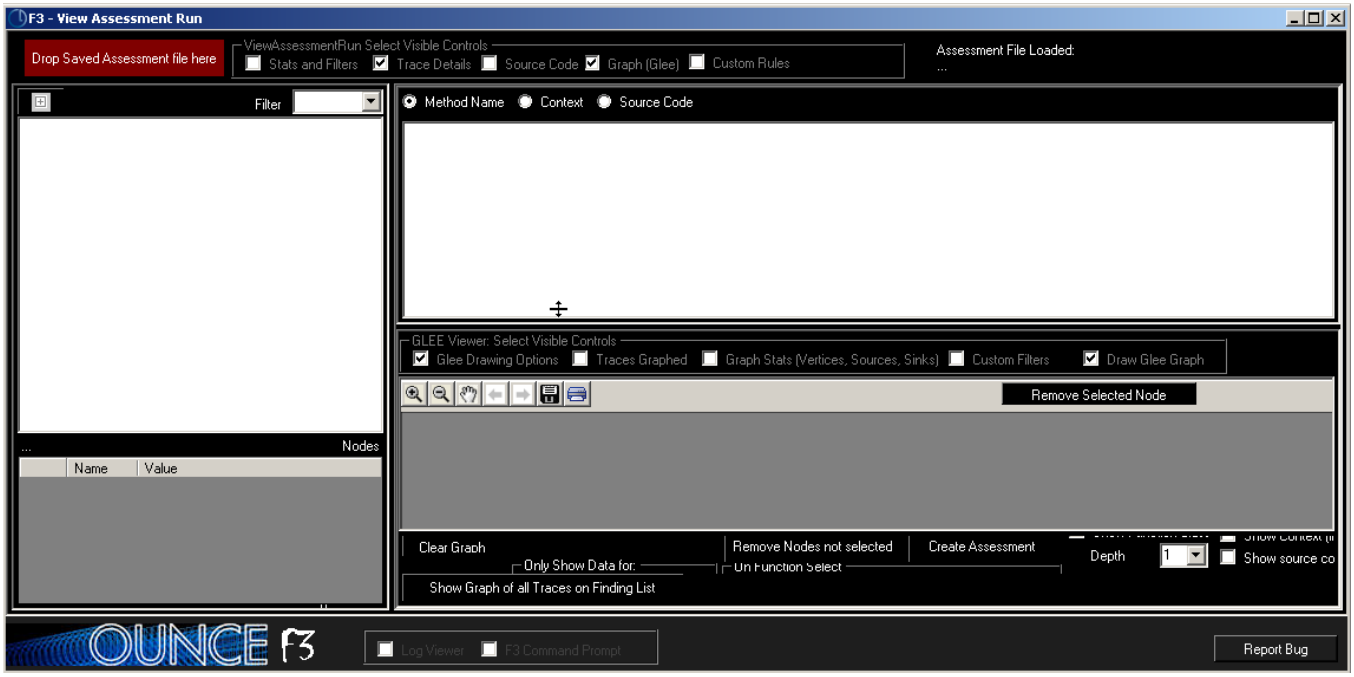




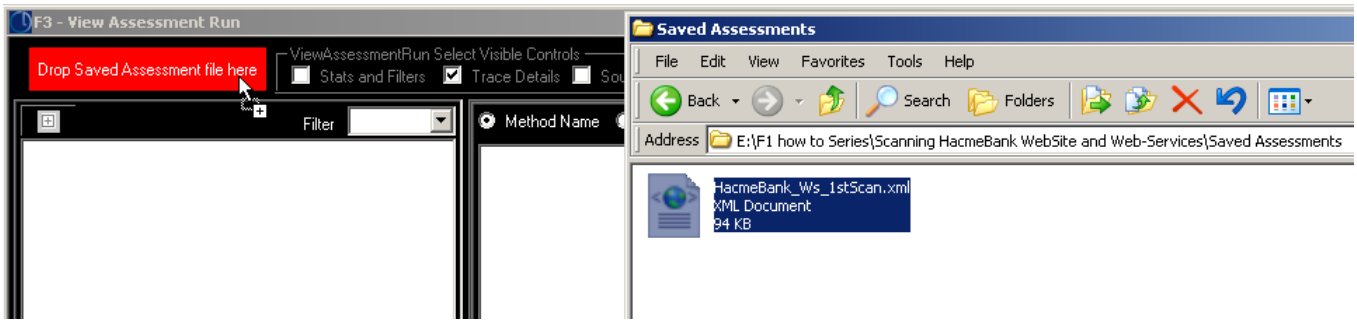
VIEWING SAVED ASSESSMENT:

Open f1AddOn_ViewAssessmentRun.exe

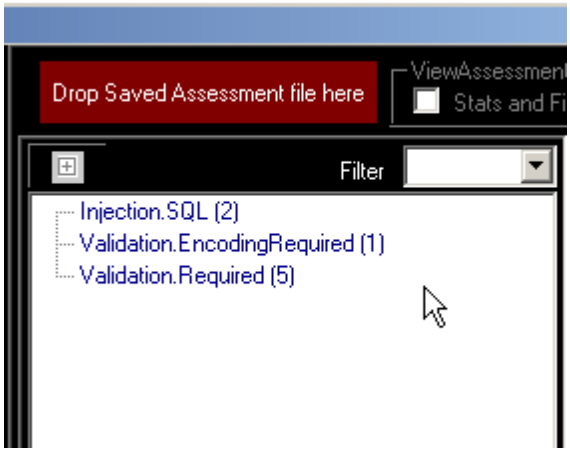




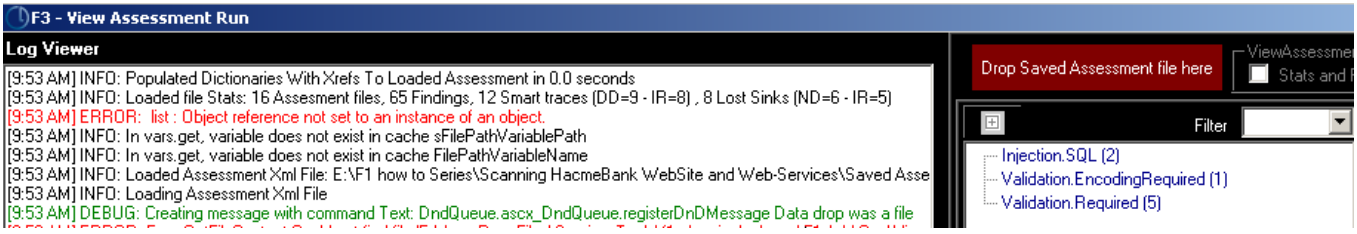
Drag and drop the previously created Saved assessment file into the ‘Drop Saved Assessment file here’ area:



This will load the saved assessment file which in this first pass will have 7 unique traces:

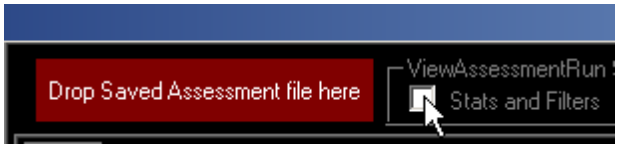


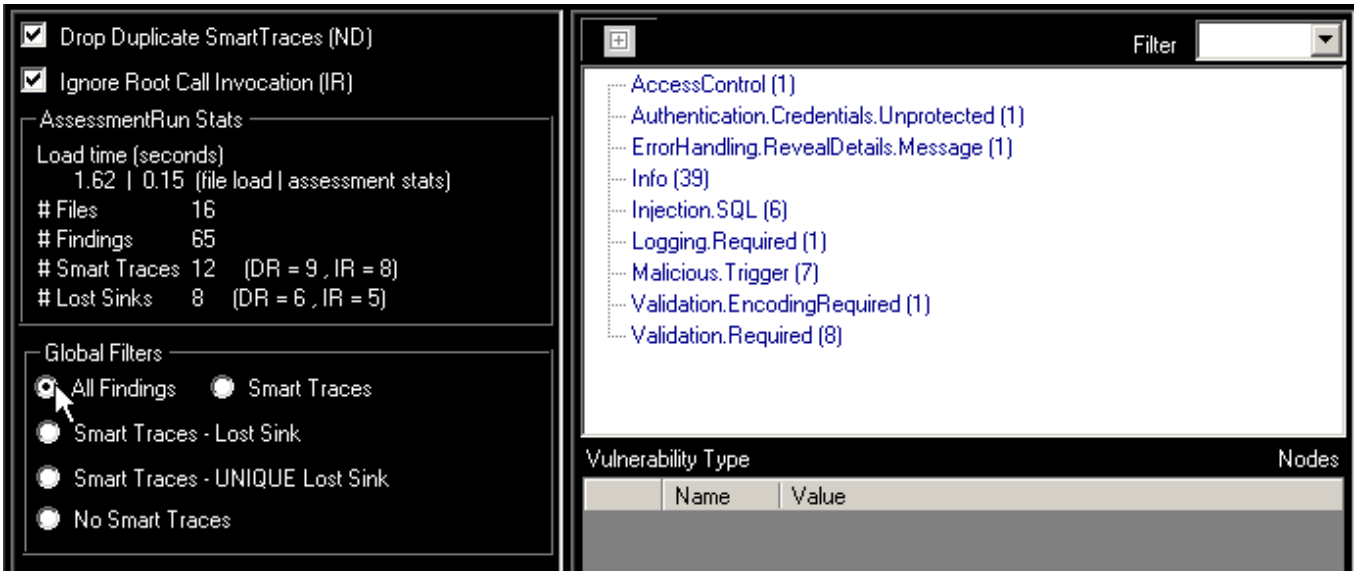
To see the log viewer click on the 'Log Viewer' checkbox at the bottom of the F3 gui:

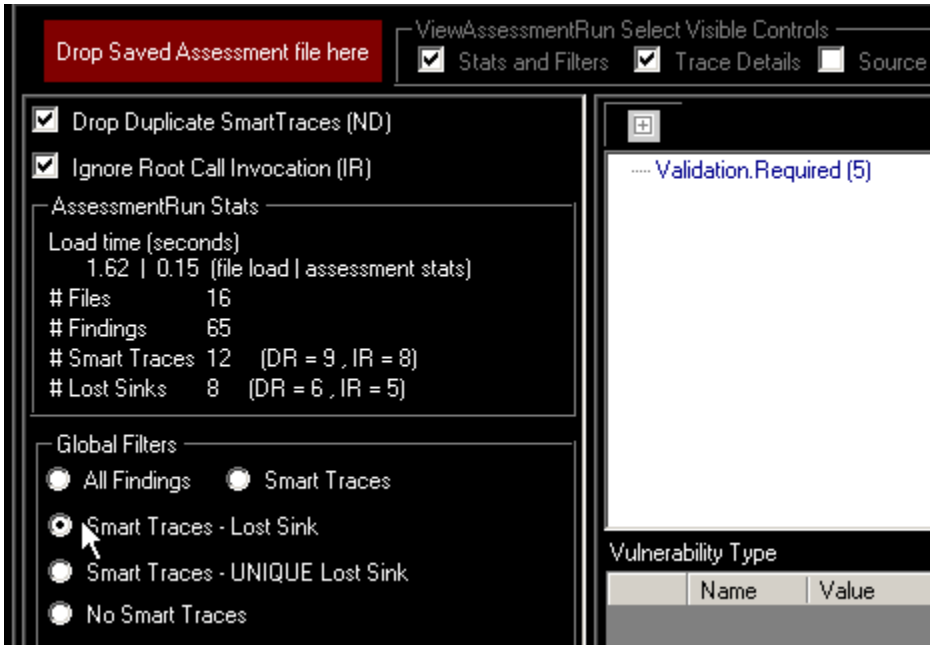


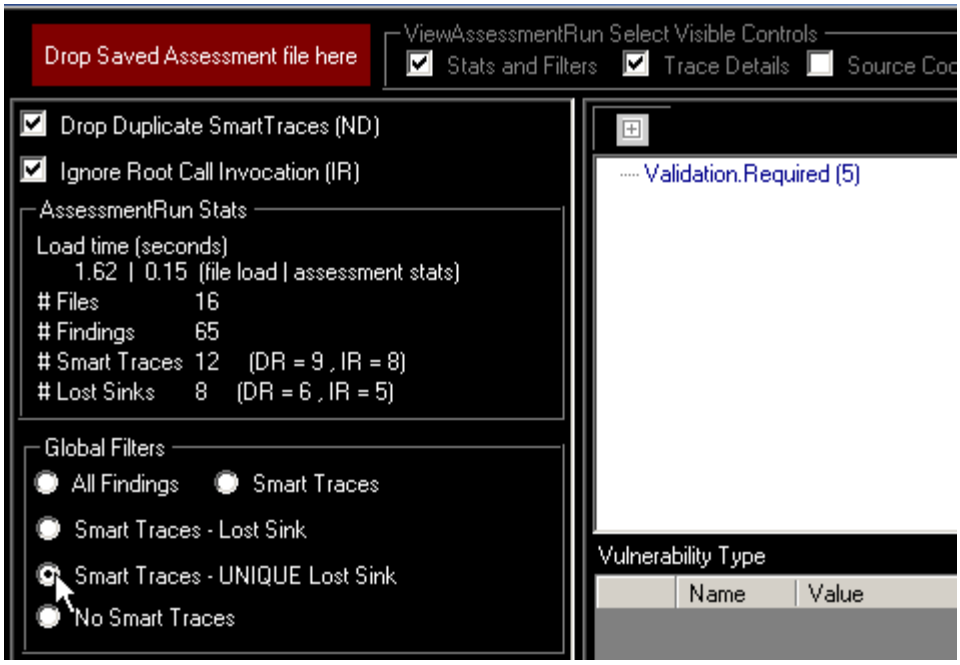
..

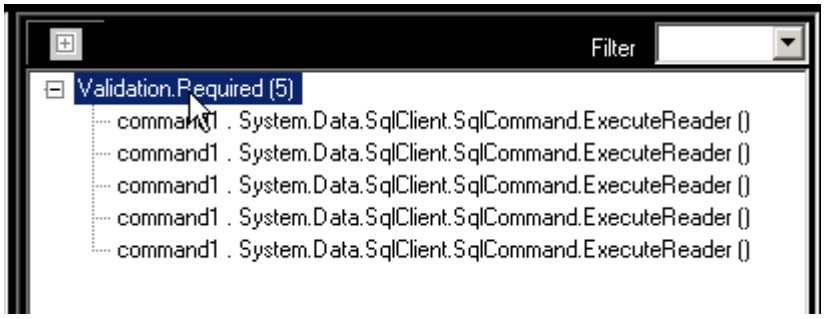
[9:53 AM] INFO: Populated Dictionaries With Xrefs To Loaded Assessment in 0.0 seconds
[9:53 AM] INFO: Loaded file Stats: 16 Assesment files, 65 Findings, 12 Smart traces (DD=9 - IR=8) , 8 Lost Sinks (ND=6 - IR=5)
(Explain how to read the stats)

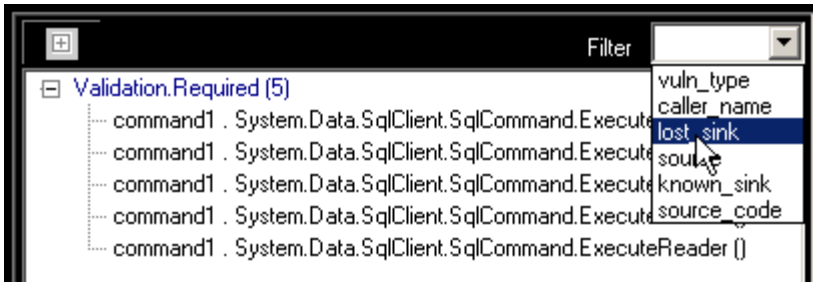


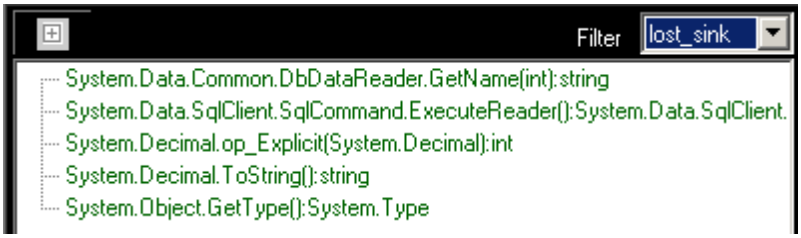

Drop Duplicate SmartTraces (ND)
Ignore Root Call Invocation (IR)
AssessmentRun Stats
Load time (seconds)
1.62 | 0.15 (file load | assessment stats)
Files 16
Findings 65
Smart Traces 12 (DR = 9 , IR = 8)
Lost Sinks 8 (DR = 6 , IR = 5)
Global Filters
All Findings
Smart Traces
Smart Traces - Lost Sink
Smart Traces - UNIQUE Lost Sink
No Smart Traces
Filter
AccessControl (1)
Authentication.Credentials.Unprotected (1)
ErrorHandling.RevealDetails.Message (1)
Info (39)
Injection.SQL (6)
Logging.Required (1)
Malicious.Trigger (7)
Validation.EncodingRequired (1)
Validation.Required (8)
Vulnerability Type
Nodes
Name
Value

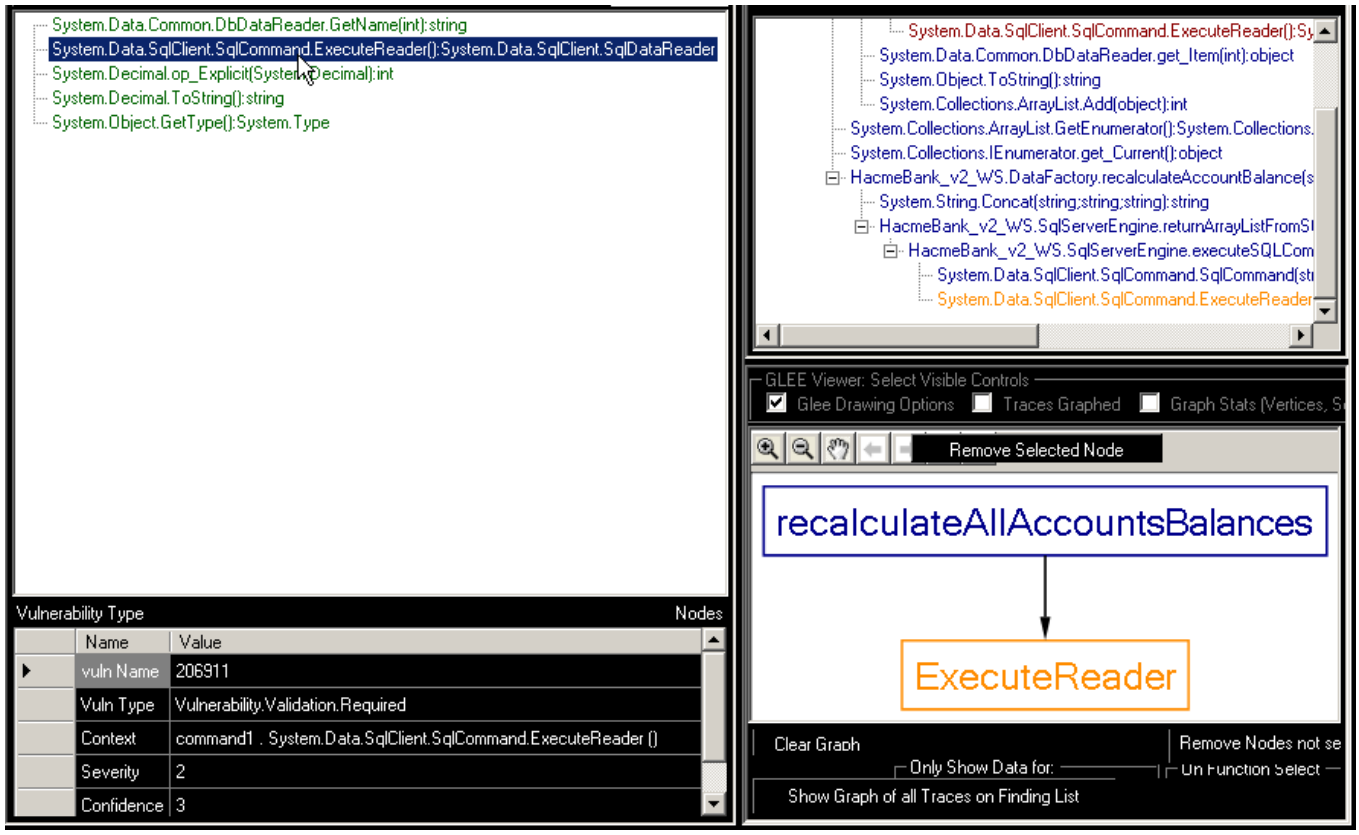

Drop Saved Assessment file here
ViewAssessmentRun Select Visible Controls
Stats and Filters
Trace Details
Source
Drop Duplicate SmartTraces (ND)
Ignore Root Call Invocation (IR)
AssessmentRun Stats
Load time (seconds)
1.62 | 0.15 (file load | assessment stats)
Files 16
Findings 65
Smart Traces 12 (DR = 9 , IR = 8)
Lost Sinks 8 (DR = 6 , IR = 5)
Global Filters
All Findings
Smart Traces
Smart Traces - Lost Sink
Smart Traces - UNIQUE Lost Sink
No Smart Traces
Validation.Required (5)
Vulnerability Type
Name
Value


Drop Saved Assessment file here
ViewAssessmentRun Select Visible Controls
Stats and Filters
Trace Details
Drop Duplicate SmartTraces (ND)
Ignore Root Call Invocation (IR)
AssessmentRun Stats
Load time (seconds)
1.62 | 0.15 (file load | assessment stats)
Files 16
Findings 65
Smart Traces 12 (DR = 9 , IR = 8)
Lost Sinks 8 (DR = 6 , IR = 5)
Global Filters
All Findings
Smart Traces
Smart Traces - Lost Sink
Smart Traces - UNIQUE Lost Sink
No Smart Traces
Validation.Required (5)
Vulnerability Type
Name
Value

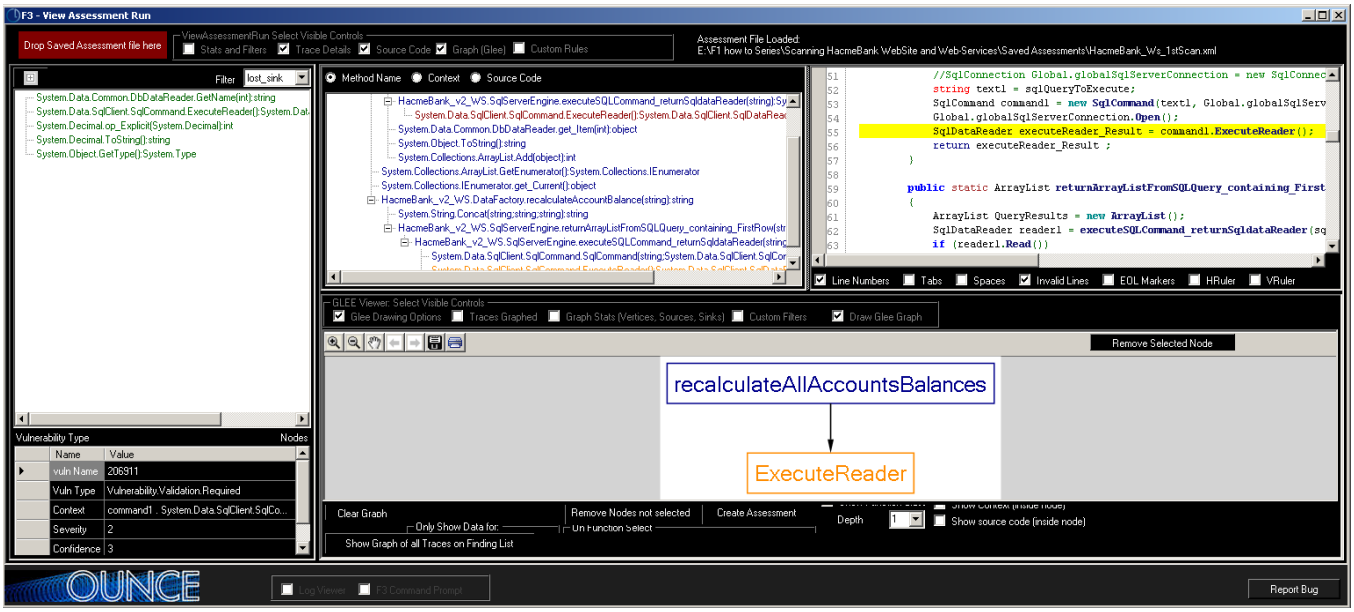

Filter
Validation.Required (5)
command1 . System.Data.SqlClient.SqlCommand.ExecuteReader ()
command1 . System.Data.SqlClient.SqlCommand.ExecuteReader ()
command1 . System.Data.SqlClient.SqlCommand.ExecuteReader ()
command1 . System.Data.SqlClient.SqlCommand.ExecuteReader ()
command1 . System.Data.SqlClient.SqlCommand.ExecuteReader ()

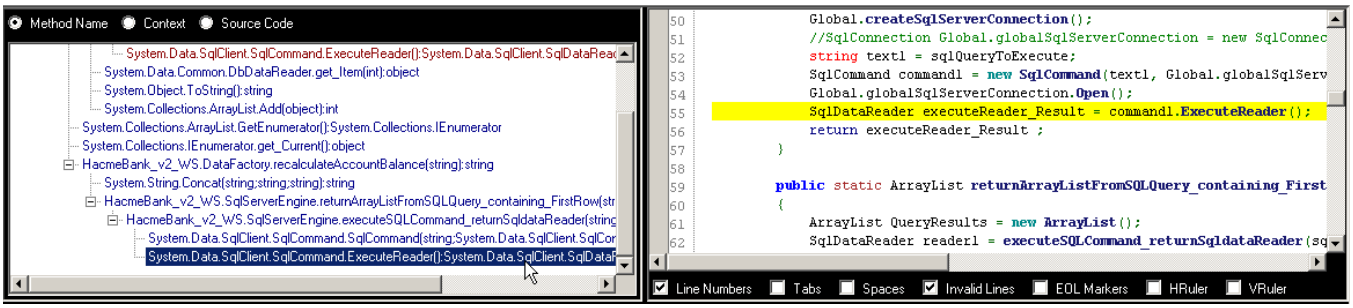

Filter
Validation.Required (5)
vuln_type
caller_name
lost_sink
known_sink
source_code
command1 . System.Data.SqlClient.SqlCommand.ExecuteReader ()


Filter
lost_sink
System.Data.Common.DbDataReader.GetName(int):string
System.Data.SqlClient.SqlCommand.ExecuteReader():System.Data.SqlClient.
System.Decimal.op_Explicit(System.Decimal):int
System.Decimal.ToString():string
System.Object.GetType():System.Type

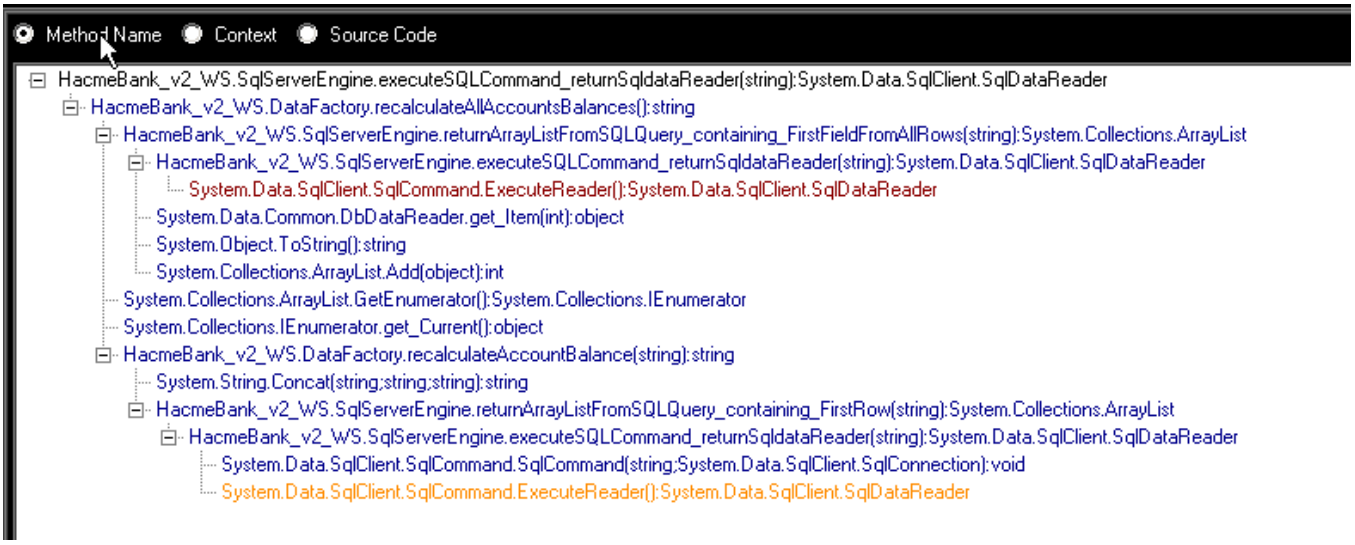

System.Data.Common.DbDataReader.GetName(int):string
System.Data.SqlClient.SqlCommand.ExecuteReader():System.Data.SqlClient.SqlDataReader
System.Decimal.op_Explicit(System.Decimal):int
System.Decimal.ToString():string
System.Object.GetType():System.Type
Vulnerability Type
Nodes
Name
Value
vuln Name
206911
Vuln Type
Vulnerability.Validation.Required
Context
command1 . System.Data.SqlClient.SqlCommand.ExecuteReader ()
Severity
2
Confidence
3
System.Data.SqlClient.SqlCommand.ExecuteReader():Sy
System.Data.Common.DbDataReader.get_Item(int):object
System.Object.ToString():string
System.Collections.ArrayList.Add(object):int
System.Collections.ArrayList.GetEnumerator():System.Collections.
System.Collections.IEnumerator.get_Current():object
HacmeBank_v2_WS.DataFactory.recalculateAccountBalance(s
System.String.Concat(string;string;string):string
HacmeBank_v2_WS.SqlServerEngine.executeSQLCom
System.Data.SqlClient.SqlCommand.SqlCommand(st
System.Data.SqlClient.SqlCommand.ExecuteReader
GLEE Viewer: Select Visible Controls
Glee Drawing Options
Traces Graphed
Graph Stats (Vertices,
Remove Selected Node
recalculateAllAccountsBalances
ExecuteReader
Clear Graph
Remove Nodes not se
Only Show Data for:
Un Function Select
Show Graph of all Traces on Finding List


F3 - View Assessment Run
Drop Saved Assessment file here
ViewAssessmentRun Select Visible Controls
Stats and Filters
Trace Details
Source Code
Graph (Glee)
Custom Rules

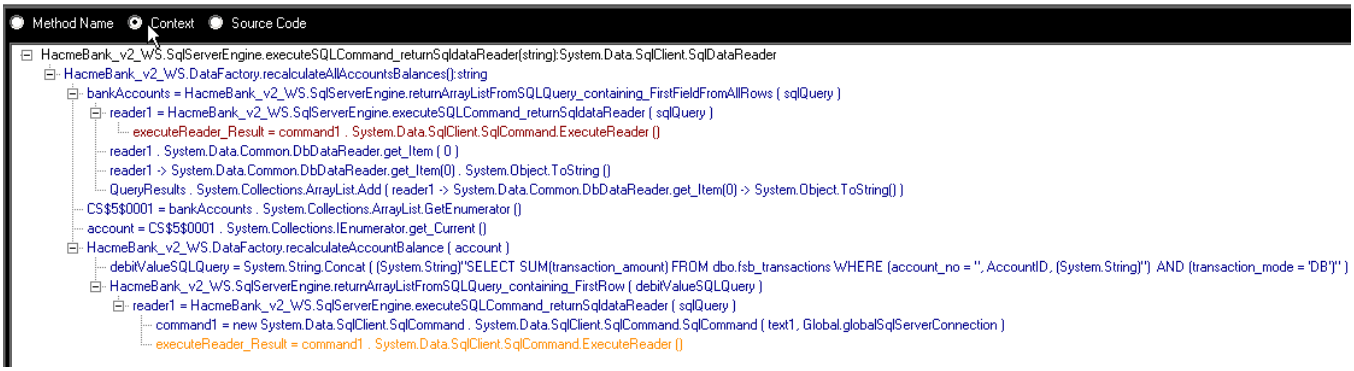




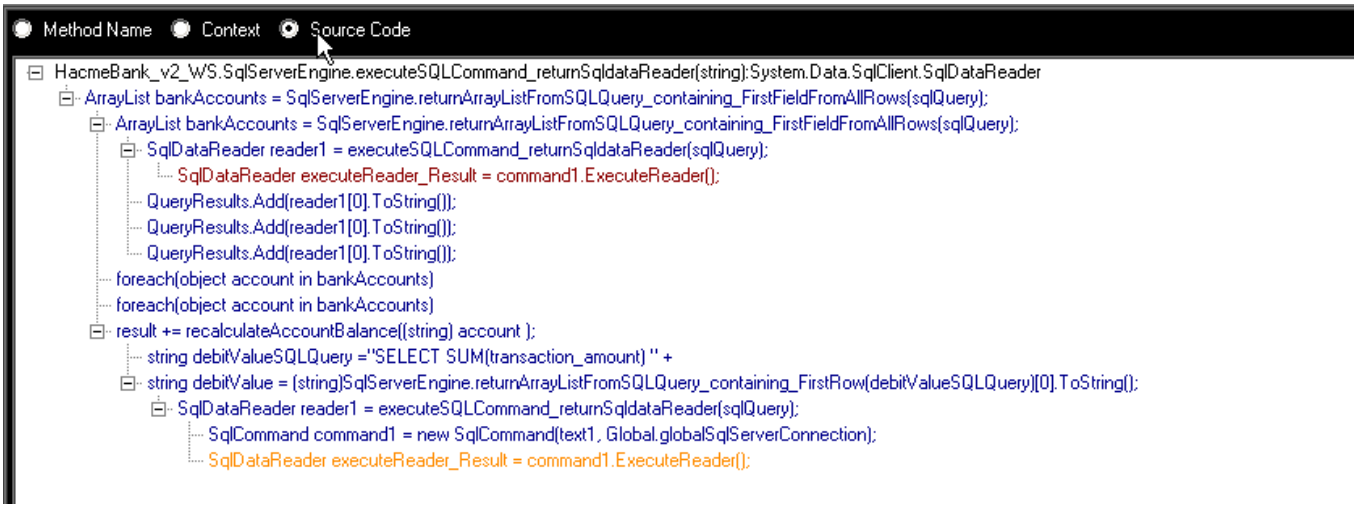
Multiple views



By Context:



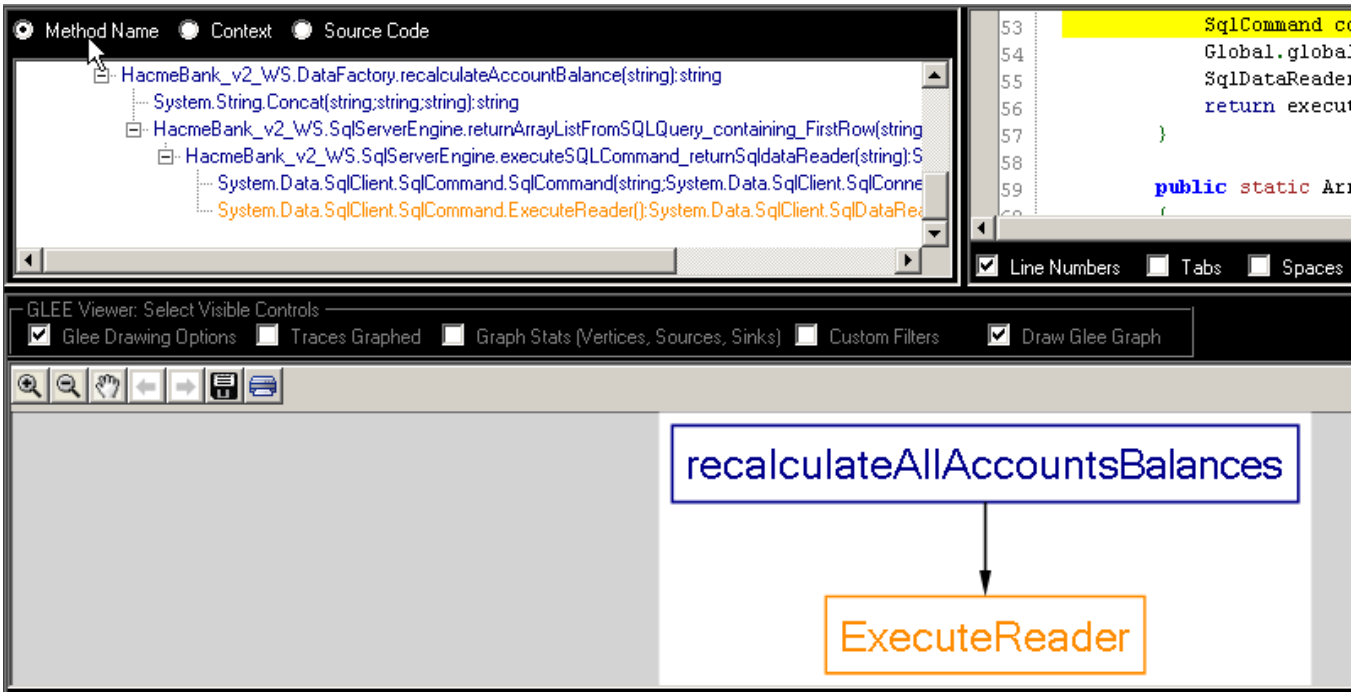
(note how the SQL injection is easier to read)



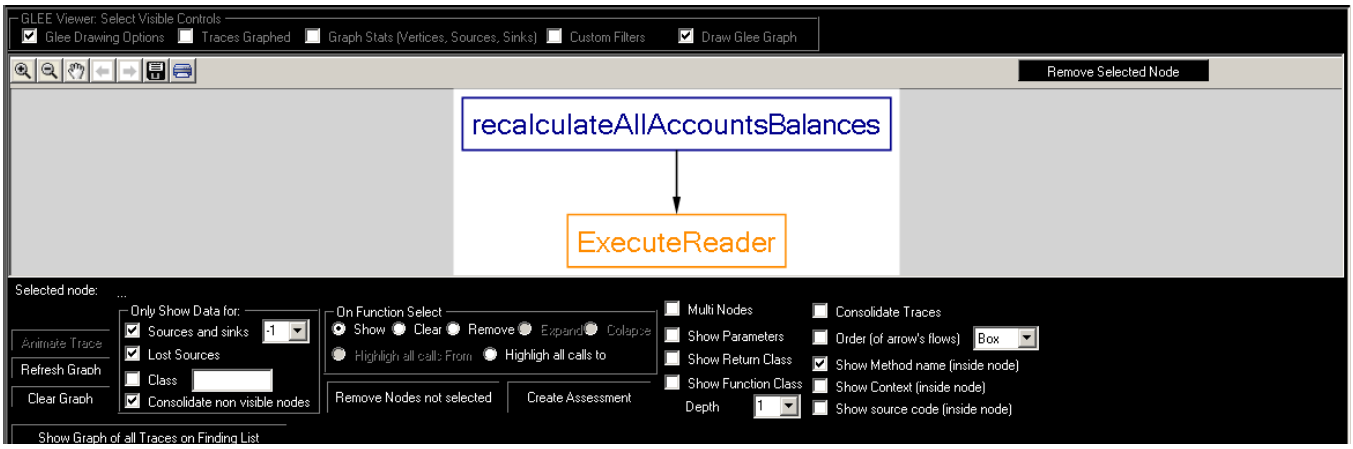
GLEE GRAPH

Let's now focus on the GLEE Graph

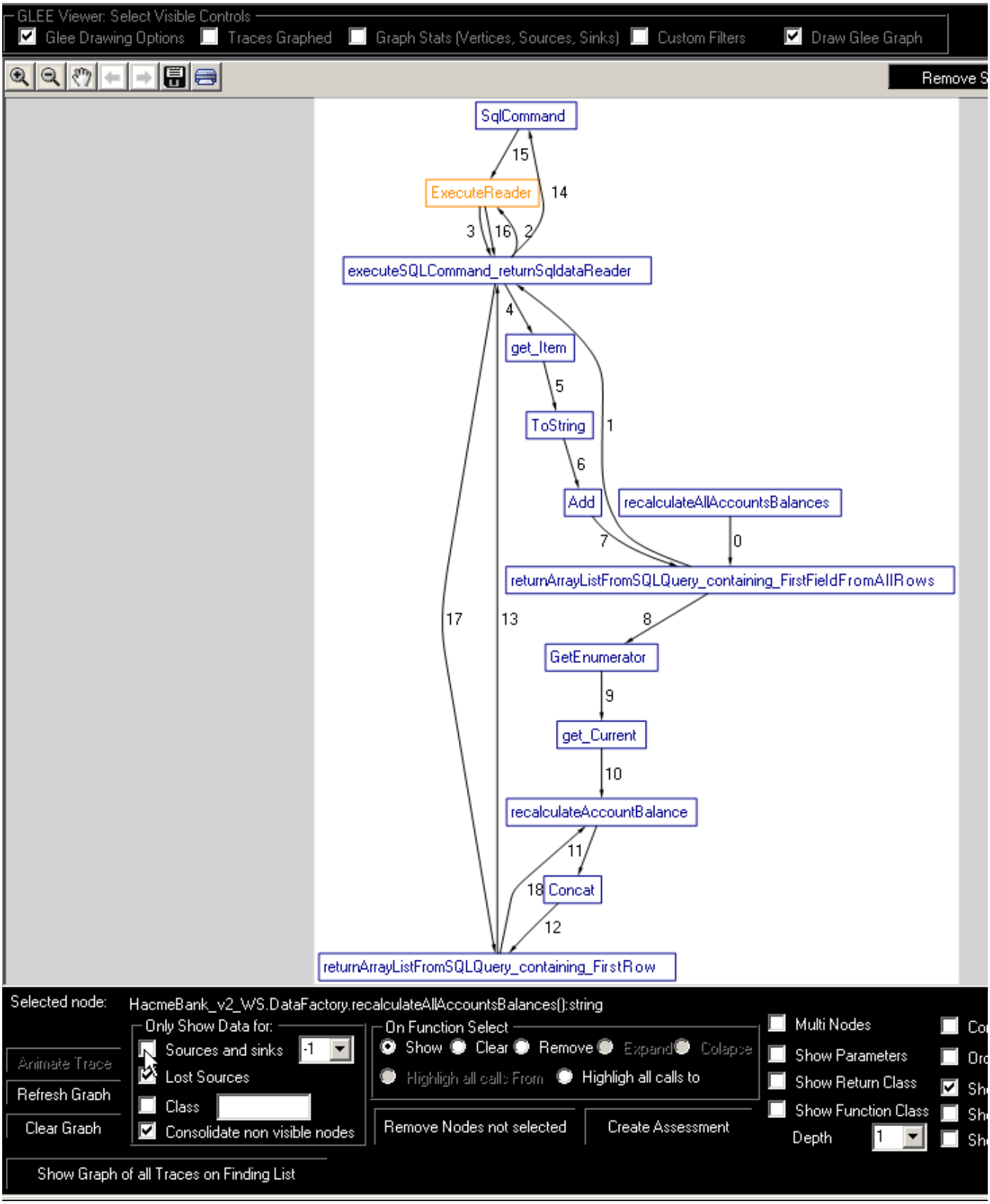
Go back to the 'Method Name' Filter



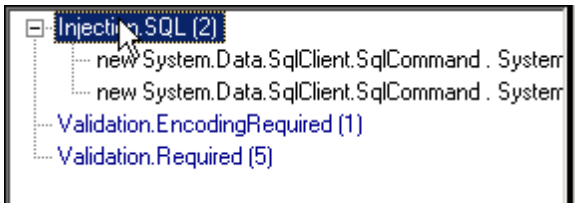
You might need to resize the options section ‘Glee Drawing options’

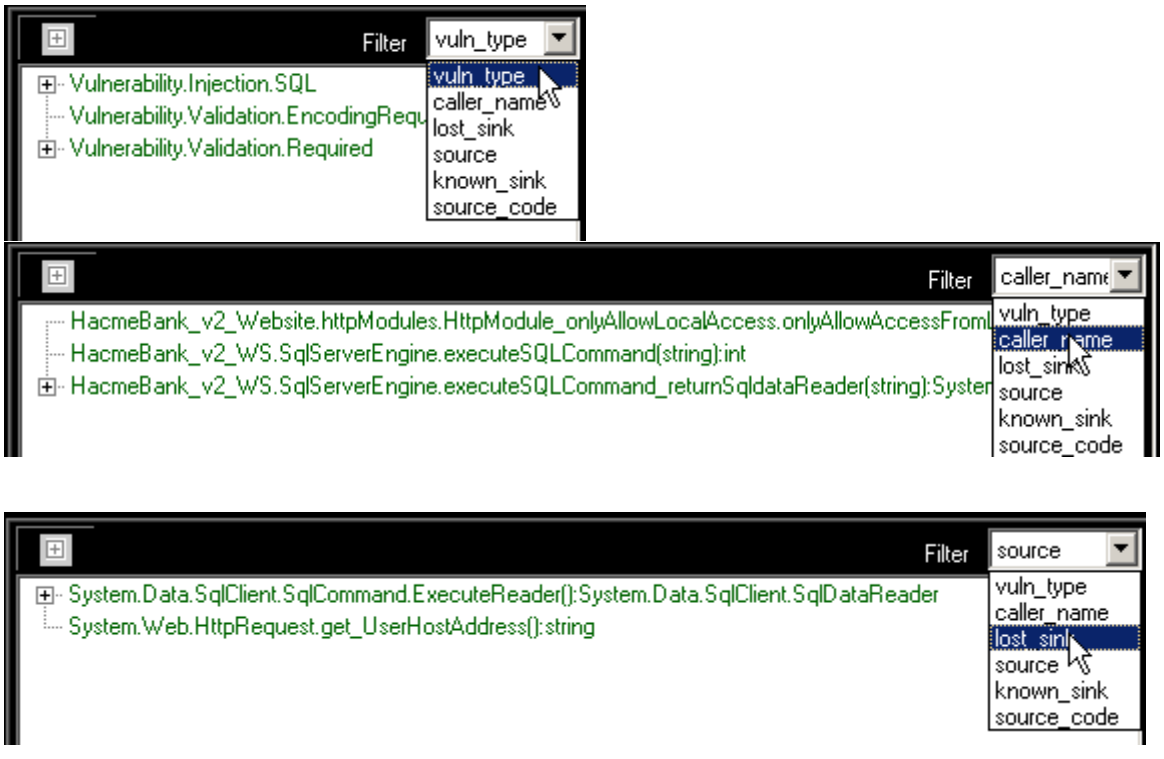


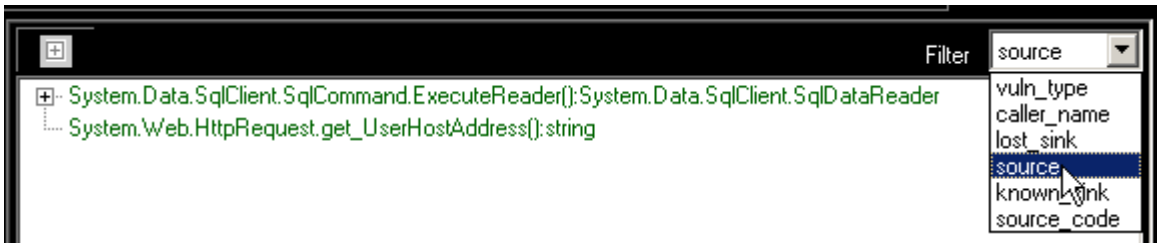
The numbers between the nodes are the sequence (from 0 till 16)

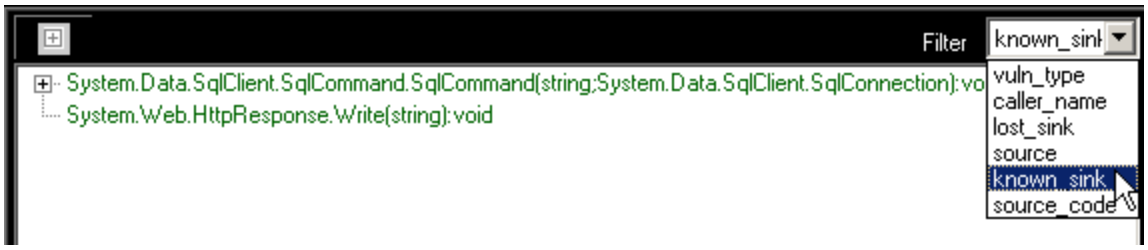

GLEE Viewer: Select Visible Controls
Glee Drawing Options
Traces Graphed
Graph Stats (Vertices, Sources, Sinks)
Custom Filters
Draw Glee Graph
Remove S
SqlCommand
15
ExecuteReader
14
3
16
2
executeSQLCommand_returnSqldataReader
4
get_Item
5
ToString
1
6
Add
recalculateAllAccountsBalances
7
0
returnArrayListFromSQLQuery_containing_FirstFieldFromAllRows
17
13
8
GetEnumerator
9
get_Current
10
recalculateAccountBalance
11
18
Concat
12
returnArrayListFromSQLQuery_containing_FirstRow
Selected node:
HacmeBank_v2_WS.DataFactory.recalculateAllAccountsBalances():string
Only Show Data for:
Sources and sinks
-1
Lost Sources
Class
Consolidate non visible nodes
On Function Select
Show
Clear
Remove
Expand
Collapse
Highligh all calls From
Highligh all calls to
Multi Nodes
Show Parameters
Show Return Class
Show Function Class
Depth
1
Animate Trace
Refresh Graph
Clear Graph
Remove Nodes not selected
Create Assessment
Show Graph of all Traces on Finding List

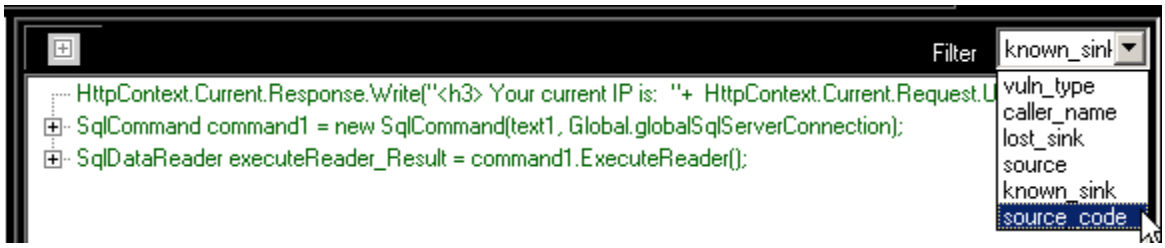
This is not be best trace to Highlight GLEEs features, lets pick up another one:



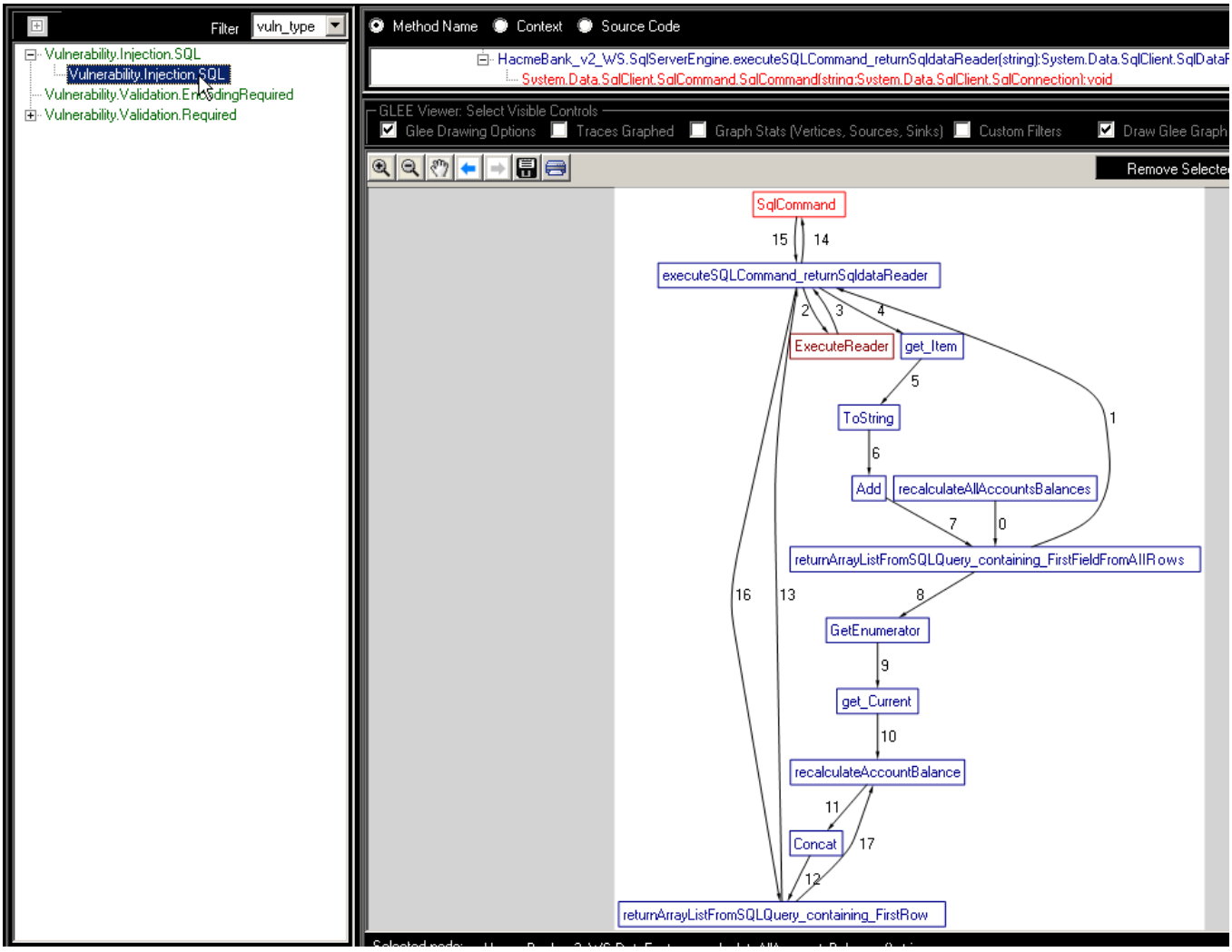


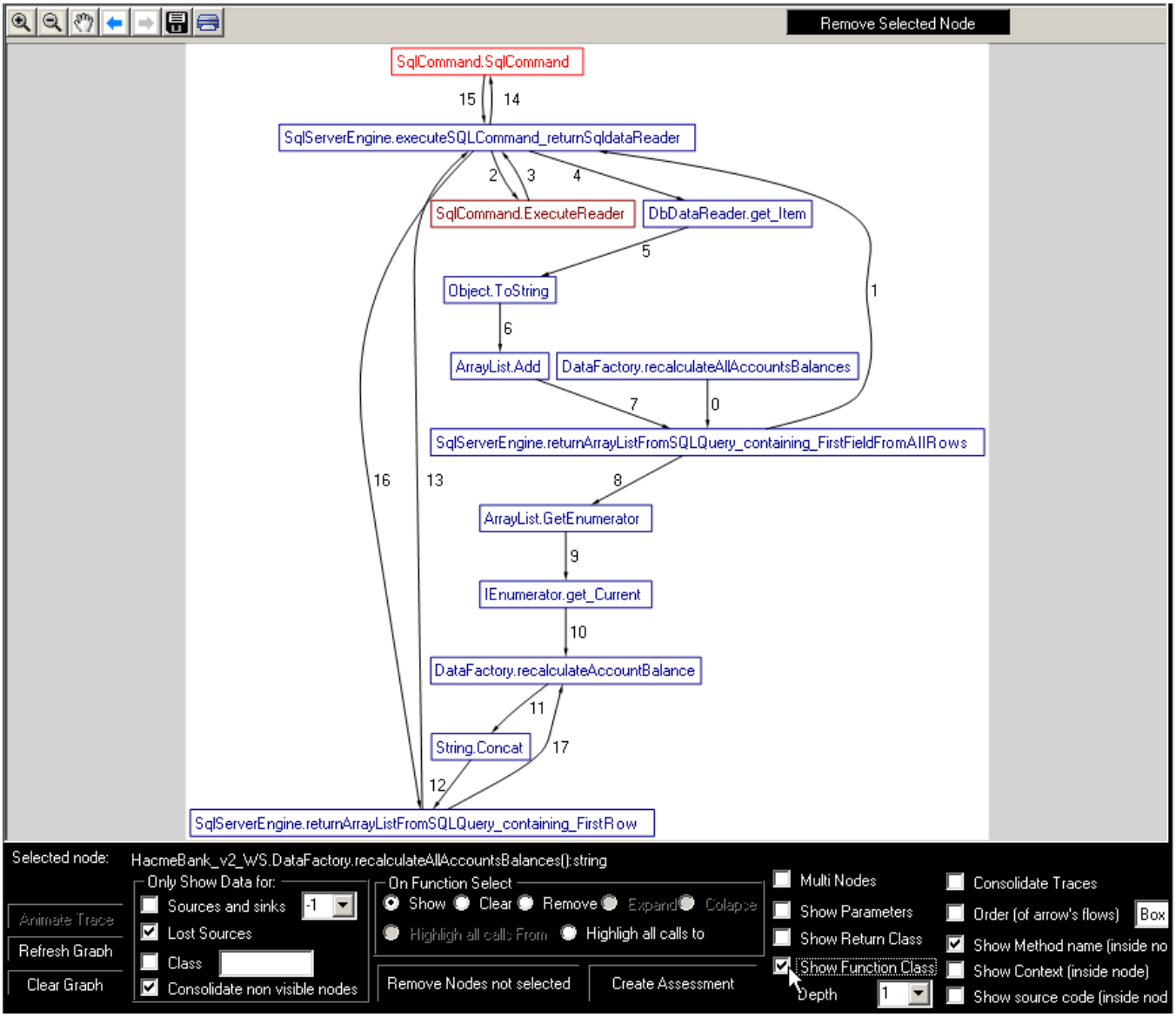


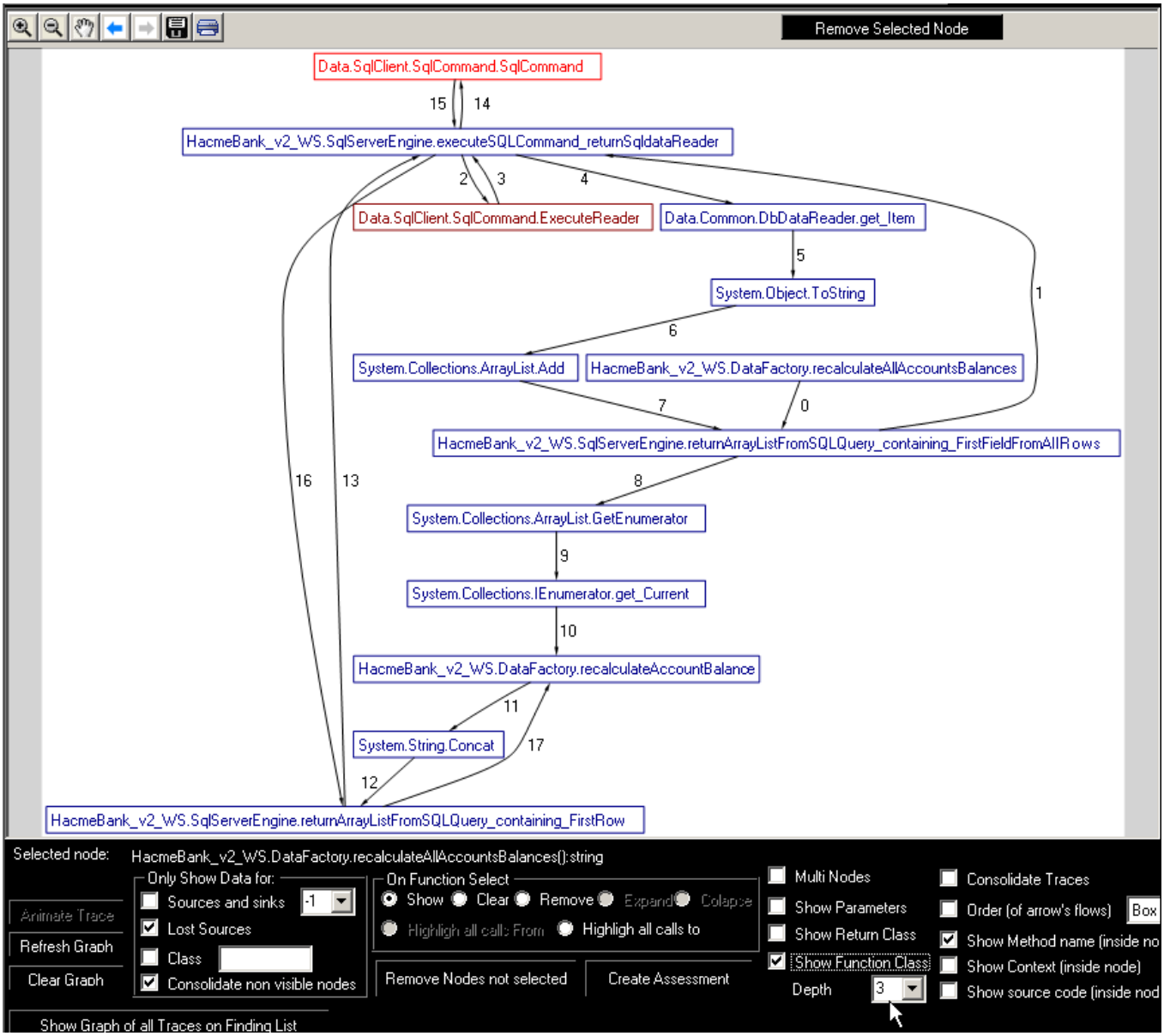


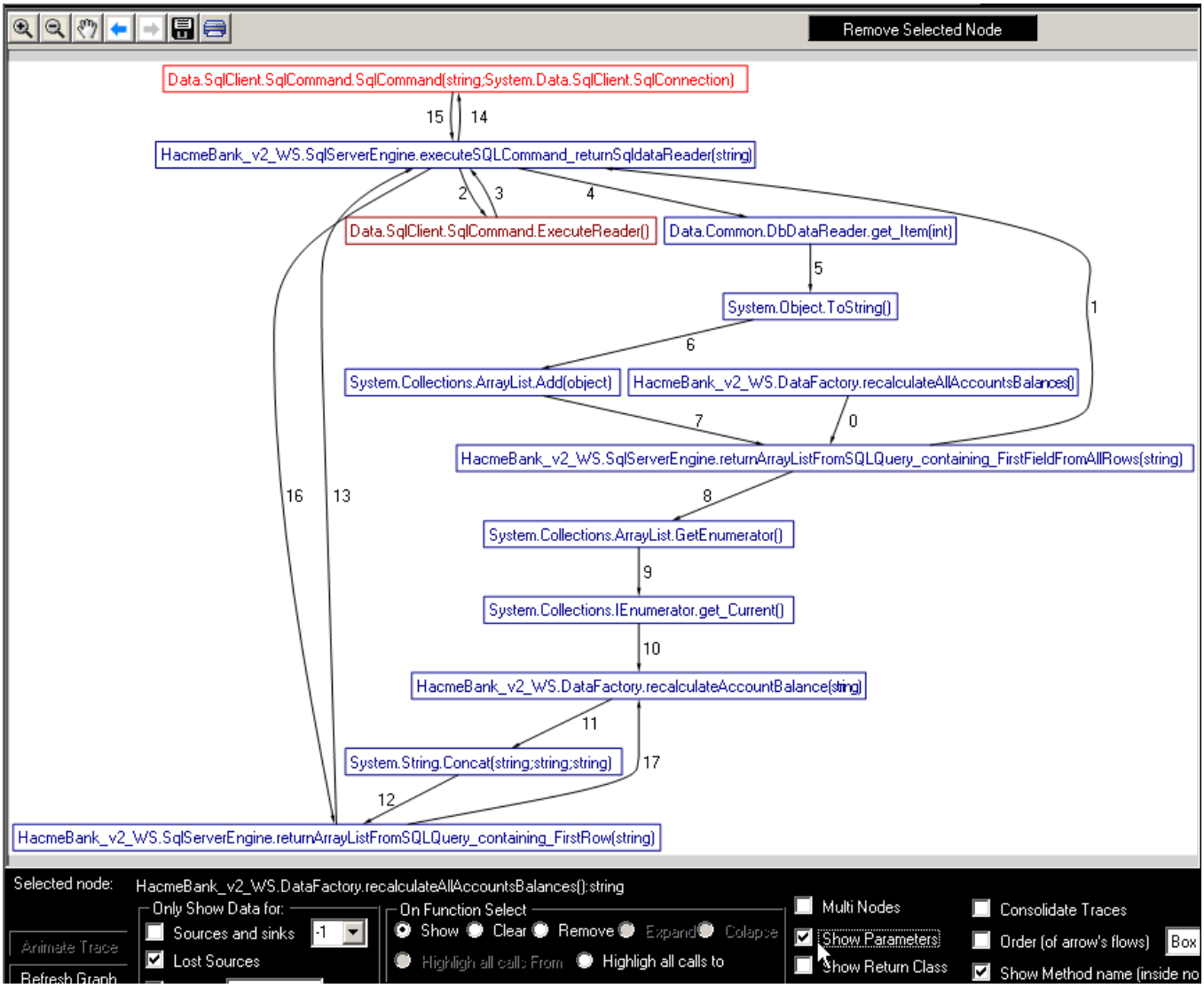


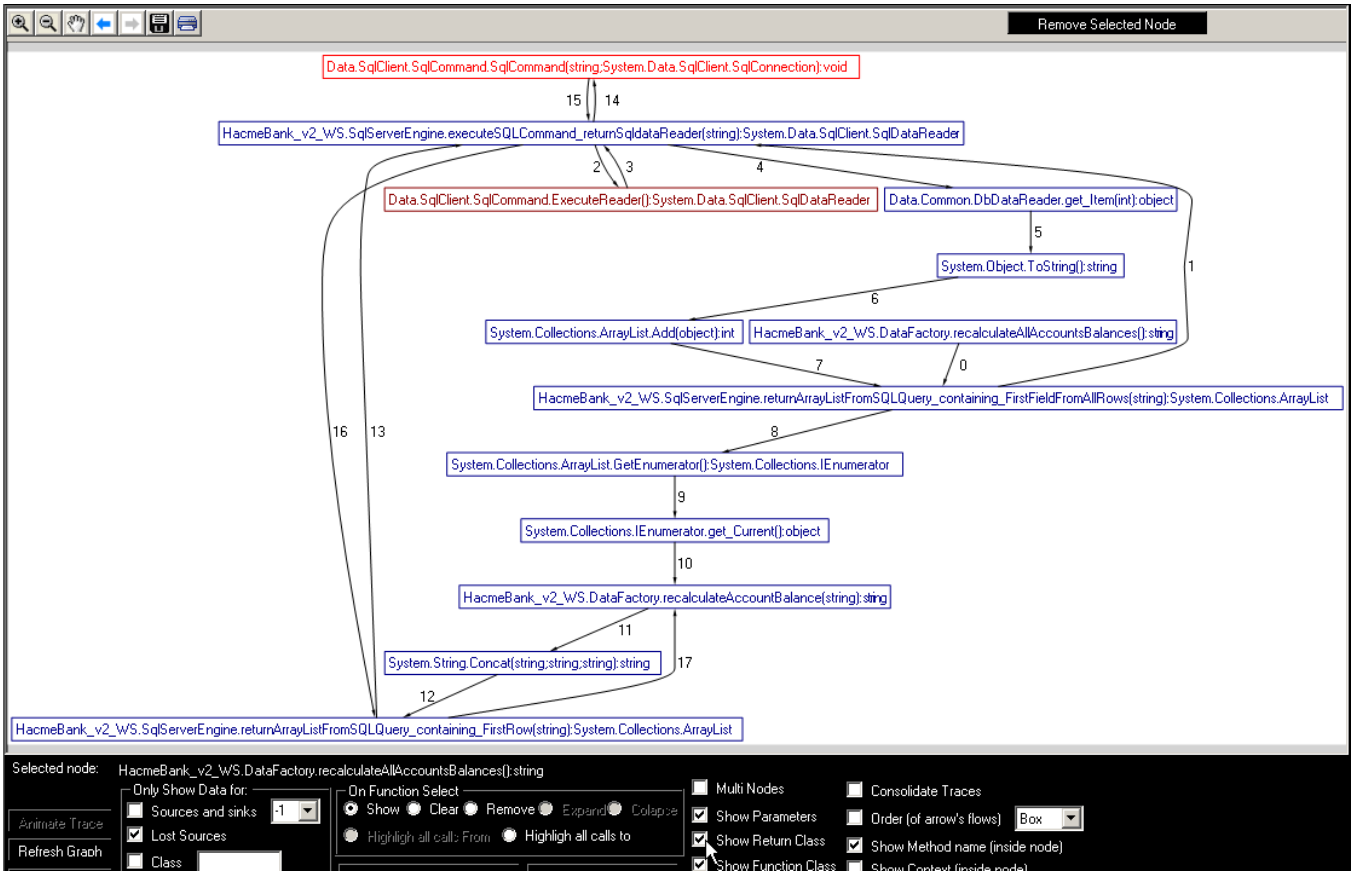
Go back to the vuln_type filter and select 2nd 'Vulnerability.Injectin.SQL'


Filter
vuln_type
Vulnerability.Injection.SQL
Vulnerability.Injection.SQL
Vulnerability.Validation.EncodingRequired
Vulnerability.Validation.Required
Method Name
Context
Source Code
HacmeBank_v2_WS.SqlServerEngine.executeSQLCommand_returnSqldataReader(string):System.Data.SqlClient.SqlDataR
System.Data.SqlClient.SqlCommand.SqlCommand(string:System.Data.SqlClient.SqlConnection):void
GLEE Viewer: Select Visible Controls
Glee Drawing Options
Traces Graphed
Graph Stats (Vertices, Sources, Sinks)
Custom Filters
Draw Glee Graph
Remove Selecte
SqlCommand
executeSQLCommand_returnSqldataReader
ExecuteReader
get_Item
ToString
Add
recalculateAllAccountsBalances
returnArrayListFromSQLQuery_containing_FirstFieldFromAllRows
GetEnumerator
get_Current
recalculateAccountBalance
Concat
returnArrayListFromSQLQuery_containing_FirstRow


Remove Selected Node
SqlCommand.SqlCommand
15
14
SqlServerEngine.executeSQLCommand_returnSqldataReader
2
3
4
SqlCommand.ExecuteReader
DbDataReader.get_Item
5
Object.ToString
1
6
ArrayList.Add
DataFactory.recalculateAllAccountsBalances
7
0
SqlServerEngine.returnArrayListFromSQLQuery_containing_FirstFieldFromAllRows
16
13
8
ArrayList.GetEnumerator
9
IEnumerator.get_Current
10
DataFactory.recalculateAccountBalance
11
String.Concat
17
12
SqlServerEngine.returnArrayListFromSQLQuery_containing_FirstRow
Selected node:
HacmeBank_v2_WS.DataFactory.recalculateAllAccountsBalances():string
Animate Trace
Refresh Graph
Clear Graph
Only Show Data for:
Sources and sinks
-1
Lost Sources
Class
Consolidate non visible nodes
On Function Select
Show
Clear
Remove
Expand
Colapse
Highligh all calls: From
Highligh all calls to
Remove Nodes not selected
Create Assessment
Multi Nodes
Show Parameters
Show Return Class
Show Function Class
Depth
1
Consolidate Traces
Order (of arrow's flows)
Box
Show Method name (inside no
Show Context (inside node)
Show source code (inside nod

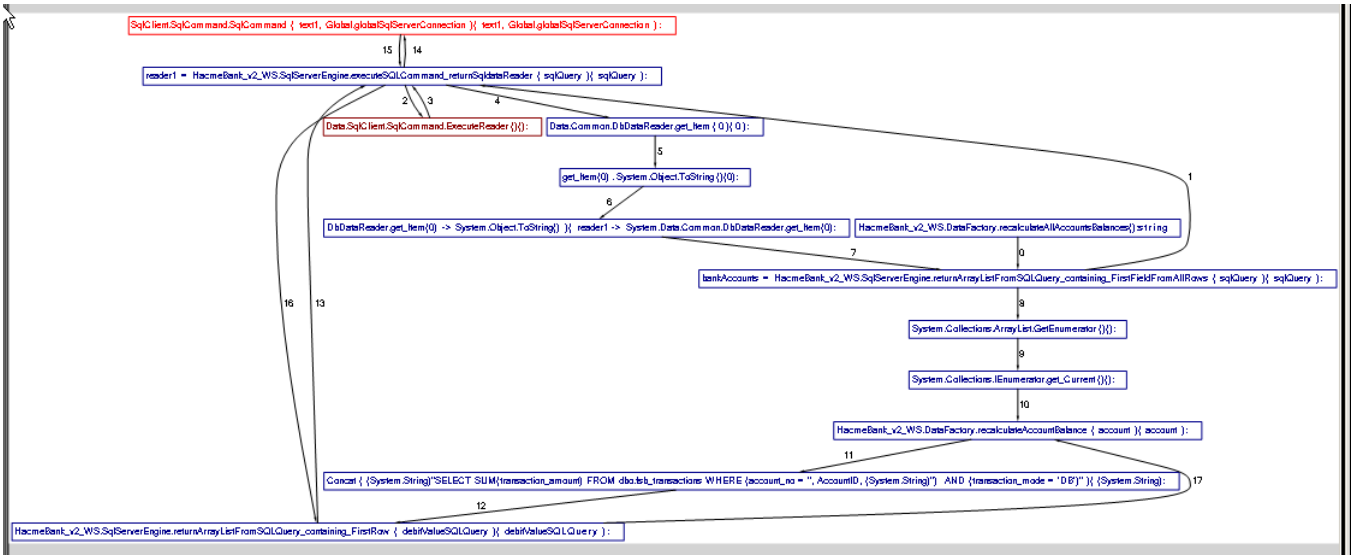

Remove Selected Node
Data.SqlClient.SqlCommand.SqlCommand
15
14
HacmeBank_v2_WS.SqlServerEngine.executeSQLCommand_returnSqldataReader
2
3
4
Data.SqlClient.SqlCommand.ExecuteReader
Data.Common.DbDataReader.get_Item
5
System.Object.ToString
1
6
System.Collections.ArrayList.Add
HacmeBank_v2_WS.DataFactory.recalculateAllAccountsBalances
7
0
HacmeBank_v2_WS.SqlServerEngine.returnArrayListFromSQLQuery_containing_FirstFieldFromAllRows
16
13
8
System.Collections.ArrayList.GetEnumerator
9
System.Collections.IEnumerator.get_Current
10
HacmeBank_v2_WS.DataFactory.recalculateAccountBalance
11
System.String.Concat
17
12
HacmeBank_v2_WS.SqlServerEngine.returnArrayListFromSQLQuery_containing_FirstRow
Selected node:
HacmeBank_v2_WS.DataFactory.recalculateAllAccountsBalances():string
Only Show Data for:
Sources and sinks
-1
Lost Sources
Class
Consolidate non visible nodes
On Function Select
Show
Clear
Remove
Expand
Colapse
Highligh all calls From
Highligh all calls to
Multi Nodes
Show Parameters
Show Return Class
Show Function Class
Depth
3
Consolidate Traces
Order (of arrow's flows)
Box
Show Method name (inside no
Show Context (inside node)
Show source code (inside nod
Animate Trace
Refresh Graph
Clear Graph
Remove Nodes not selected
Create Assessment
Show Graph of all Traces on Finding List


Remove Selected Node
Data.SqlClient.SqlCommand.SqlCommand(string;System.Data.SqlClient.SqlConnection)
15
14
HacmeBank_v2_WS.SqlServerEngine.executeSQLCommand_returnSqldataReader(string)
2
3
4
Data.SqlClient.SqlCommand.ExecuteReader()
Data.Common.DbDataReader.get_Item(int)
5
System.Object.ToString()
1
6
System.Collections.ArrayList.Add(object)
HacmeBank_v2_WS.DataFactory.recalculateAllAccountsBalances()
7
0
HacmeBank_v2_WS.SqlServerEngine.returnArrayListFromSQLQuery_containing_FirstFieldFromAllRows(string)
16
13
8
System.Collections.ArrayList.GetEnumerator()
9
System.Collections.IEnumerator.get_Current()
10
HacmeBank_v2_WS.DataFactory.recalculateAccountBalance(string)
11
System.String.Concat(string;string;string)
17
12
HacmeBank_v2_WS.SqlServerEngine.returnArrayListFromSQLQuery_containing_FirstRow(string)
Selected node:
HacmeBank_v2_WS.DataFactory.recalculateAllAccountsBalances():string
Animate Trace
Refresh Graph
Only Show Data for:
Sources and sinks
-1
Lost Sources
On Function Select
Show
Clear
Remove
Expand
Colapse
Highligh all calls From
Highligh all calls to
Multi Nodes
Show Parameters
Show Return Class
Consolidate Traces
Order (of arrow's flows)
Box
Show Method name (inside no


Remove Selected Node
Data.SqlClient.SqlCommand.SqlCommand(string;System.Data.SqlClient.SqlConnection):void
HacmeBank_v2_WS.SqlServerEngine.executeSQLCommand_returnSqldataReader(string):System.Data.SqlClient.SqlDataReader
Data.SqlClient.SqlCommand.ExecuteReader():System.Data.SqlClient.SqlDataReader
Data.Common.DbDataReader.get_Item(int):object
System.Object.ToString():string
System.Collections.ArrayList.Add(object):int
HacmeBank_v2_WS.DataFactory.recalculateAllAccountsBalances():string
HacmeBank_v2_WS.SqlServerEngine.returnArrayListFromSQLQuery_containing_FirstFieldFromAllRows(string):System.Collections.ArrayList
System.Collections.ArrayList.GetEnumerator():System.Collections.IEnumerator
System.Collections.IEnumerator.get_Current():object
HacmeBank_v2_WS.DataFactory.recalculateAccountBalance(string):string
System.String.Concat(string;string;string):string
HacmeBank_v2_WS.SqlServerEngine.returnArrayListFromSQLQuery_containing_FirstRow(string):System.Collections.ArrayList
Selected node:
HacmeBank_v2_WS.DataFactory.recalculateAllAccountsBalances():string
Animate Trace
Refresh Graph
Only Show Data for:
Sources and sinks
Lost Sources
Class
On Function Select
Show
Clear
Remove
Expand
Collapse
Highligh all calls: From
Highligh all calls to
Multi Nodes
Show Parameters
Show Return Class
Consolidate Traces
Order (of arrow's flows)
Box
Show Method name (inside node)


Method Name
Context
Source Code


Selected node:
Animate Trace
Refresh Graph
Clear Graph

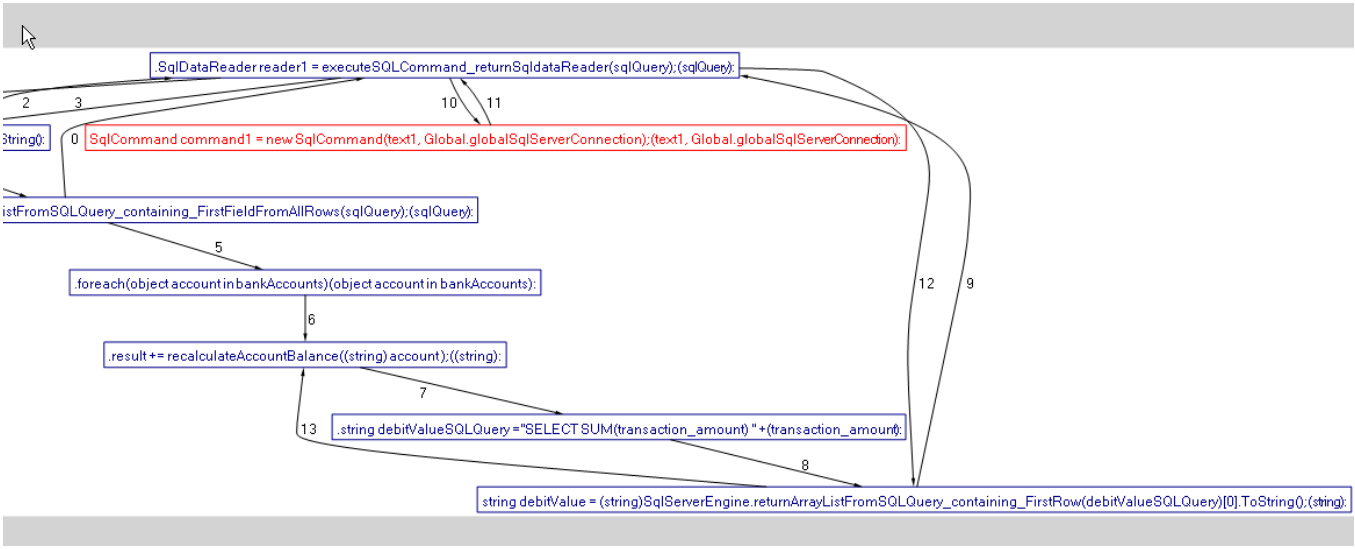


Zoom in

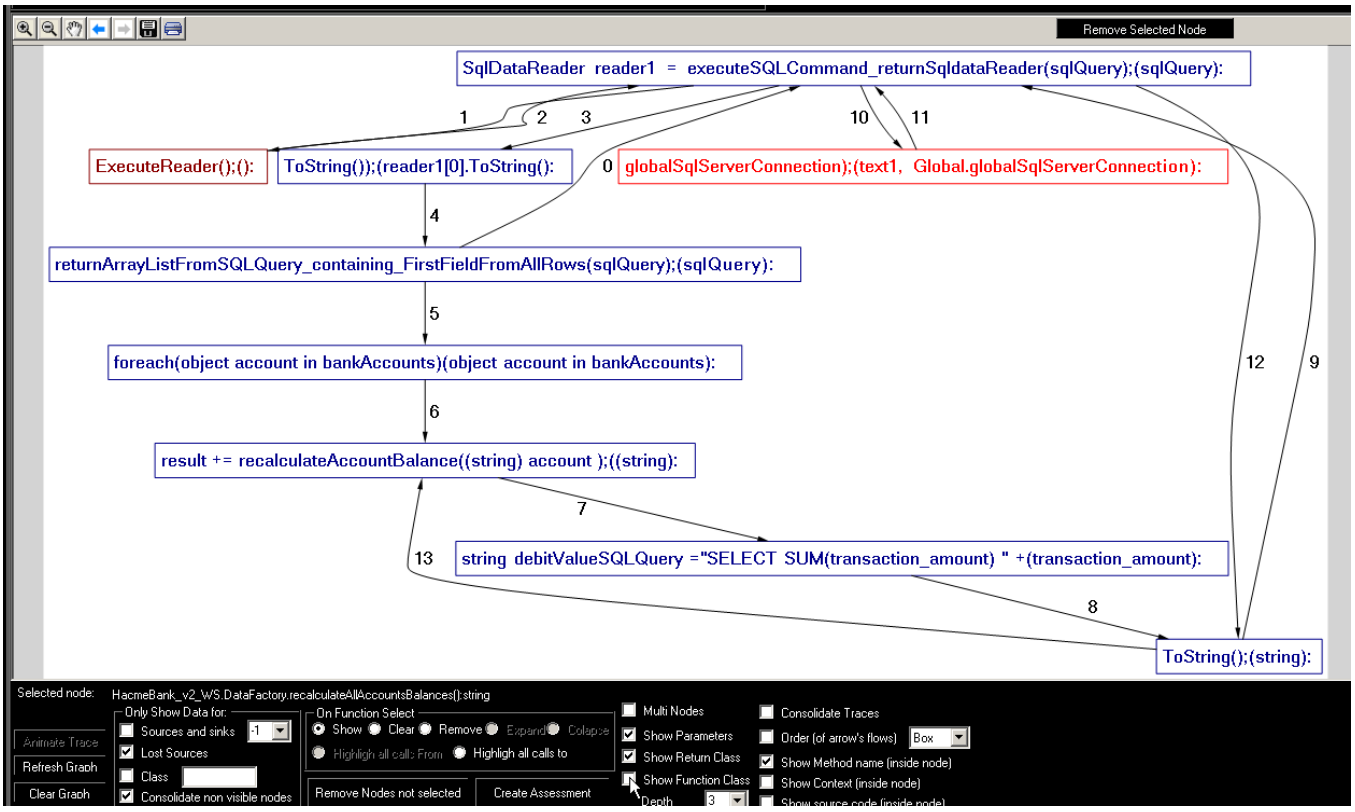
And note how with viewing the Context it becomes easier to read what is happening:

But the best view is the source code one:

and 



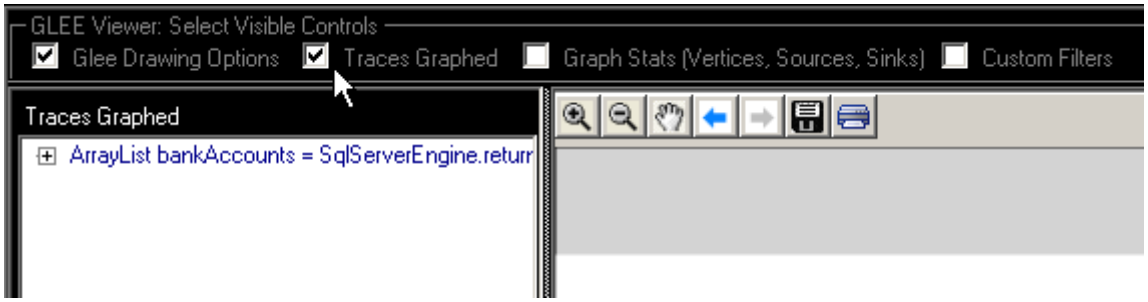
(uncheck *Show Function Class*)



This Smart trace supports viewing multiple traces at the same time.

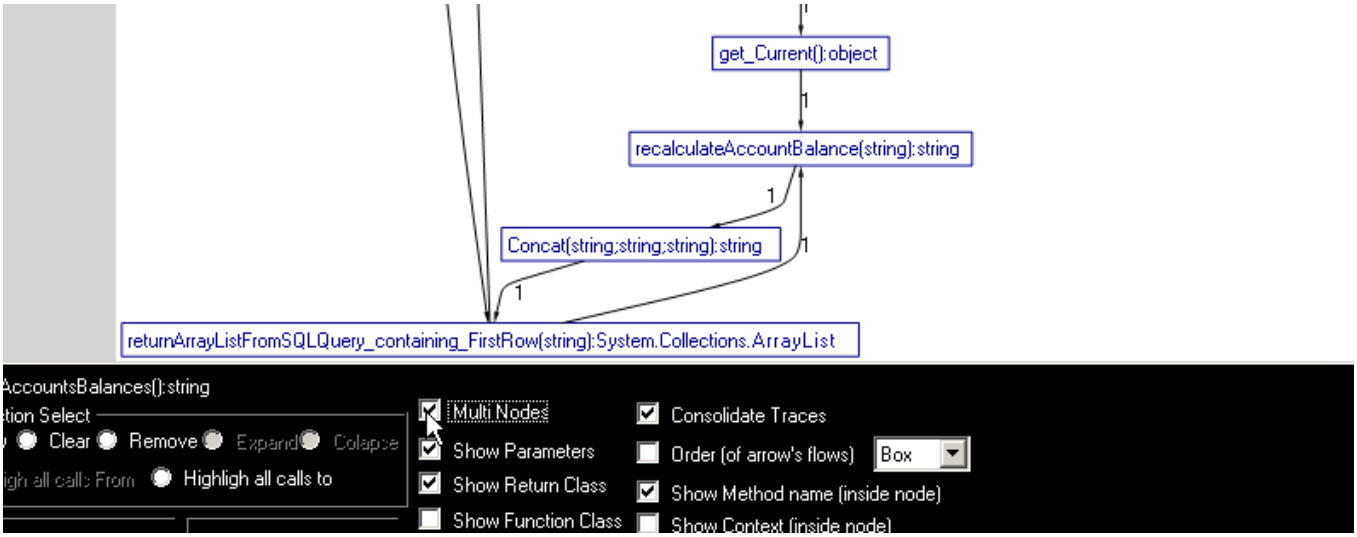
By default only one is shown. To see which on check the *Traces Graphed* checkbox

And note how the listbox only has one entry



Before adding more nodes go back to the *Method Name* view mode:

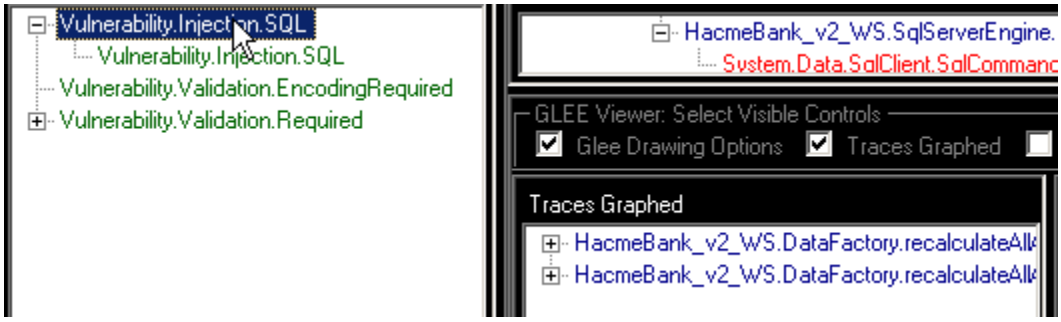
And check the Multi nodes



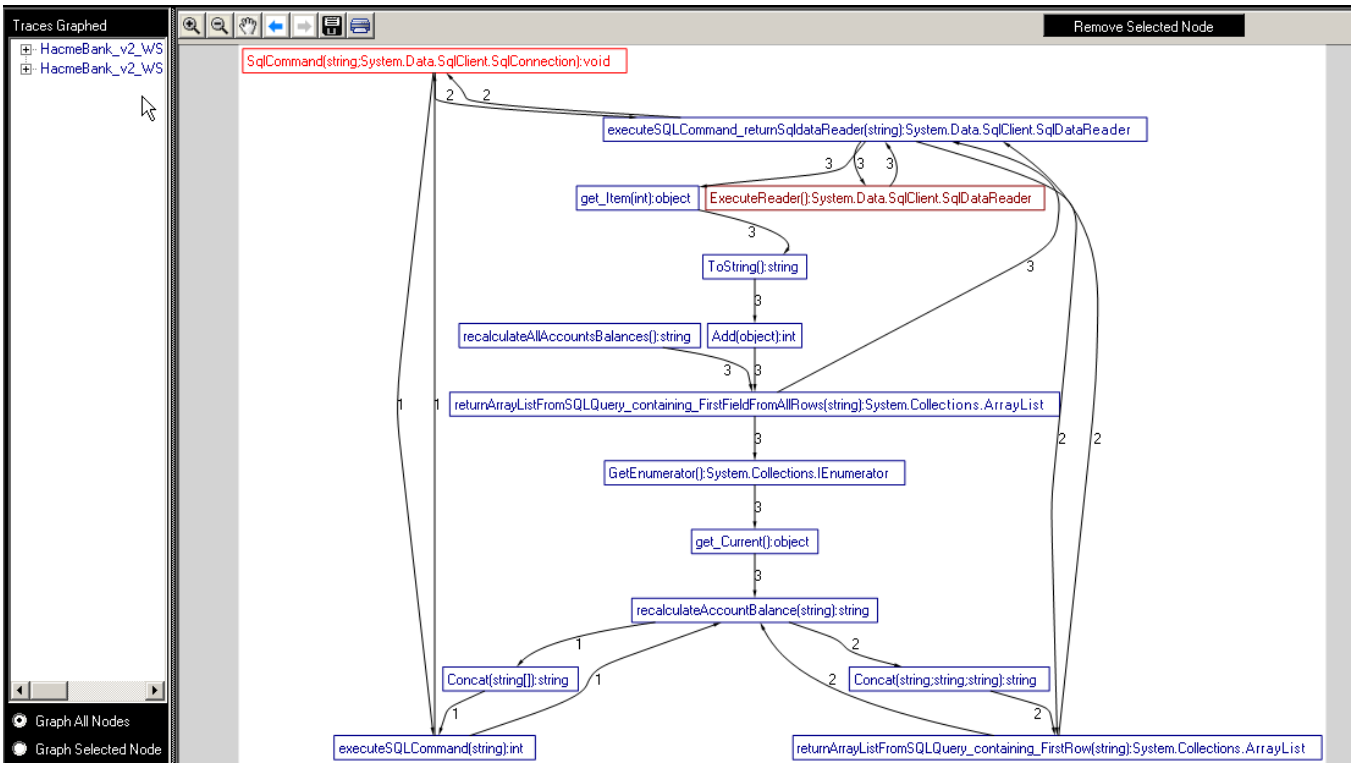
(note how all the numbers changed to 1.

The reason is because the selecting *Multi Nodes* automatically selects the *Consolidate Traces*(more details on why bellow))

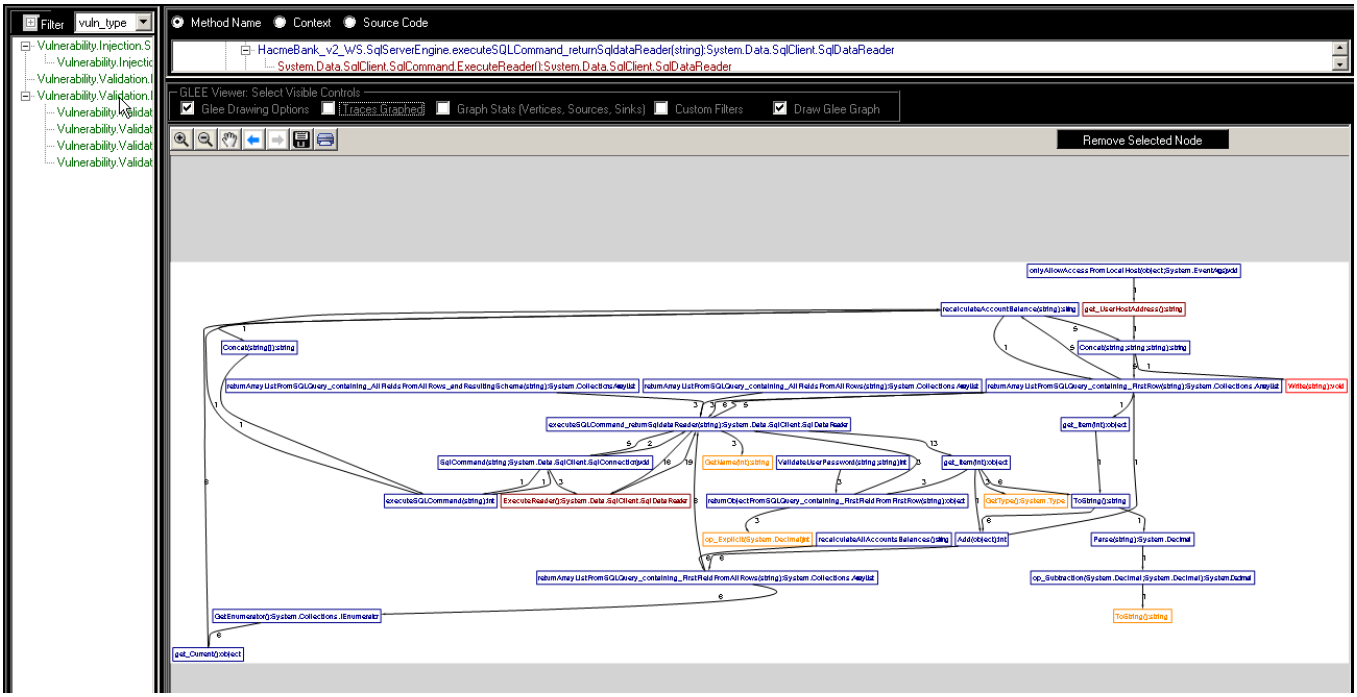
Now go back to the list of findings with traces and select the other SQL Injection trace:



And note how the GLEE viewer is now showing both traces:



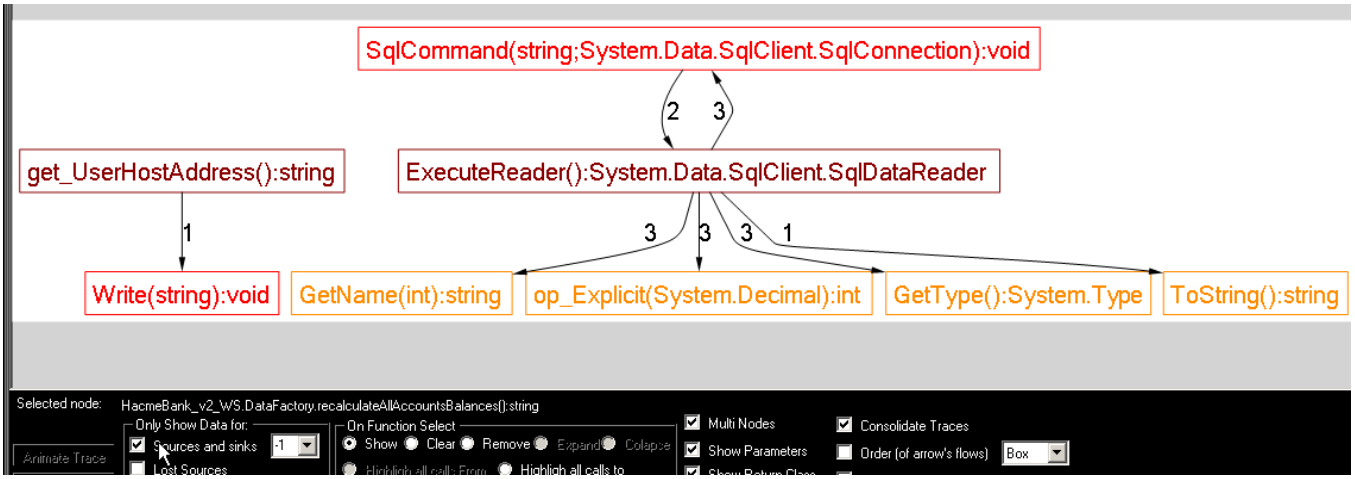
Click on the other traces so draw them all (when there are sub notes, all will be added when selecting the root node)



Note that is this graph we now have all sources sinks and lost sinks (the reason why there are only 4 lost sinks is because ExecuteReader is both a source and Lost Sink)

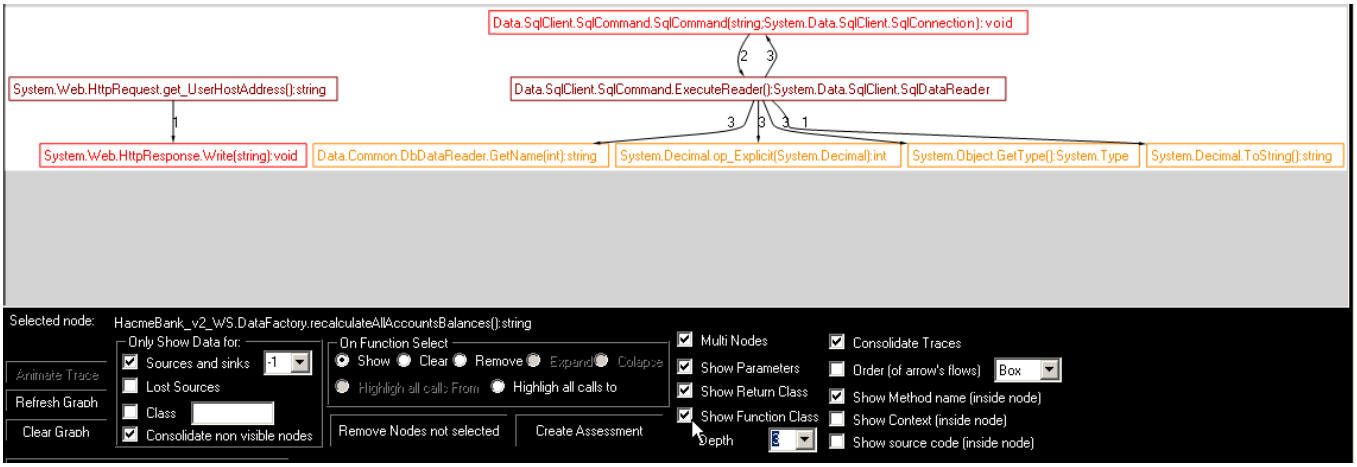
Now to make this graph more readable, select the 

and the resulting graph is much more interesting:

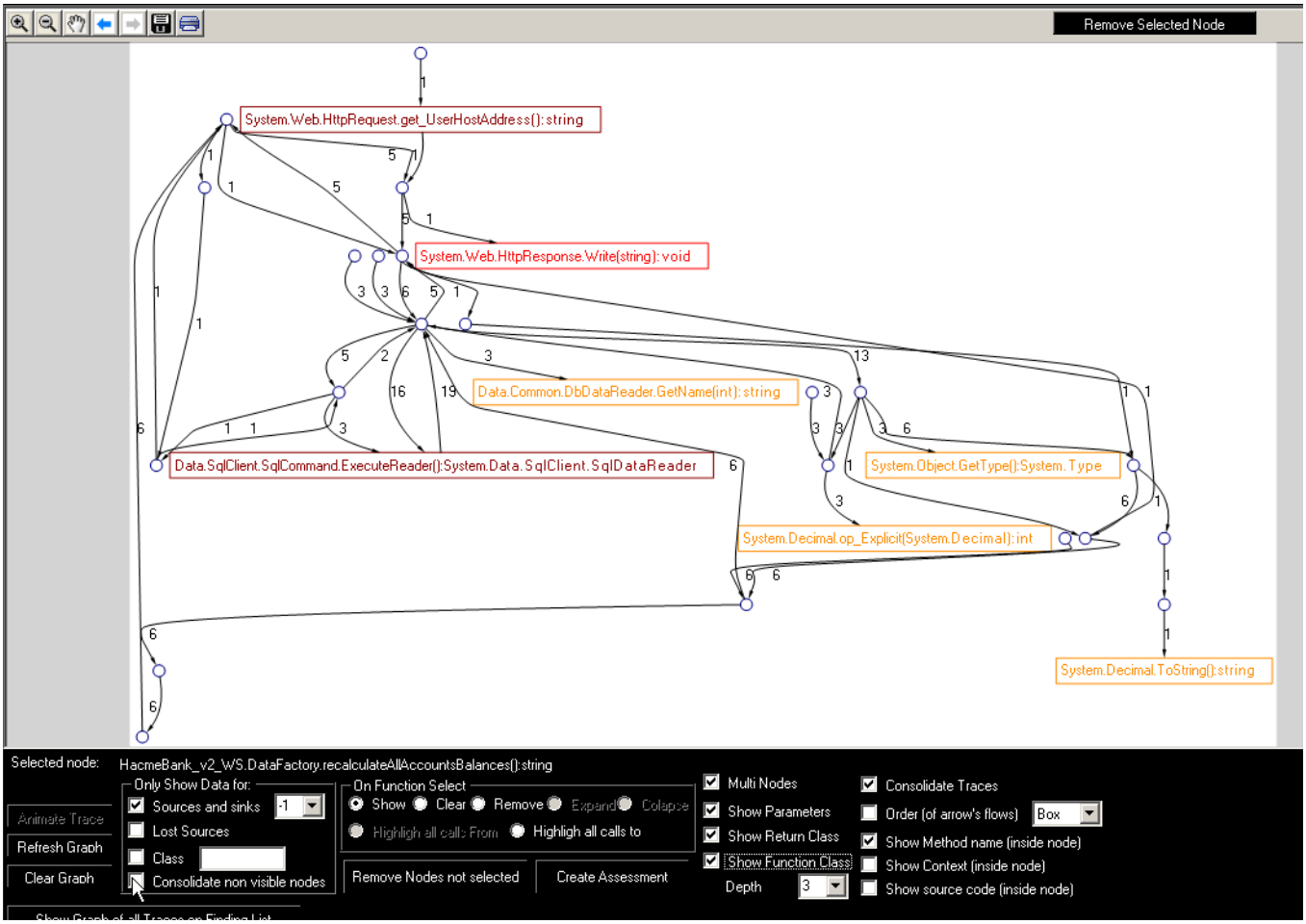


This is a graph that only shows the sources and sinks (the numbers in the lines are the number of cases

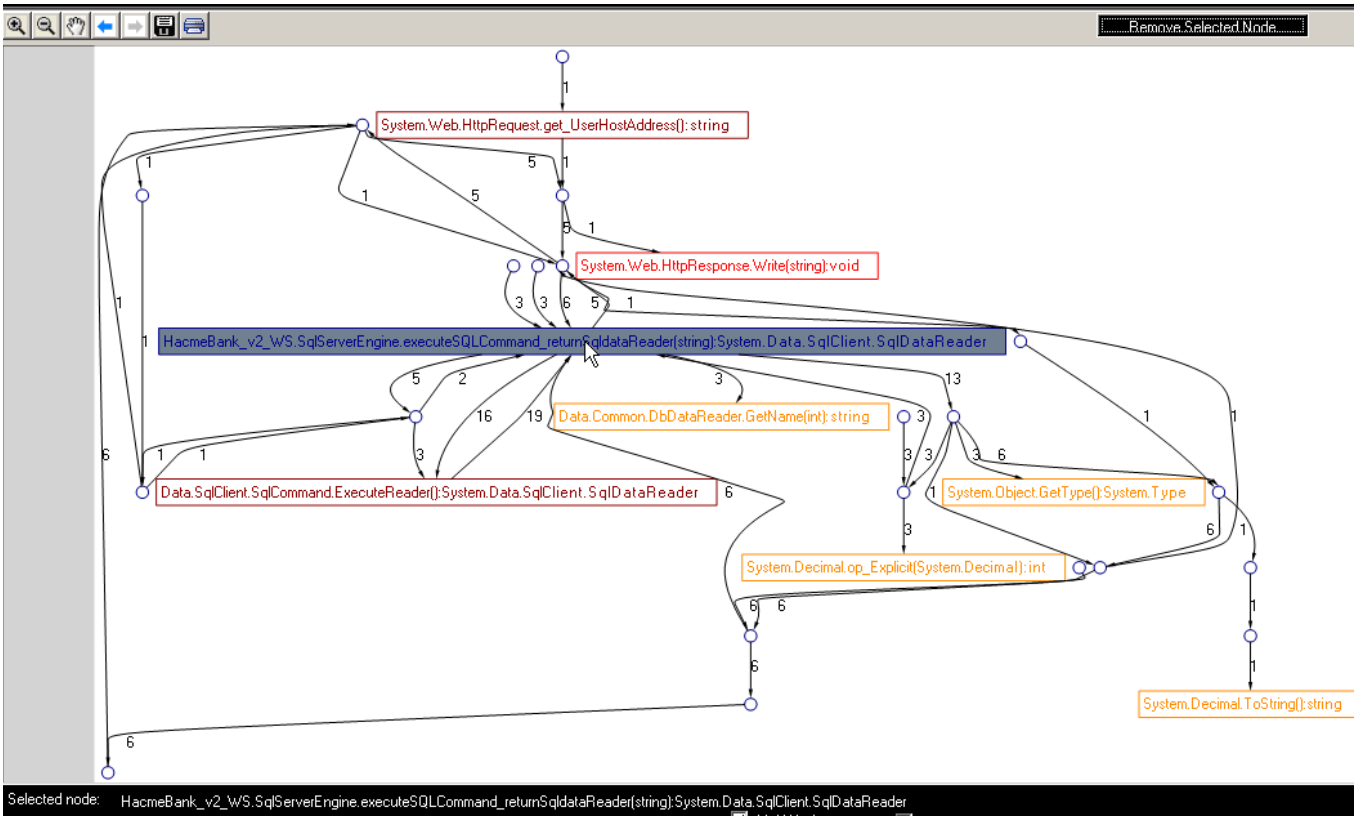
with that particular source->target method)



To get a fell how many methods exist between source and sink uncheck *Consolidate non visible nodes*



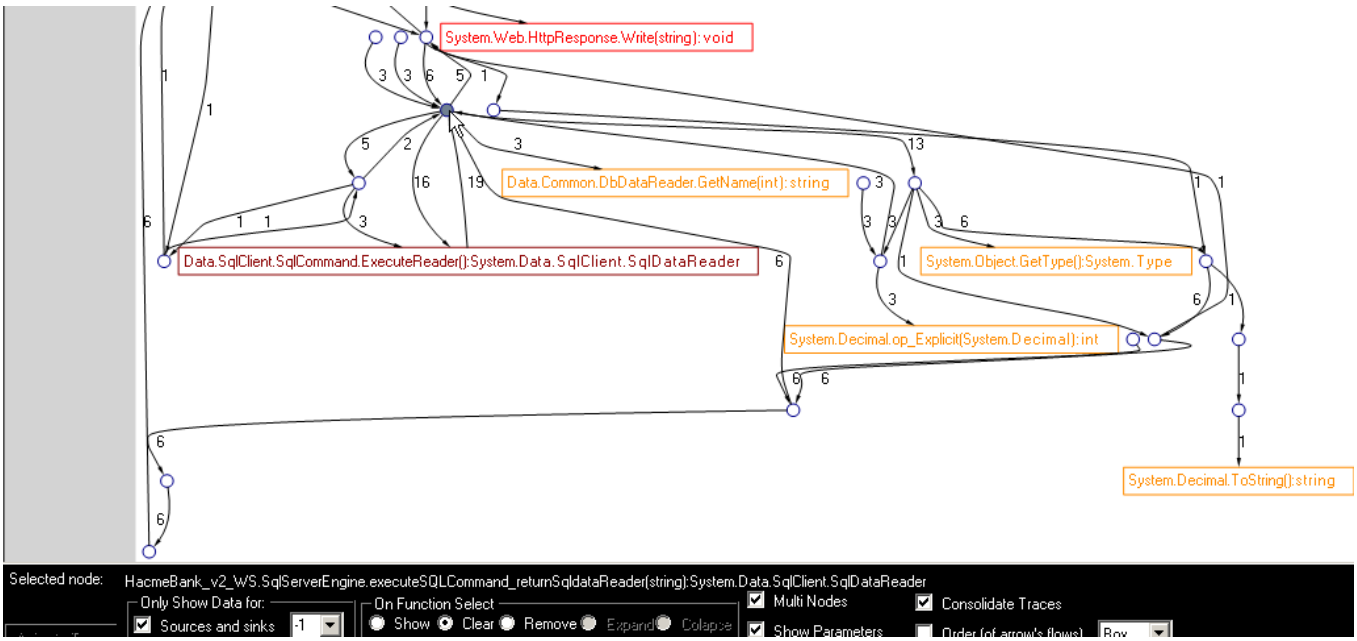
To see what a particular node is, just click on the small circle



The behaviour of the node-left-click-event is controlled by the following settings:

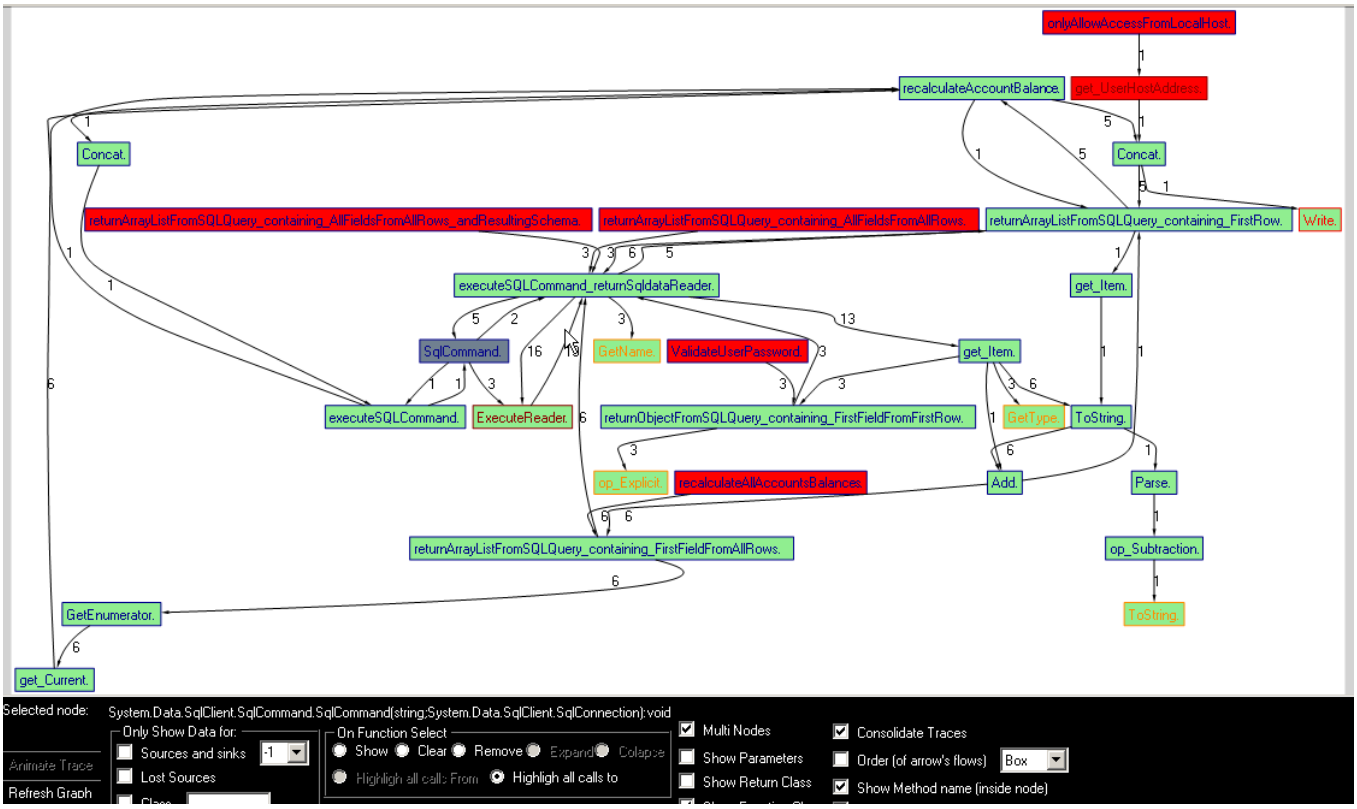
In the case above the default *Show* option was used.

The *Clear* option is set to the Right-click and will colapse the node:



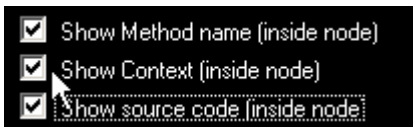
Selecting  allows to remove nodes with left-click

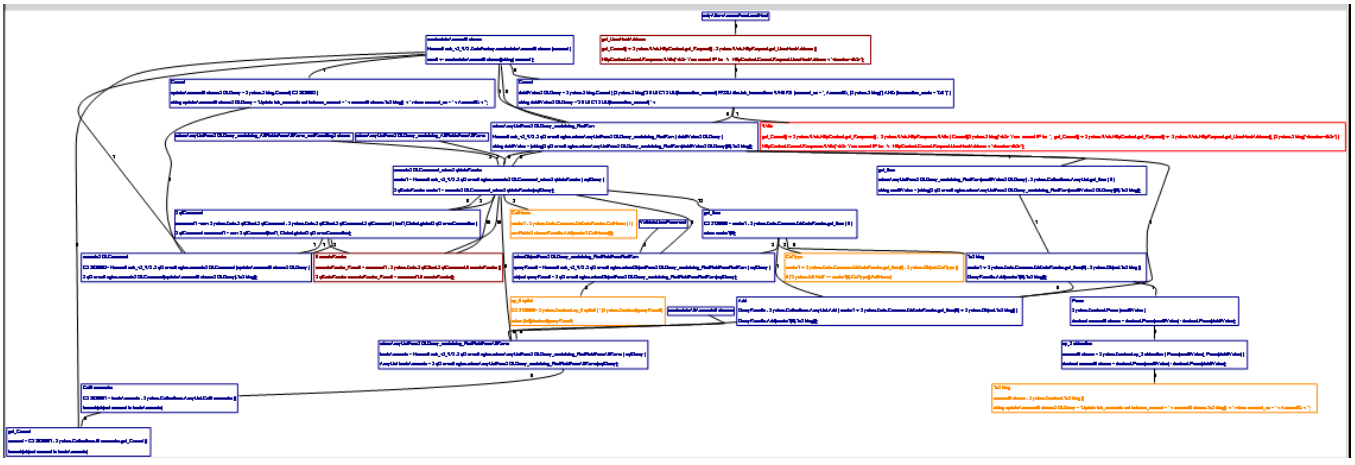
 will colour code calls TO the selected method (in red) and calls FROM the selected method (in green).



(although this example is not the best one (it works best when there are different types of vulnerabilities shown))

It is also possible to view the source code and context inside each box:



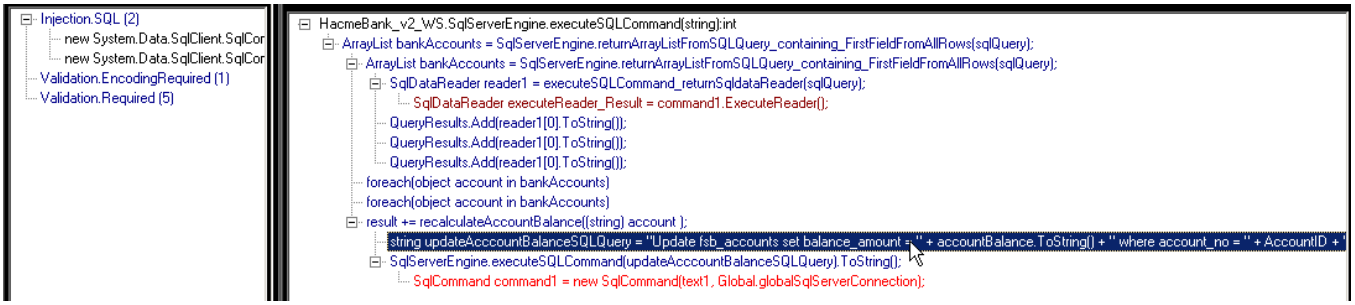


```
Web.HttpResponse.Write
get_Current()-> System.Web.HttpContext.get_Response() . System.Web.HttpResponse.Write ( Concat((System.String)"<h3> Your current IP is: ", get_Current()-> System.Web.HttpConte
HttpContext.Current.Response.Write("<h3> Your current IP is: "+ HttpContext.Current.Request.UserHostAddress+ "</center></h3>");
```

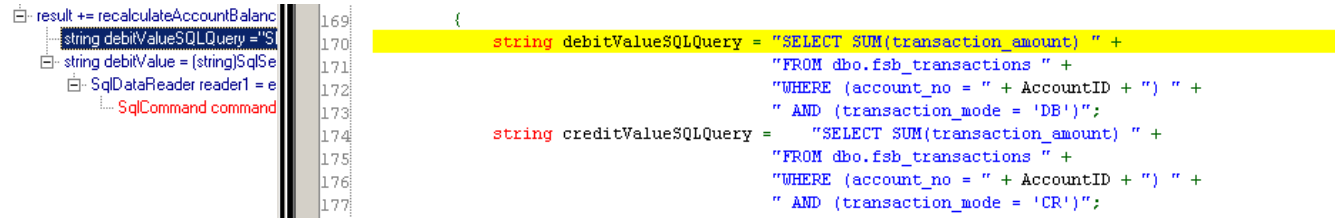
To go back to the normal GLEE Viewing mode uncheck 

SQL INJECTION CASES

A quick look at the two traces with SQL Injection show that they are only exploitable if the attacker is able to control the account_id value:



…..



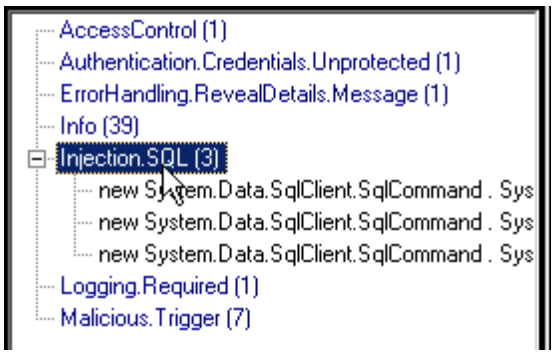
So unless we find another vulnerability that allows the injection of payloads on the AccountID value this will not be exploitable.

Since we don't have more traces let's look at the findings with no traces.

Select again the *Stats and Filters*

and in the left-hand options, select the *No Traces* radio button

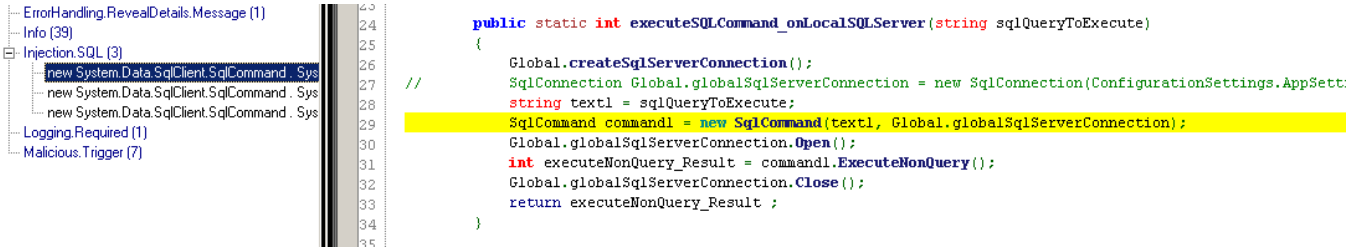
The original set of findings had 3 SQL injections with no traces:

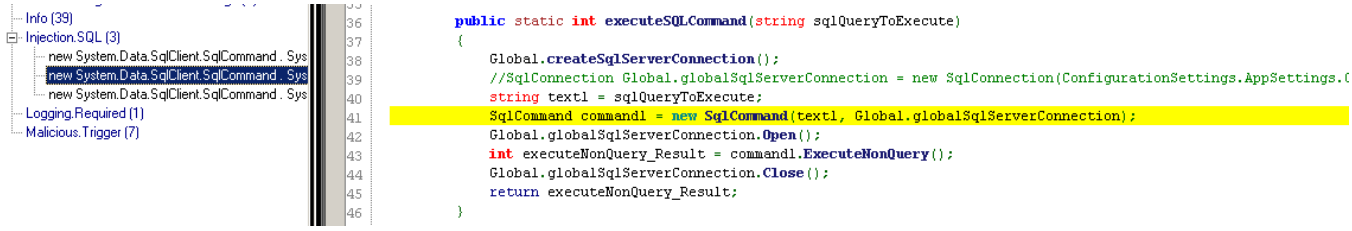


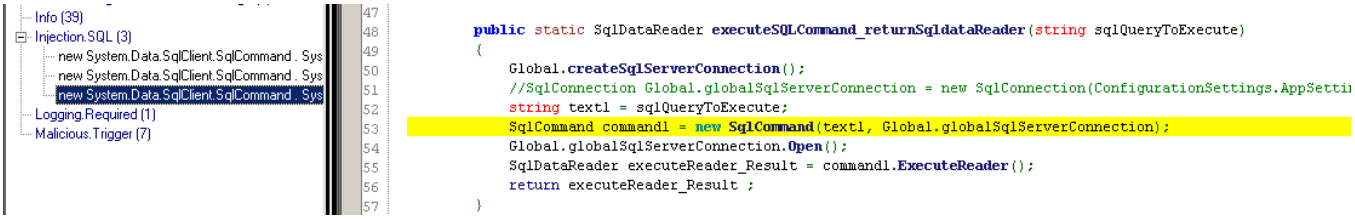
(uncheck all but the 

)

Here are the 3 findings:







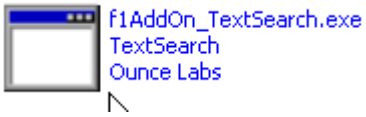
Looking at the above 3 examples one quickly see that if tainted data arrives at the following methods we have an SQL injection:

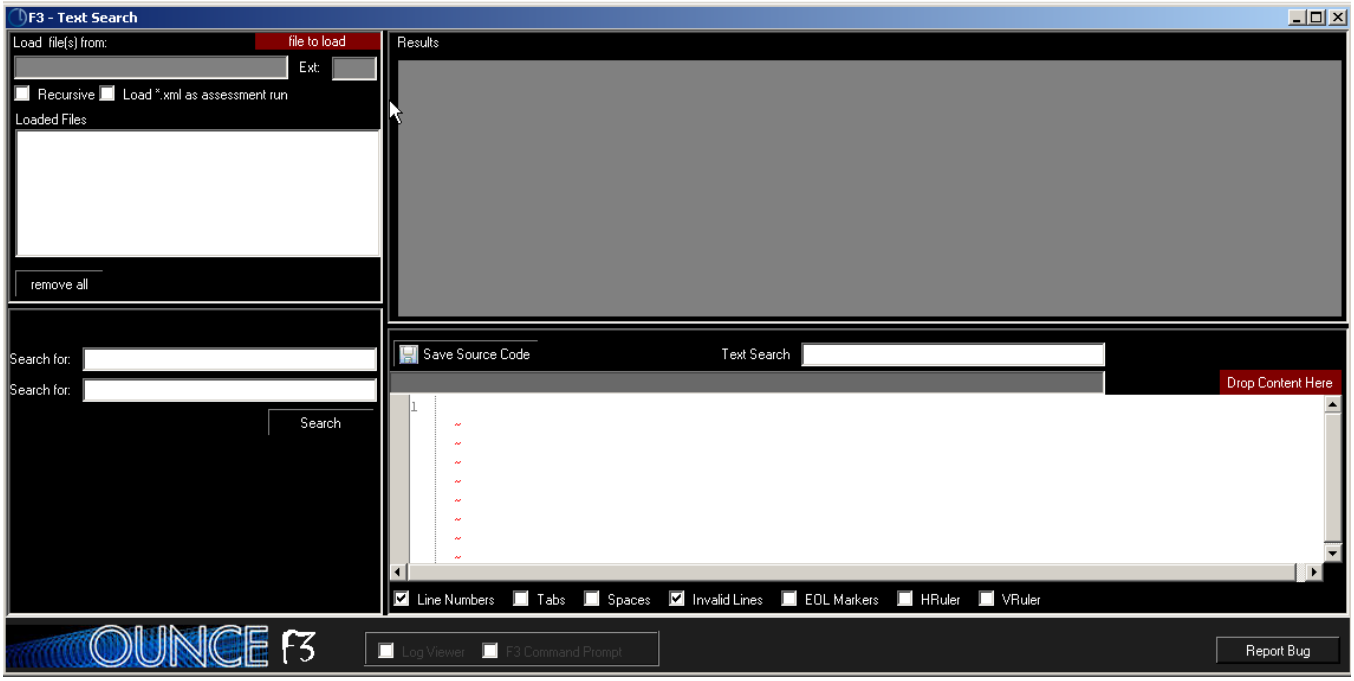
```
public static int executeSQLCommand_onLocalSQLServer(string sqlQueryToExecute)
public static SqlDataReader executeSQLCommand_returnSqldataReader(string sqlQueryToExecute)
public static int executeSQLCommand(string sqlQueryToExecute)
```

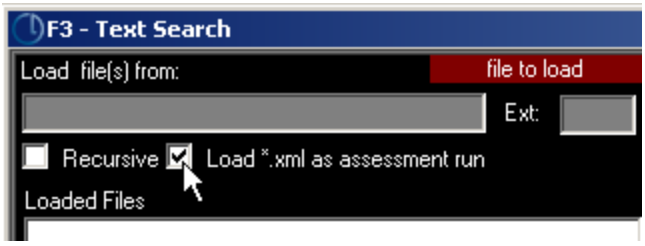
TEXT SEARCH

Using RegEx search to find out if there is a path from the exposed webservices functions to these methods

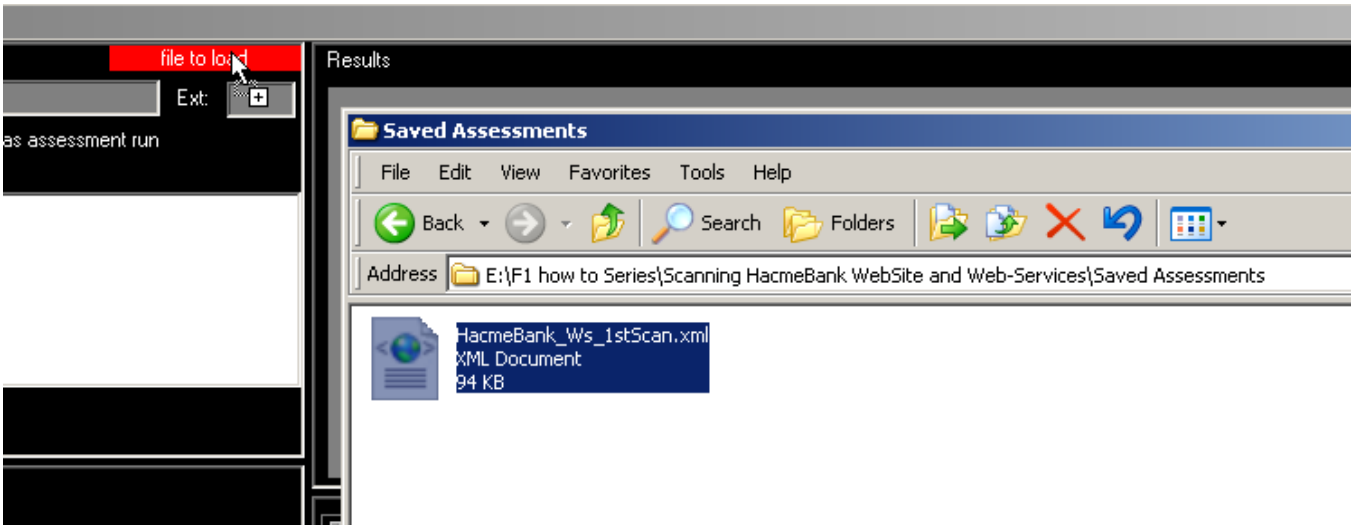
Open f1AddOn_TextSearch.exe



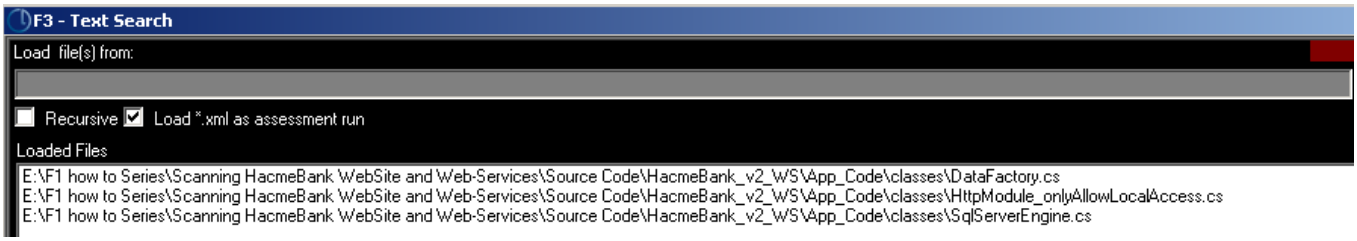


Check : 

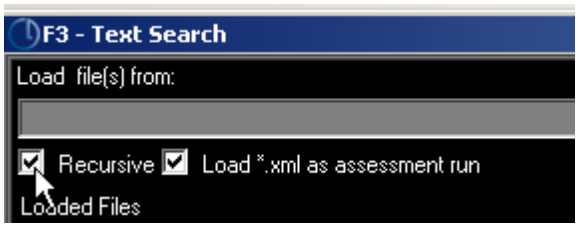
Drag and Drop the previously created Saved Assessment file



And that will load for search all files that had a finding in this project:

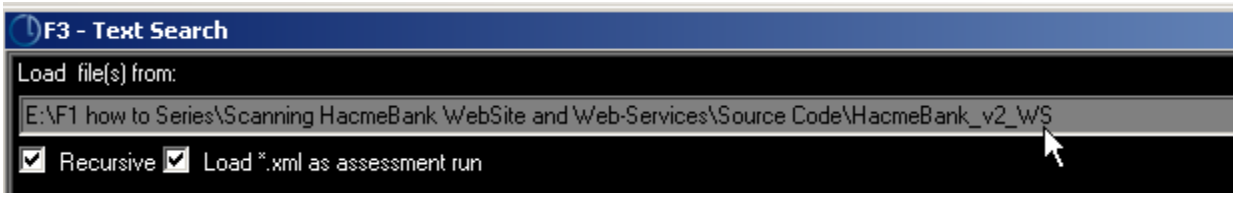


Just to make sure we have all files

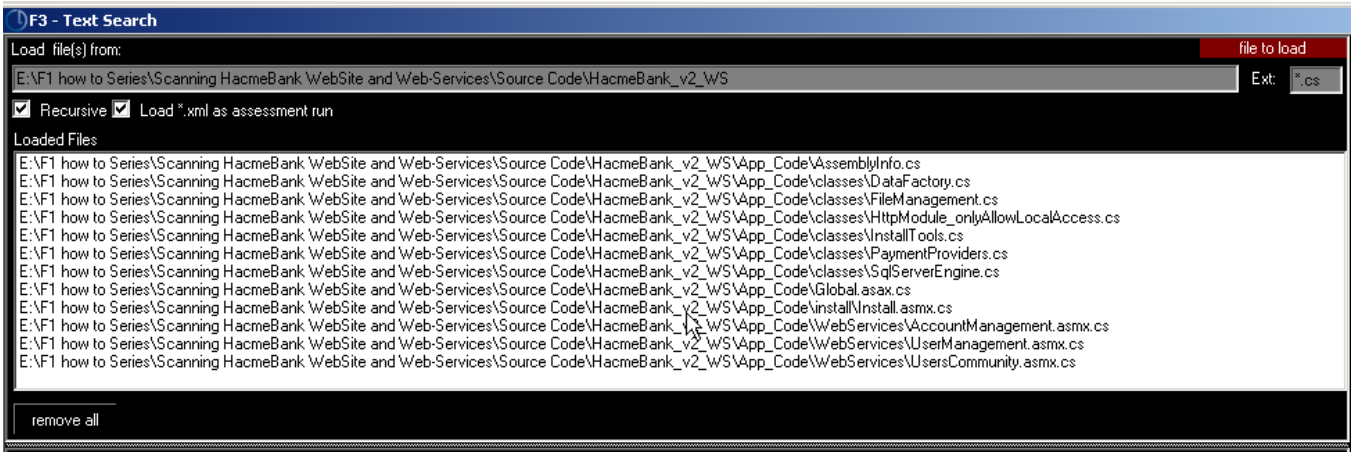
Check: 

Enter *.cs on 

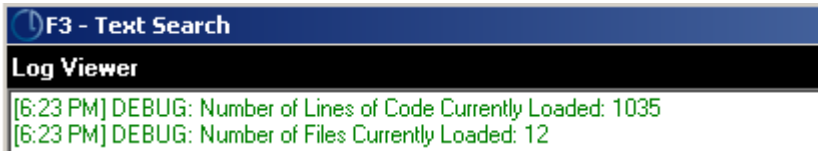
Enter the path to the source code files:



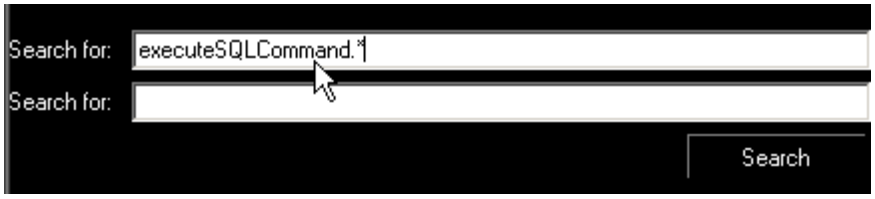
Press enter to load all *.cs from this directory



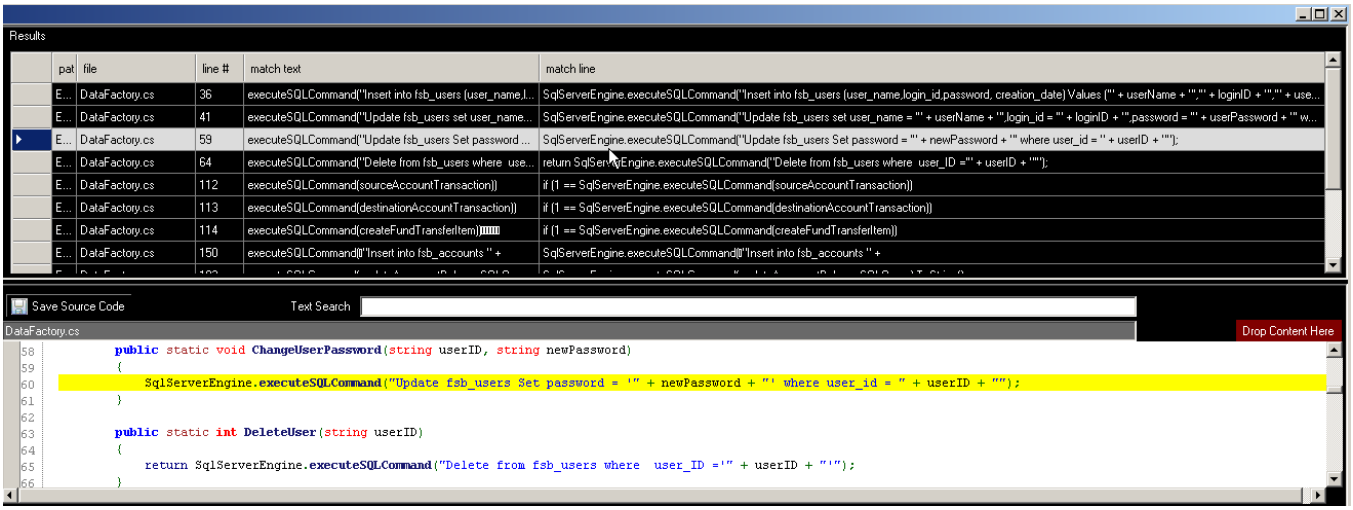
The LogView will provide details of how many files and text lines are loaded



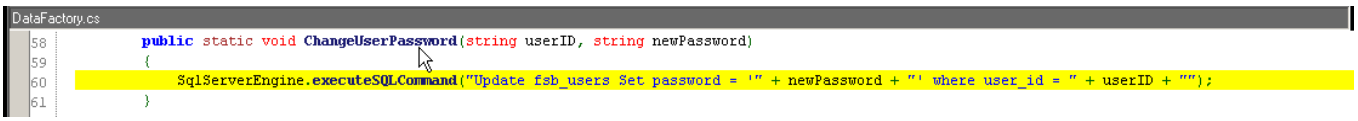
Enter *executeSQLCommand.** on the *Search for* textbox and click *Search*



And the search results will show many cases of vulnerable code:

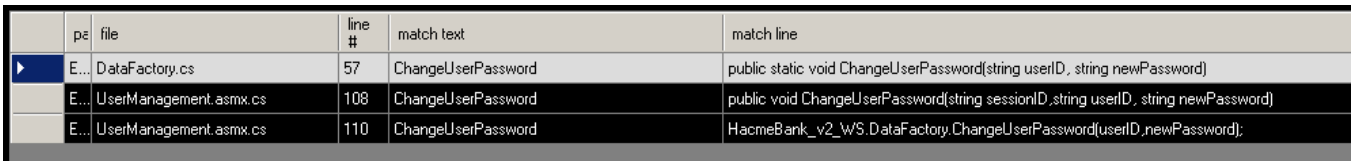


For example the *ChangeUserPassword* function of the *DataFactory.cs* file:



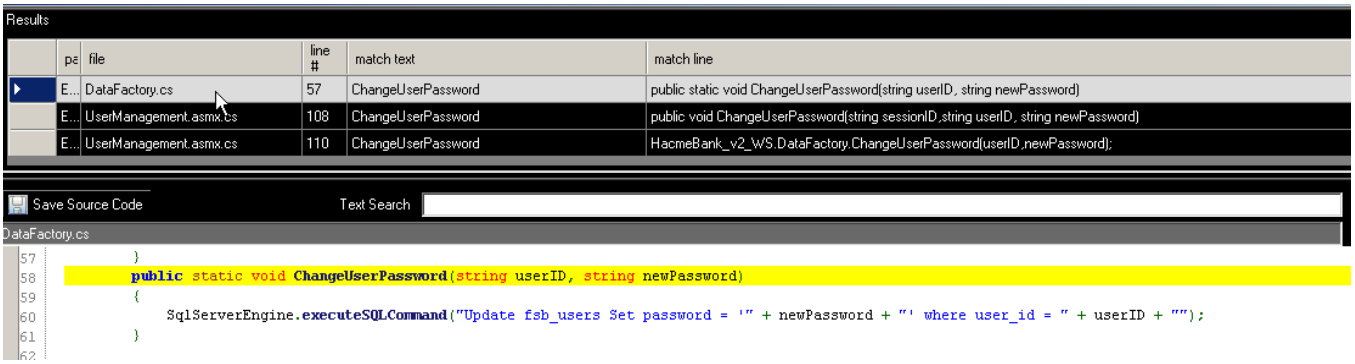
A quick search for *ChangeUserPassword*

Results in three mathes:



	pe	file	line #	match text	match line
▶	E...	DataFactory.cs	57	ChangeUserPassword	public static void ChangeUserPassword(string userID, string newPassword)
	E...	UserManagement.asmx.cs	108	ChangeUserPassword	public void ChangeUserPassword(string sessionID,string userID, string newPassword)
	E...	UserManagement.asmx.cs	110	ChangeUserPassword	HacmeBank_v2_WS.DataFactory.ChangeUserPassword(userID,newPassword);

The first one is the ChangeUserPassword (which we've seen before)



Results

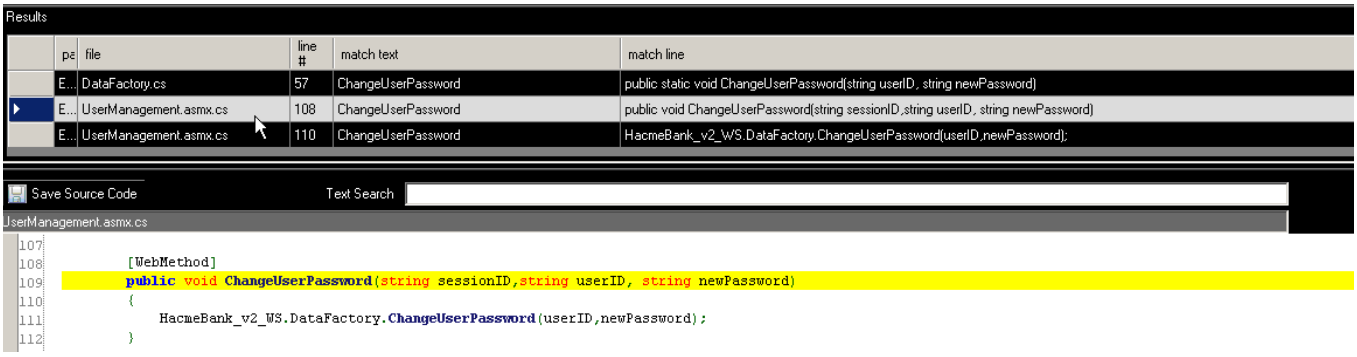
	pe	file	line #	match text	match line
▶	E...	DataFactory.cs	57	ChangeUserPassword	public static void ChangeUserPassword(string userID, string newPassword)
	E...	UserManagement.asmx.cs	108	ChangeUserPassword	public void ChangeUserPassword(string sessionID,string userID, string newPassword)
	E...	UserManagement.asmx.cs	110	ChangeUserPassword	HacmeBank_v2_WS.DataFactory.ChangeUserPassword(userID,newPassword);

Save Source Code Text Search

DataFactory.cs

```
57          }
58          public static void ChangeUserPassword(string userID, string newPassword)
59          {
60              SqlServerEngine.executeSQLCommand("Update fsb_users Set password = '" + newPassword + "' where user_id = " + userID + "");
61          }
```

The 2nd is a WebMethod call



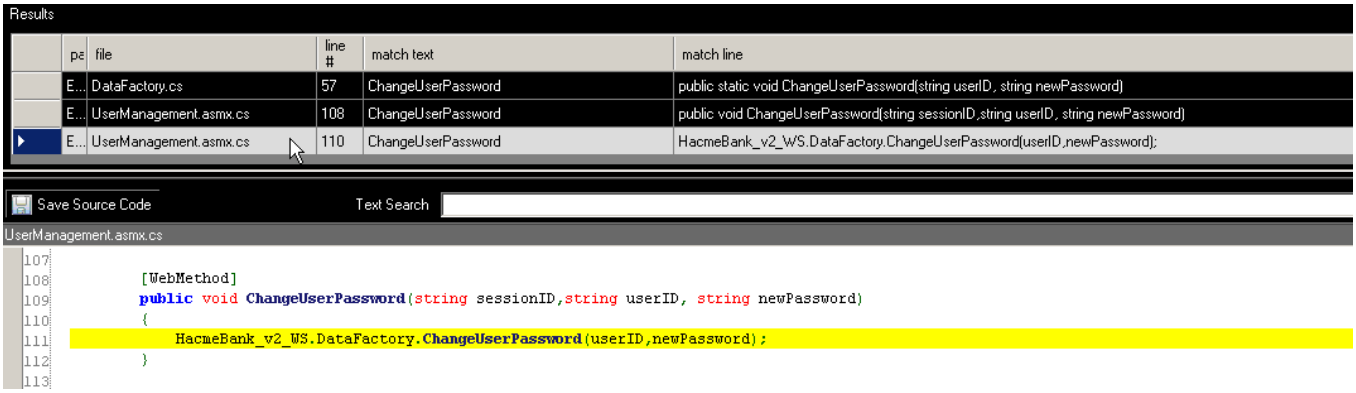
Results

	pe	file	line #	match text	match line
	E...	DataFactory.cs	57	ChangeUserPassword	public static void ChangeUserPassword(string userID, string newPassword)
▶	E...	UserManagement.asmx.cs	108	ChangeUserPassword	public void ChangeUserPassword(string sessionID,string userID, string newPassword)
	E...	UserManagement.asmx.cs	110	ChangeUserPassword	HacmeBank_v2_WS.DataFactory.ChangeUserPassword(userID,newPassword);

Save Source Code Text Search

UserManagement.asmx.cs

```
107
108         [WebMethod]
109         public void ChangeUserPassword(string sessionID,string userID, string newPassword)
110         {
111             HacmeBank_v2_WS.DataFactory.ChangeUserPassword(userID,newPassword);
112         }
```



So clearly this is exploitable since there is a direct path from the externally faced WebMethod to the Sql Injection entry point.:

This should be an easy issue for Ounce to detect. The problem is that at the moment the Ounce AE doesn't recognize the WebMethod as a location where Tainted data will originate. In Ounce's nomenclature this is called a **Callback** (a good example of a callback is the *main(String[] args)* where the AE threats all parameters from that methods as tainted).

In 6.0 there is support for callbacks from the GUI, but in 5.0x we need to do it differently.

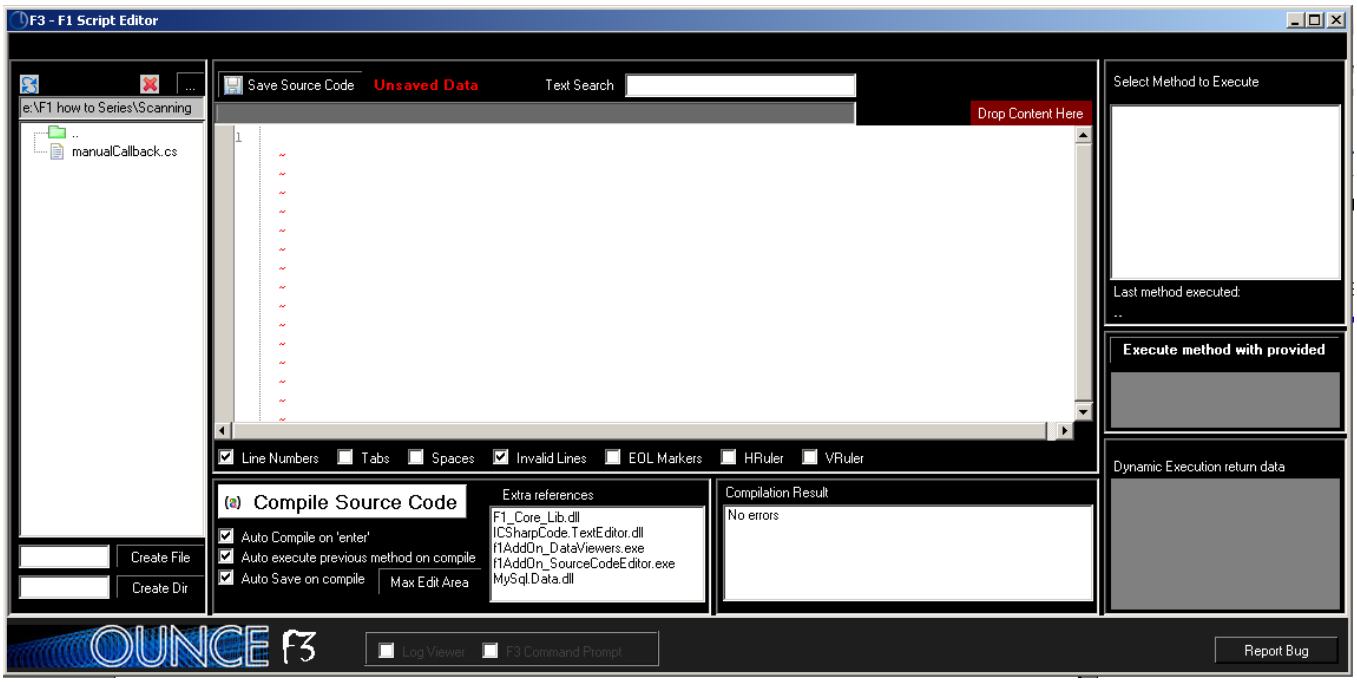
Using F1, there are 4 ways to create a Callback:

1) If there is a trace that covers the target method, we can use the *Custom Rules* Add-On (in this case there isn't one, so the *Custom Rules* Add-on will covered in another *F1 how-to* document)
2) Once a consolidated F1CirData (from Cir Dumps) file is created we can use it with the *Custom Rules* Add-On (the problem is that the CIR doesn't contain the attributes data (i.e the [WebMethod]) so in this case we would still need to manually selected which methods were WebMethods and should be marked as Callbacks (this will be covered in another *F1 how-to)*
3) Call the exposed F1 API that creates the Callbacks (see below)
4) Use the F1 Add-On *dotNet_CallbacksMaker_5_03* (see below)

USING F1 API TO CREATE A CALLBACK

F1 exposes most of its internal functions for invocation and scripting inside F1. In this first example we will invoke directly the function that creates the rules callbacks

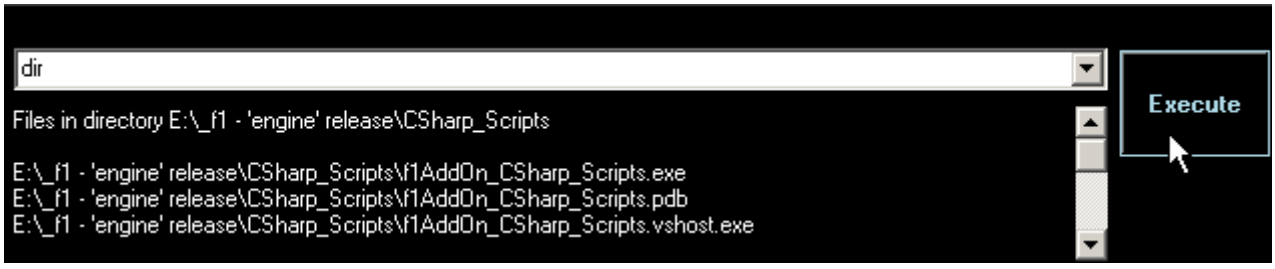
Start the *f1AddOn_CSharp_Scripts.exe*



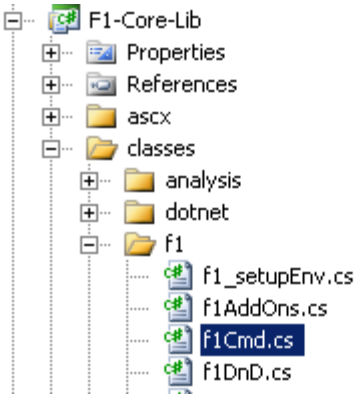
Before we get into the script example let's confirm that the desired API function is there.
Open the F3 Command Prompt

The Command Prompt allows the direct execution of commands, with some support of Intelisense
Click on the TextBox and press space to see a list of available commands and classes:

Typing *dir* and clicking on Execute (or pressing on Enter) will show the files in the current directory



dir in this case is a public static function located inside the F1_core_lib.dll dll and inside the f1Cmd.cs class file




```
namespace ounceLabs.F1.core_lib.classes.f1
{
    [core_lib.classes.F1_ExposedAndInvokableByF1]
    public class f1Cmd
    {
```

.....

```
        public static String dir()
        {
            List<String> lsFiles = new List<string>();
            files.getListOfAllFilesFromDirectory(lsFiles,sCurrentf1CmdWorkDirectory,false, "*.*",false);
            StringBuilder sbResult = new StringBuilder();
            sbResult.AppendLine("Files in directory " + sCurrentf1CmdWorkDirectory);
            sbResult.AppendLine();
            foreach (string sFile in lsFiles)
                sbResult.AppendLine(sFile);
            return sbResult.ToString();
        }
```

EXAMPLE OF USING THE COMMAND PROMPT

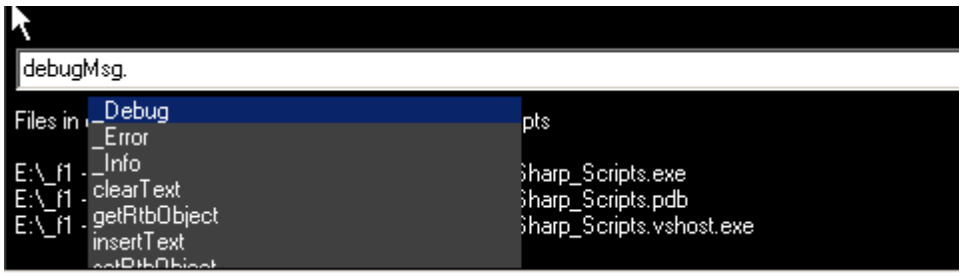
A good example of interaction between the *Command Prompt* and F1 is using debugMsg class to send a message to the LogViewer:

Make the LogViewer Visible: 

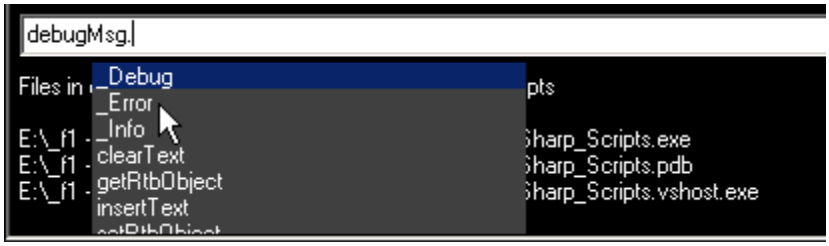
And type the letters *de* on the *CommantPrompt* TextBox

(note how the intelisense changes as the letters are presed)

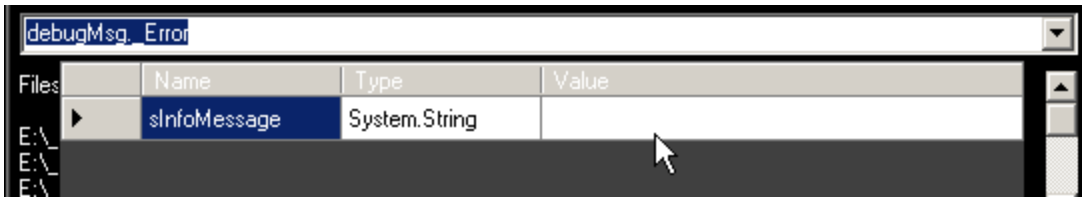
To select the current highlighted choice press on ‘Left Arrow’



Or use the mouse to manually select from the ListBox the desired method:

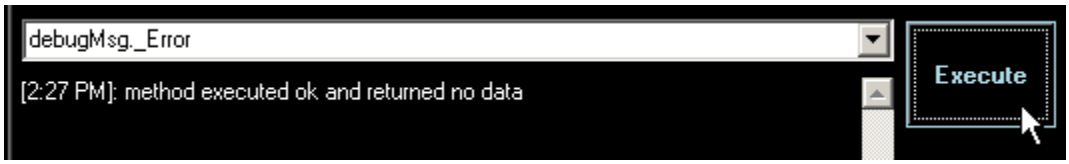


Once a function is selected, a DataGridView will be dynamically populated with placeholders for the required parameters:



In this case just click on the *Value* cell and enter the message to send:

Click *execute*:



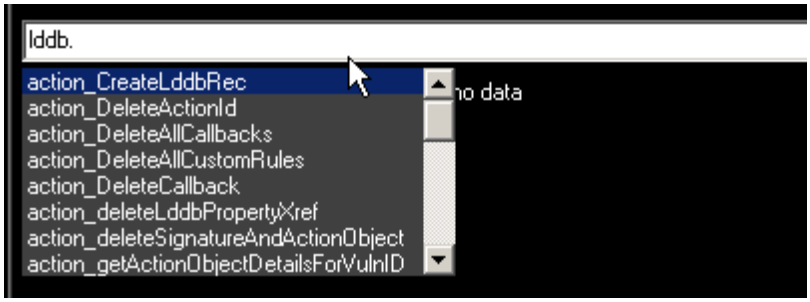
And note on the LogViewer the new entry:

Note: all past commands are added to a local cache which is visible by opening up the dropdown list:

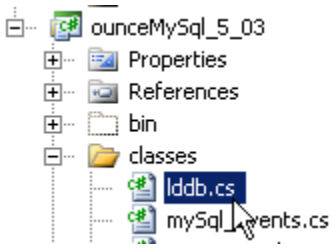
And accessed using the up and down arrows

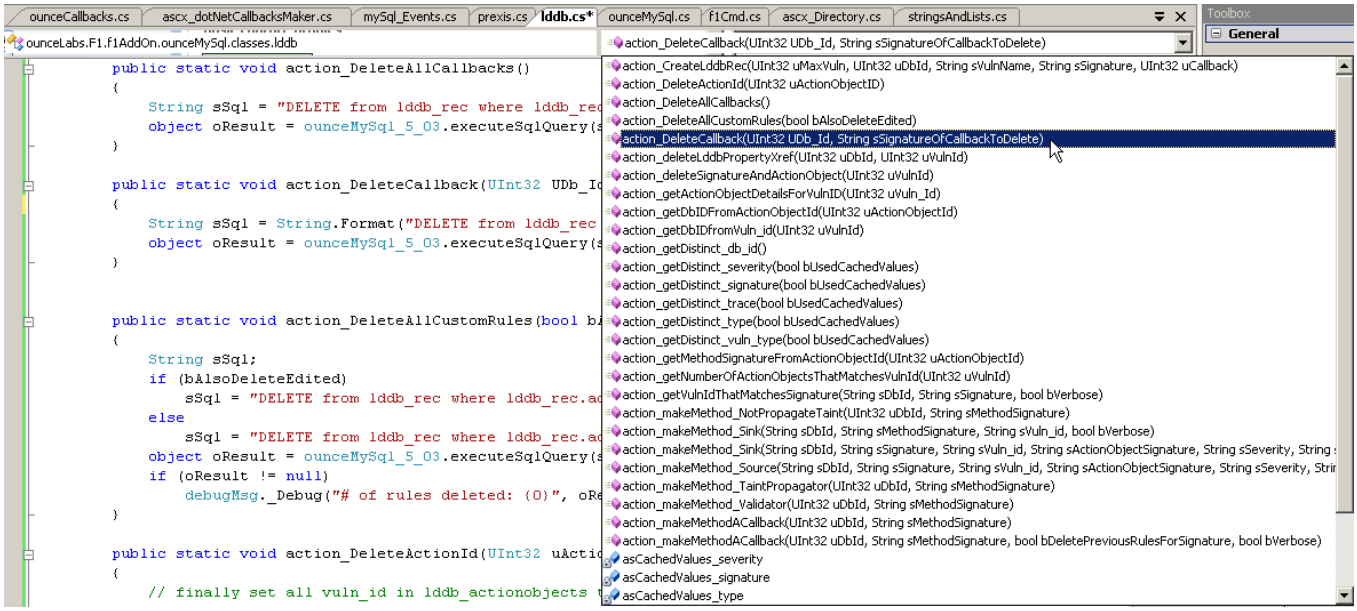
INVOKING THE CREATE CALLBACK FROM THE *COMMAND PROMPT*

In the *Command Prompt* enter lddb.

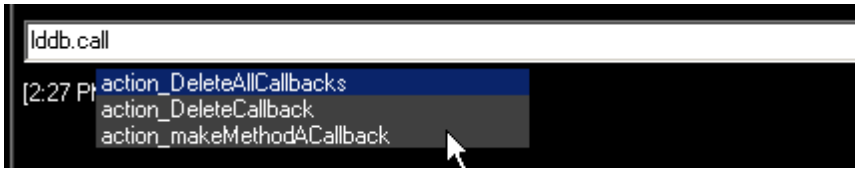


The list shown are all exposed methods from the lddb class which is part of the *ounceMySql_5_03* dll

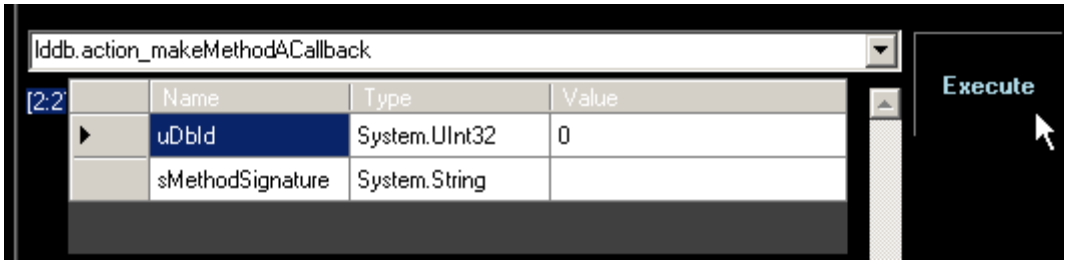




In *Command Prompt* enter *call* and select *action_makeMethodACallback*



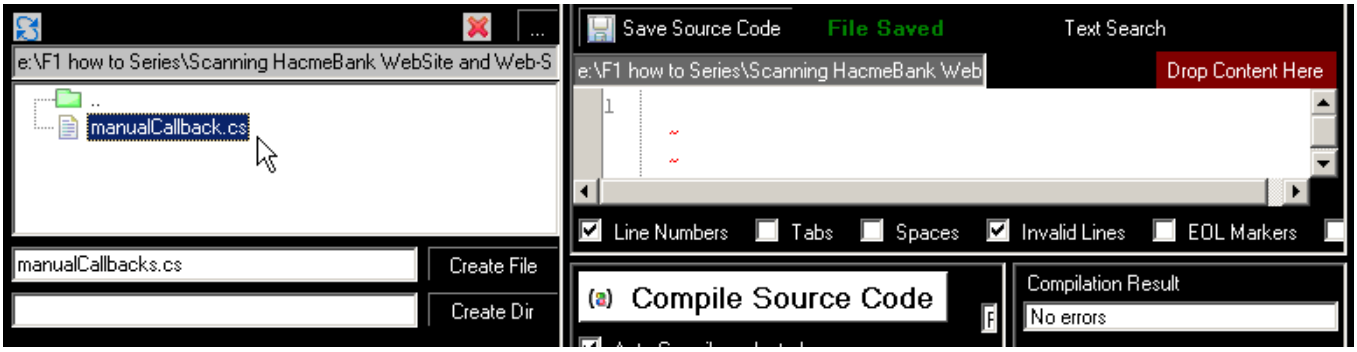
And if the Db_Id and signature are known, enter the values and click execute:



INVOKING THE CREATE CALLBACK FROM A SCRIPT

Going back to the *f1AddOn_CSharp_Scripts* creade a new file called manualCallbacks.cs

Click on the newly created file:

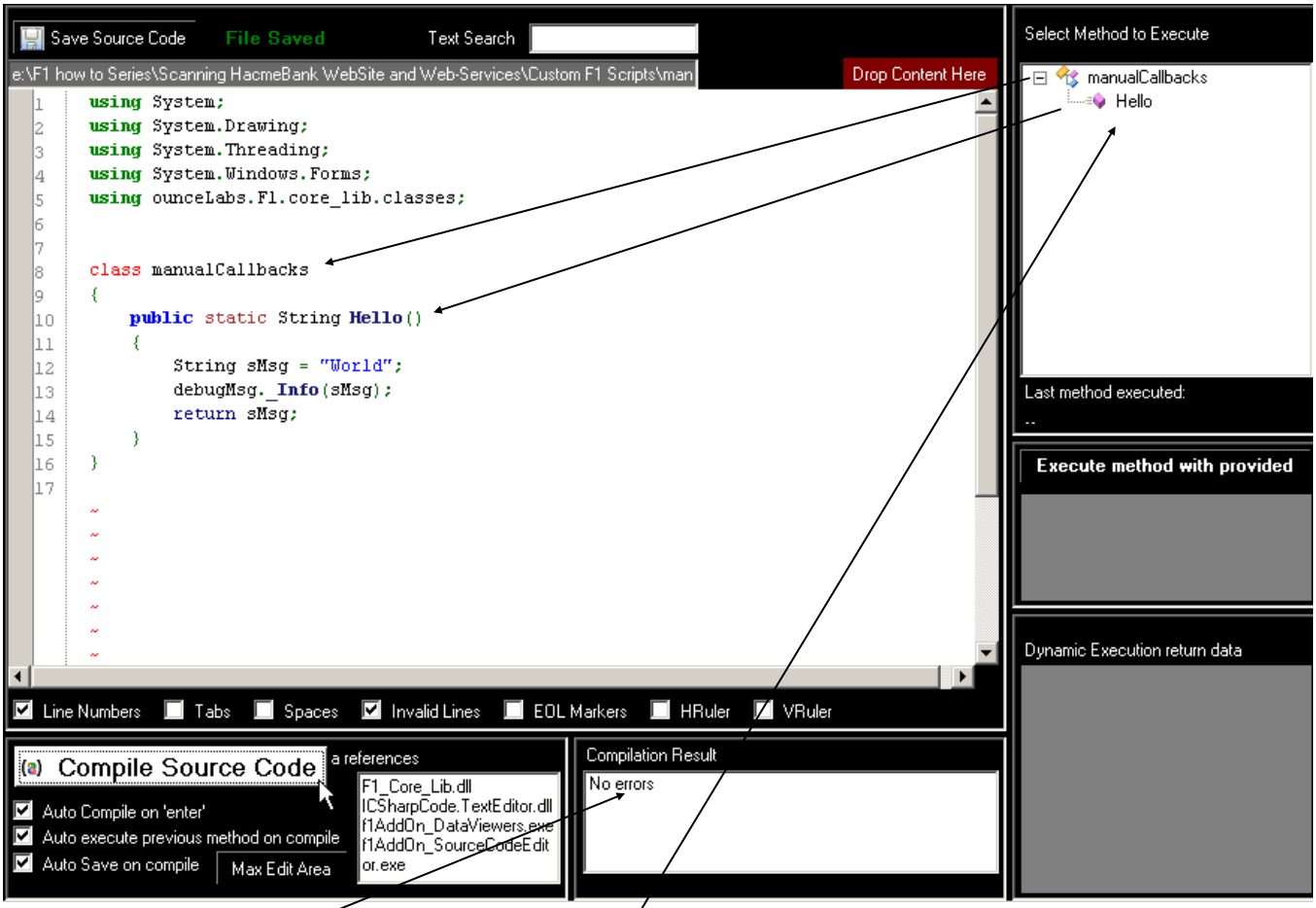


And paste the following code :

```
using System;
using System.Drawing;
using System.Threading;
using System.Windows.Forms;
using ounceLabs.F1.core_lib.classes;


class manualCallbacks
{
	public static String Hello()
	{
		String sMsg = "World";
		debugMsg._Info(sMsg);
		return sMsg;
	}
}
```

BUG ALERT: Sometimes when opening the CSharp Code editor on new files (i.e files with no content, the Editor Control will stop behaving normally. If that happens, just close the *f1AddOn_CSharp_Scripts* Add-On and try again (the 2nd time around always works ok)

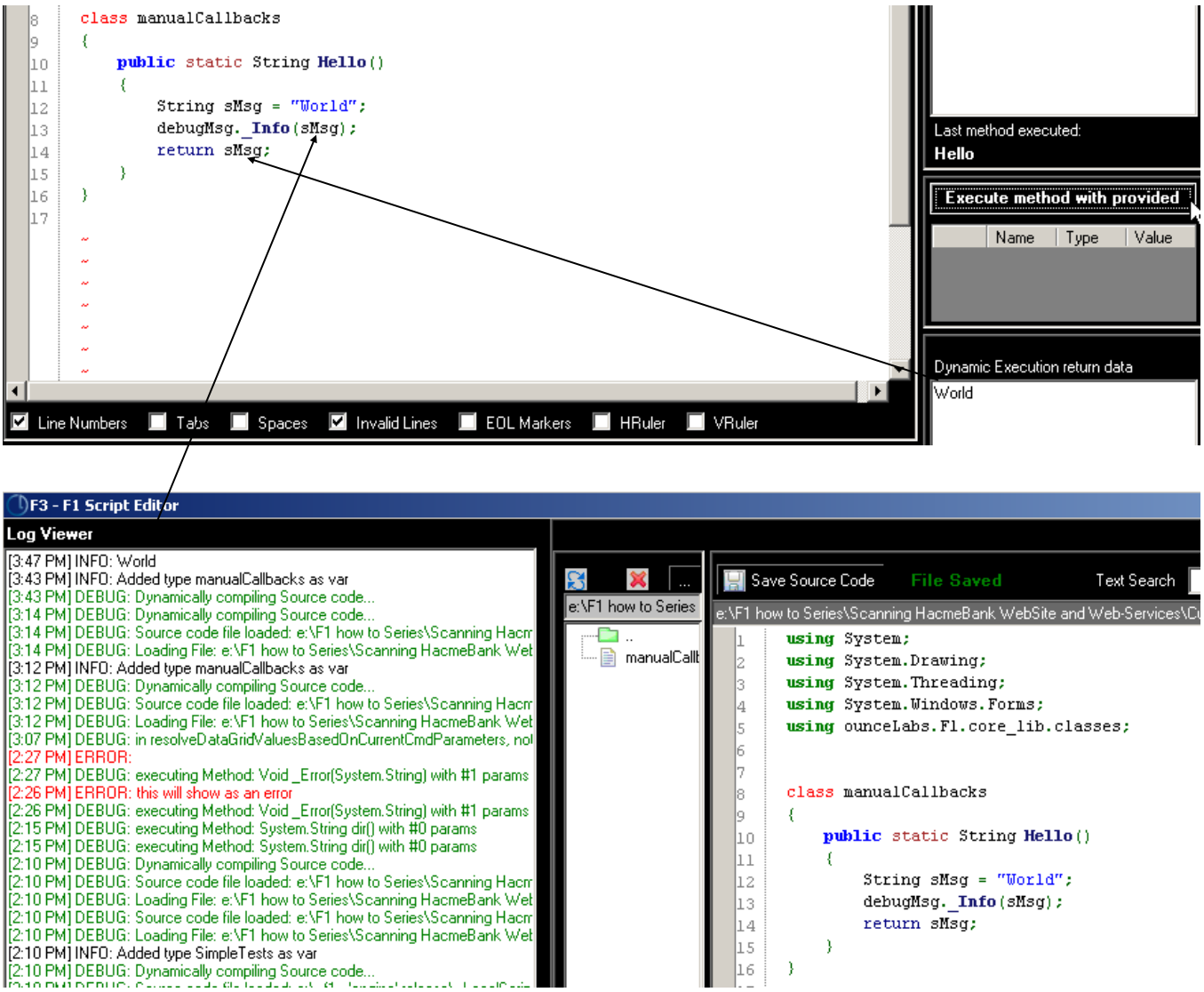
Now click on *Compile Source Code*



There should be no errors and the top right ListBox should now be populated with a dynamic list of the *manualCallbacks* class

}

Click on  and on  to execute this method:

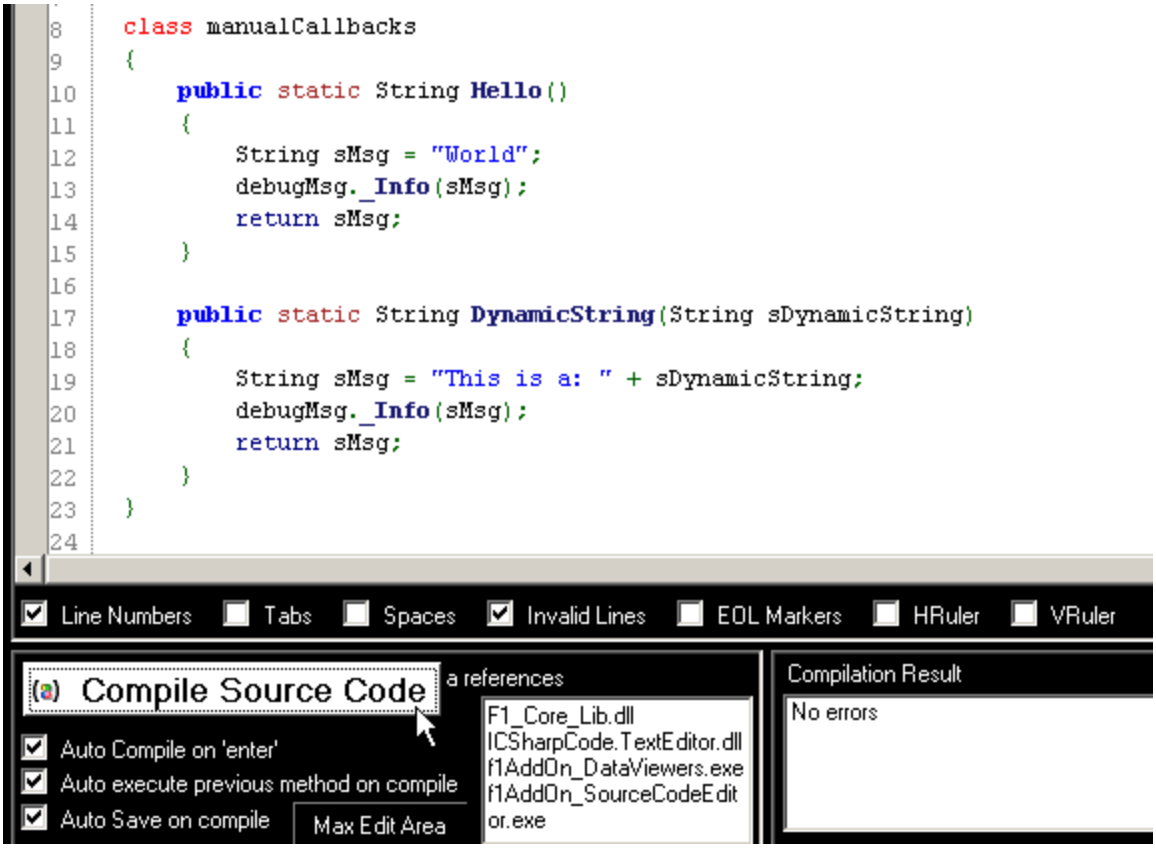


Just to show a couple more features of the Script Editor:

Enter the following code below the *Hello* function

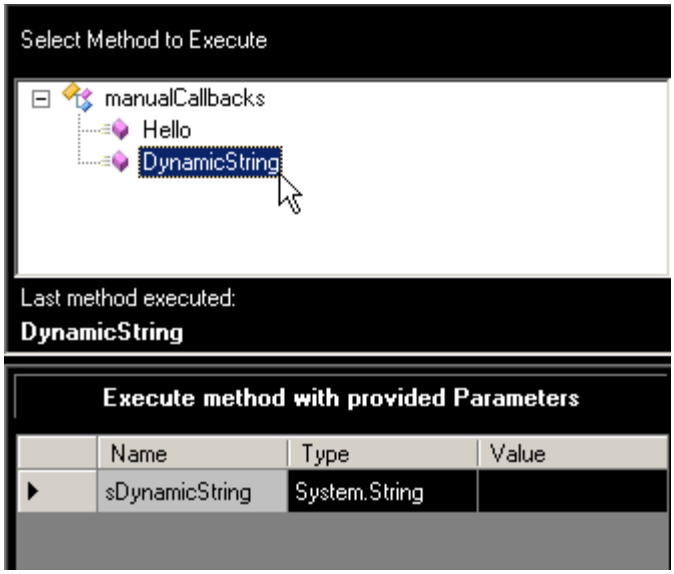
```
public static String DynamicString(String sDynamicString)
{
	String sMsg = "This is a: " + sDynamicString;
	debugMsg._Info(sMsg);
	return sMsg;
}
```

And click on *Compile Source Code*

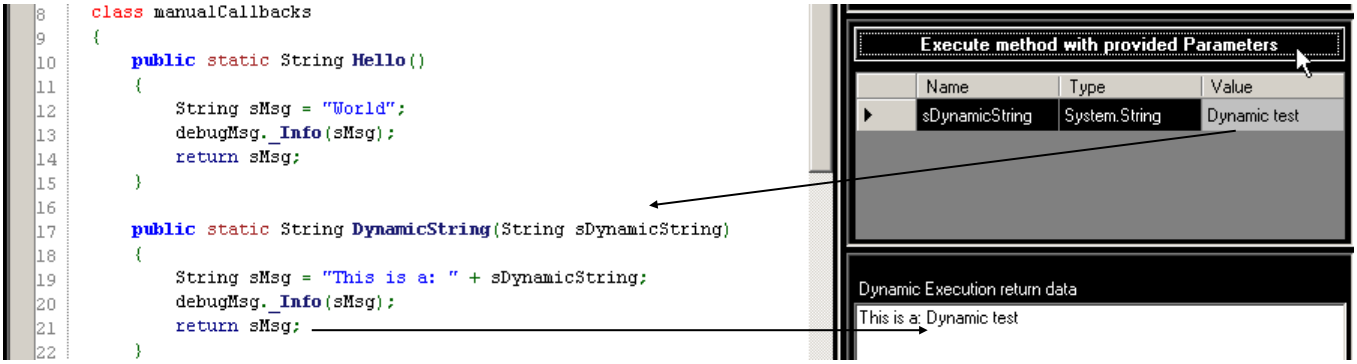


The compilation result should show no errors and the dynamic list of methods should now have the *DynamicString* function

Click on this *DynamicString* node and note how the DataGridView below now has a row with the required parameters (in this case the String *sDynamicString*)

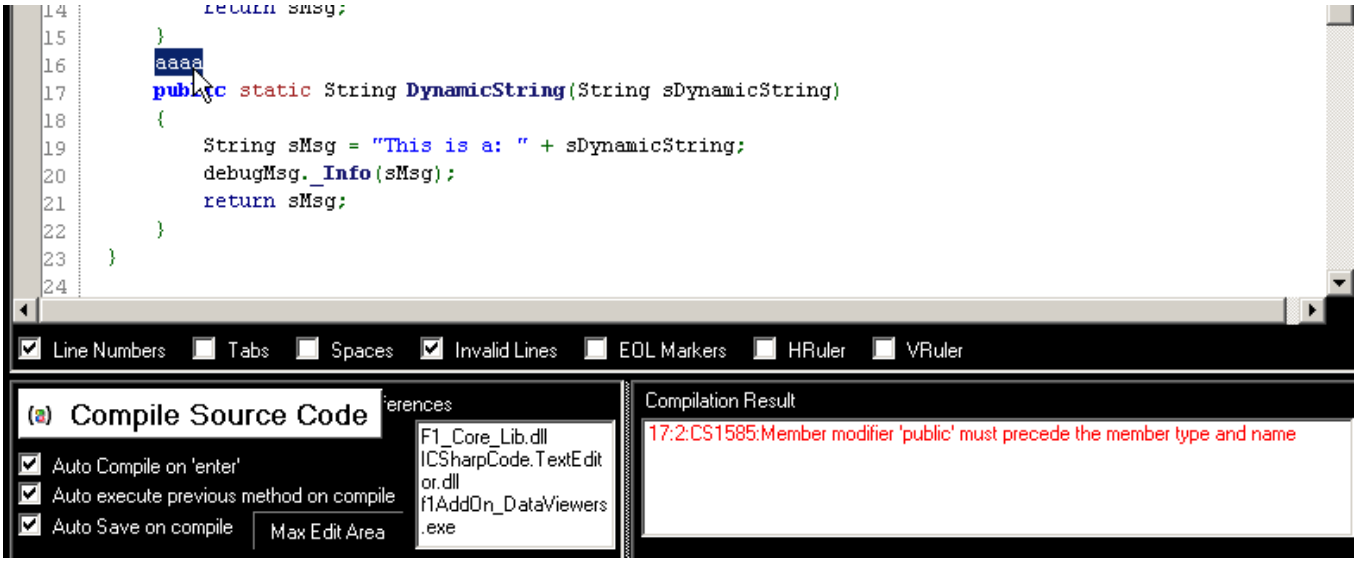

Select Method to Execute
manualCallbacks
Hello
DynamicString
Last method executed:
DynamicString
Execute method with provided Parameters
Name
Type
Value
sDynamicString
System.String

Enter a value in the editable cell and click on *'Execute method with provided Parameters'*

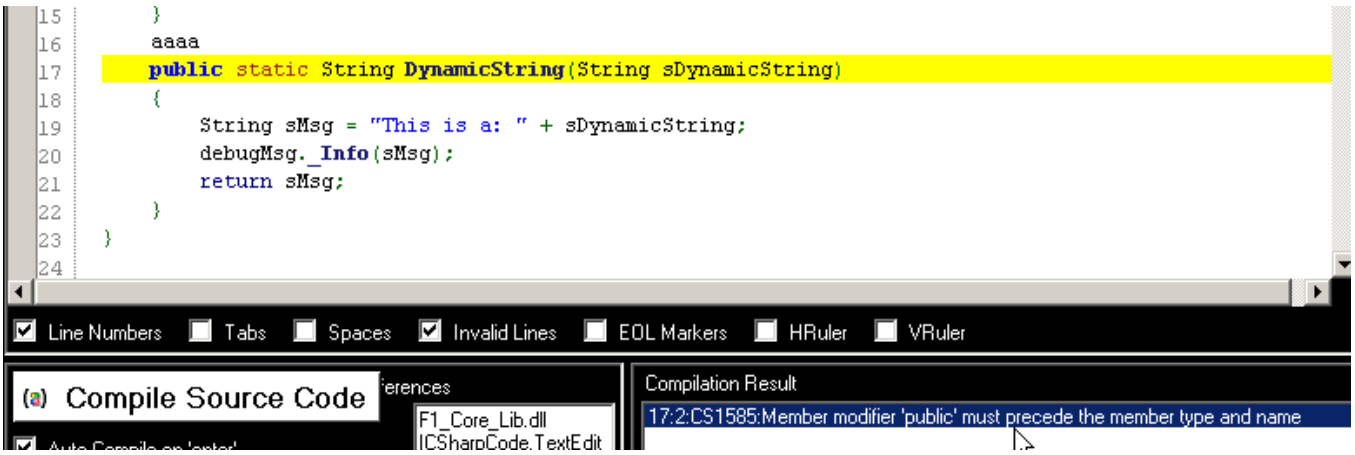


ERROR HANDLING

When errors occur in the C# code detailed error messages are shown in the Compilation Result TextBox



Click on the error and its location will be highlighted



TIPs: When the cursor is inside the Source Code Edit area, if the Auto Compile on 'enter' is set automatic compilation can be triggered by the following sequence: {Select an text area (Shitt+right-arrow)} + Enter. Then if Auto execute previous method on compile is also set, the last executed method will be automatically re-executed (this simple feature saves considerable about of time since it automates a process that would take at least 3 clicks and at least 20 secs).

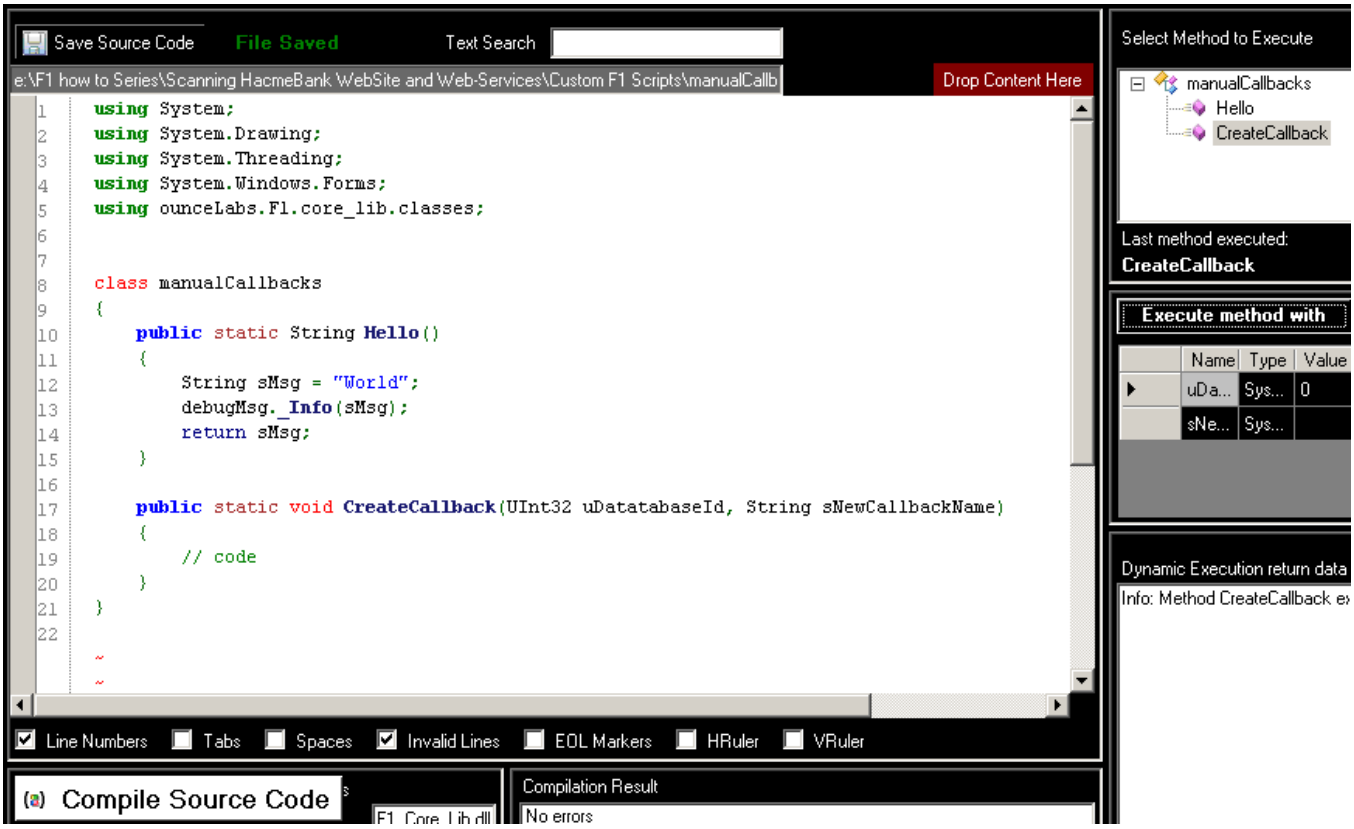
The Auto Save on compile auto saves on compile and Max Edit Area will collapse the left and right panels (only leaving the script editor and preferences panels)

CREATING CALLBACK

On the script editor, replace the existing code with

```
using System;
using System.Drawing;
using System.Threading;
using System.Windows.Forms;
using ounceLabs.F1.core_lib.classes;


class manualCallbacks
{
	public static String Hello()
	{
		String sMsg = "World";
		debugMsg._Info(sMsg);
		return sMsg;
	}

	public static void CreateCallback(UInt32 uDatatabaseId, String sNewCallbackName)
	{
		// code
	}
}
```

The code should compile with no errors:



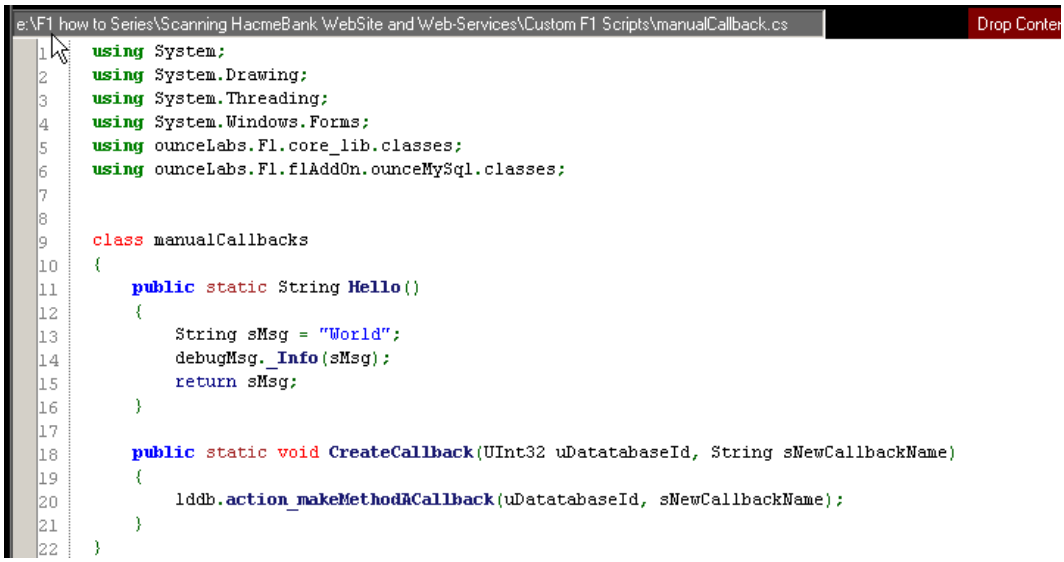
Add the following using reference:

```
using ounceLabs.F1.f1AddOn.ounceMySql.classes;
```

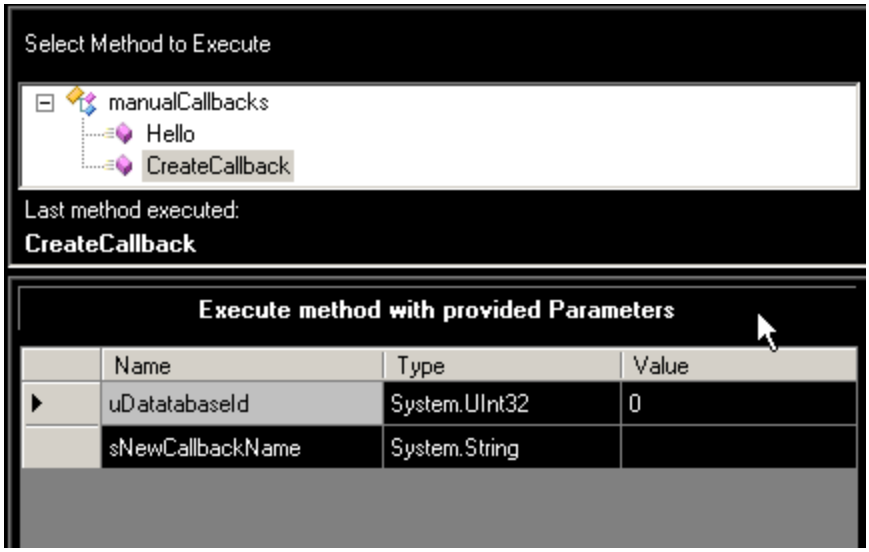
and line inside the CreateCallback function:

```
lddb.action_makeMethodACallback(uDatatabaseId, sNewCallbackName);
```

and that's it:



Compile it and execute it using the right-hand side dynamic execution environment:



EXECUTING SCRIPT CREATED USING COMMAND PROMPT

Another way to execute this method is to use the *CommandPrompt.*

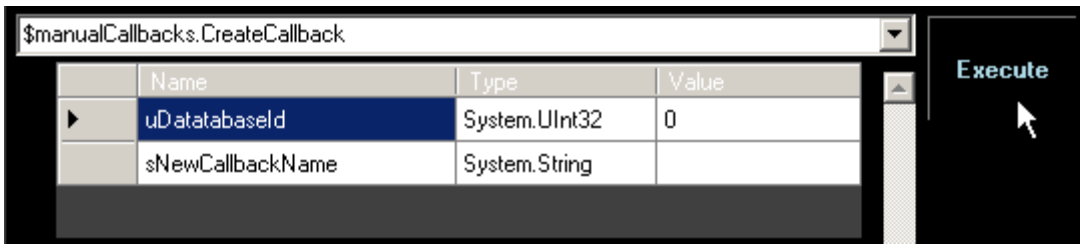
On successful compilation the newly created class is placed on a global (to F1) dynamic variable which can be invoked from the *Command Prompt.*

Make the *Command Prompt* visible 

Enter $ to see the list of available variables:

Select the *manualCallbacks* and press . (dot) to show the functions available)

Select *CreateCallback* enter the desired values and click execute

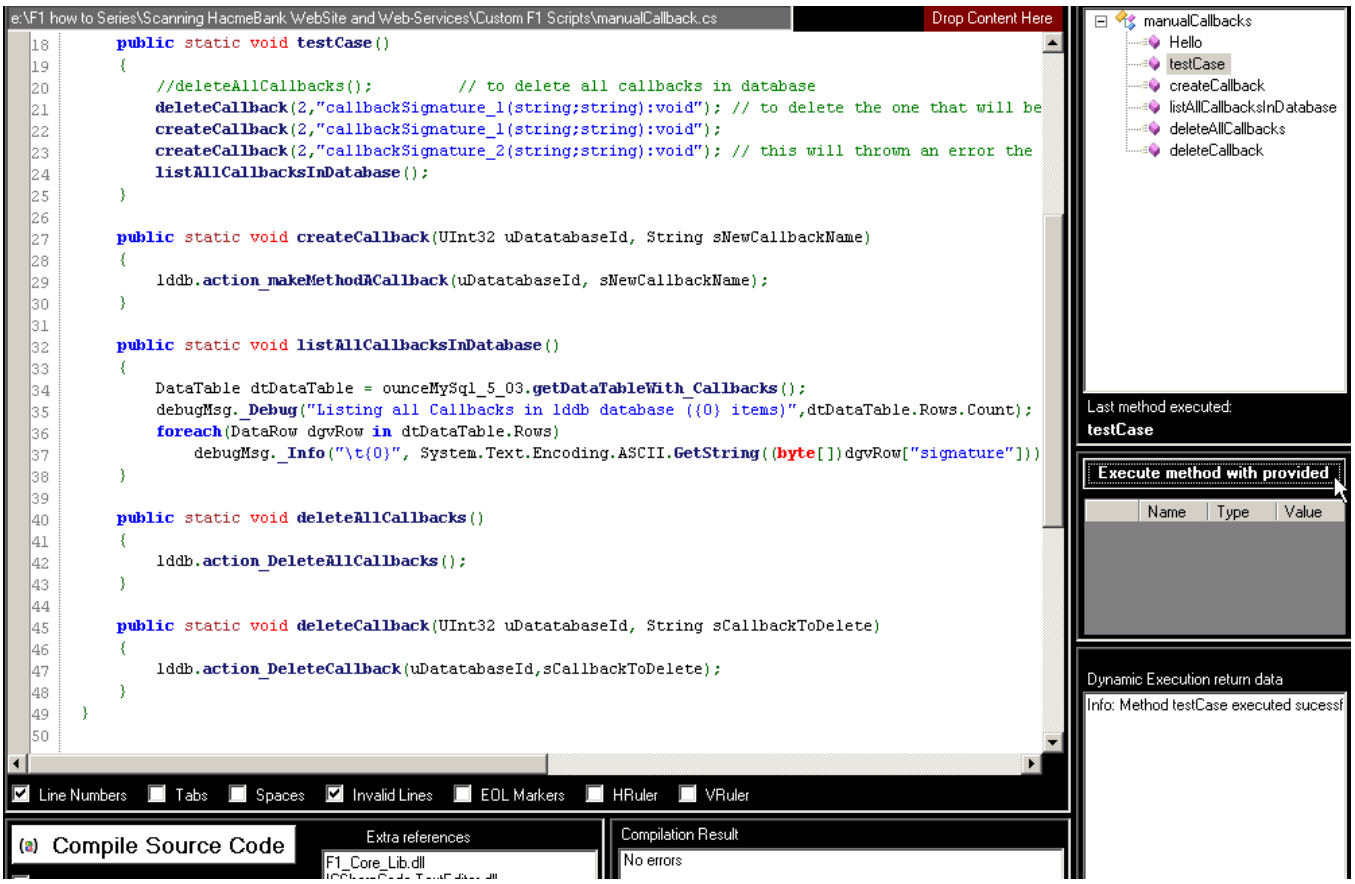


ADDING MORE USEFUL FUNCTIONS TO SCRIPT

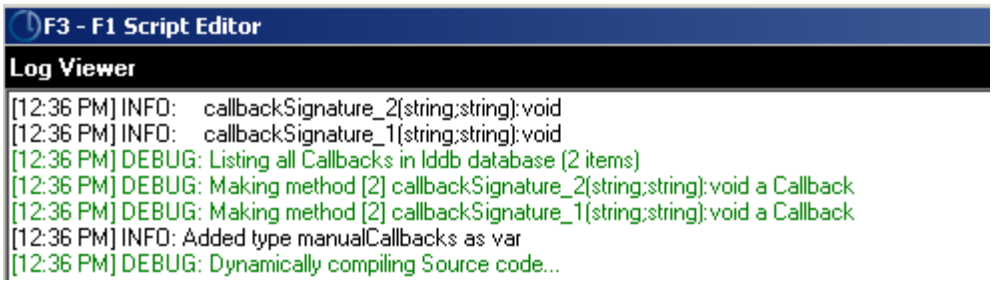
Replace the previous script with this code:

```
using System;
using System.Data;
using System.Drawing;
using System.Threading;
using System.Windows.Forms;
using ounceLabs.F1.core_lib.classes;
using ounceLabs.F1.f1AddOn.ounceMySql.classes;


class manualCallbacks
{
	public static String Hello()
	{
		String sMsg = "World";
		debugMsg._Info(sMsg);
		return sMsg;
	}
	public static void testCase()
	{
		//deleteAllCallbacks();			// to delete all callbacks in database
		deleteCallback(2,"callbackSignature_1(string;string):void"); // to delete the
one that will be created next
		createCallback(2,"callbackSignature_1(string;string):void");
		createCallback(2,"callbackSignature_2(string;string):void"); // this will thrown
an error the 2nd time it is executed (duplicate callback entries are not allowed)
		listAllCallbacksInDatabase();
	}

	public static void createCallback(UInt32 uDatatabaseId, String sNewCallbackName)
	{
		lddb.action_makeMethodACallback(uDatatabaseId, sNewCallbackName);
	}

	public static void listAllCallbacksInDatabase()
	{
		DataTable dtDataTable =
ounceMySql_5_03.getDataTableWith_Callbacks();
		debugMsg._Debug("Listing all Callbacks in lddb database ({0}
items)",dtDataTable.Rows.Count);
		foreach(DataRow dgvRow in dtDataTable.Rows)
			debugMsg._Info("\t{0}",
System.Text.Encoding.ASCII.GetString((byte[])dgvRow["signature"]));
	}

	public static void deleteAllCallbacks()
	{
		lddb.action_DeleteAllCallbacks();
	}

	public static void deleteCallback(UInt32 uDatatabaseId, String sCallbackToDelete)
	{
		lddb.action_DeleteCallback(uDatatabaseId,sCallbackToDelete);
	}
}
```

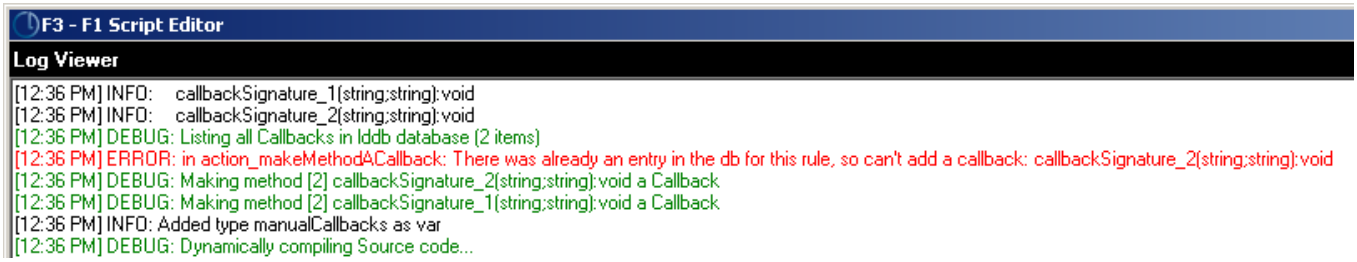
The script should compile with no errors.



Execute *testCase* and the LogViewer should look like this the first time it is executed:



And like this the 2nd time:



USING DOTNET_CALLBACKSMAKER_5_03 ADD-ON TO AUTOMATICALLY FIND WEB SERVICES FUNCTIONS AND MAKE THEM ALL CALLBACKS

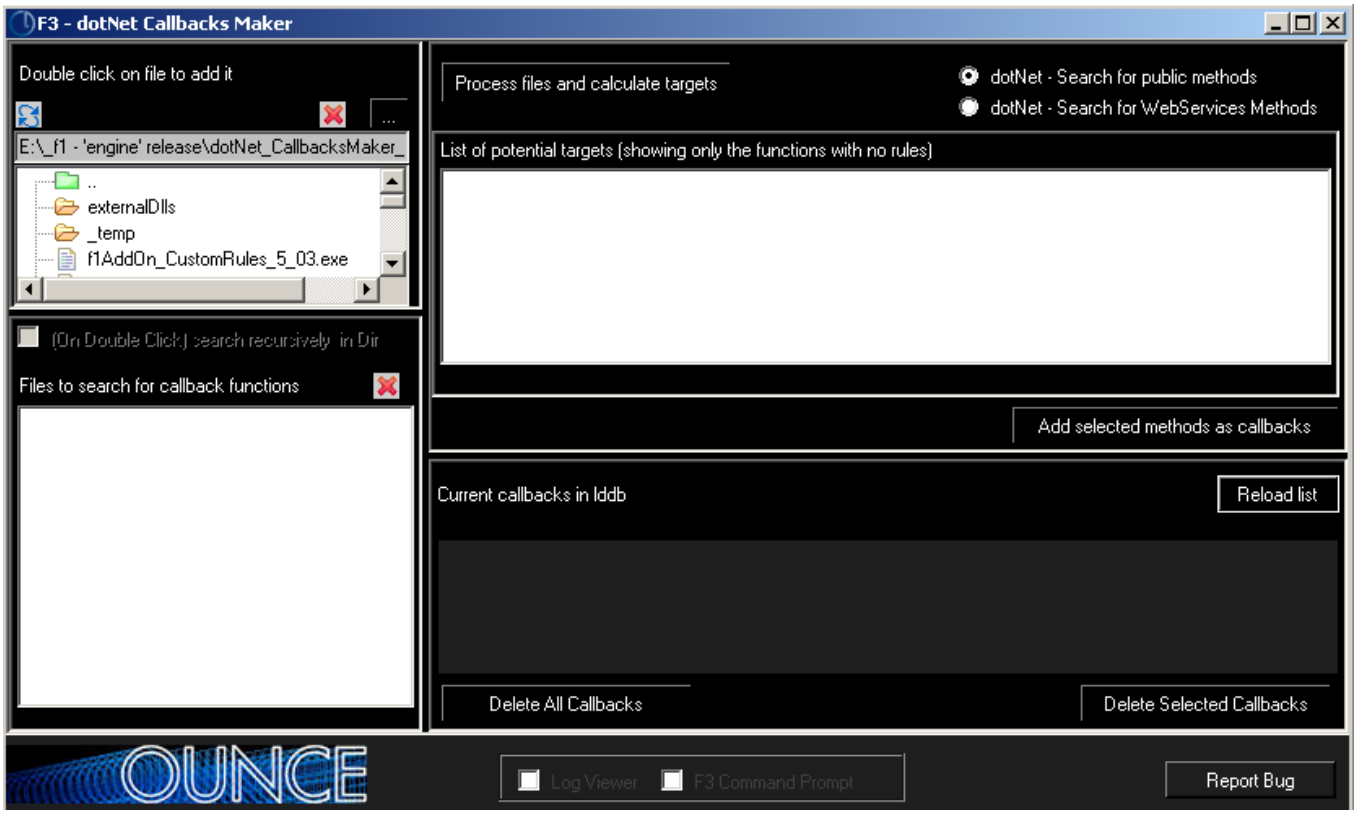
There were three main problems with the previous examples:

1) the need to know the correct function signature in a format that will be recognized by CORE's AE
2) the detection process of the methods marked with the [WebMethod] attribute
3) the manual (one function at the time) process

Ideally what we need is to have an automated process to analyze a .Net *.dll file and automatically create the Callbacks rules.
And this is exactly what the *dotNet_CallbacksMaker_5_03* Add-on does

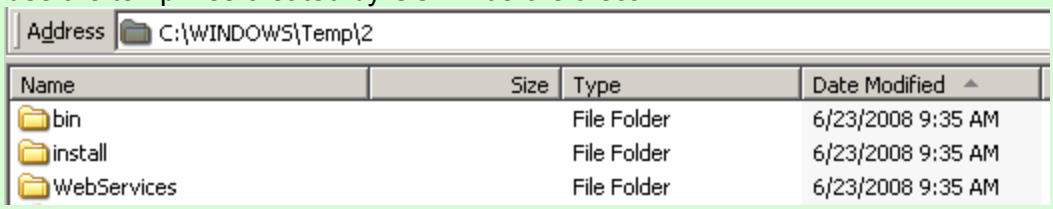
FINDING WEBSERVICES FUNCTIONS AND ADDING 1 CALLBACK

Open *f1AddOn_dotNet_CallbacksMaker.exe*



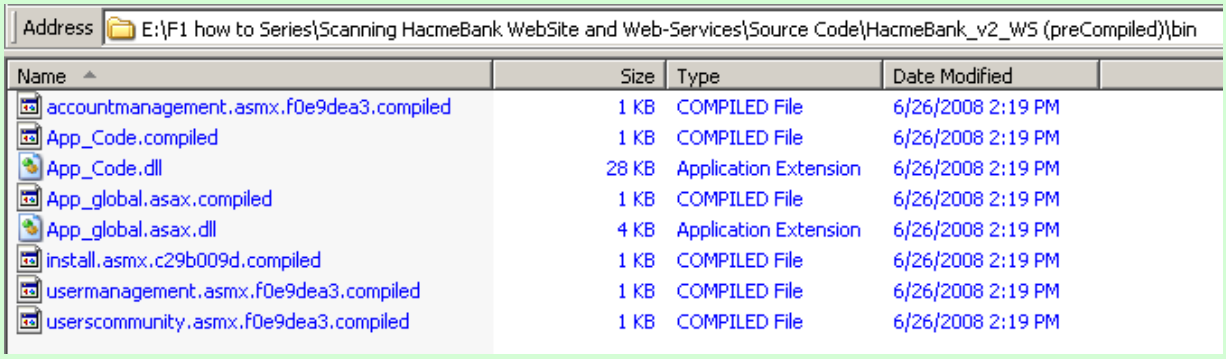
TIPs: For ASP.NET websites the dlls are not automatically created on build, so to access one can:

1) use the temp files created by CORE before a scan:

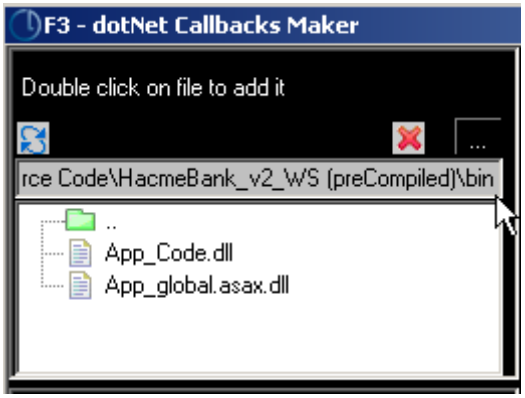


2) run the aspnet_compiler.exe tool

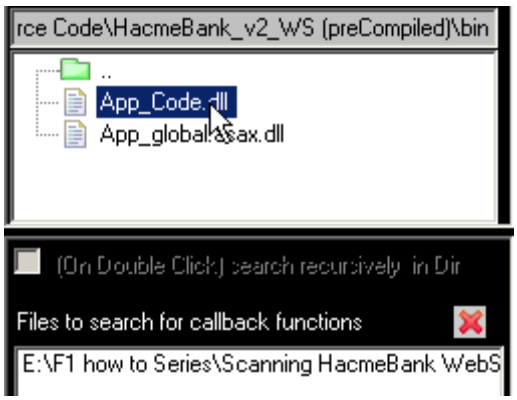
```
aspnet_compiler -p HacmeBank_v2_WS -v /HacmeBank_v2_WS "HacmeBank_v2_WS (preCompiled)"
```



Using the top-left directory control, navigate to the directory with the precompiled .Net dll (enter the address and press enter: *E:\F1 how to Series\Scanning HacmeBank WebSite and Web-Services \Source Code\HacmeBank_v2_WS (preCompiled)\bin)*

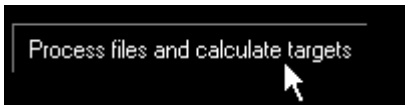


Double click on App_Code.dll to add it to the ListBox with *Files to search for callbacks functions*



Select the *dotnet – Search for WebServices Methods* radio box so that only the Web Methods are shown

Click on *Process files and calculate targets*



And all methods with the [WebMethod] attribute are listed:

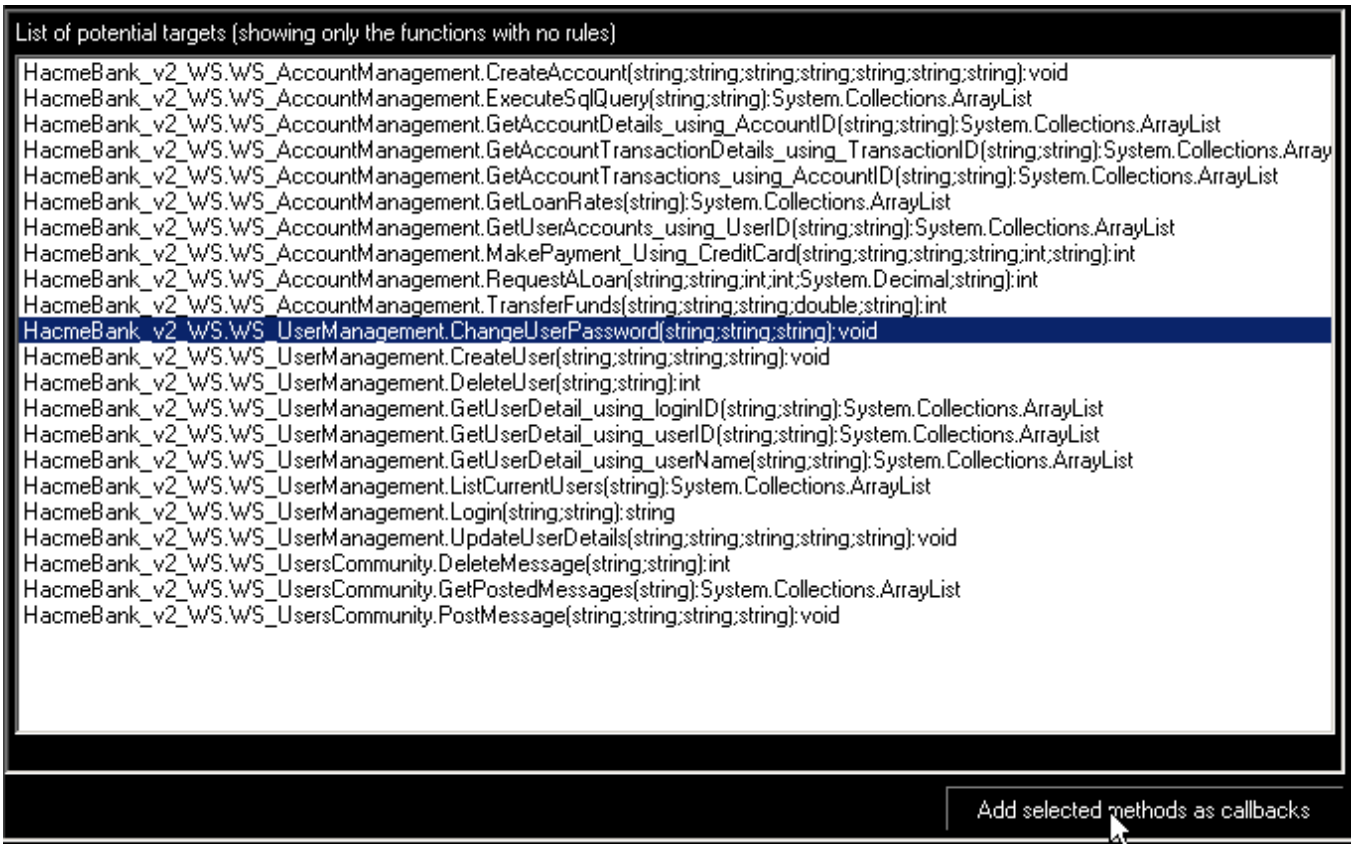
Following the example shown before in this document, we will use the ChangeUserPassword function which was vulnerable to SQL Injection

:

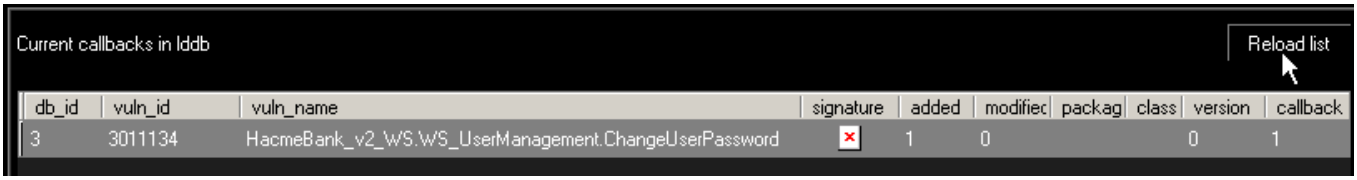
UserManagement.asmx.cs

```
107
108         [WebMethod]
109         public void ChangeUserPassword(string sessionID,string userID, string newPassword)
110         {
111             HacmeBank_v2_WS.DataFactory.ChangeUserPassword(userID,newPassword);
112         }
113
```

To *ChangeUserPassword* a callback, select it and press on *Add selected methods as callbacks*



Click on *Reload List* and confirm that the rule has been added

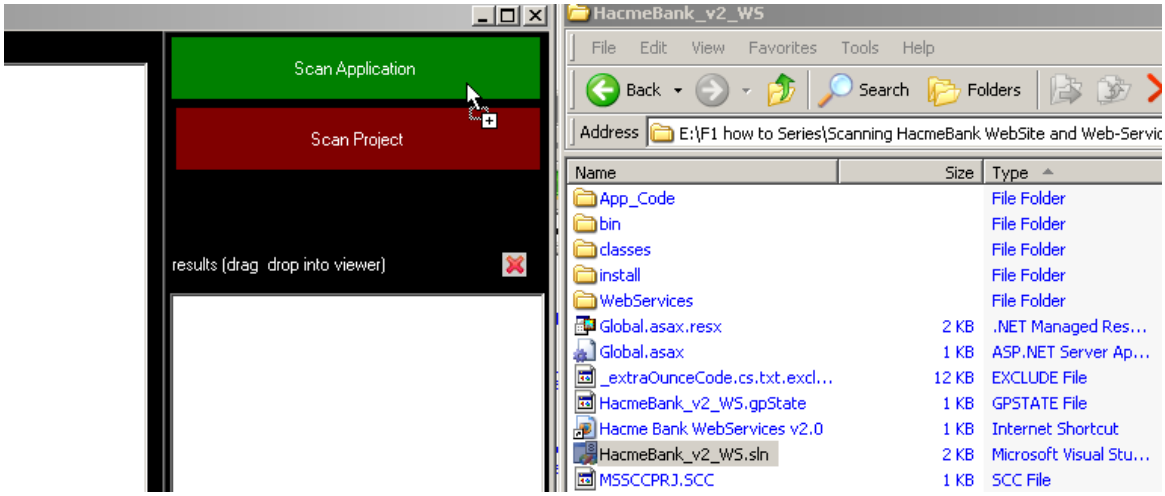


RE-SCANNING APPLICATION AND ANALYSING NEW TRACE

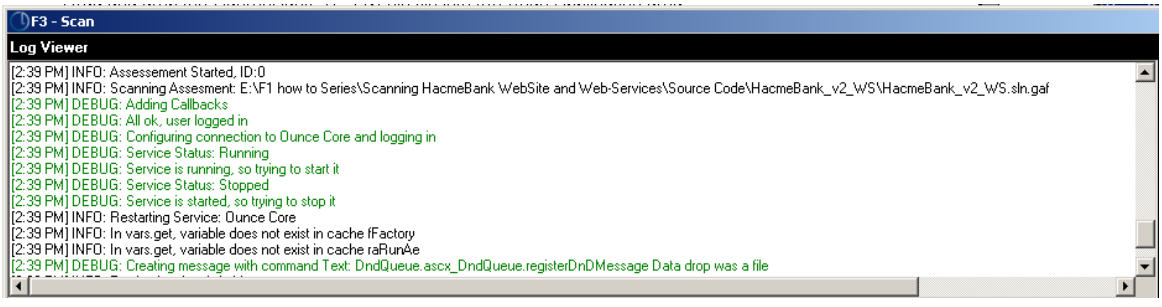
With the new Callback in place we are need to rescan the application:

Open 

Drag and drop the HacmeBank_v2_WS.sln file into the *Scan Application* area

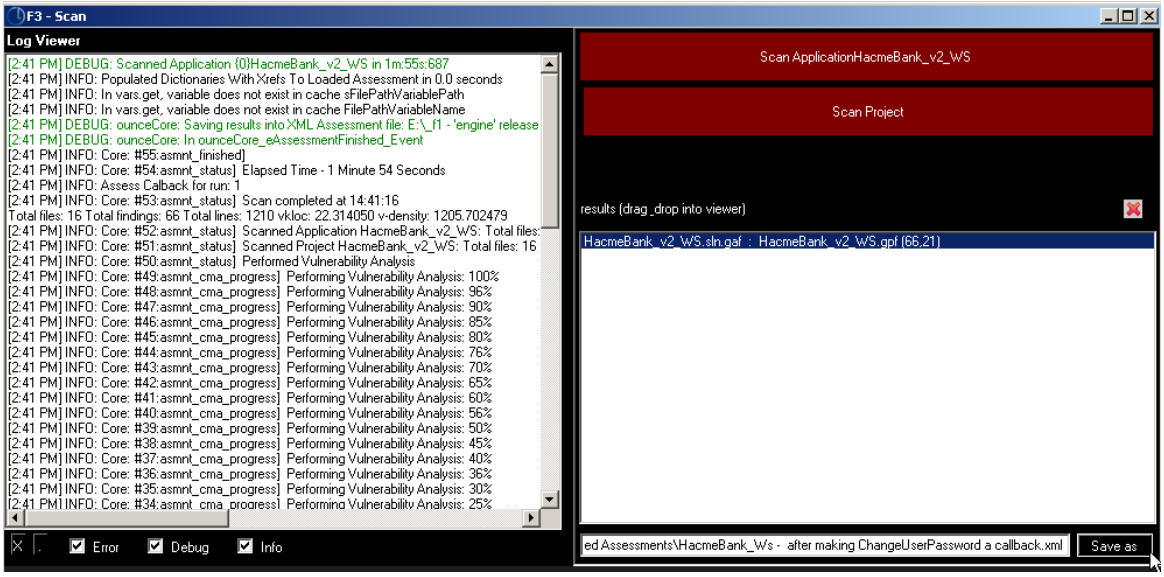


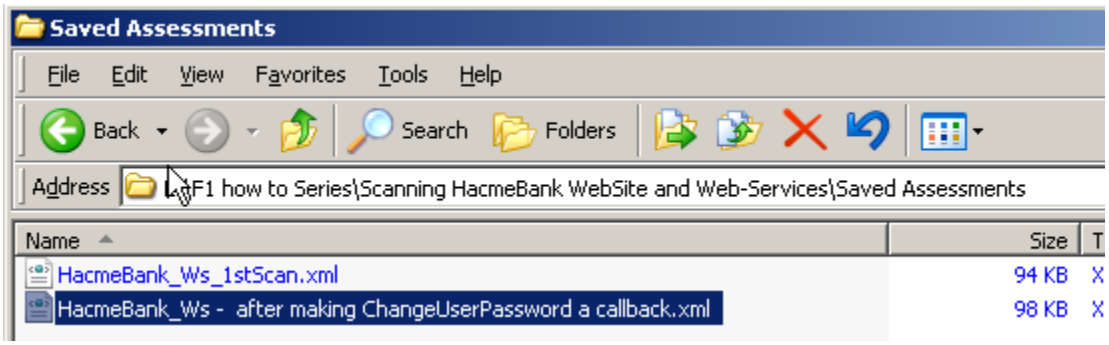
If this is the first time a scan is executed from the current open instance of f1AddOn_Scan_5_03.exe the Ounce Core Service will be restarted and the connection re-established:



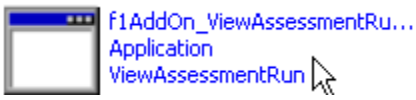
IMPORTANT NOTE: After **making any Custom Rules changes** on the CORE database, **the Ounce CORE services must be restarted!** F1 makes direct changes to the CORE's database and at the moment those changes cannot be dynamically reloaded

Once scan is completed select the new line in the *results* ListBox to save the assessment to a local folder:

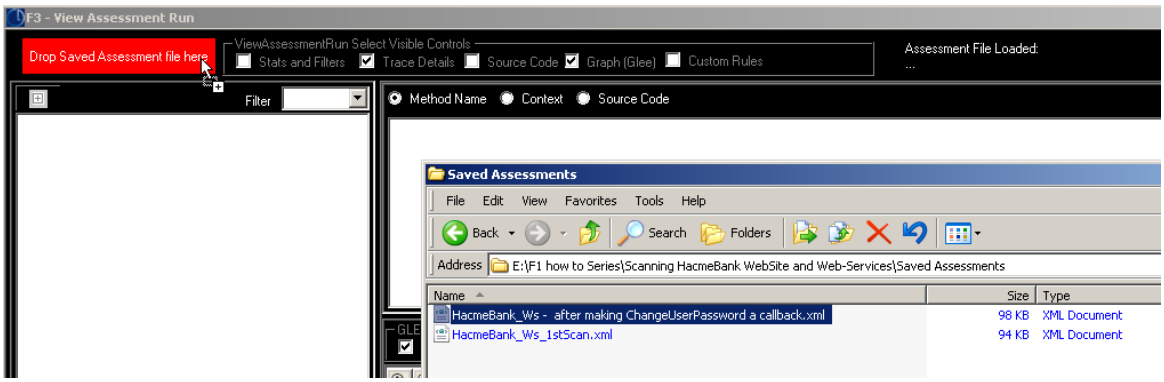




Open to *f1AddOn_ViewAssessmentRun_5_03.exe*

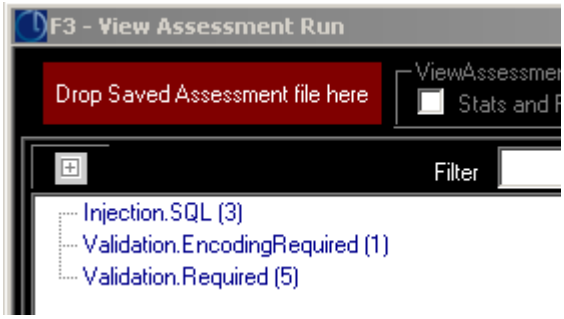


Drag and drop the Saved Assessment file into the *Drop Saved Assessment file here* control area

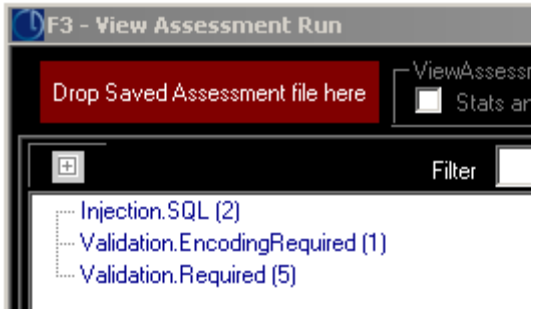


And the scan will be loaded (below are the number of traces in the current scan (with the **ChangeUserPassword** callback) and in the previous one (with no callbacks))

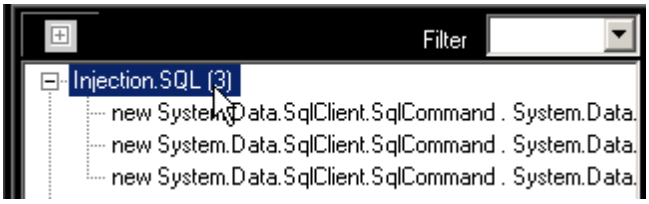
Current Scan



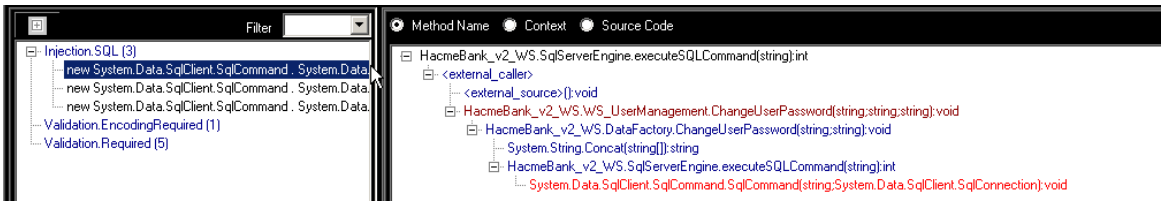
Previous scan



Click on *Injection.SQL (3)* to expand the node[1]

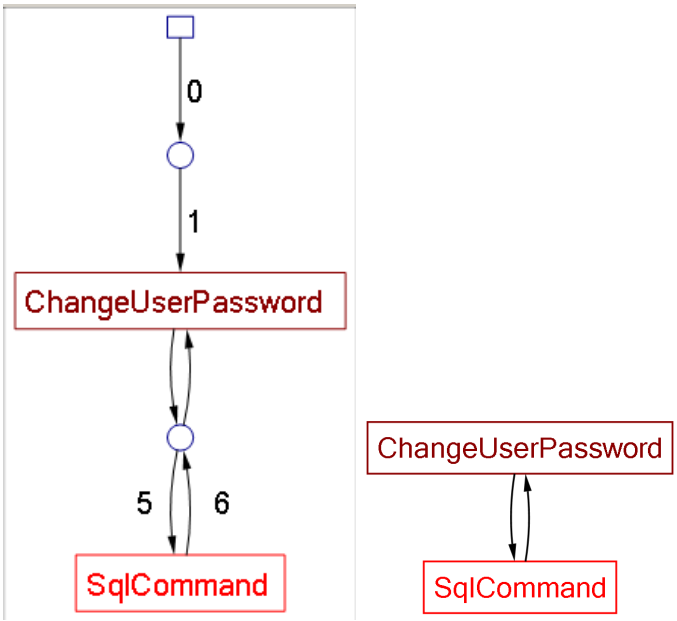


Click on the first child node and see that now we have a trace from the ChangeUserPassword source to the SqlCommand Sink

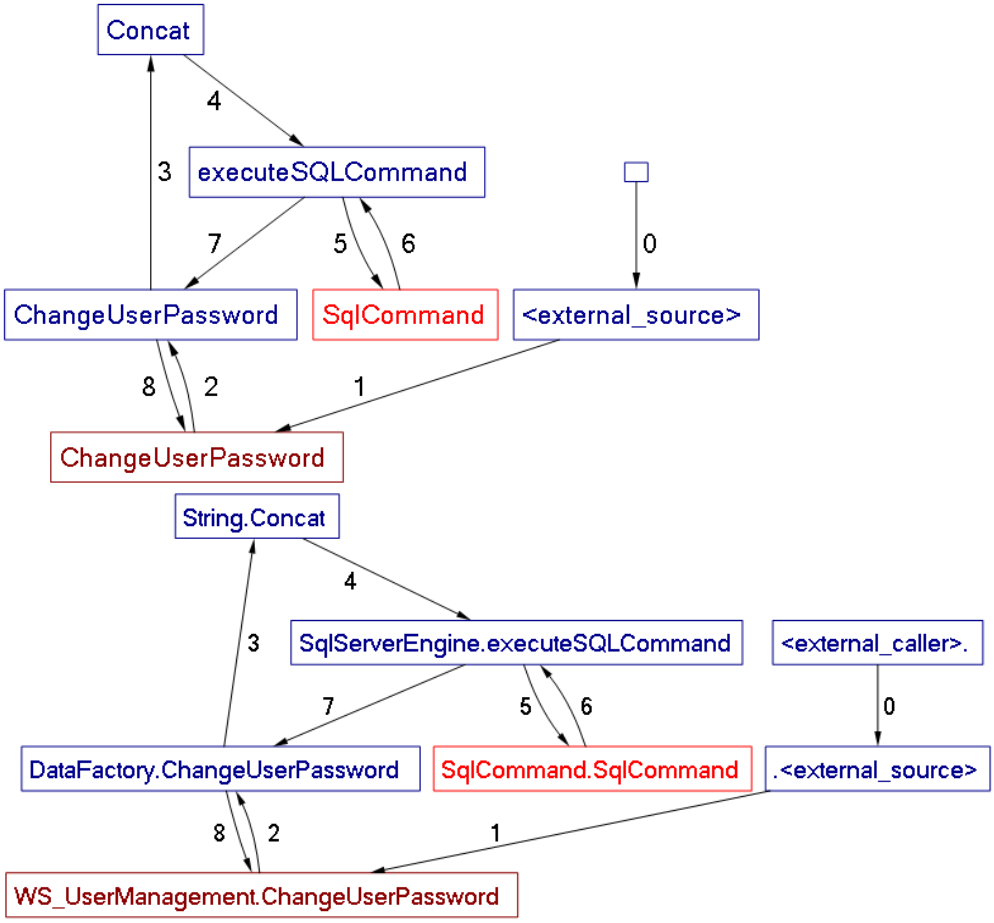


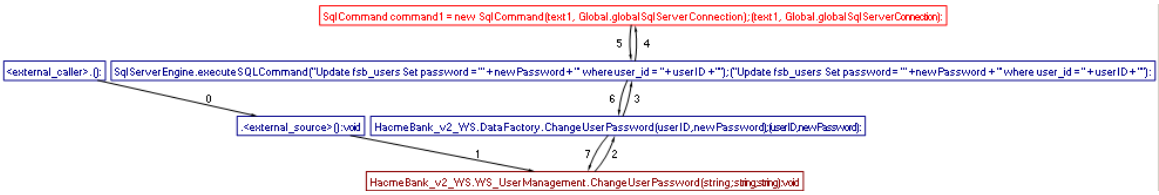
The GLEE graph can also be used to analyze this trace:

[1] This TreeView delay loads the findings (i.e. only when selected) which allows faster rendering of this control for assessment files that have large number of findings


0
1
ChangeUserPassword
5
6
SqlCommand

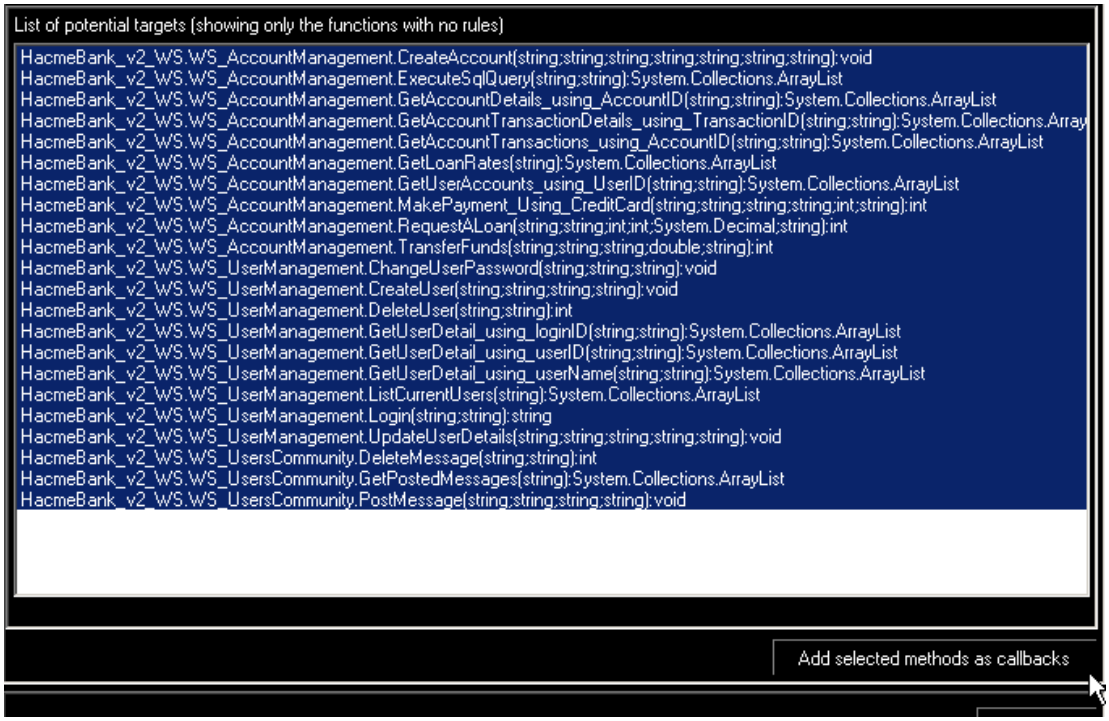
ChangeUserPassword
SqlCommand


Concat
4
3
executeSQLCommand
7
5
6
0
ChangeUserPassword
SqlCommand
<external_source>
8
2
1
ChangeUserPassword
String.Concat
4
3
SqlServerEngine.executeSQLCommand
<external_caller>.
7
5
6
0
DataFactory.ChangeUserPassword
SqlCommand.SqlCommand
.<external_source>
8
2
1
WS_UserManagement.ChangeUserPassword

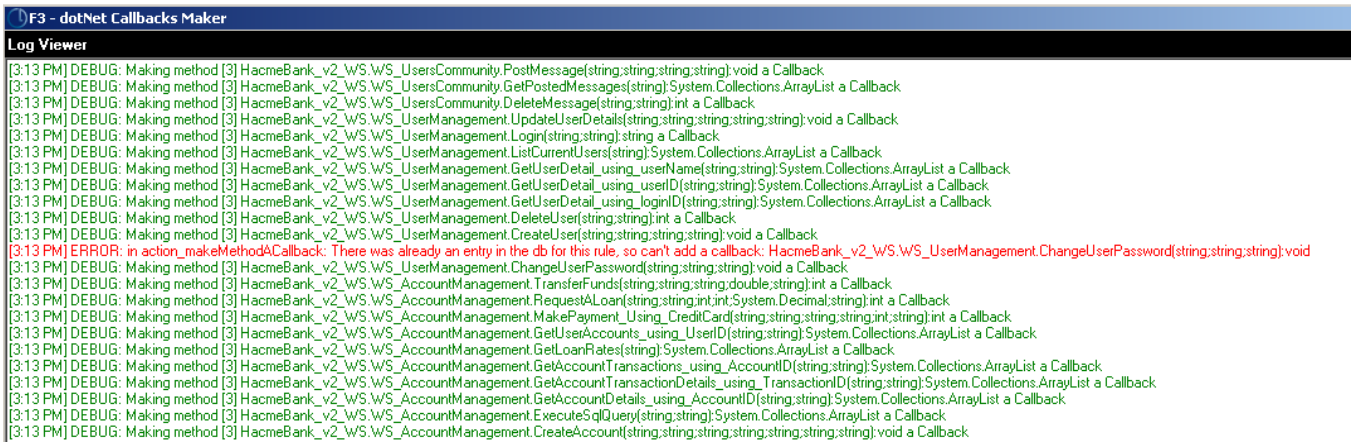


ADDING ALL CALLBACKS, RESCANNING AND VIEWING

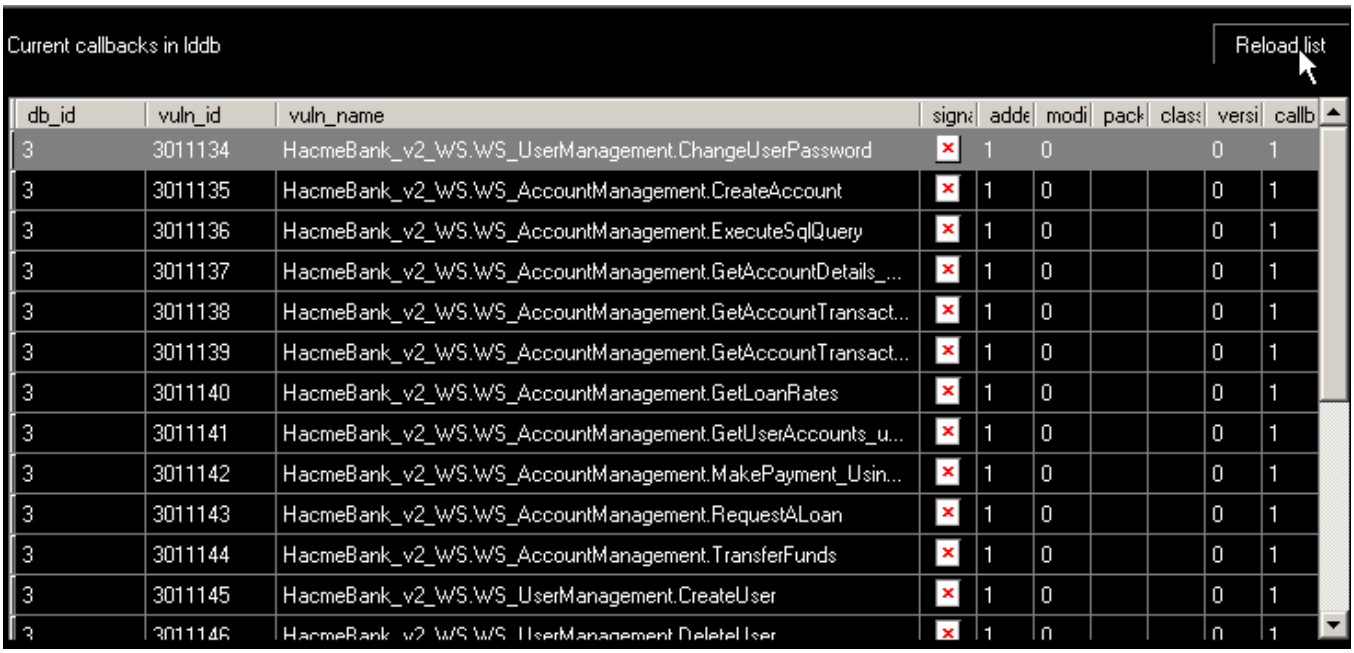
Go back to the *f1AddOn_CustomRules_5_03.exe*, select all functions and add them:



The LogViewer should look like this (the error is caused by the fact that we previously added the *ChangeUserPassword* function as a callback):



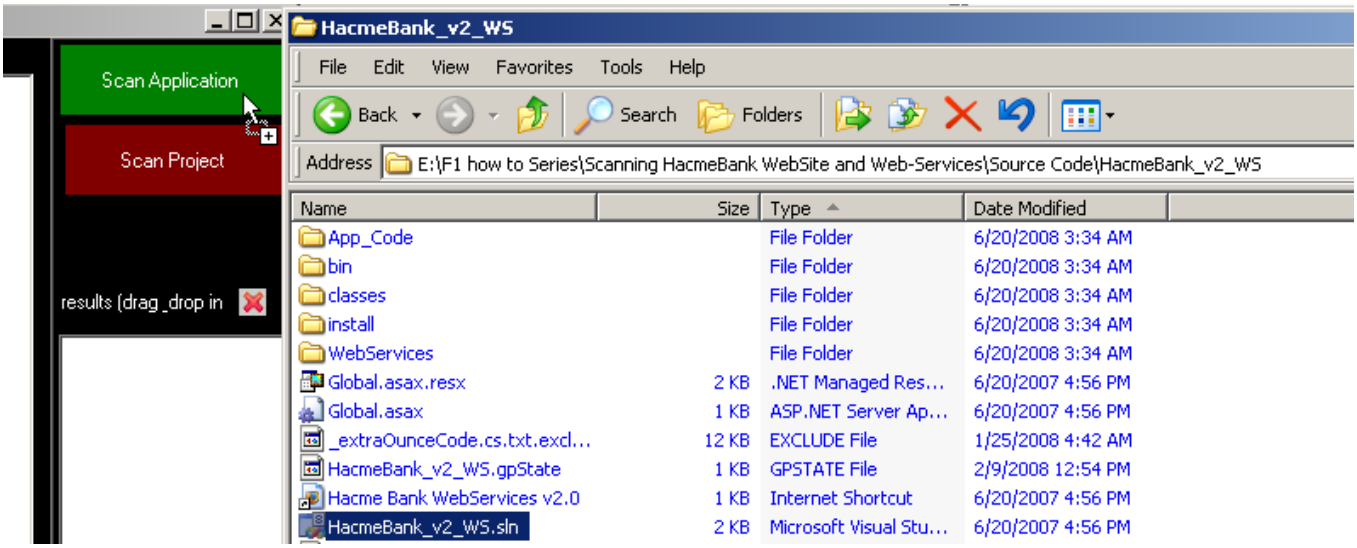
Click on *Reload List* will also confirm that the callbacks are in place:



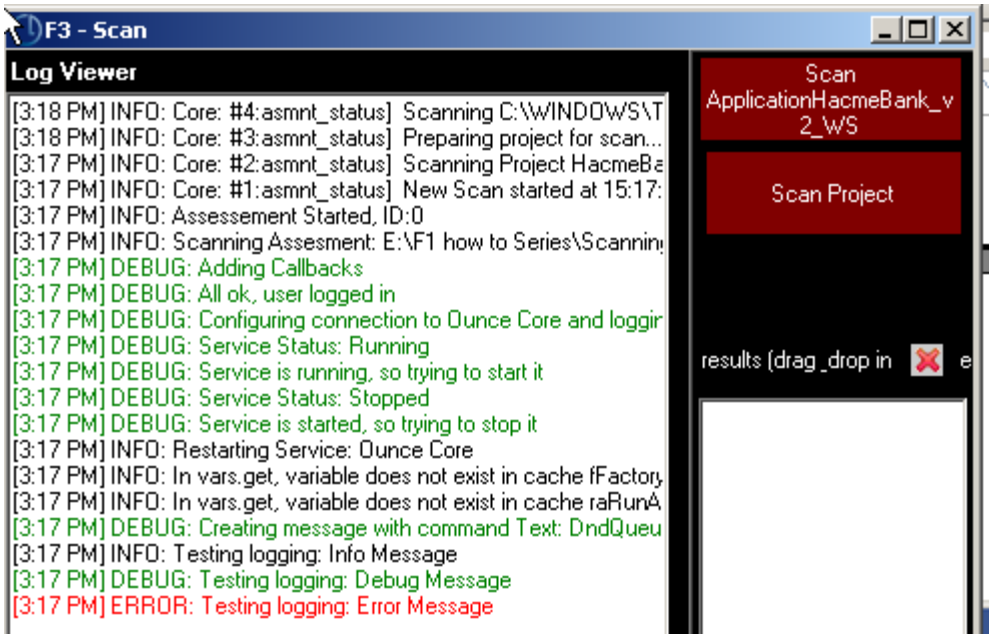
If the Scan Add-On is still open, close it

And reopen it:

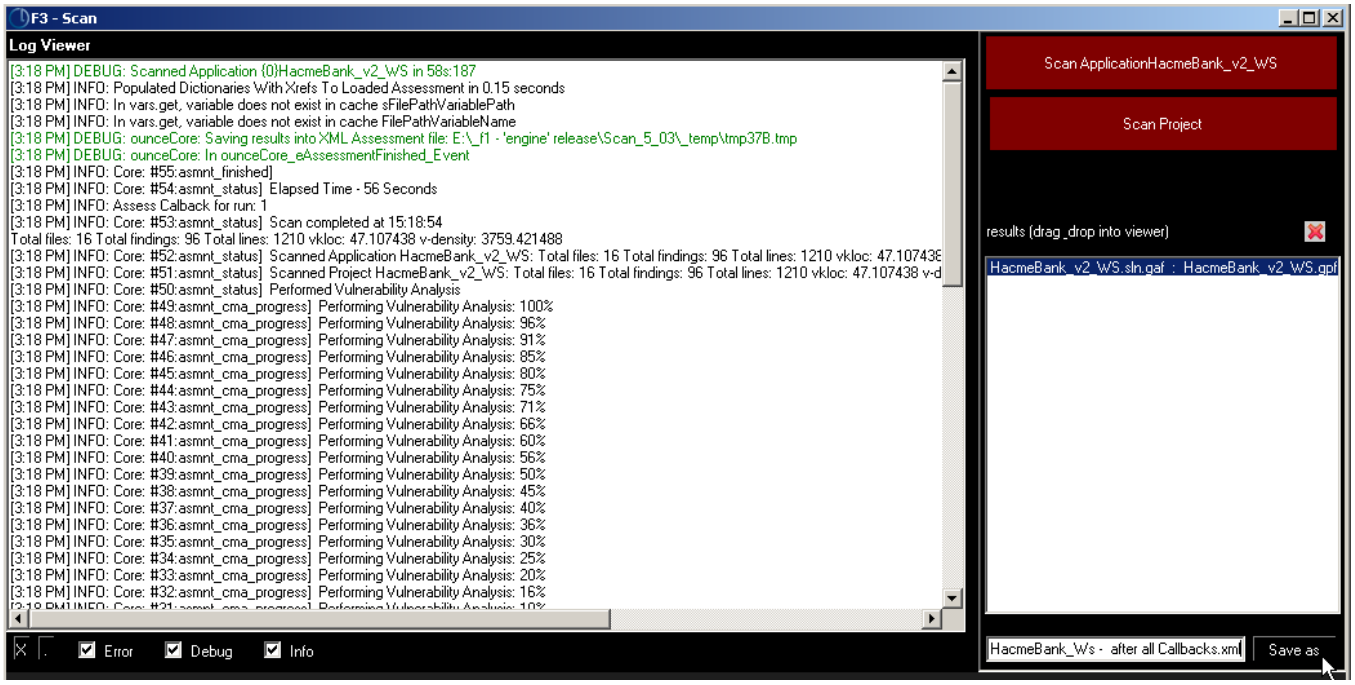
Drag and drop the solution to scan it:

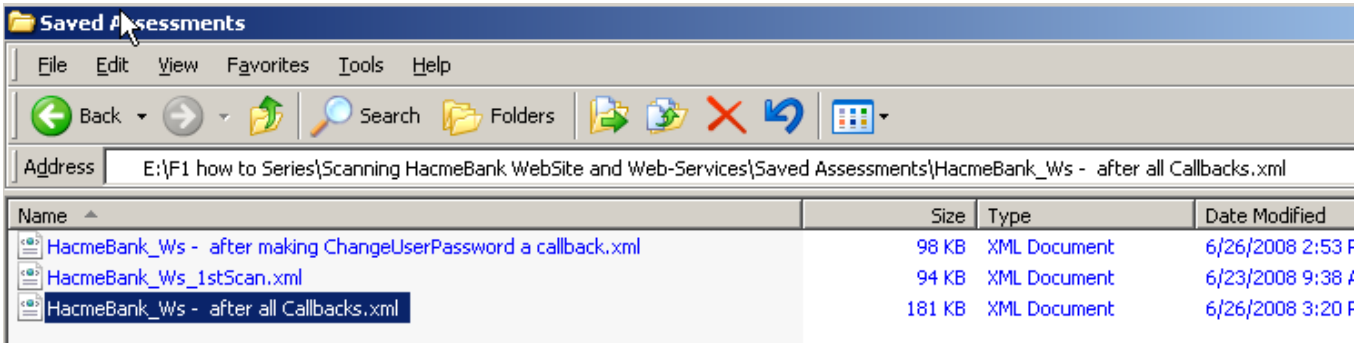


The Ounce Core will be restarted and a new scan will be triggered:

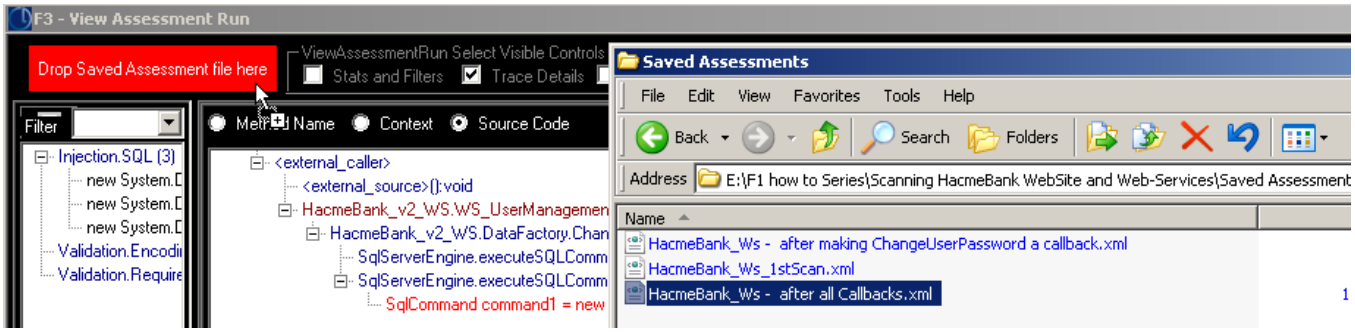


Once the scan is completed, select the new item in the *results* list and save it:

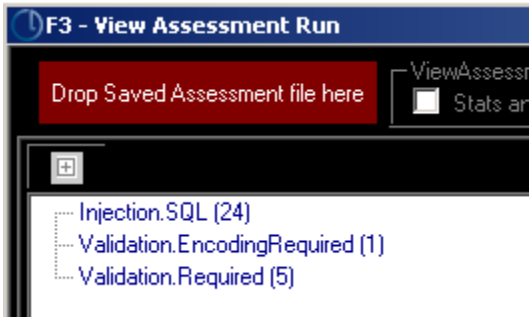


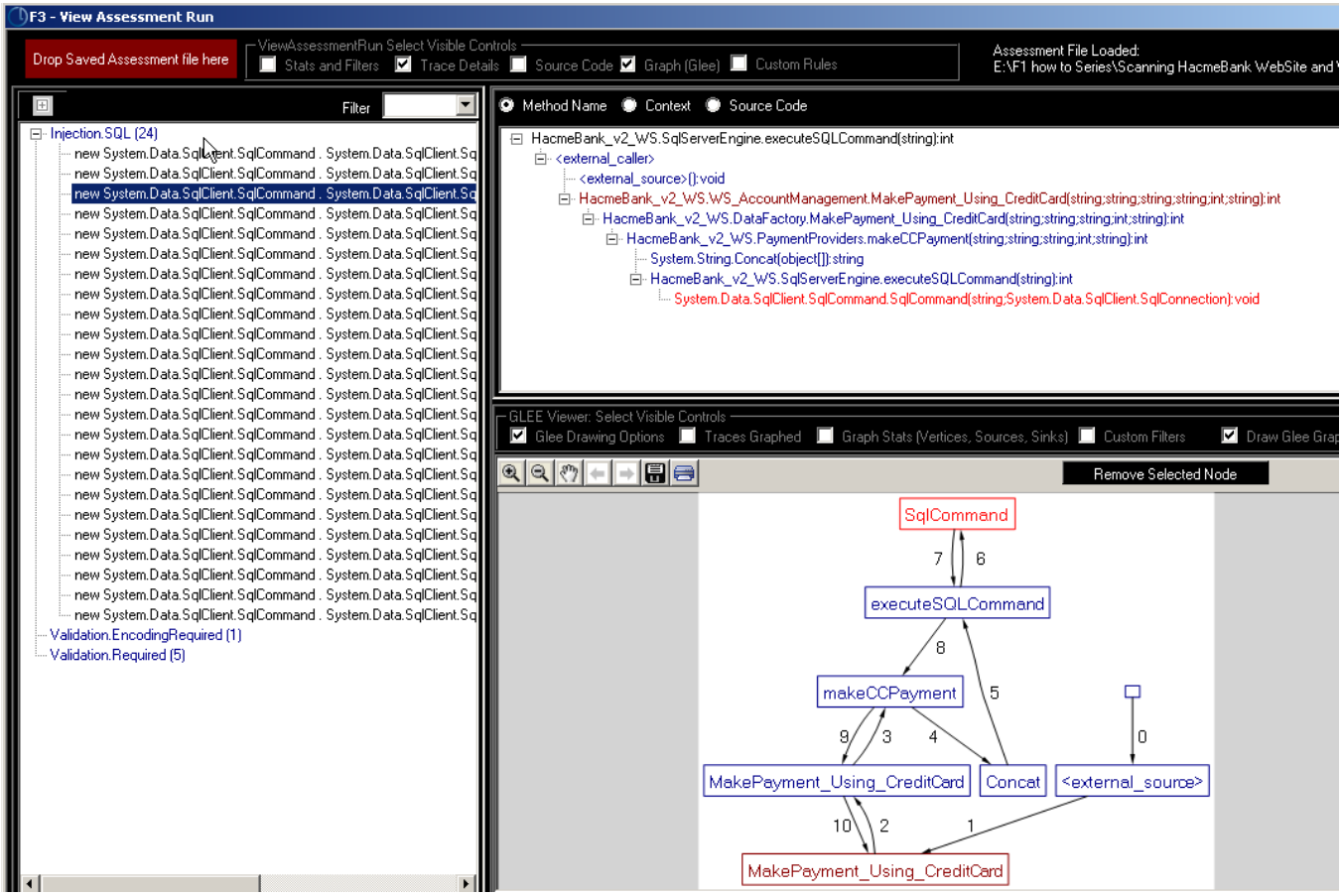


Drag and drop the assessment in to the *View Assessment Run* Add-On:



And note that there are 24 SQL Injection traces:

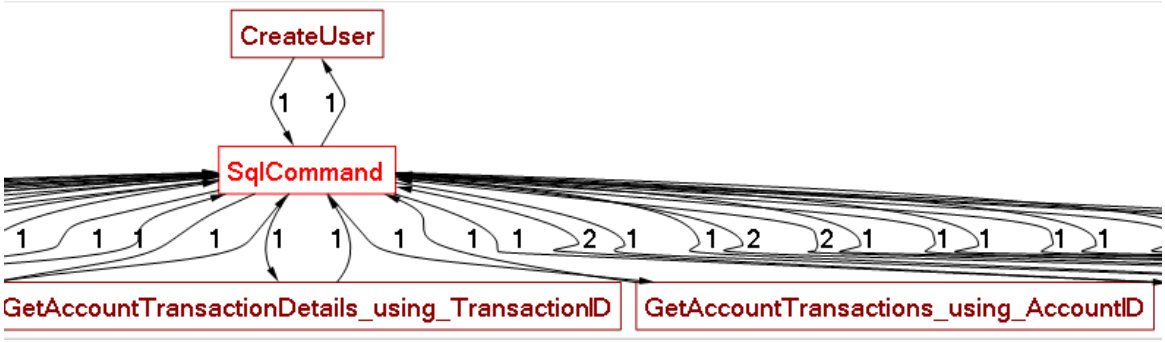




VIEWING 'INSECURE PATTERN' FOR SQL INJECTION

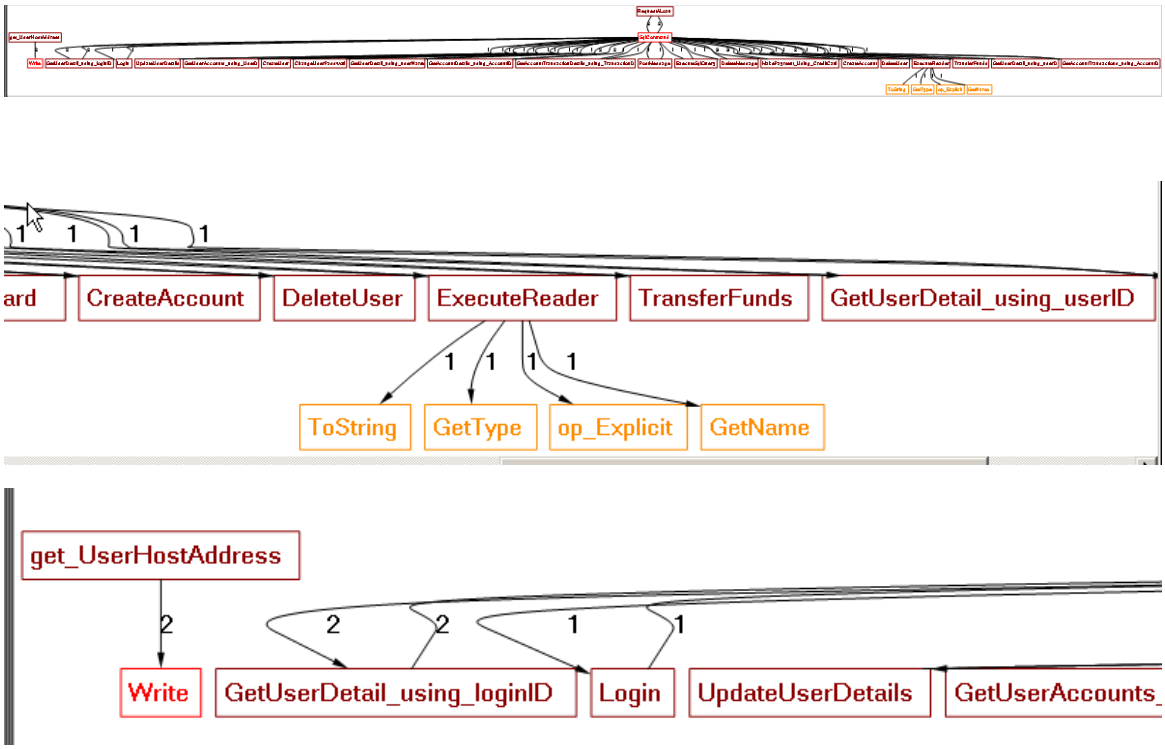
Here is an interesting view created using GLEE analysis.

This shows that all SQL injection vulnerabilities are created by one sink



VIEWING ONLY SOURCE AND SINKS FOR ALL TRACES

This is a view with all traces (the SQL Injection and the Validation ones)



(NOT COMPLETED)