QT 4 You
Qt Reversing Tutorial by ar1vr
http://picturoku.blogspot.com

I was asked a couple months ago by a friend of mine to “tweak” a software protection as he wanted to “evaluate” more “thoroughly” the targeted program. As I went in to it, I found the protection scheme very interesting as it involved a couple of cross platforms based engines interacting together to serve as anti-hacking security licensing system. One was Qt, the other being JavaScript. It was the second time that I came against a Qt based software. The first time thought, I just used the basic cracking skills every reverser uses with Windows GDI apps. But this time I needed to go deeper to understand the cross VM interactions, and a bigger understanding of the Qt framework was needed, so, I engaged into searching for more info on the subject. The results were quite poor, except for an article by Daniel Pistelli, who now works at Hex Rays (coincidence that IDA has been reassembled using this framework?).
Besides his article, not much on the subject of reversing Qt code seemed to be around. Since then I was faced with a fair amount of Qt applications, even Portuguese ones, so I decided to post my thoughts on the subject, and build a tutorial for the 4th version of Qt (coincidently its also the same version of IDA 6.1, hihi):

Note: The Qt platform has an OO approach, so consider before reading this tutorial, gather some insight about reversing C++ stuff, like RTTI, MS calling convention, virtual functions, etc.

For the purpose of this demonstration I used a demo application who accompanies the Qt SDk called *sdi.exe*, located in the examples*/mainwindows/sdi* folder. You can use any program you wish, to follow this explanation, thought.

1. Recognizing the Entrypoint

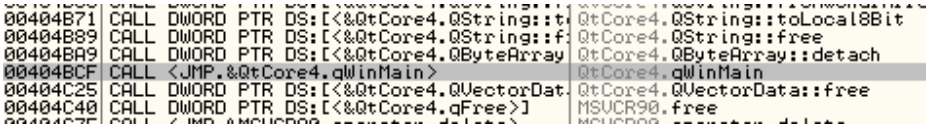
Usually looks like this and Olly is smart enough to identify it as it is the EP defined in the PE header.

2. How to find Main?

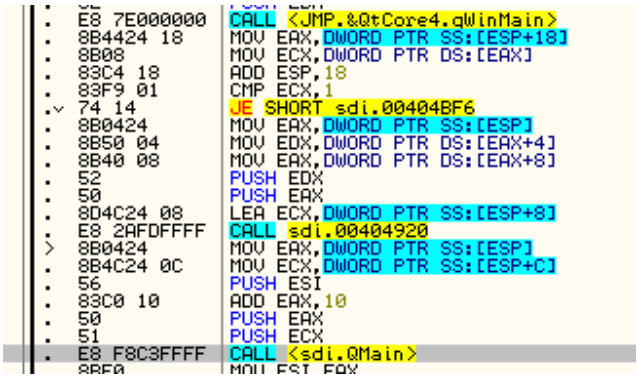
```
int main(int argc, char *argv[])
{
    Q_INIT_RESOURCE(sdi);
    QApplication app(argc, argv);
    app.setApplicationName("SDI Example");
    app.setOrganizationName("Trolltech");
    MainWindow *mainWin = new MainWindow;
    mainWin->show();
    return app.exec();
}
```

Consider Main function source code. Let's see what do we need to find it's corresponding assembly code, and what this code looks like after compilation.

2.1. First step is to search for all inter-modular calls and locate the imported dispatcher function **qWinMain** defined in QtCore4.dll.



2.2. Second step, select QtCore4.qWinMain and jump into it's code definition.



2.3. The first call after QtCore4.qWinMain sets up memory. But, we are impatient and we want QMain. QMain function will always be the second call, as highlighted in the figure.

To verify if we've selected the correct function call, go up some lines and the following api calls should be present:

- GetCommandLineW
- QString::fromWCharArray
- QString::toLocal8Bit
- QByteArray::detach

```
$  83EC 10      SUB ESP,10
.  8D4424 04    LEA EAX,DWORD PTR SS:[ESP+4]
.  50           PUSH EAX
.  6A FF        PUSH -1
.  FF15 0060400 CALL DWORD PTR DS:[<&KERNEL32.GetCommandLineW>]
.  50           PUSH EAX
.  8D4C24 14    LEA ECX,DWORD PTR SS:[ESP+14]
.  51           PUSH ECX
.  FF15 9C61400 CALL DWORD PTR DS:[<&QtCore4.QString::fromWCharArray>]
.  83C4 0C      ADD ESP,0C
.  8BC8         MOV ECX,EAX
.  FF15 E060400 CALL DWORD PTR DS:[<&QtCore4.QString::toLocal8Bit>]
.  8B5424 08    MOV EDX,DWORD PTR SS:[ESP+8]
.  83C8 FF      OR EAX,FFFFFFFF
.  F0:0FC102    LOCK XADD DWORD PTR DS:[EDX],EAX
.  75 0E        JNZ SHORT sdi.00404B92
.  8B4C24 08    MOV ECX,DWORD PTR SS:[ESP+8]
.  51           PUSH ECX
.  FF15 9061400 CALL DWORD PTR DS:[<&QtCore4.QString::free>]
.  83C4 04      ADD ESP,4
>  6A 08        PUSH 8
.  8D4C24 04    LEA ECX,DWORD PTR SS:[ESP+4]
.  C74424 10 00 MOV DWORD PTR SS:[ESP+10],0
.  E8 2BFFFFFF  CALL sdi.00404AD0
.  8D4C24 04    LEA ECX,DWORD PTR SS:[ESP+4]
.  FF15 DC60400 CALL DWORD PTR DS:[<&QtCore4.QByteArray::detach>]
.  8B4C24 20    MOV ECX,DWORD PTR SS:[ESP+20]
```

2.4. And we're in QMain

```
$  6A FF        PUSH -1
.  68 864C4000  PUSH sdi.00404C86
.  64:A1 000000 MOV EAX,DWORD PTR FS:[0]
.  50           PUSH EAX
.  83EC 10      SUB ESP,10
.  53           PUSH EBX
.  56           PUSH ESI
.  57           PUSH EDI                                  ntdll.7C910738
.  A1 10D14000  MOV EAX,DWORD PTR DS:[40D110]
.  33C4         XOR EAX,ESP
.  50           PUSH EAX
.  8D4424 20    LEA EAX,DWORD PTR SS:[ESP+20]
.  64:A3 000000 MOV DWORD PTR FS:[0],EAX
.  E8 152D0000  CALL sdi.00403D40
.  8B4424 34    MOV EAX,DWORD PTR SS:[ESP+34]             sdi.<ModuleEntryPoin
.  68 02060400  PUSH 40602
.  50           PUSH EAX
.  8D4C24 38    LEA ECX,DWORD PTR SS:[ESP+38]
.  51           PUSH ECX
.  8D4C24 24    LEA ECX,DWORD PTR SS:[ESP+24]
.  FF15 C863400 CALL DWORD PTR DS:[<&QtGui4.QApplication::QApplication>]  QtGui4.QApplication:
.  8B35 8861400 MOV ESI,DWORD PTR DS:[<&QtCore4.QString::fromAscii_helper>]  QtCore4.QString::fro
.  6A FF        PUSH -1
.  33DB         XOR EBX,EBX
.  68 24644000  PUSH sdi.00406424                         ASCII "SDI Example"
.  895C24 30    MOV DWORD PTR SS:[ESP+30],EBX
.  FFD6         CALL ESI
.  894424 18    MOV DWORD PTR SS:[ESP+18],EAX
.  8D5424 18    LEA EDX,DWORD PTR SS:[ESP+18]
.  52           PUSH EDX                                  ntdll.KiFastSystemCa
.  C64424 34 01 MOV BYTE PTR SS:[ESP+34],1
.  FF15 8C61400 CALL DWORD PTR DS:[<&QtCore4.QCoreApplication::setApplicationN  QtCore4.QCoreApplica
.  8B4424 1C    MOV EAX,DWORD PTR SS:[ESP+1C]
```

2.5. The startup address is usually pointing to the start of the .CODE section

00400000	00001000	sdi		PE header	Imag	R	RWE
00401000	00005000	sdi	.text	code	Imag	R	RWE
00406000	00007000	sdi	.rdata	imports	Imag	R	RWE
0040D000	00001000	sdi	.data	data	Imag	R	RWE
0040E000	00001000	sdi	.rsrc	resources	Imag	R	RWE

That is 401000 in this case.

3. Code analysis

3.1. Instruction 1 – Initialize resources

```
    Q_INIT_RESOURCE(sdi);
    QApplication app(argc, argv);
    app.setApplicationName("SDI Example");
```

```
8D4424 20     LEA EAX,DWORD PTR SS:[ESP+20]
64:A3 000000  MOV DWORD PTR FS:[0],EAX
E8 152D0000   CALL sdi.00403D40
8B4424 34     MOV EAX,DWORD PTR SS:[ESP+34]
68 02060400   PUSH 40602
50            PUSH EAX
8D4C24 38     LEA ECX,DWORD PTR SS:[ESP+38]
51            PUSH ECX
8D4C24 24     LEA ECX,DWORD PTR SS:[ESP+24]
FF15 C863400  CALL DWORD PTR DS:[<&QtGui4.QApplication::QApplication>]
```

If we enter the function, we should see the resources being pushed and the **qRegisterResourceData** function being called:

```
68 206C4000   PUSH sdi.00406C20
68 208D4000   PUSH sdi.00408D20
68 B88D4000   PUSH sdi.00408DB8
6A 01         PUSH 1
FF15 EC60400  CALL DWORD PTR DS:[<&QtCore4.qRegisterResourceData>]
83C4 10       ADD ESP,10
B8 01000000   MOV EAX,1
C3            RETN
```

3.2. Instruction 2 – Instantiate QApplication

```
Q_INIT_RESOURCE(sdi);
QApplication app(argc, argv);
app.setApplicationName("SDI Example");
```

```
8B4424 34     MOV EAX,DWORD PTR SS:[ESP+34]
68 02060400   PUSH 40602
50            PUSH EAX
8D4C24 38     LEA ECX,DWORD PTR SS:[ESP+38]
51            PUSH ECX
8D4C24 24     LEA ECX,DWORD PTR SS:[ESP+24]
FF15 C863400  CALL DWORD PTR DS:[<&QtGui4.QApplication::QApplication>]
```

```
0012FF18  Arg1 = 0012FF18
003D59F0  Arg2 = 003D59F0
00040602  Arg3 = 00040602
```

Pretty easy to follow right? The **QApplication** class is calling it's constructor with the following arguments: Arg1 = argc, Arg2 = argv, Arg3 = compile_version

3.3. Instruction 3

```
Q_INIT_RESOURCE(sdi);
QApplication app(argc, argv);
app.setApplicationName("SDI Example");
app.setOrganizationName("Trolltech");
```

```
33DB          XOR EBX,EBX
68 24644000   PUSH sdi.00406424                    ASCII "SDI Example"
895C24 30     MOV DWORD PTR SS:[ESP+30],EBX
FFD6          CALL ESI                            QtCore4.QString::fr
894424 18     MOV DWORD PTR SS:[ESP+18],EAX
8D5424 18     LEA EDX,DWORD PTR SS:[ESP+18]
52            PUSH EDX                            QtGui4.654D800A
C64424 34 01  MOV BYTE PTR SS:[ESP+34],1
FF15 8C61400  CALL DWORD PTR DS:[<&QtCore4.QCoreApplication::setApplicationName>]  QtCore4.QCoreApplic
8B4424 1C     MOV EAX,DWORD PTR SS:[ESP+1C]
```

3.4. Instruction 4

```
    app.setApplicationName("SDI Example");
    app.setOrganizationName("Trolltech");
    MainWindow *mainWin = new MainWindow;
    mainWin->show();
    return app.exec();
```

```
PUSH -1
PUSH sdi.00406418                                   ASCII "Trolltech"
CALL ESI                                            QtCore4.QString::f
MOV DWORD PTR SS:[ESP+18],EAX
LEA EAX,DWORD PTR SS:[ESP+18]
PUSH EAX
MOV BYTE PTR SS:[ESP+34],2
CALL DWORD PTR DS:[<&QtCore4.QCoreApplication::setOrganizationName>]   QtCore4.QCoreAppli
MOV ECX,DWORD PTR SS:[ESP+1C]
ADD ESP,0C
MOV BYTE PTR SS:[ESP+28],BL
```

3.5. Instruction 5

```
    app.setOrganizationName("Trolltech");
    MainWindow *mainWin = new MainWindow;
    mainWin->show();
    return app.exec();
```

```
CALL <JMP.&MSVCR90.operator new>
ADD ESP,4
MOV DWORD PTR SS:[ESP+14],EAX
MOV BYTE PTR SS:[ESP+28],3
CMP EAX,EBX
JE SHORT sdi.004010EC
MOV ECX,EAX
CALL sdi.00402A60
```

This function calls all the subclasses constructors

```
MOV DWORD PTR SS:[ESP+14],ESP
PUSH 0
MOV DWORD PTR DS:[EAX],0
CALL DWORD PTR DS:[<&QtGui4.QMainWindow::QMainWindow>]
MOV DWORD PTR DS:[ESI],sdi.0040699C
MOV DWORD PTR DS:[ESI+8],sdi.0040697C
```

3.6. Instruction 6

```
    app.setOrganizationName("Trolltech");
    MainWindow *mainWin = new MainWindow;
    mainWin->show();
    return app.exec();
```

```
MOV ECX,EAX
MOV BYTE PTR SS:[ESP+28],BL
CALL DWORD PTR DS:[<&QtGui4.QWidget::show>]
```

3.7. Instruction 7

```
    MainWindow *mainWin = new MainWindow;
    mainWin->show();
    return app.exec();
}
```

```
CALL DWORD PTR DS:[<&QtGui4.QWidget::show>]
CALL DWORD PTR DS:[<&QtGui4.QApplication::exec>]
LEA ECX,DWORD PTR SS:[ESP+18]
```

4. Intercept Messages in Qt

Messages in Qt are set thru slots definition. Slots are then connected using the connect macro to a **QWidget** based class. As sampled in the code:

```
saveAsAct = new QAction(tr("Save &As..."), this);
saveAsAct->setShortcuts(QKeySequence::SaveAs);
saveAsAct->setStatusTip(tr("Save the document under a new name"));
connect(saveAsAct, SIGNAL(triggered()), this, SLOT(saveAs()));

closeAct = new QAction(tr("&Close"), this);
closeAct->setShortcut(tr("Ctrl+W"));
closeAct->setStatusTip(tr("Close this window"));
connect(closeAct, SIGNAL(triggered()), this, SLOT(close()));

exitAct = new QAction(tr("E&xit"), this);
exitAct->setShortcuts(QKeySequence::Quit);
exitAct->setStatusTip(tr("Exit the application"));
connect(exitAct, SIGNAL(triggered()), qApp, SLOT(closeAllWindows()));
```

So we have here 3 types of message redirection, the first one "**saveAsAct**" is defined in the slot set of the **MainWindow** class. "**saveAsAct**" will be made part of the **MainWindow** class dispatcher table.

```
//! [class definition with macro]
class MainWindow : public QMainWindow
{
    Q_OBJECT
```

The second one, "**closeAct**", is defined as an overload of **QMainWindow** superclass, so it will not be part of **MainWindow** class dispatcher table, but will be dispatched thru **MainWindow** dispatcher function. But as "**closeAct**" is not found in **MainWindow** class dispatcher table it will be forwarded to **QMainWindow** superclass dispatcher using an **Event** class. The definition of "**closeAct**" is:

```
protected:
    void closeEvent(QCloseEvent *event);
```

Finally "**exitAct**" is applied or connected to **qApp**. **qApp** is an instance of **QApplication**, so it can't be intercepted in the **MainWindow** dispatcher function, but in **QApplication** dispatcher function.

Considering that everything was easy till now, this will be the "hard" part of the process, getting to trace the GDI interaction. Let's see how we can accomplish this task. Basically there are three approaches that can be used:

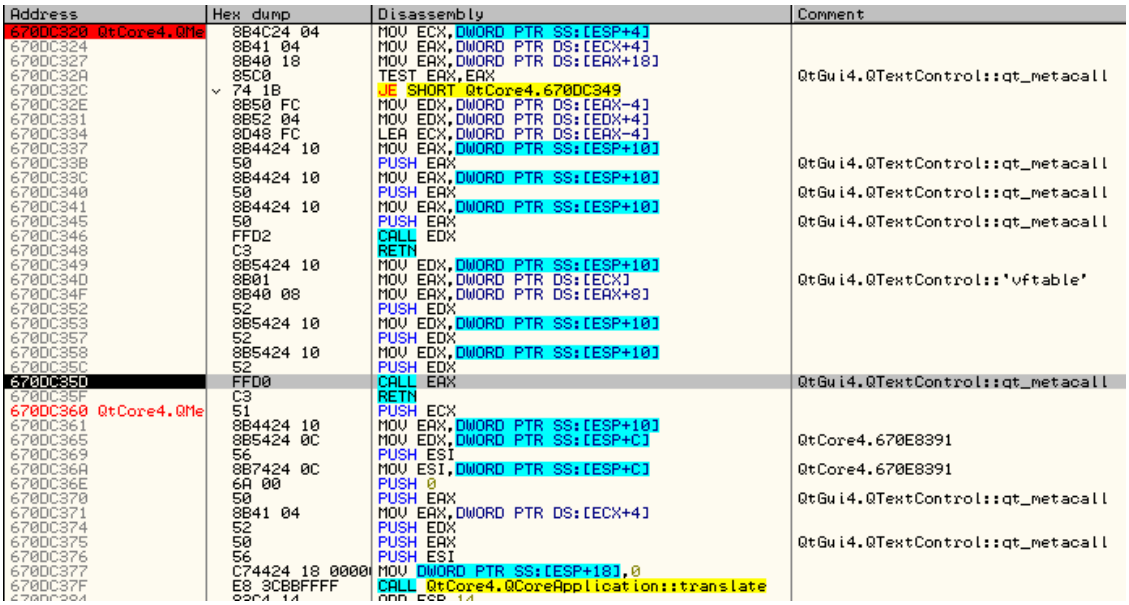
4.1. Find and set a conditional breakpoint at **QMetaObject::metacall** with [esp+4] == object addr

The object is any GDI particular object like a QTextControl, QMainWindow, etc. that you might know it's memory address.

4.2. Set a conditional breakpoint on **Q<class of object>::qt_metacall** with [esp+8] == WM_

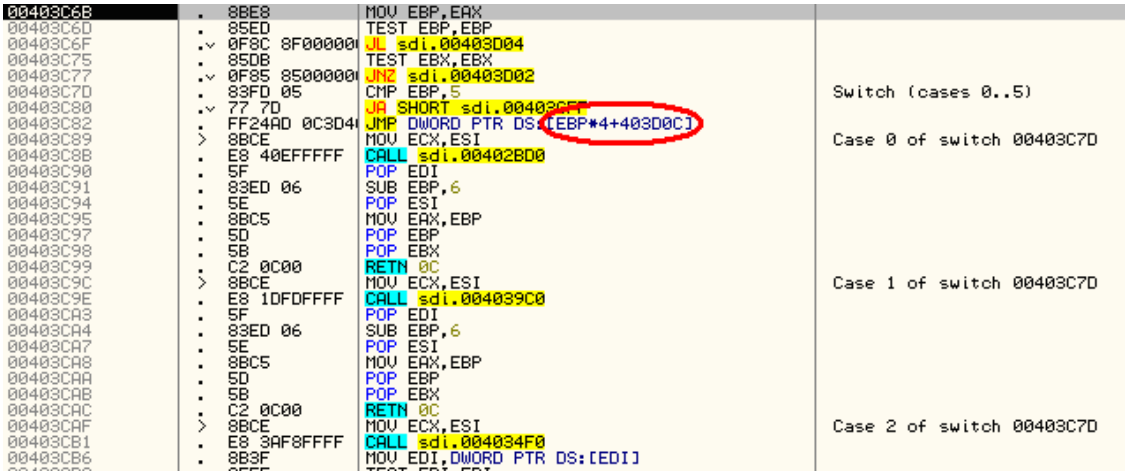
4.3. Set a breakpoint on **QMainWindow::qt_metacall** (QMainWindow is defined in QtGui4.dll).

On stop set a breakpoint on .code section of main executable, it should stop on the Qt dispatcher.



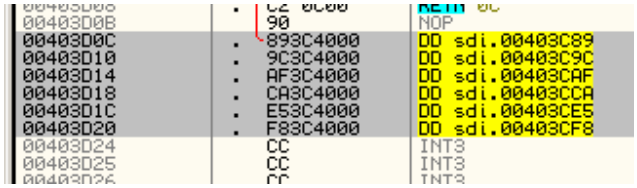
Address	Hex dump	Disassembly	Comment
670DC320 QtCore4.QMe	8B4C24 04	MOV ECX,DWORD PTR SS:[ESP+4]	
670DC324	8B41 04	MOV EAX,DWORD PTR DS:[ECX+4]	
670DC327	8B40 18	MOV EAX,DWORD PTR DS:[EAX+18]	
670DC32A	85C0	TEST EAX,EAX	QtGui4.QTextControl::qt_metacall
670DC32C	74 1B	JE SHORT QtCore4.670DC349	
670DC32E	8B50 FC	MOV EDX,DWORD PTR DS:[EAX-4]	
670DC331	8B52 04	MOV EDX,DWORD PTR DS:[EDX+4]	
670DC334	8D48 FC	LEA ECX,DWORD PTR DS:[EAX-4]	
670DC337	8B4424 10	MOV EAX,DWORD PTR SS:[ESP+10]	
670DC33B	50	PUSH EAX	QtGui4.QTextControl::qt_metacall
670DC33C	8B4424 10	MOV EAX,DWORD PTR SS:[ESP+10]	
670DC340	50	PUSH EAX	QtGui4.QTextControl::qt_metacall
670DC341	8B4424 10	MOV EAX,DWORD PTR SS:[ESP+10]	
670DC345	50	PUSH EAX	QtGui4.QTextControl::qt_metacall
670DC346	FFD2	CALL EDX	
670DC348	C3	RETN	
670DC349	8B5424 10	MOV EDX,DWORD PTR SS:[ESP+10]	
670DC34D	8B01	MOV EAX,DWORD PTR DS:[ECX]	QtGui4.QTextControl::'vftable'
670DC34F	8B40 08	MOV EAX,DWORD PTR DS:[EAX+8]	
670DC352	52	PUSH EDX	
670DC353	8B5424 10	MOV EDX,DWORD PTR SS:[ESP+10]	
670DC357	52	PUSH EDX	
670DC358	8B5424 10	MOV EDX,DWORD PTR SS:[ESP+10]	
670DC35C	52	PUSH EDX	
670DC35D	FFD0	CALL EAX	QtGui4.QTextControl::qt_metacall
670DC35F	C3	RETN	
670DC360 QtCore4.QMe	51	PUSH ECX	
670DC361	8B4424 10	MOV EAX,DWORD PTR SS:[ESP+10]	
670DC365	8B5424 0C	MOV EDX,DWORD PTR SS:[ESP+C]	QtCore4.670E8391
670DC369	56	PUSH ESI	
670DC36A	8B7424 0C	MOV ESI,DWORD PTR SS:[ESP+C]	QtCore4.670E8391
670DC36E	6A 00	PUSH 0	
670DC370	50	PUSH EAX	QtGui4.QTextControl::qt_metacall
670DC371	8B41 04	MOV EAX,DWORD PTR DS:[ECX+4]	
670DC374	52	PUSH EDX	
670DC375	50	PUSH EAX	QtGui4.QTextControl::qt_metacall
670DC376	56	PUSH ESI	
670DC377	C74424 18 0000	MOV DWORD PTR SS:[ESP+18],0	
670DC37F	E8 3CBBFFFF	CALL QtCore4.QCoreApplication::translate	

In the main Qt dispatcher built by the compiler, Olly tells us that there are 6 cases on switch.



00403C6B	8BE8	MOV EBP,EAX	
00403C6D	85ED	TEST EBP,EBP	
00403C6F	0F8C 8F00000(	JL sdi.00403D04	
00403C75	85DB	TEST EBX,EBX	
00403C77	0F85 8500000(	JNZ sdi.00403D02	
00403C7D	83FD 05	CMP EBP,5	Switch (cases 0..5)
00403C80	77 7D	JA SHORT sdi.00403CFF	
00403C82	FF24AD 0C3D40(	JMP DWORD PTR DS:[EBP*4+403D0C]	
00403C89	8BCE	MOV ECX,ESI	Case 0 of switch 00403C7D
00403C8B	E8 40EFFFFF	CALL sdi.00402BD0	
00403C90	5F	POP EDI	
00403C91	83ED 06	SUB EBP,6	
00403C94	5E	POP ESI	
00403C95	8BC5	MOV EAX,EBP	
00403C97	5D	POP EBP	
00403C98	5B	POP EBX	
00403C99	C2 0C00	RETN 0C	
00403C9C	8BCE	MOV ECX,ESI	Case 1 of switch 00403C7D
00403C9E	E8 1DFDFFFF	CALL sdi.004039C0	
00403CA3	5F	POP EDI	
00403CA4	83ED 06	SUB EBP,6	
00403CA7	5E	POP ESI	
00403CA8	8BC5	MOV EAX,EBP	
00403CAA	5D	POP EBP	
00403CAB	5B	POP EBX	
00403CAC	C2 0C00	RETN 0C	
00403CAF	8BCE	MOV ECX,ESI	Case 2 of switch 00403C7D
00403CB1	E8 3AF8FFFF	CALL sdi.004034F0	
00403CB6	8B3F	MOV EDI,DWORD PTR DS:[EDI]	

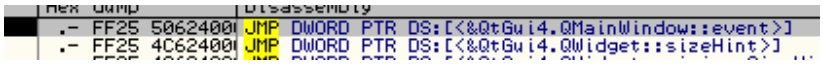
The six destinations are set here:

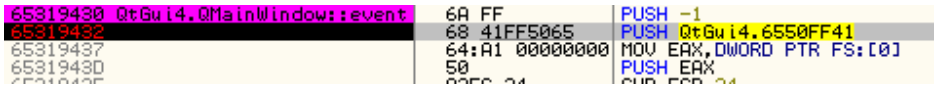


Which correspond to the slots definitions of MainWindow class:

```
private slots:
    void newFile();
    void open();
    bool save();
    bool saveAs();
    void about();
    void documentWasModified();
```

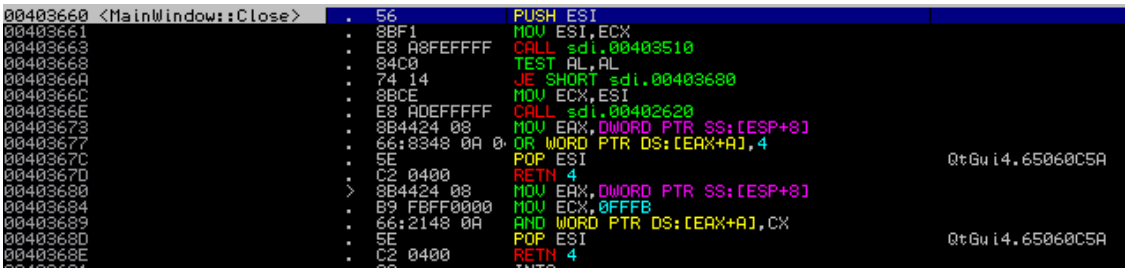
4.4. To target messages attached to events, set a breakpoint on **QMainWindow::qt_metacall**, on stop set a breakpoint on **QMainWindow::event**. After break set a new breakpoint in the .code section of main executable, it should stop on the Qt function you're after.



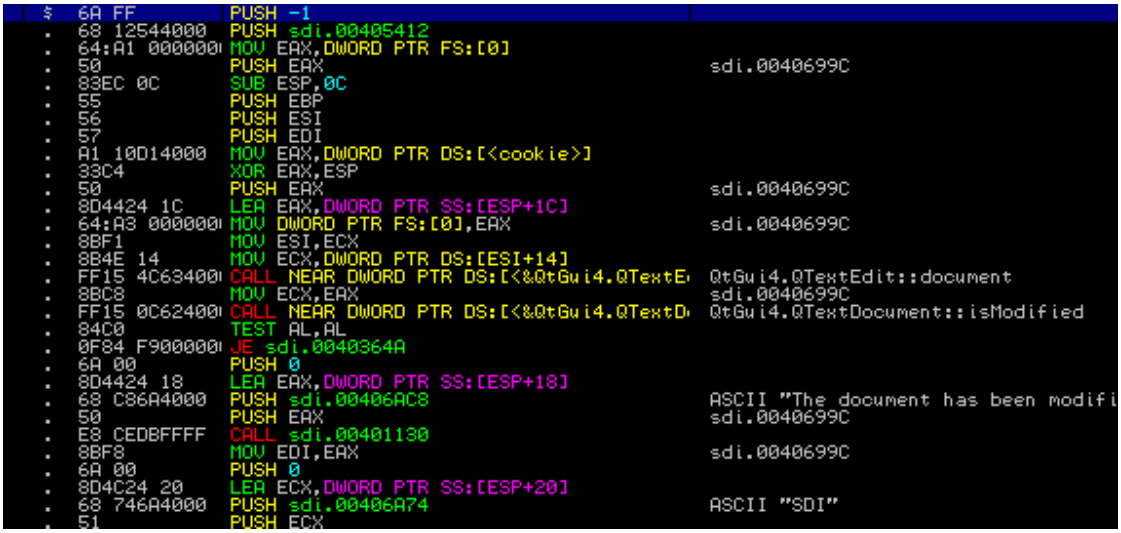


For example, tracing the **MainWindow::Close**, after following the steps defined before we reach:

```
void MainWindow::closeEvent(QCloseEvent *event)
{
    if (maybeSave()) {
        writeSettings();
        event->accept();
    } else {
        event->ignore();
    }
}
```



Just to confirm we're in the right place let's enter the first call:



Which is maybeSave() function who's definition is:

```
bool MainWindow::maybeSave()
{
    if (textEdit->document()->isModified()) {
    QMessageBox::StandardButton ret;
        ret = QMessageBox::warning(this, tr("SDI"),
                     tr("The document has been modified.\n"
                        "Do you want to save your changes?"),
                     QMessageBox::Save | QMessageBox::Discard
              | QMessageBox::Cancel);
        if (ret == QMessageBox::Save)
            return save();
        else if (ret == QMessageBox::Cancel)
            return false;
    }
    return true;
```

See the corresponding text? We're in the right place.

For the sake of completeness, there's a caveat you need to be aware regarding the first approach I presented: Setting a breakpoint in **QMetaObject::metacall**. After entering

QMetaObject::metacall and setting a breakpoint on the user section code of the main module, it won't enter immediately in the function code we're after; instead the metacall will just try to find the virtual stub function responsible for the dispatching. For resolving this, all you need is let the code enter in the Qt core dispatcher and then set a second section code breakpoint in the main module and finally you'll be there.

Ufff… Good luck in you reversing.

Hope you enjoyed it!