AUG 2, 2018 - 7 MINUTE READ - COMMENTS - REVERSE ENGINEERING DVTA WRITEUP

DVTA - Part 4 - Traffic Tampering with dnSpy

Who am I?

I am Parsia, a security engineer at Electronic Arts.

I write about application security, reverse engineering, Go, cryptography, and (obviously) videogames.

Click on About Me! to know more.

Collections

After doing network recon in part three, it's time to do some traffic manipulation. We will learn how to capture and modify network traffic using dnSpy. This is much easier than trying to intercept and modify traffic after it's transmitted.

Previous parts are at:

- DVTA - Part 1 - Setup
- DVTA - Part 2 - Cert Pinning and Login Button
- DVTA - Part 3 - Network Recon

General Traffic Manipulation Intro

Previously we used Wireshark to capture network traffic. Passive sniffing is usually easy but only useful to a degree. If the application was using TLS, we would have seen garbage after the TLS handshake [1]. In these cases, Man-in-the-Middling (MitM-ing) the traffic with a proxy tool (e.g. Burp) is usually the way to go. But that introduces new challenges.

1. Redirecting the traffic to the proxy.
2. Masquerading as the server (e.g. make client accept our proxy's certificate instead of server).
3. Modifying packets.

I will need a lot of pages to talk about these and document what I have learned through the years. This is not the place for it.

Depending on the interception method, you can bypass some of these challenges. For example, by hooking application's function calls that send the data, you can omit the first two

Thick Client Proxying
Go/Golang
Blockchain/Distributed Ledgers
Automation
Reverse Engineering
Crypto(graphy)
CTFs/Writeups
WinAppDbg
AWSome.pw - S3 bucket squatting - my very legit branded vulnerability

(traffic redirection and server emulation). This is exactly what we are going to do to manipulate traffic in two ways:

1. Debugging with dnSpy - this part.
2. ~~Hooking with WinAppDbg - next part.~~ Seems like this is much harder than I expected. I need to learn about hooking .NET functions with WinAppDbg (or in general). Added to my TODO list.

Debugging with dnSpy

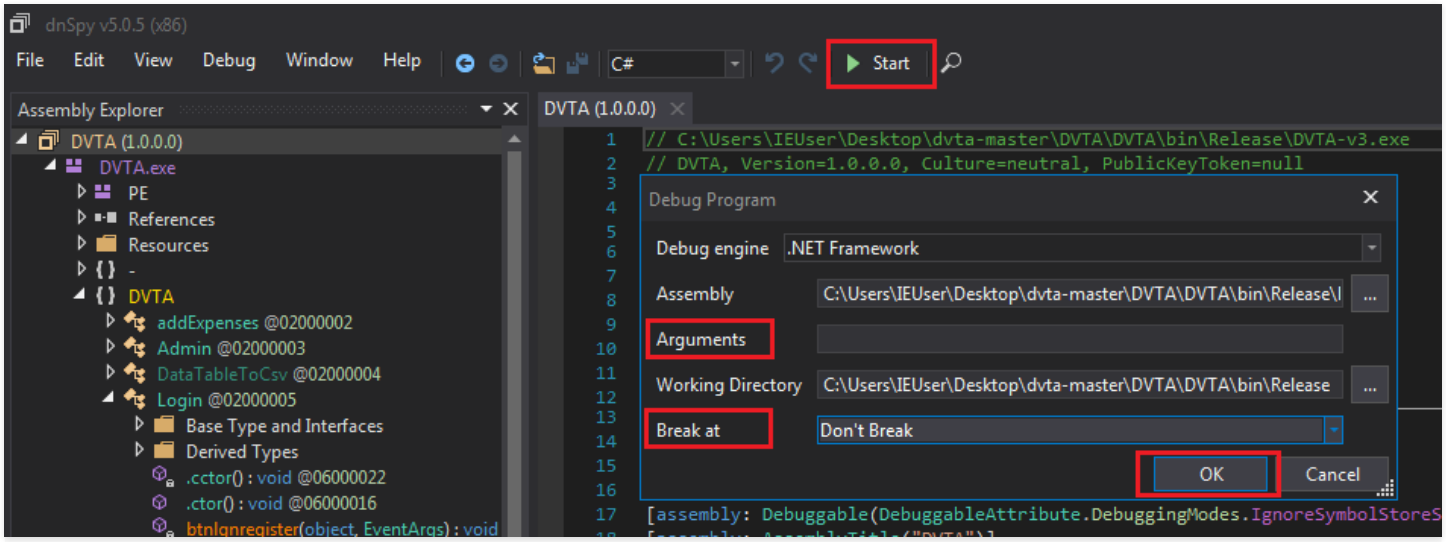
My first interaction with dnSpy was when version 1 was released. I used it to modify the outgoing traffic and make myself admin. It was one of my first thick client tests and I was so proud of myself. We are going to do the same here. We will debug the application with dnSpy and then view/modify the outgoing data. We need to:

1. Identify the function/code where data is assembled before transmission.
2. Set a breakpoint.
3. Debug the application with dnSpy.
4. Use the application.
5. Modify the traffic when the breakpoint is triggered.
6. ???
7. Profit.

Putting a breakpoint where the traffic is being transmitted is also viable in some use-cases. But in this case with the direct connection to MSSQL server, we want to manipulate queries.

Login

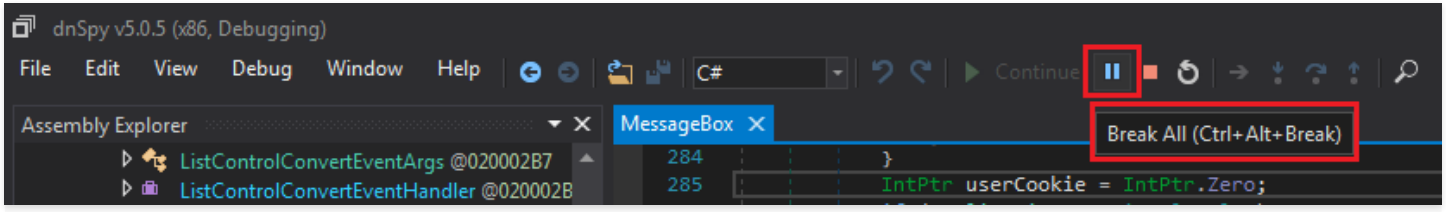
We will start with the login request. We already know where it happens but let's pretend we do not[2]. Drag and drop `dvta.exe` into dnSpy. Then click on `Start`. Note the dialog box allows you to enter command line parameters and set initial breakpoints. None is needed in our case so we will just press `Ok`.



Starting the application with dnSpy

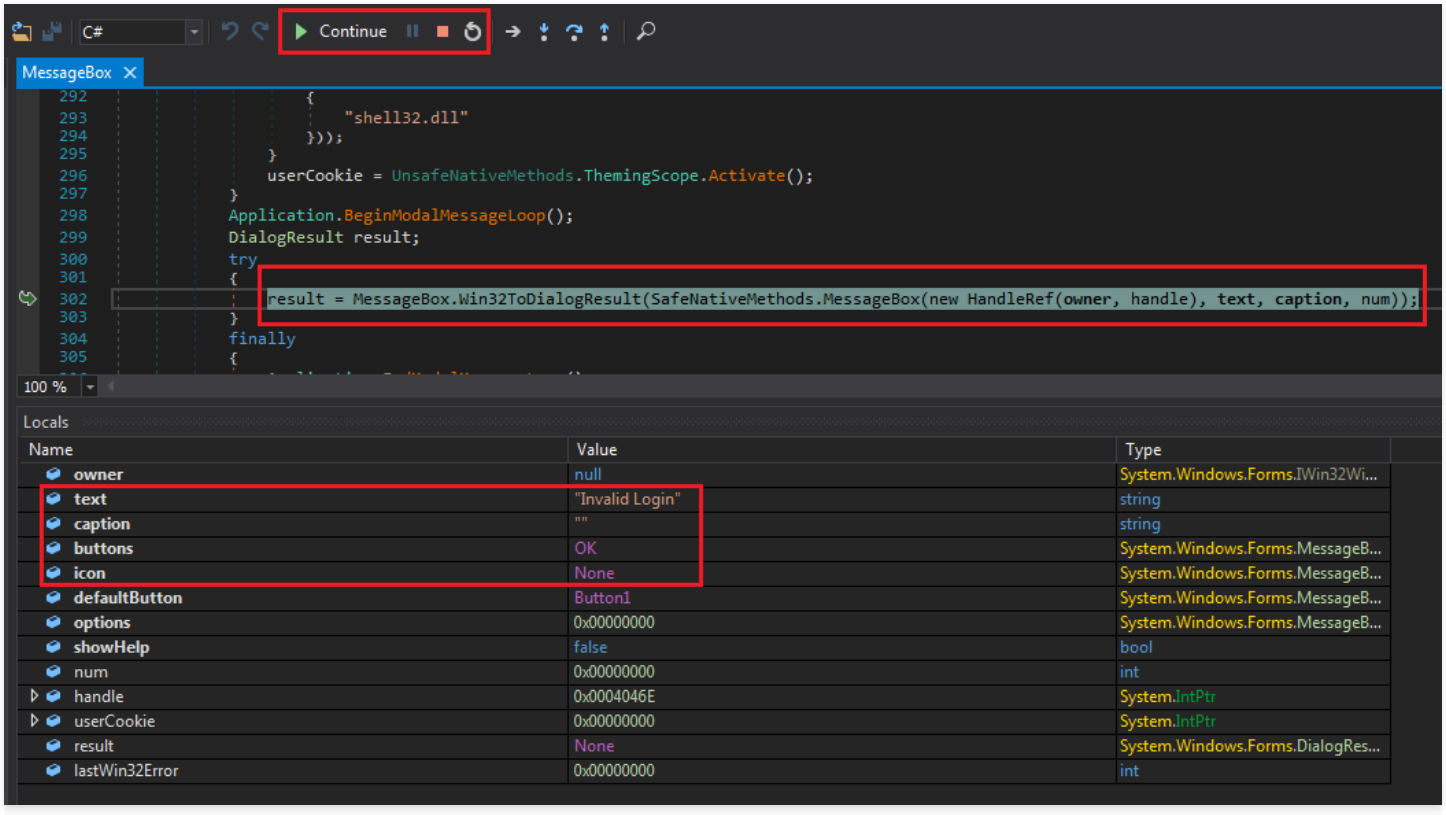
The anti-debug does not get triggered. We could have easily removed it anyway. Fetch the login token and try to login with dummy credentials. After it fails, do not close the `Invalid Login` button.

In dnSpy click on the pause button (tooltip says `Break All`).



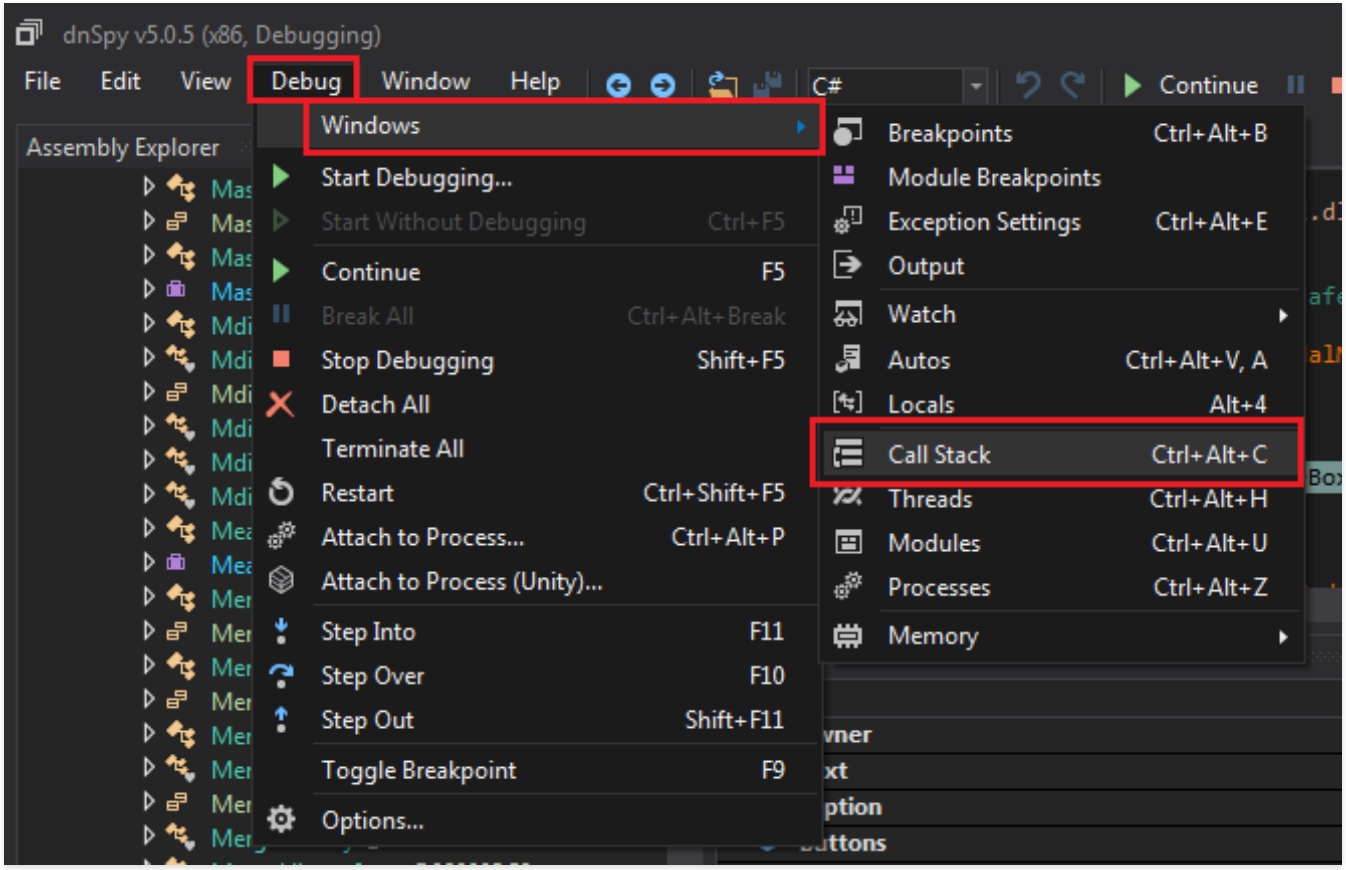
"Break All" button

We break in `System.Windows.Forms.dll > MessageBox`.



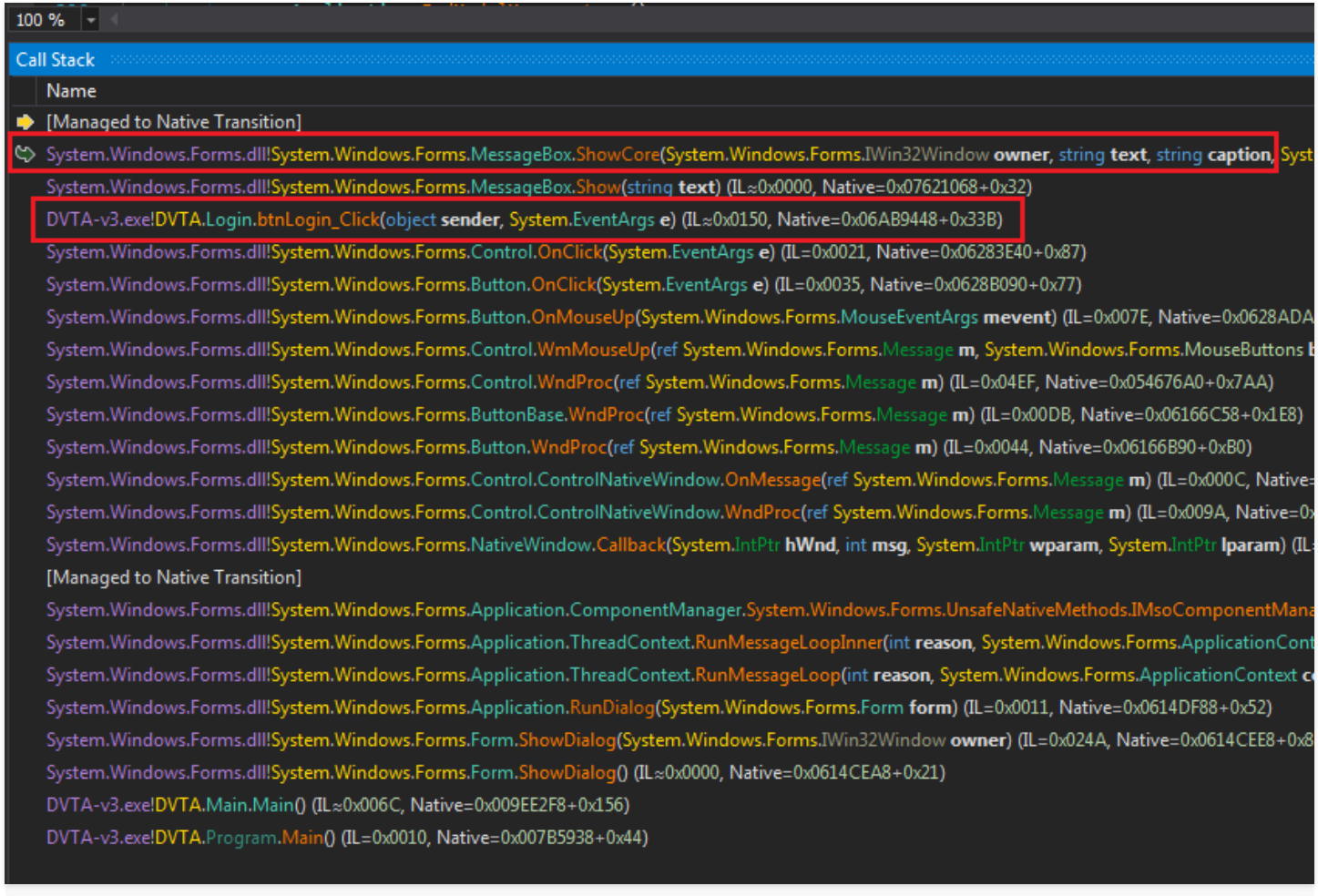
MessageBox break

This is a system DLL and not part of the application. Time for another useful dnSpy feature. Use `Debug (menu) > Windows > Call Stack` or `Ctrl+Alt+C`.



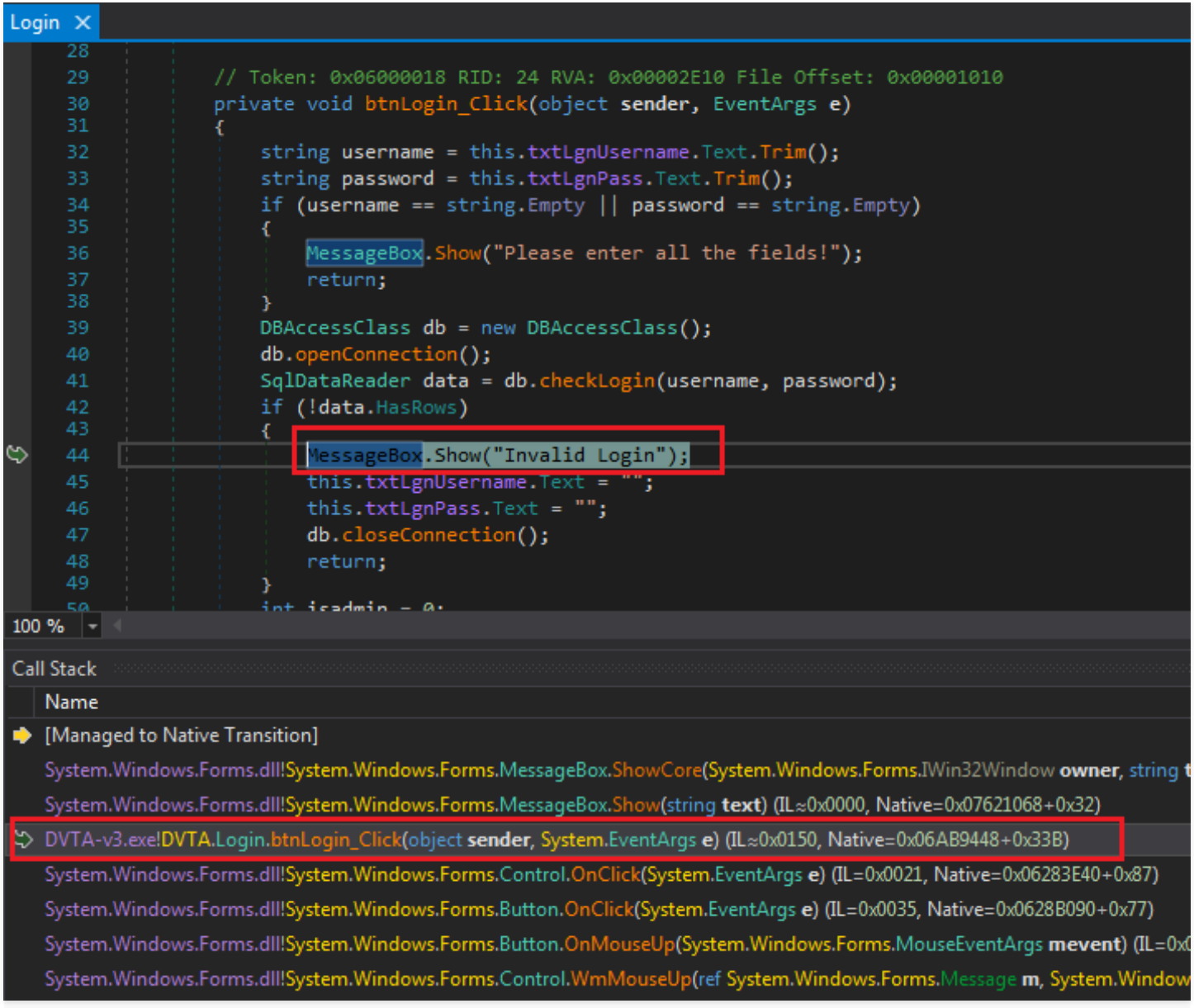
Viewing call Stack

Call stack allows us to see how we got here.



Call stack displayed in dnSpy

`Login.btnLogin_Click` is in the call chain. We can double-click on it to get to the code.



btnLogin_Click

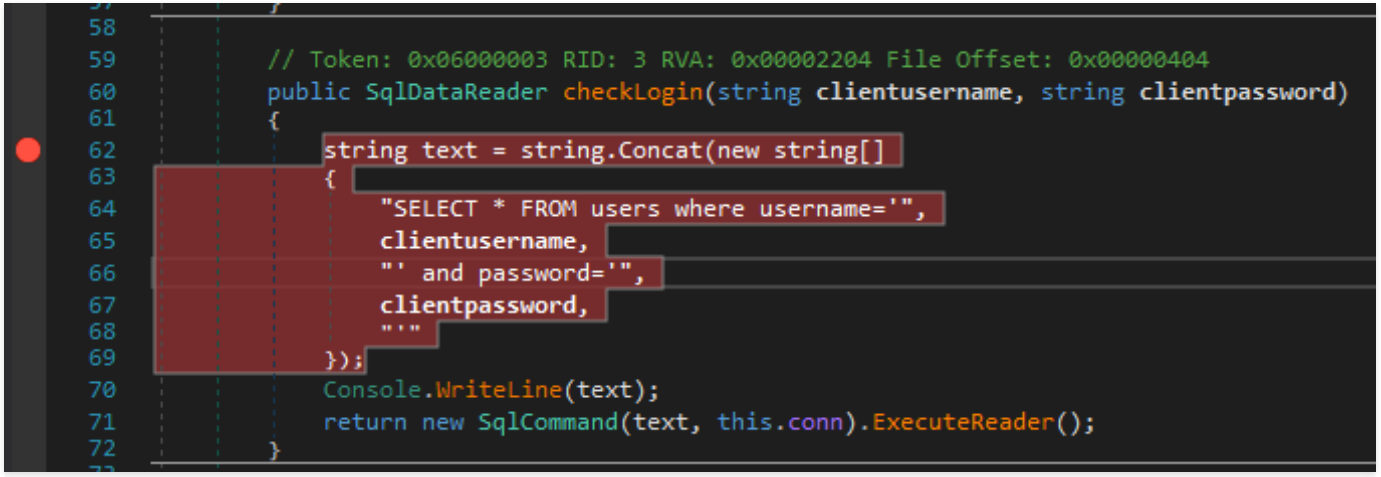
Username and password are passed to `db.checkLogin`. Click on it:

db.checkLogin

```
// Token: 0x06000003 RID: 3 RVA: 0x00002204 File Offset: 0x00000404
public SqlDataReader checkLogin(string clientusername, string clientpassword)
{
    string text = string.Concat(new string[]
    {
        "SELECT * FROM users where username='",
        clientusername,
        "' and password='",
        clientpassword,
        "'"
    });
    Console.WriteLine(text);
    return new SqlCommand(text, this.conn).ExecuteReader();
}
```

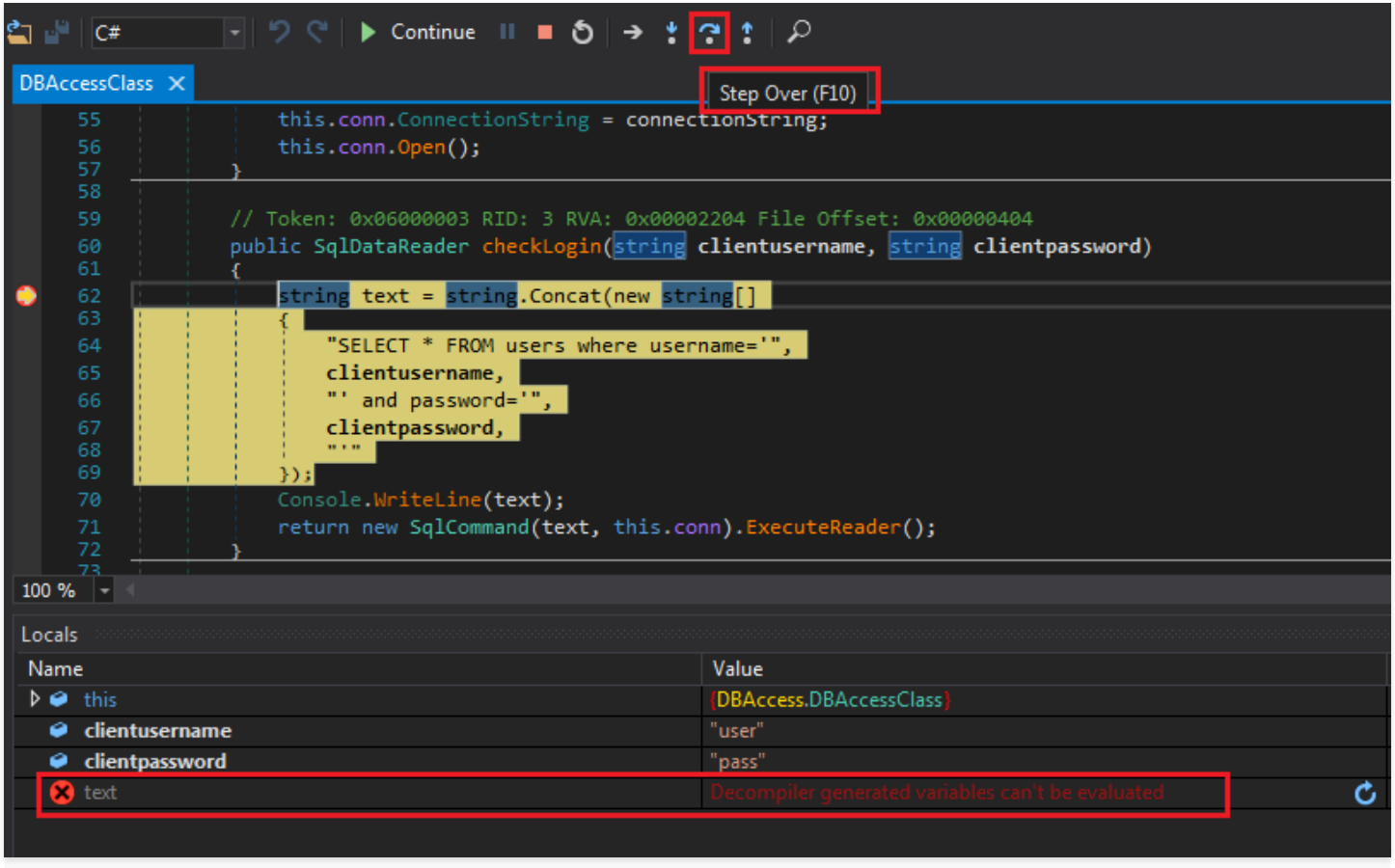
Query is created in a way that is vulnerable to SQL injection (but we were expecting that in a damn vulnerable application). Put a breakpoint here to see the query in action.

Right-click on the `string text =` line and select `Add Breakpoint` or click on the grey edge to the left of the line number (where the red circle is in the following image):



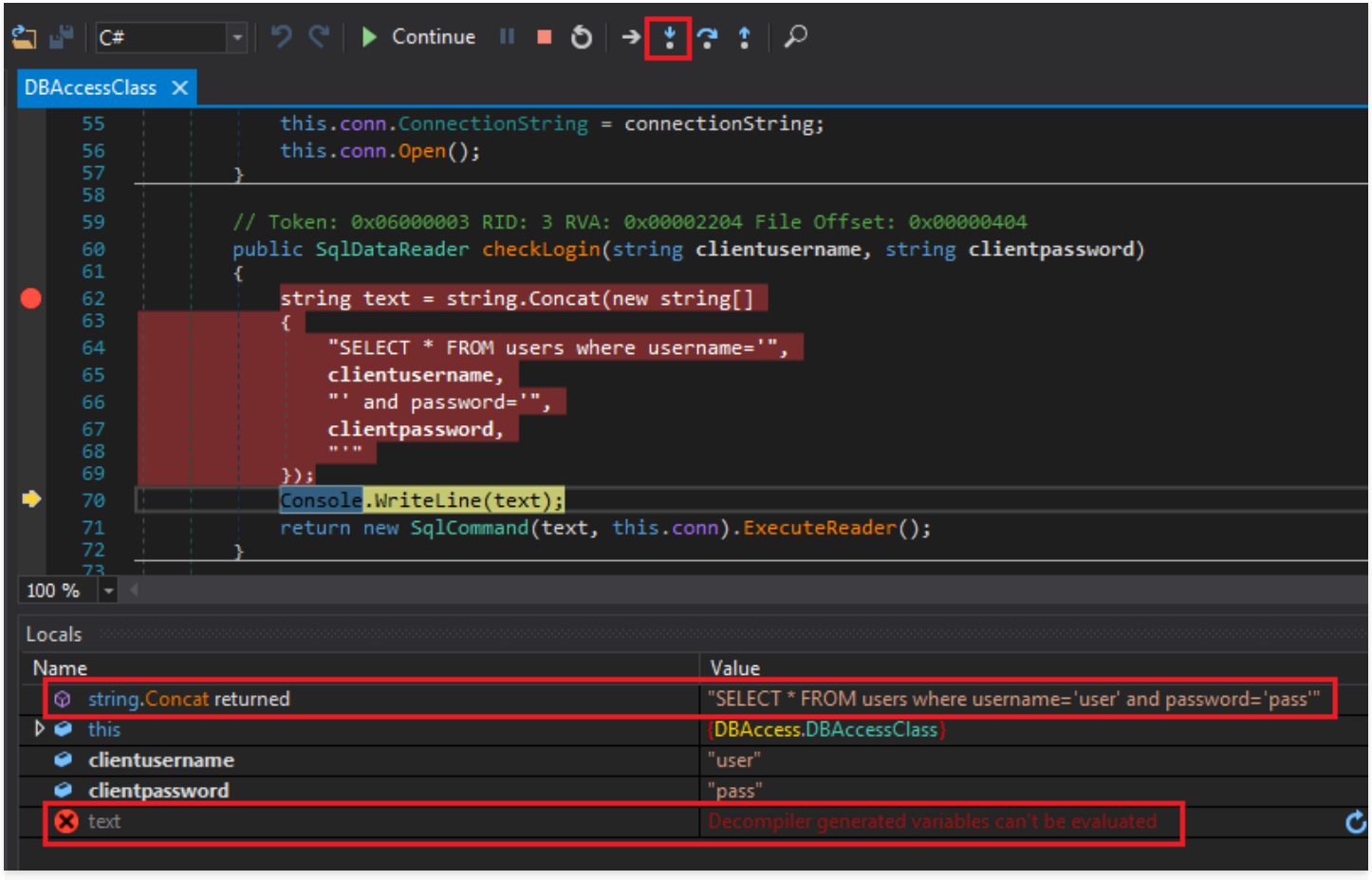
Breakpoint set

Click on `Continue` and try to login again. The breakpoint will get triggered. Close the call stack window and you should see a new window named `Locals`. This window is used to view and modify the value of variables in scope.



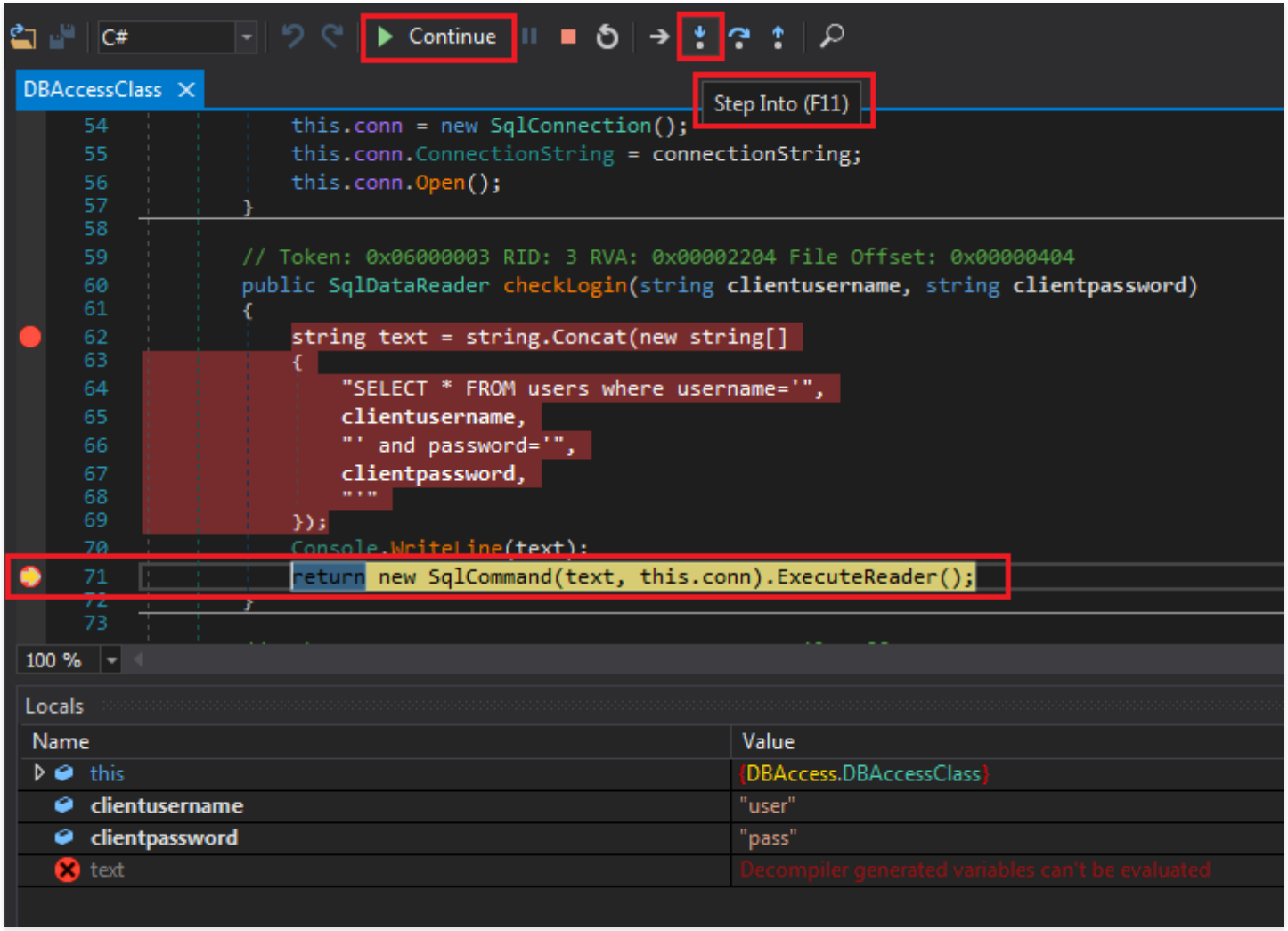
Breakpoint triggered

Like any other debugger, we can `Step Into`, `Step Over`, and the rest of the usual control. You can navigate with the shortcut keys or the buttons to the right of `Start/Continue`. Press `F10` or `Step Over` to get to the next decompiled instruction which is `Console.WriteLine(text);`.



text cannot be modified

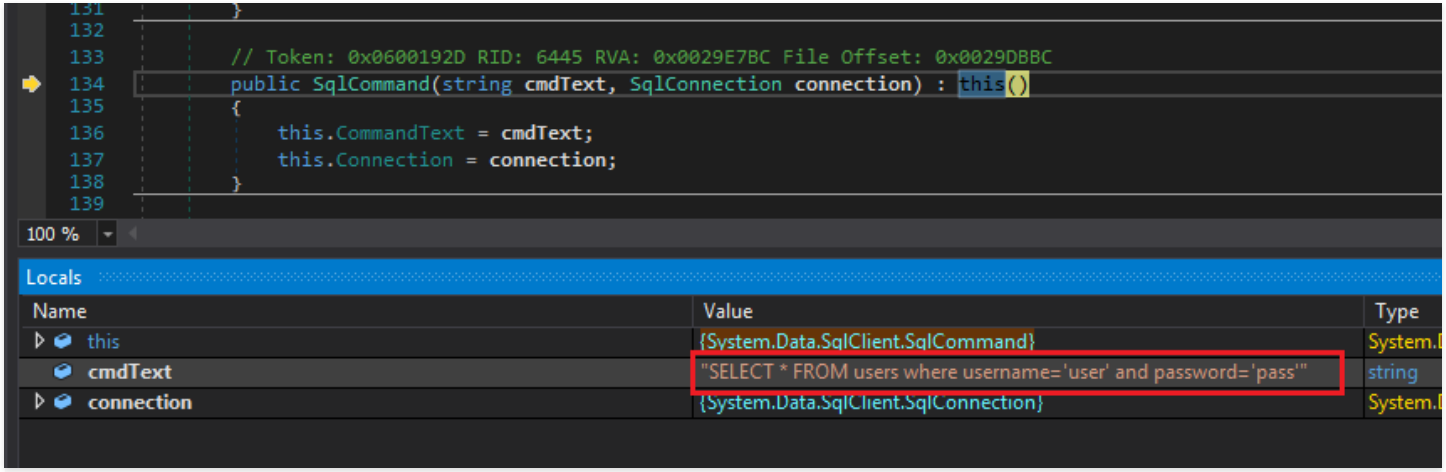
We have a problem inside dnSpy. We cannot modify the value of `text`. The cs0103 error means variable does not exist (e.g. not in scope). I am not sure why this is happening but we can modify the value in a different place. Set a breakpoint on `return new SqlCommand ...` and click `Continue`.



Breakpoint at return triggered

Bypassing Login

This time, we want to jump inside the function call. Click `Step Into`.

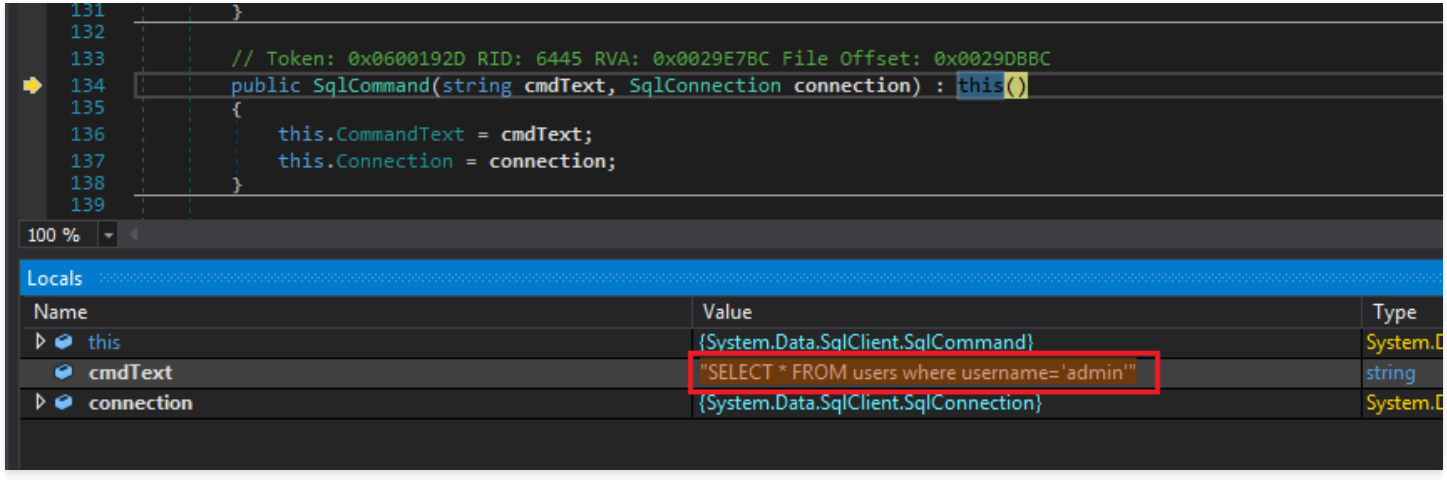


Inside SqlCommand constructor

Here we can modify the value of the query. Double-click on the value in the `Locals` window and type the following (don't forget the double quotes because we are modifying a string):

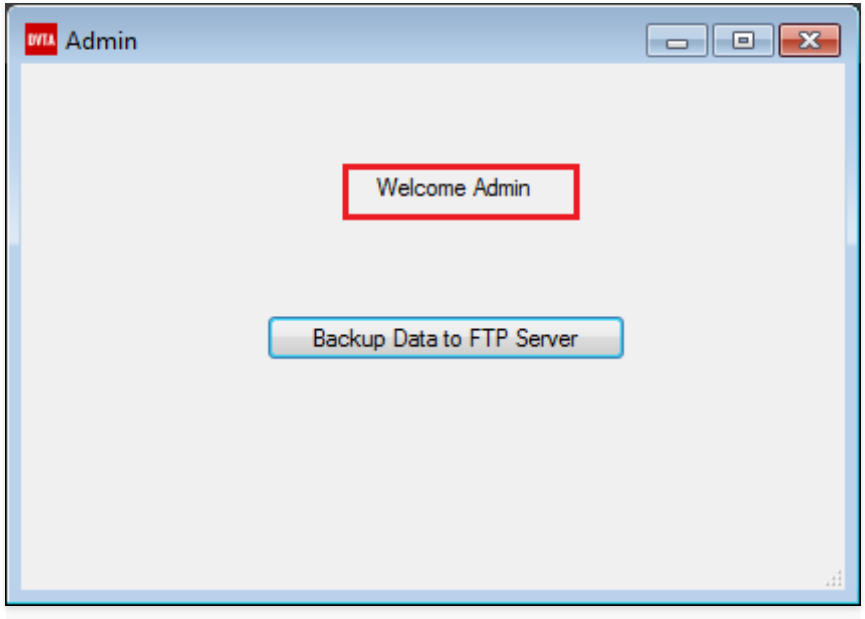
- `"SELECT * FROM users where username='admin'"`

Then press `Enter` and notice the modified value is highlighted:



SQL query modified

Press `Continue` and let this query run. We are logged in as admin.

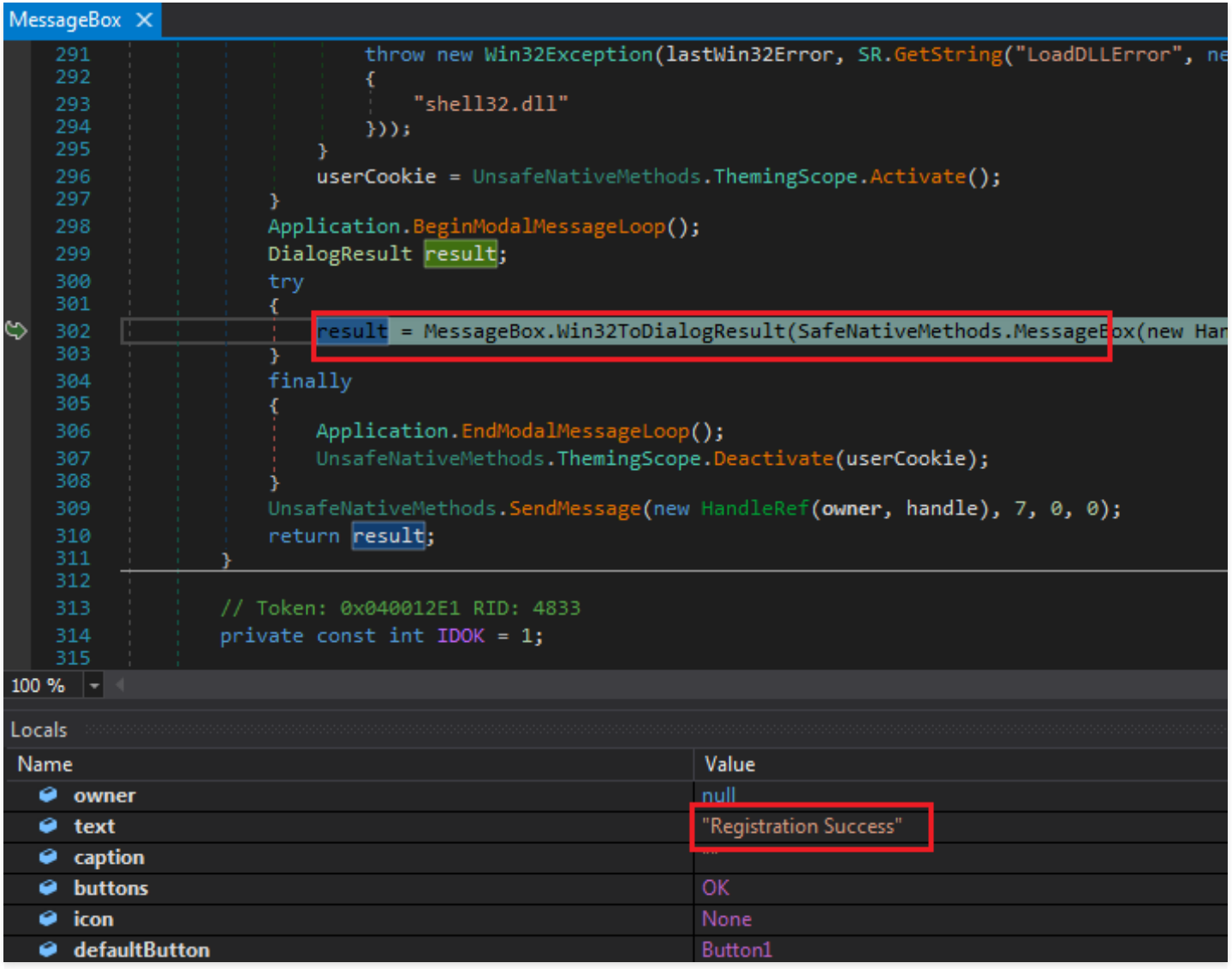


Logged in as admin

Note that we can change this query to anything we want (e.g. `INSERT` or `DELETE`).

Register

Messing with the register function is similar. Run the application with dnSpy and attempt to register any user. Do not close the message box and stop dnSpy with `Break All` like we saw before.



dnSpy after Break All

Next, use the call stack to discover where it was called.

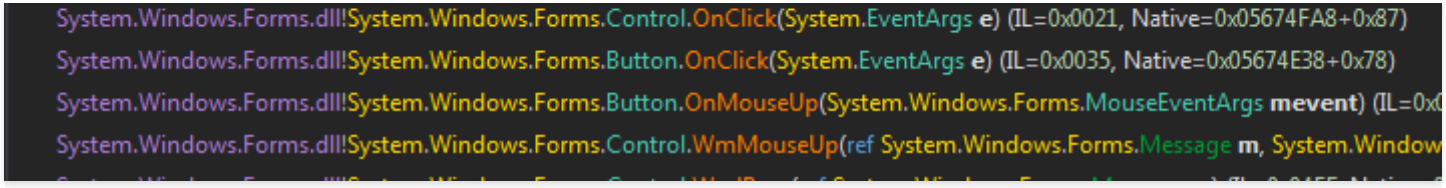
Register

```
45          // Token: 0x06000029 RID: 41 RVA: 0x000035D0 File Offset: 0x000017D0
46          private void btnReg_Click(object sender, EventArgs e)
47          {
48              string username = this.txtRegUsername.Text.Trim();
49              string password = this.txtRegPass.Text.Trim();
50              string confirmpassword = this.txtRegCfmPass.Text.Trim();
51              string email = this.txtRegEmail.Text.Trim();
52              if (username == string.Empty || password == string.Empty || confirmpassword == s
53              {
54                  MessageBox.Show("Please enter all the fields!");
55                  return;
56              }
57              if (password != confirmpassword)
58              {
59                  MessageBox.Show("Passwords do not match");
60                  return;
61              }
62              DBAccessClass dbaccessClass = new DBAccessClass();
63              dbaccessClass.openConnection();
64              if (dbaccessClass.RegisterUser(username, password, email))
65              {
66                  this.txtRegUsername.Text = "";
67                  this.txtRegPass.Text = "";
68                  this.txtRegCfmPass.Text = "";
69                  this.txtRegEmail.Text = "";
70                  MessageBox.Show("Registration Success");
71              }
72              else
73              {
74                  MessageBox.Show("Registration Failed");
75              }
76              dbaccessClass.closeConnection();
77          }
78
```

100 %

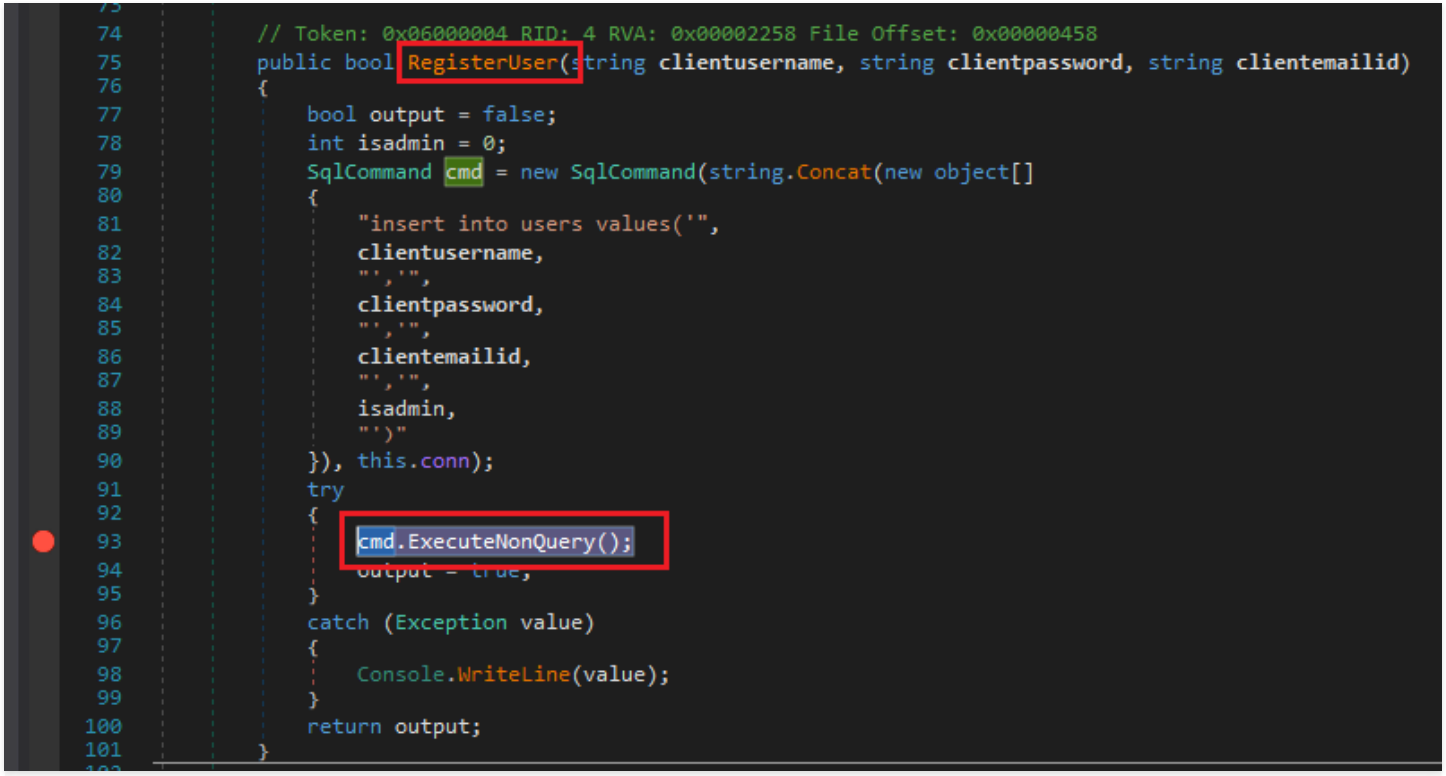
Call Stack

Name
[Managed to Native Transition]
System.Windows.Forms.dll!System.Windows.Forms.MessageBox.ShowCore(System.Windows.Forms.IWin32Window **owner**, string t
System.Windows.Forms.dll!System.Windows.Forms.MessageBox.Show(string **text**) (IL≈0x0000, Native=0x0706AF38+0x32)
DVTA-v3.exe!DVTA.Register.btnReg_Click(object **sender**, System.EventArgs **e**) (IL≈0x00EF, Native=0x0567C068+0x228)



btnReg_Click

Click on `RegisterUser` in line 64 `if (dbaccessClass.RegisterUser(username, password, email))` to see the query being created. Set a breakpoint on line 93 `cmd.ExecuteNonQuery();` and press `Continue`.

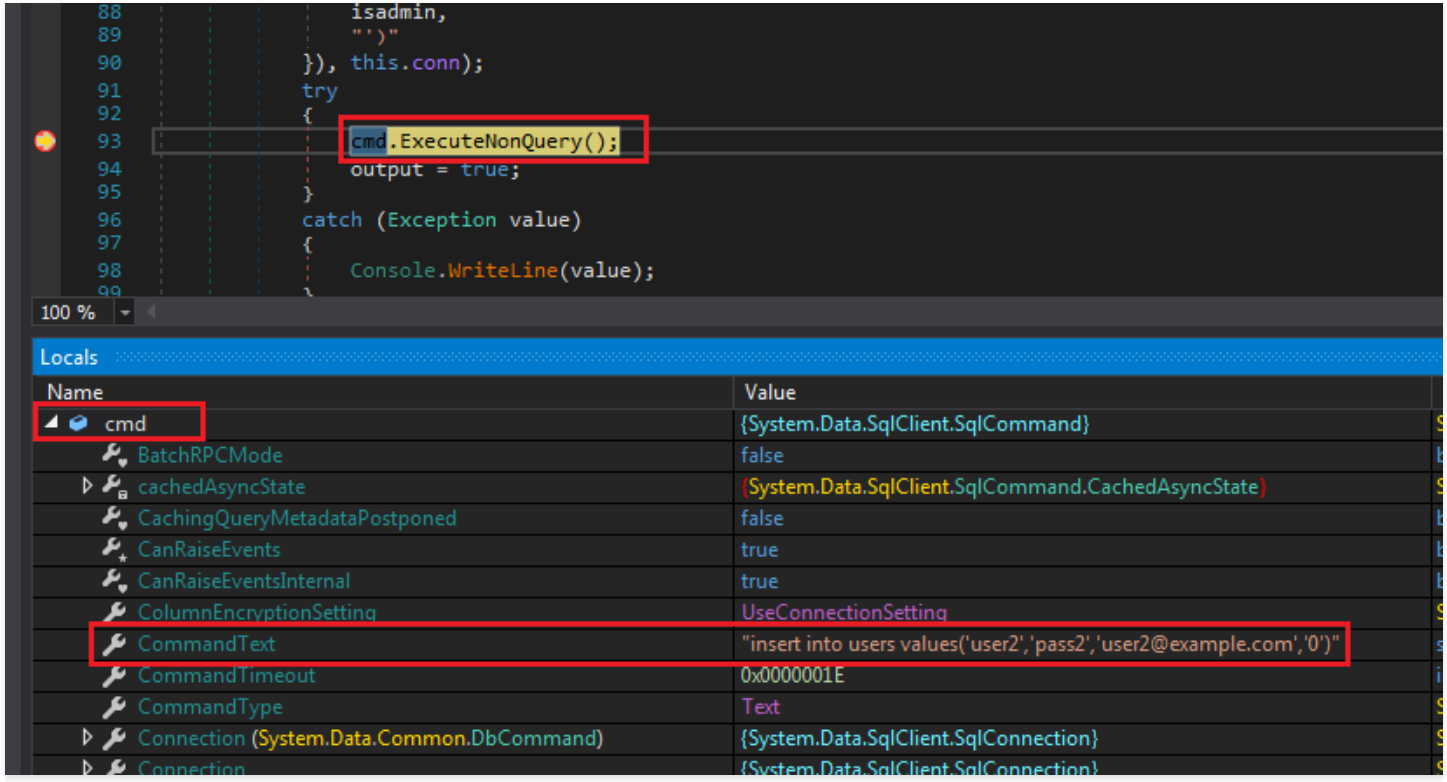


RegisterUser

Registering Admins

New users cannot be admin. The admin account is hardcoded. We can bypass this restriction and register a new admin.

Try to register again. When the breakpoint is reached, expand the `cmd` object in the `Locals` window to see the `CommandText`:

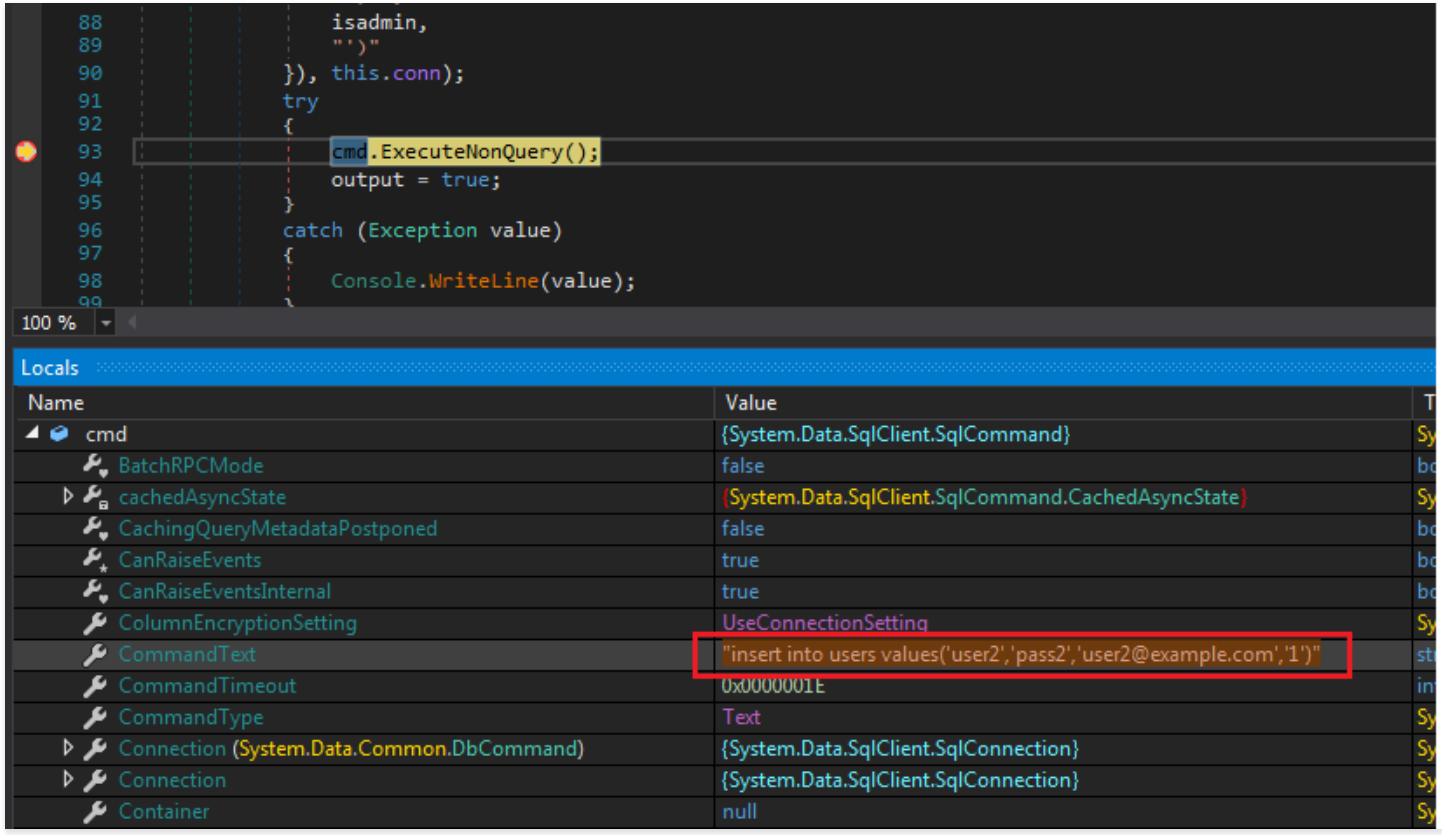


Register user SQL statement

The statement looks like this:

- `"insert into users values('user2','pass2','user2@example.com','0')"`

We already know the last value is `isAdmin`. We can modify this to create a new admin.



Modified register payload

Press `Continue` and login as admin with `user2:pass2`.

New admin user in the database

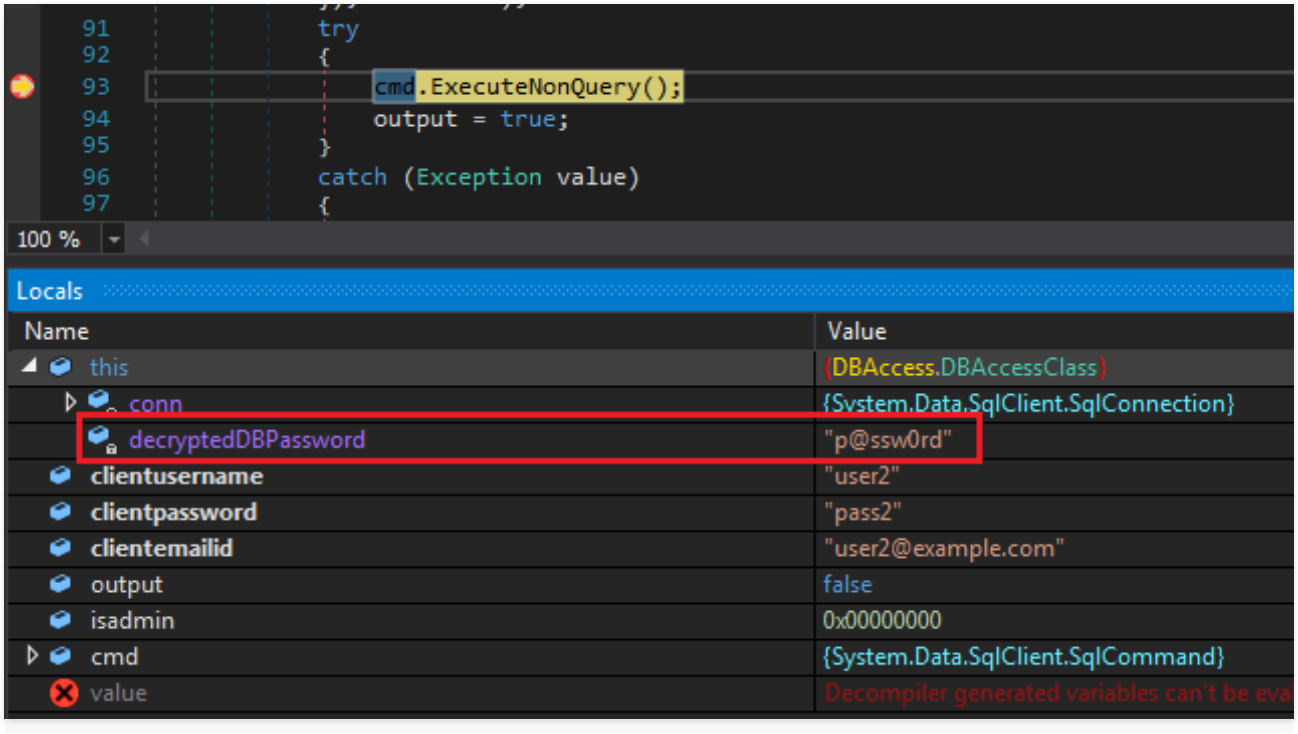
Note: We could have done this in different ways. Another way (because in the real world you are not usually creating queries client-side and contacting the DB directly), was to put a breakpoint where the SQL statement is created and flip the value of `isadmin` to `1`.

Grabbing the Database Credentials

Database credentials are hardcoded in the application. It's very easy to see them using dnSpy.

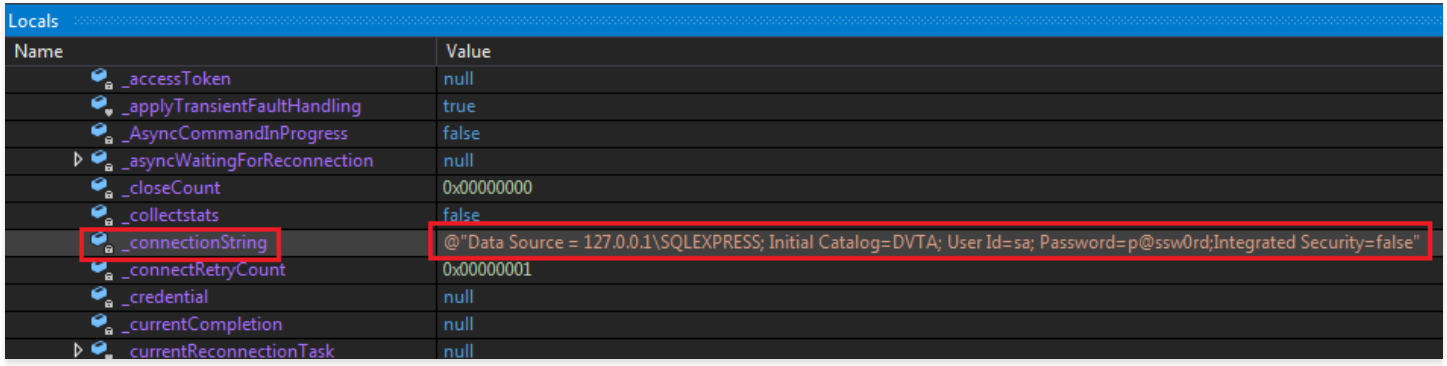
We already know where the SQL queries are created. Go back to the `cmd.ExecuteNonQuery()` line from last section. Run the application again and try to register a user. We want the breakpoint to be reached.

After the breakpoint is triggered, open the `Locals` window and expand `this`. We can see a variable called `decryptedDBPassword` with value `p@ssw0rd`. This means the password was stored in some encrypted format. In future sections we will return to figure out how it's encrypted.



DB password

To see the complete connection string, expand `conn` and scroll down to `_connectionString`:



Connection string

Conclusion

In this part, we learned how to debug with dnSpy. We used our new power to manipulate the outgoing traffic, made ourselves admin, and managed to discover the database credentials.

In the next part, we will focus on client-side and break some encryption. By client-side I mean what is stored on the machine, where, and how it can be accessed.

~~In next part, we will use WinAppDbg to hook function calls and intercept/modify traffic. To get started see my WinAppDbg posts:~~

- ~~https://parsiya.net/categories/winappdbg/~~

---

1. Same with any other sort of encryption, but those are rare these days. [return]
2. By now you should know this pattern. I use it to show new ways of doing things. If it gets boring, feel free to skip but I hope you will read and learn something new. [return]

Posted by Parsia • Aug 2, 2018 • Tags: dnSpy

DVTA - Part 3 - Network Recon

Committing Insurance Fraud with Tineola