

Hackerman's Hacking Tutorials

The knowledge of anything, since all things have causes, is not acquired or complete unless it is known by its causes. - Avicenna

[About Me!](#)[Cheat Sheet](#)[My Clone](#)[How This Website is Built](#)[The Other Guy from Wham!](#)

AUG 2, 2018 - 7 MINUTE READ - [COMMENTS](#) -

[REVERSE ENGINEERING](#)[DVTA](#)[WRITEUP](#)

DVTA - Part 4 - Traffic Tampering with dnSpy

- [General Traffic Manipulation Intro](#)
- [Debugging with dnSpy](#)
 - [Login](#)
 - [Bypassing Login](#)
 - [Register](#)
 - [Registering Admins](#)
 - [Grabbing the Database Credentials](#)
- [Conclusion](#)

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.



in

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

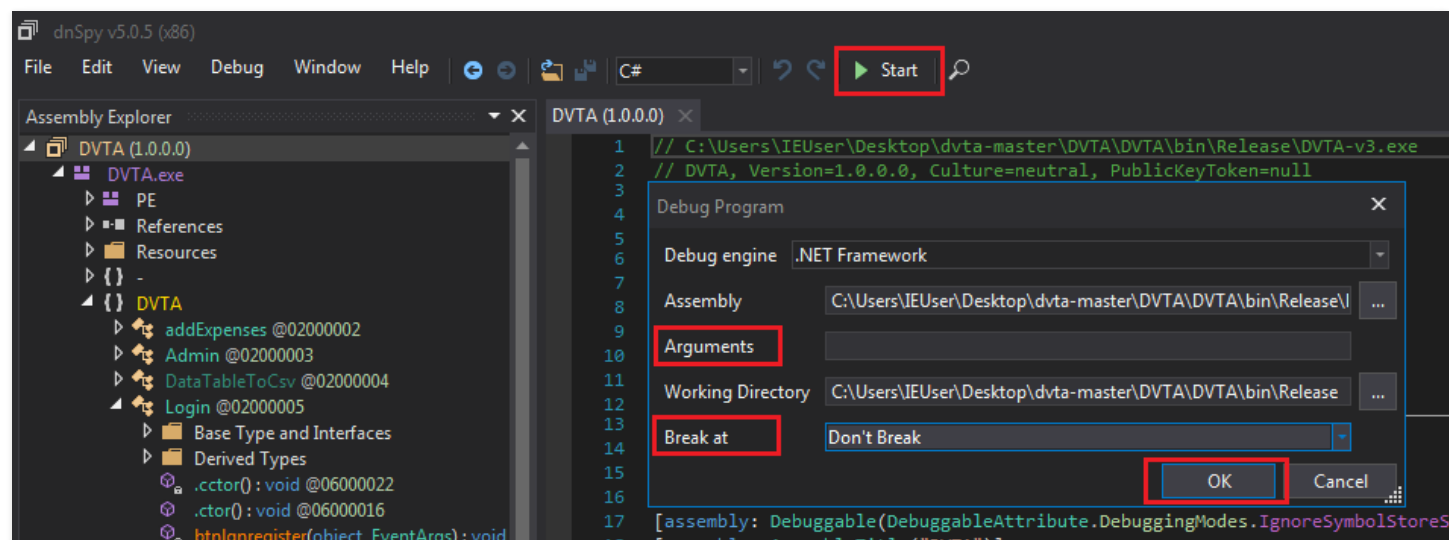
My first interaction with dnSpy was when version 1 was released. I used it to modify the outgoing traffic and make myself admin. It was one of my first thick client tests and I was so proud of myself. We are going to do the same here. We will debug the application with dnSpy and then view/modify the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

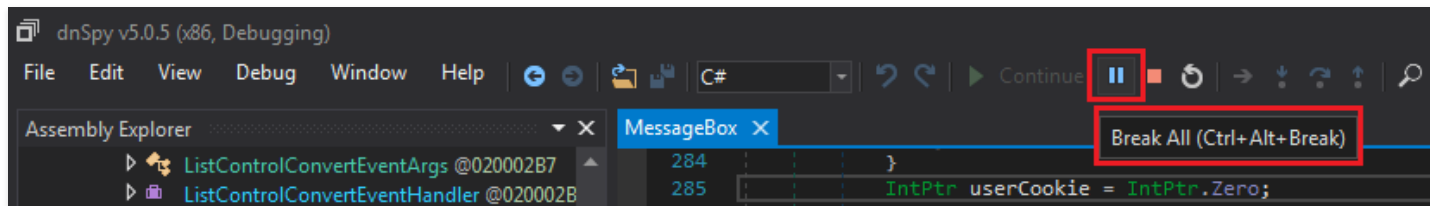
We will start with the login request. We already know where it happens but let's pretend we do not². Drag and drop `dvta.exe` into dnSpy. Then click on `Start`. Note the dialog box allows you to enter command line parameters and set initial breakpoints. None is needed in our case so we will just press `Ok`.



Starting the application with dnSpy

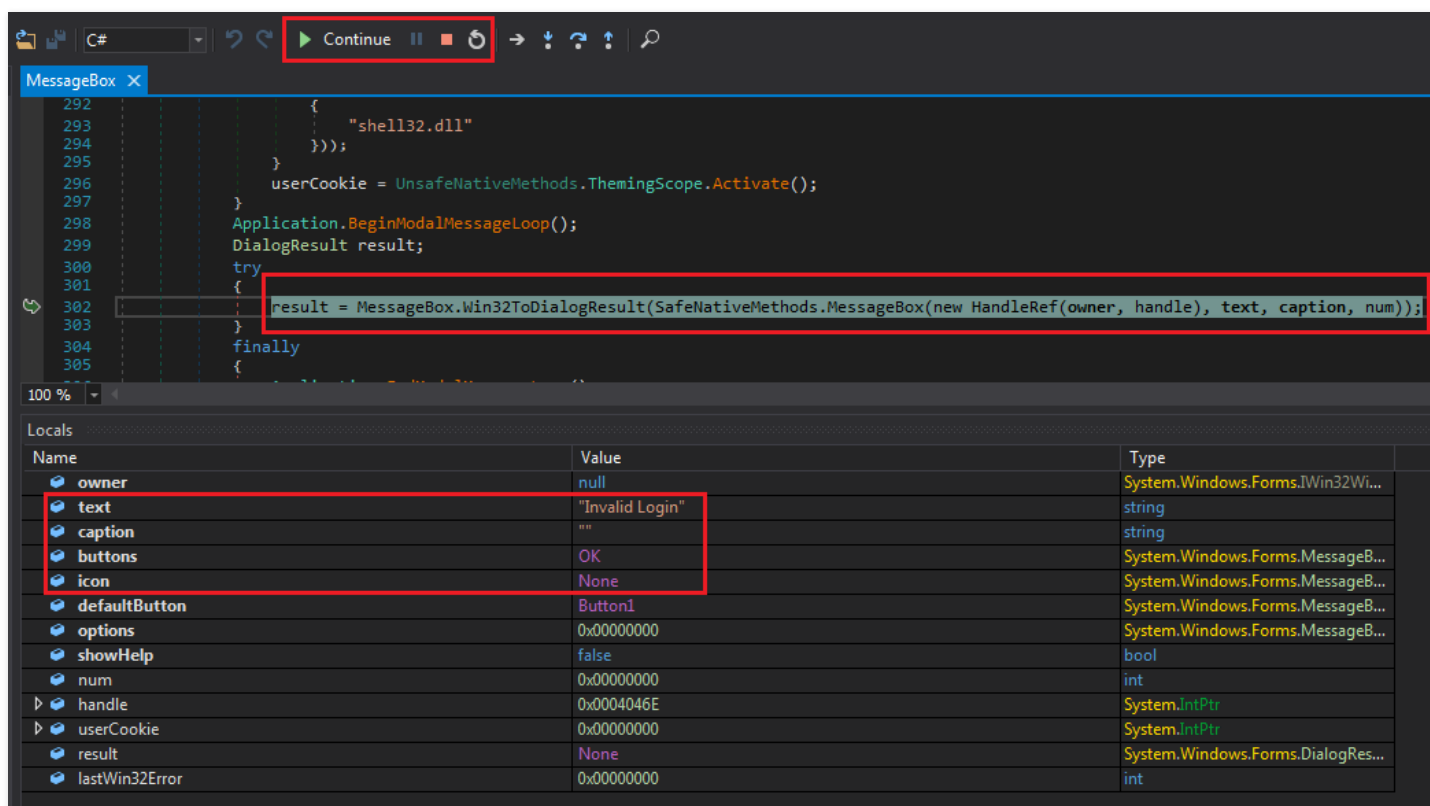
The anti-debug does not get triggered. We could have easily removed it anyway. Fetch the login token and try to login with dummy credentials. After it fails, do not close the `Invalid Login` button.

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



"Break All" button

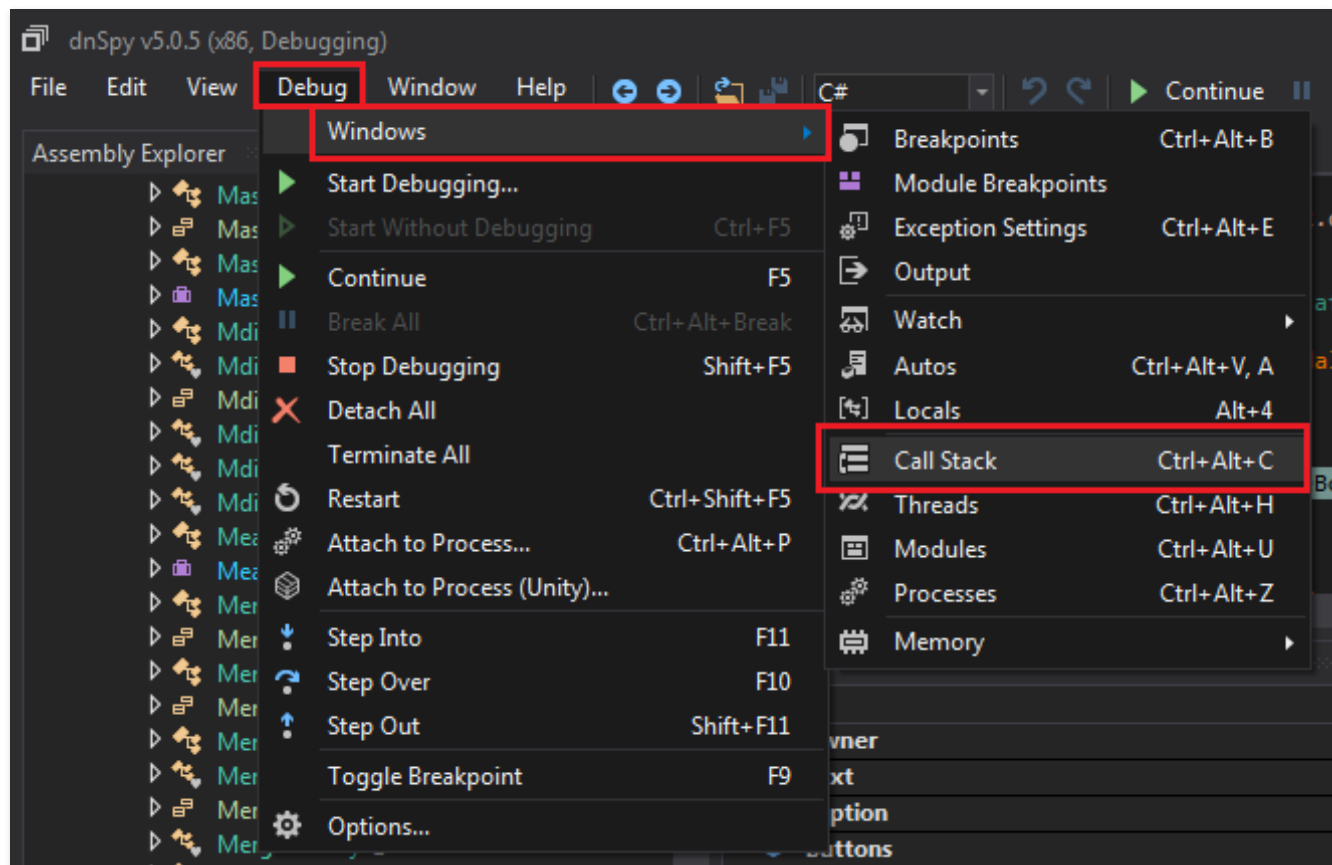
We break in `System.Windows.Forms.dll > MessageBox`.



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.

```
100 %  
Call Stack  
Name  
[Managed to Native Transition]  
System.Windows.Forms.dll!System.Windows.Forms.MessageBox.ShowCore(System.Windows.Forms.IWin32Window owner, string text, string caption, Syst  
System.Windows.Forms.dll!System.Windows.Forms.MessageBox.Show(string text) (IL≈0x0000, Native=0x07621068+0x32)  
DVTA-v3.exe!DVTA.Login.btnLogin_Click(object sender, System.EventArgs e) (IL≈0x0150, Native=0x06AB9448+0x33B)  
System.Windows.Forms.dll!System.Windows.Forms.Control.OnClick(System.EventArgs e) (IL=0x0021, Native=0x06283E40+0x87)  
System.Windows.Forms.dll!System.Windows.Forms.Button.OnClick(System.EventArgs e) (IL=0x0035, Native=0x0628B090+0x77)  
System.Windows.Forms.dll!System.Windows.Forms.Button.OnMouseUp(System.Windows.Forms.MouseEventArgs mevent) (IL=0x007E, Native=0x0628ADA  
System.Windows.Forms.dll!System.Windows.Forms.Control.WmMouseUp(ref System.Windows.Forms.Message m, System.Windows.Forms.MouseButtons b  
System.Windows.Forms.dll!System.Windows.Forms.Control.WndProc(ref System.Windows.Forms.Message m) (IL=0x04EF, Native=0x054676A0+0x7AA)  
System.Windows.Forms.dll!System.Windows.Forms.ButtonBase.WndProc(ref System.Windows.Forms.Message m) (IL=0x00DB, Native=0x06166C58+0x1E8)  
System.Windows.Forms.dll!System.Windows.Forms.Button.WndProc(ref System.Windows.Forms.Message m) (IL=0x0044, Native=0x06166B90+0xB0)  
System.Windows.Forms.dll!System.Windows.Forms.Control.ControlNativeWindow.OnMessage(ref System.Windows.Forms.Message m) (IL=0x000C, Native=  
System.Windows.Forms.dll!System.Windows.Forms.Control.ControlNativeWindow.WndProc(ref System.Windows.Forms.Message m) (IL=0x009A, Native=0x  
System.Windows.Forms.dll!System.Windows.Forms.NativeWindow.Callback(System.IntPtr hWnd, int msg, System.IntPtr wParam, System.IntPtr lParam) (IL  
[Managed to Native Transition]  
System.Windows.Forms.dll!System.Windows.Forms.Application.ComponentManager.System.Windows.Forms.UnsafeNativeMethods.IMsoComponentMana  
System.Windows.Forms.dll!System.Windows.Forms.Application.ThreadContext.RunMessageLoopInner(int reason, System.Windows.Forms.ApplicationCont  
System.Windows.Forms.dll!System.Windows.Forms.Application.ThreadContext.RunMessageLoop(int reason, System.Windows.Forms.ApplicationContext c  
System.Windows.Forms.dll!System.Windows.Forms.Application.RunDialog(System.Windows.Forms.Form form) (IL=0x0011, Native=0x0614DF88+0x52)  
System.Windows.Forms.dll!System.Windows.Forms.Form.ShowDialog(System.Windows.Forms.IWin32Window owner) (IL=0x024A, Native=0x0614CEE8+0x8  
System.Windows.Forms.dll!System.Windows.Forms.Form.ShowDialog() (IL≈0x0000, Native=0x0614CEA8+0x21)  
DVTA-v3.exe!DVTA.Main.Main() (IL≈0x006C, Native=0x009EE2F8+0x156)  
DVTA-v3.exe!DVTA.Program.Main() (IL=0x0010, Native=0x007B5938+0x44)
```

Call stack displayed in dnSpy

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

Login X

```
28
29 // Token: 0x06000018 RID: 24 RVA: 0x00002E10 File Offset: 0x00001010
30 private void btnLogin_Click(object sender, EventArgs e)
31 {
32     string username = this.txtLgnUsername.Text.Trim();
33     string password = this.txtLgnPass.Text.Trim();
34     if (username == string.Empty || password == string.Empty)
35     {
36         MessageBox.Show("Please enter all the fields!");
37         return;
38     }
39     DBAccessClass db = new DBAccessClass();
40     db.openConnection();
41     SqlDataReader data = db.checkLogin(username, password);
42     if (!data.HasRows)
43     {
44         MessageBox.Show("Invalid Login");
45         this.txtLgnUsername.Text = "";
46         this.txtLgnPass.Text = "";
47         db.closeConnection();
48         return;
49     }
50     int isAdmin = 0;
```

100 %

Call Stack

Name
➡ [Managed to Native Transition]
System.Windows.Forms.dll!System.Windows.Forms.MessageBox.ShowCore(System.Windows.Forms.IWin32Window owner, string text) (IL=0x0000, Native=0x07621068+0x32)
System.Windows.Forms.dll!System.Windows.Forms.MessageBox.Show(string text) (IL=0x0000, Native=0x07621068+0x32)
➡ DVTA-v3.exe!DVTA.Login.btnLogin_Click(object sender, System.EventArgs e) (IL=0x0150, Native=0x06AB9448+0x33B)
System.Windows.Forms.dll!System.Windows.Forms.Control.OnClick(System.EventArgs e) (IL=0x0021, Native=0x06283E40+0x87)
System.Windows.Forms.dll!System.Windows.Forms.Button.OnClick(System.EventArgs e) (IL=0x0035, Native=0x0628B090+0x77)
System.Windows.Forms.dll!System.Windows.Forms.Button.OnMouseUp(System.Windows.Forms.MouseEventArgs mevent) (IL=0x0000, Native=0x0628B090+0x77)
System.Windows.Forms.dll!System.Windows.Forms.Control.WmMouseUp(ref System.Windows.Forms.Message m, System.Windows.Forms.MouseButtons button, int x, int y) (IL=0x0000, Native=0x0628B090+0x77)

btnLogin_Click

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

`db.checkLogin`

```
1 // Token: 0x06000003 RID: 3 RVA: 0x00002204 File Offset: 0x00000404
2 public SqlDataReader checkLogin(string clientusername, string clientpassword)
3 {
4     string text = string.Concat(new string[]
5     {
6         "SELECT * FROM users where username='",
7         clientusername,
8         "' and password='",
9         clientpassword,
10        "' "
11    });
12    Console.WriteLine(text);
13    return new SqlCommand(text, this.conn).ExecuteReader();
14 }
```

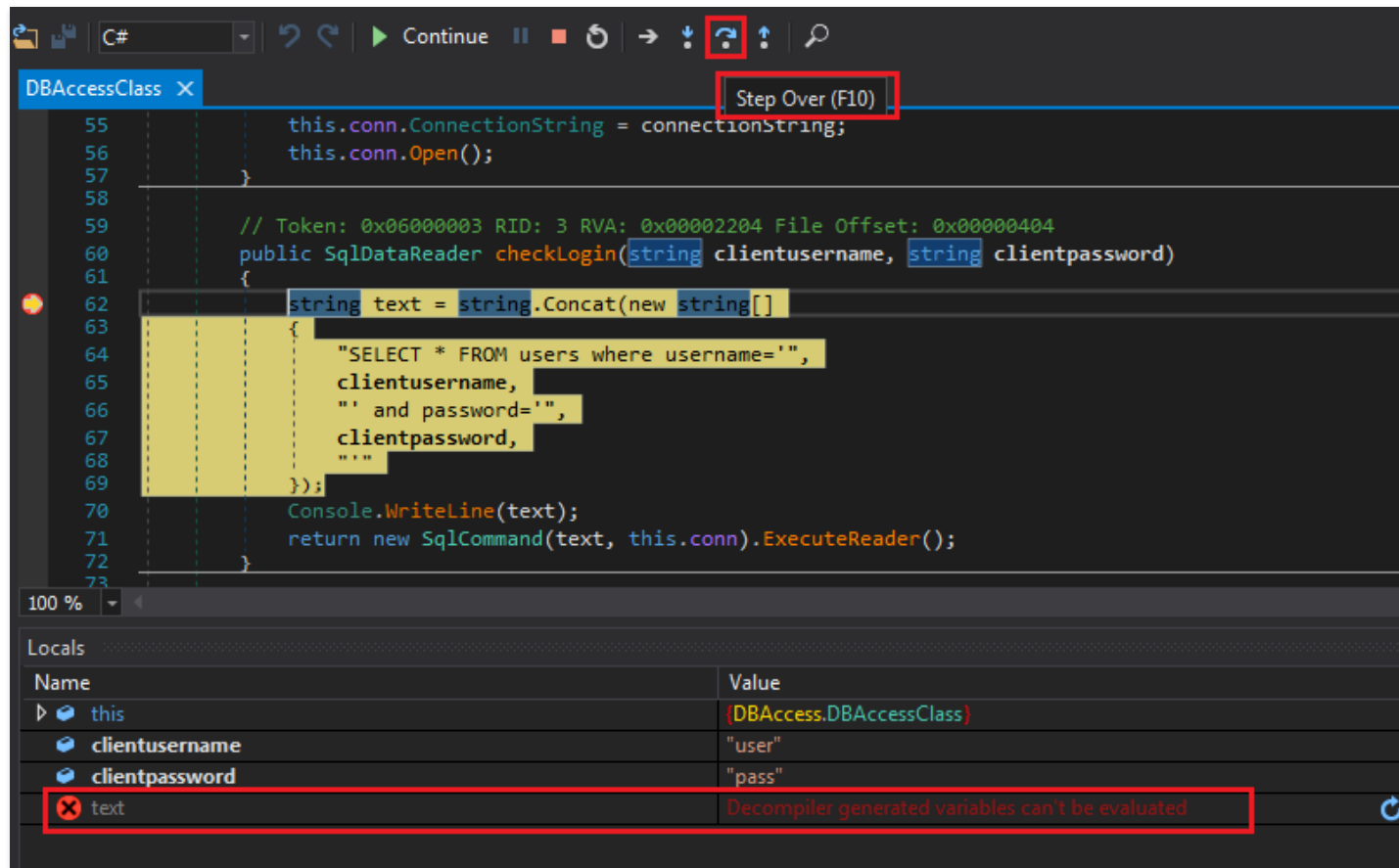
Query is created in a way that is vulnerable to SQL injection (but we were expecting that in a damn vulnerable application). Put a breakpoint here to see the query in action.

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):

```
57 // Token: 0x06000003 RID: 3 RVA: 0x00002204 File Offset: 0x00000404
58
59 public SqlDataReader checkLogin(string clientusername, string clientpassword)
60 {
61     string text = string.Concat(new string[]
62     {
63         "SELECT * FROM users where username='",
64         clientusername,
65         "' and password='",
66         clientpassword,
67         "'"
68     });
69     Console.WriteLine(text);
70     return new SqlCommand(text, this.conn).ExecuteReader();
71 }
72
```

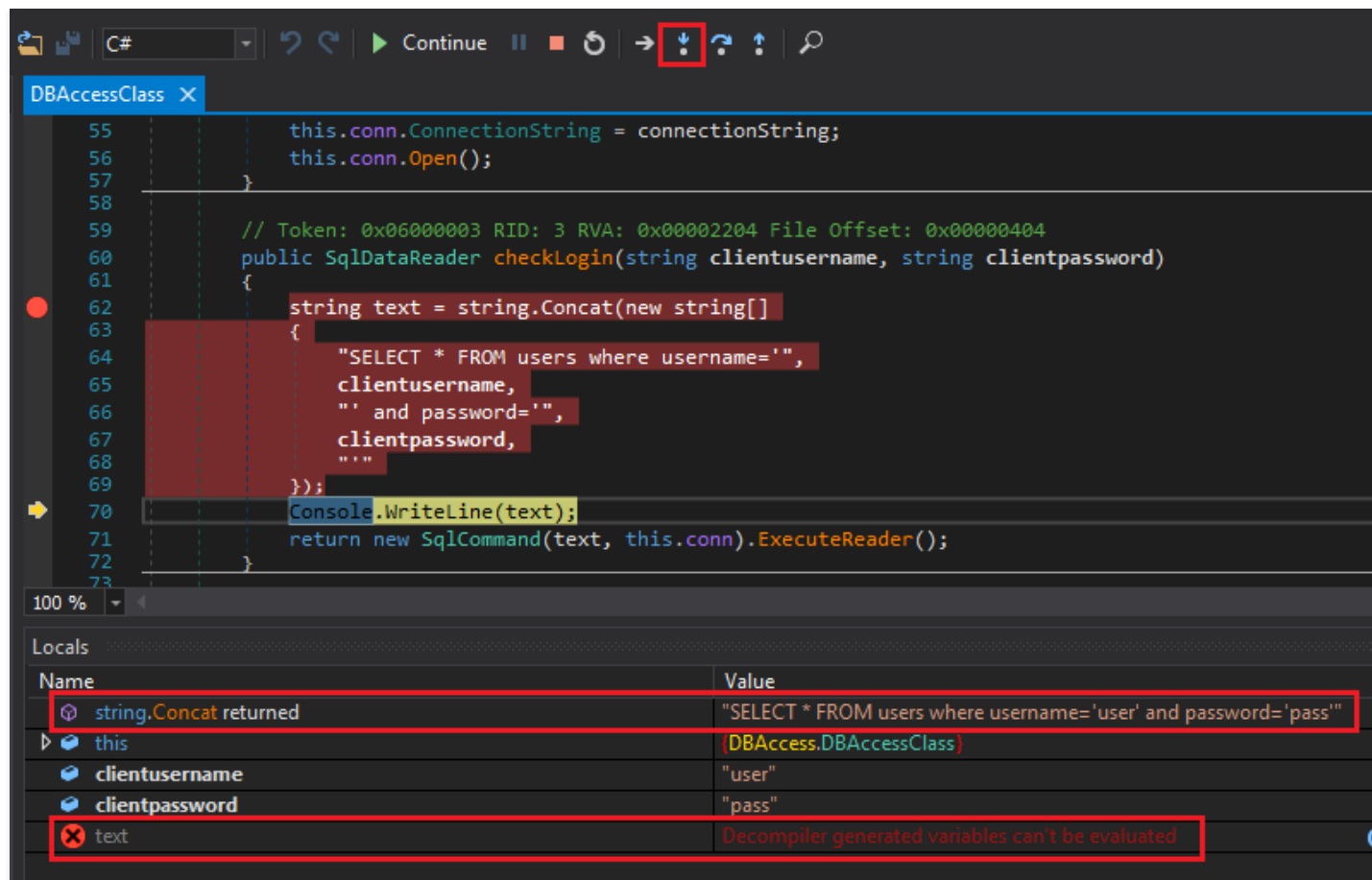
Breakpoint set

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



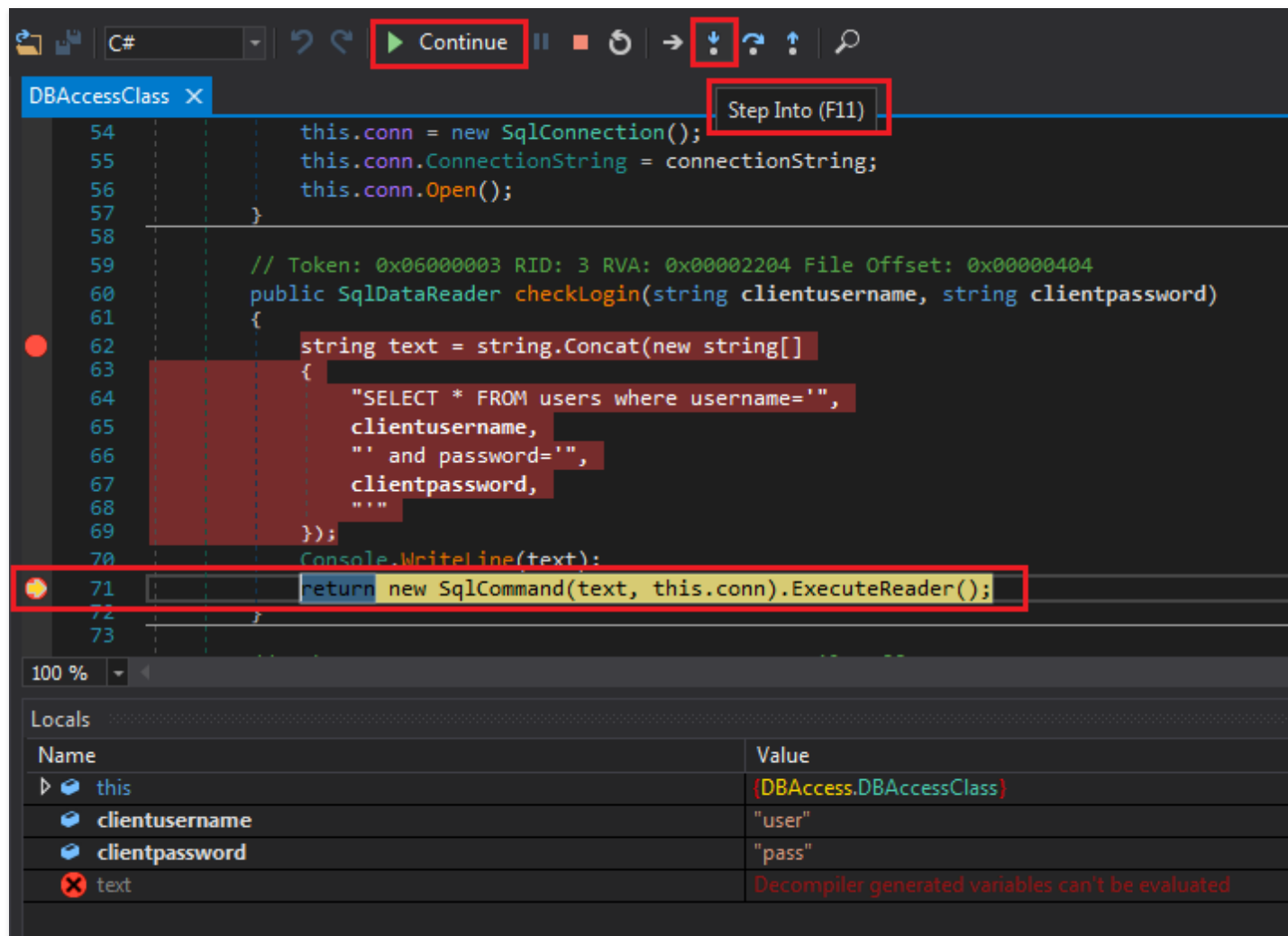
Breakpoint triggered

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

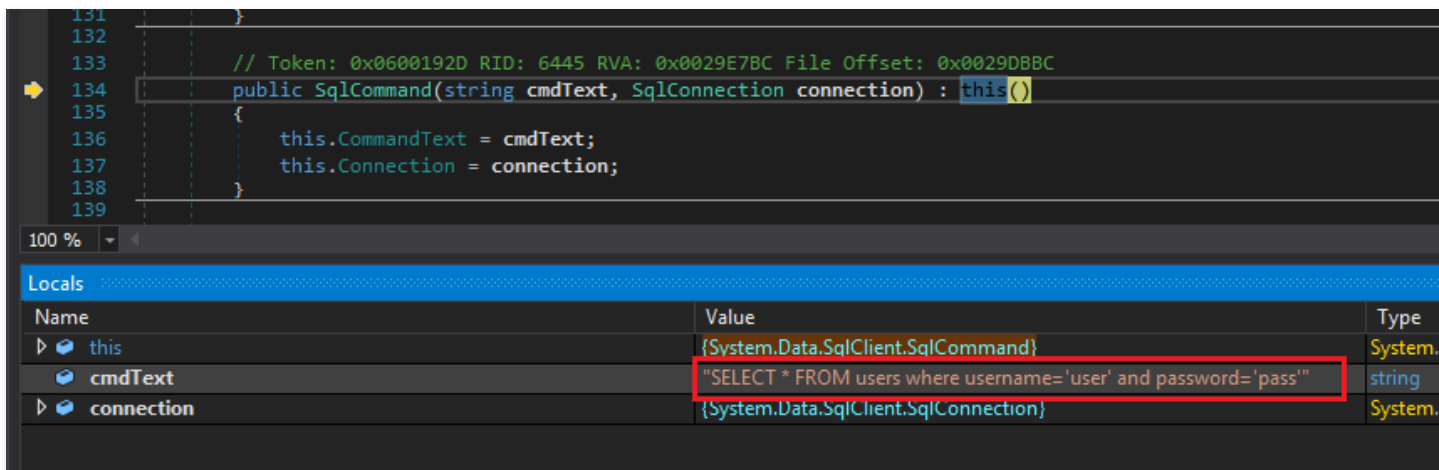
We have a problem inside dnSpy. We cannot modify the value of `text`. The [cs0103](#) error means variable does not exist (e.g. not in scope). I am not sure why this is happening but we can modify the value in a different place. Set a breakpoint on `return new SqlCommand ...` and click `Continue`.



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:

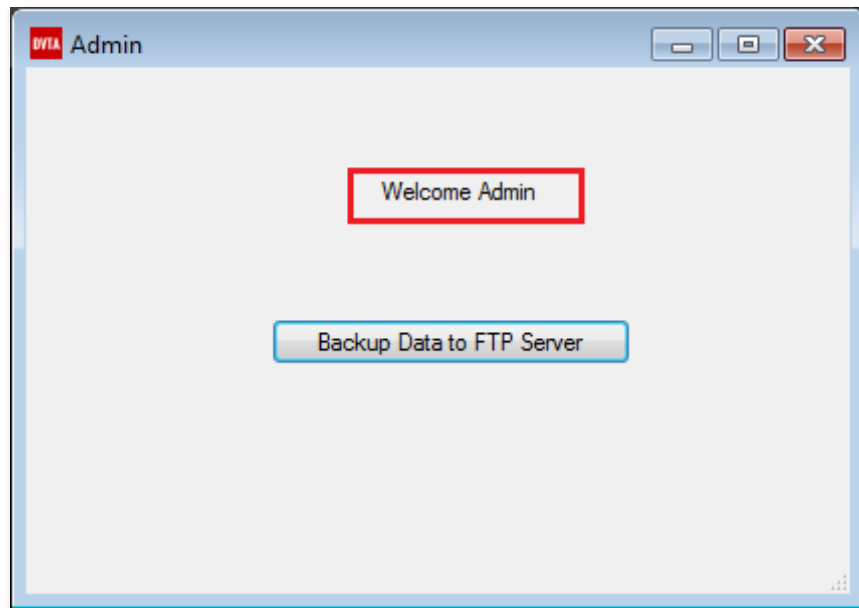
```
131     }
132
133     // Token: 0x0600192D RID: 6445 RVA: 0x0029E7BC File Offset: 0x0029D8BC
134     public SqlCommand(string cmdText, SqlConnection connection) : this()
135     {
136         this.CommandText = cmdText;
137         this.Connection = connection;
138     }
139 }
```

100 %

Name	Value	Type
▶ this	{System.Data.SqlClient.SqlCommand}	System.D
▶ cmdText	"SELECT * FROM users where username='admin'"	string
▶ connection	{System.Data.SqlClient.SqlConnection}	System.D

SQL query modified

Press 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.

MessageBox X

```
291         throw new Win32Exception(lastWin32Error, SR.GetString("LoadDLLError", ne
292         {
293             "shell32.dll"
294         });
295     }
296     userCookie = UnsafeNativeMethods.ThemingScope.Activate();
297 }
298 Application.BeginModalMessageLoop();
299 DialogResult result;
300 try
301 {
302     result = MessageBox.Win32ToDialogResult(SafeNativeMethods.MessageBox(new Har
303 }
304 finally
305 {
306     Application.EndModalMessageLoop();
307     UnsafeNativeMethods.ThemingScope.Deactivate(userCookie);
308 }
309 UnsafeNativeMethods.SendMessage(new HandleRef(owner, handle), 7, 0, 0);
310 return result;
311 }
312
313 // Token: 0x040012E1 RID: 4833
314 private const int IDOK = 1;
315
```

100 %

Locals

Name	Value
owner	null
text	"Registration Success"
caption	
buttons	OK
icon	None
defaultButton	Button1

dnSpy after Break All

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

```

Register X
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)

```
System.Windows.Forms.dll!System.Windows.Forms.Control.OnClick(System.EventArgs e) (IL=0x0021, Native=0x05674FA8+0x87)  
System.Windows.Forms.dll!System.Windows.Forms.Button.OnClick(System.EventArgs e) (IL=0x0035, Native=0x05674E38+0x78)  
System.Windows.Forms.dll!System.Windows.Forms.Button.OnMouseUp(System.Windows.Forms.MouseEventArgs mevent) (IL=0x0041, Native=0x05674E38+0x78)  
System.Windows.Forms.dll!System.Windows.Forms.Control.WmMouseUp(ref System.Windows.Forms.Message m, System.Windows.Forms.MouseButtons mouseButtons, int x, int y) (IL=0x0041, Native=0x05674E38+0x78)
```

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`.

```

73
74 // Token: 0x06000004 RID: 4 RVA: 0x00002258 File Offset: 0x00000458
75 public bool RegisterUser(string clientusername, string clientpassword, string clientemailid)
76 {
77     bool output = false;
78     int isadmin = 0;
79     SqlCommand cmd = new SqlCommand(string.Concat(new object[]
80     {
81         "insert into users values(",
82         clientusername,
83         "','",
84         clientpassword,
85         "','",
86         clientemailid,
87         "','",
88         isadmin,
89         "')"
90     })), this.conn);
91     try
92     {
93         cmd.ExecuteNonQuery();
94         output = true;
95     }
96     catch (Exception value)
97     {
98         Console.WriteLine(value);
99     }
100     return output;
101 }

```

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`:

```
88         isadmin,
89         "")
90     }, this.conn);
91     try
92     {
93         cmd.ExecuteNonQuery();
94         output = true;
95     }
96     catch (Exception value)
97     {
98         Console.WriteLine(value);
99     }
```

100 %

Locals

Name	Value
cmd	{System.Data.SqlClient.SqlCommand}
BatchRPCMode	false
cachedAsyncState	{System.Data.SqlClient.SqlCommand.CachedAsyncState}
CachingQueryMetadataPostponed	false
CanRaiseEvents	true
CanRaiseEventsInternal	true
ColumnEncryptionSetting	UseConnectionSetting
CommandText	"insert into users values('user2','pass2','user2@example.com','0')"
CommandTimeout	0x0000001E
CommandType	Text
Connection (System.Data.Common.DbCommand)	{System.Data.SqlClient.SqlConnection}
Connection	{System.Data.SqlClient.SqlConnection}

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.

```
88         isadmin,  
89         "'')"  
90     }}, this.conn);  
91     try  
92     {  
93         cmd.ExecuteNonQuery();  
94         output = true;  
95     }  
96     catch (Exception value)  
97     {  
98         Console.WriteLine(value);  
99     }  
100 }
```

100 %

Name	Value	Type
cmd	{System.Data.SqlClient.SqlCommand}	System.Data.SqlClient.SqlCommand
BatchRPCMode	false	bool
cachedAsyncState	{System.Data.SqlClient.SqlCommand.CachedAsyncState}	System.Data.SqlClient.SqlCommand.CachedAsyncState
CachingQueryMetadataPostponed	false	bool
CanRaiseEvents	true	bool
CanRaiseEventsInternal	true	bool
ColumnEncryptionSetting	UseConnectionSetting	System.Data.SqlClient.SqlColumnEncryptionSetting
CommandText	"insert into users values('user2','pass2','user2@example.com','1')"	string
CommandTimeout	0x0000001E	int
CommandType	Text	System.Data.CommandType
Connection (System.Data.Common.DbCommand)	{System.Data.SqlClient.SqlConnection}	System.Data.SqlClient.SqlConnection
Connection	{System.Data.SqlClient.SqlConnection}	System.Data.SqlClient.SqlConnection
Container	null	System.Data.Common.DbCommand

Modified register payload

Press and login as admin with .

SQLQuery1.sql - IE11WIN7\...\L...r (52))

```

/***** Script for SelectTopNRows command from SSMS *****/
SELECT TOP 1000 [id]
, [username]
, [password]
, [email]
, [isadmin]
FROM [DVTIA] . [dbo] . [users]

```

Results Messages

	id	username	password	email	isadmin
1	0	admin	admin123	admin@damnvulnerablethickclientapp.com	1
2	1	rebecca	rebecca	rebecca@test.com	0
3	2	raymond	raymond	raymond@test.com	0
4	7	user2	pass2	user2@example.com	1

New admin user in the database

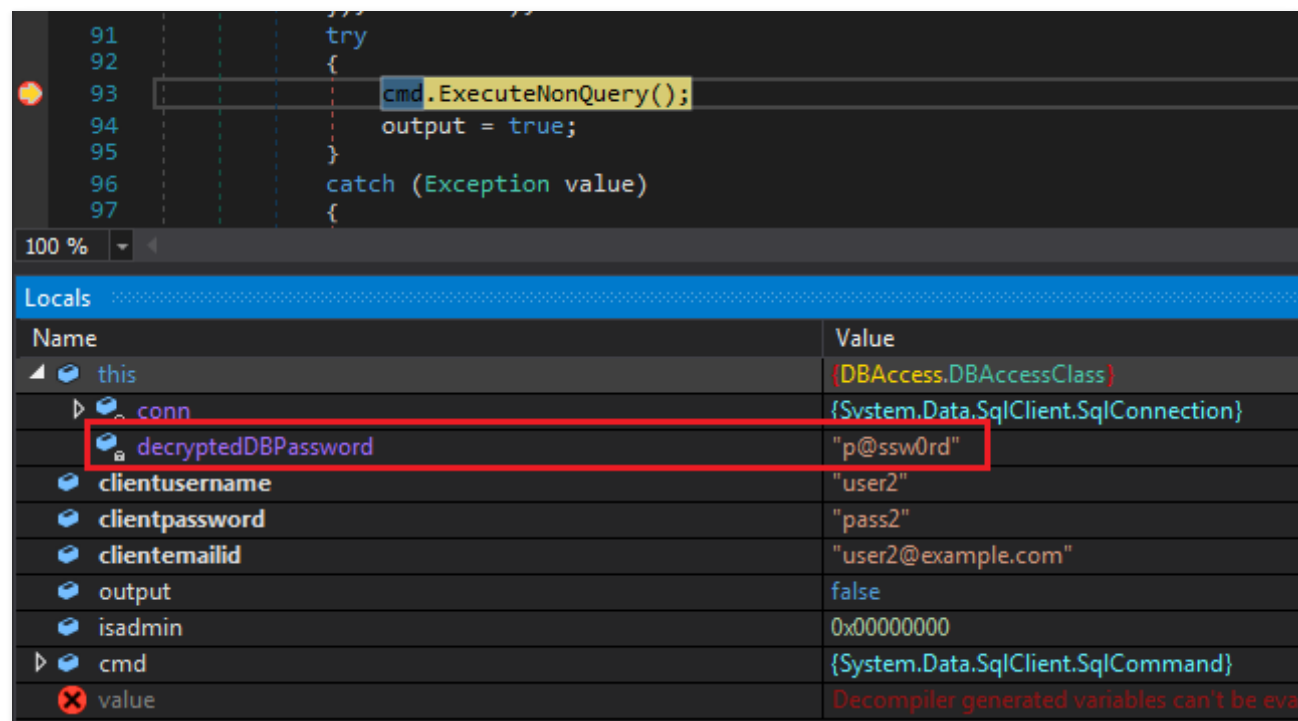
Note: We could have done this in different ways. Another way (because in the real world you are not usually creating queries client-side and contacting the DB directly), was to put a breakpoint where the SQL statement is created and flip the value of `isadmin` to `1`.

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`:

Locals	
Name	Value
_accessToken	null
_applyTransientFaultHandling	true
_AsyncCommandInProgress	false
_asyncWaitingForReconnection	null
_closeCount	0x00000000
_collectstats	false
_connectionString	@ "Data Source = 127.0.0.1\SQLEXPRESS; Initial Catalog=DVTA; User Id=sa; Password=p@ssw0rd;Integrated Security=false"
_connectRetryCount	0x00000001
_credential	null
_currentCompletion	null
currentReconnectionTask	null

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](#)

0 Comments

Parsiya

 Login ▾

 Recommend

 Tweet

 Share

Sort by Best ▾



Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 



Name

Be the first to comment.

 Subscribe

 Add Disqus to your site

 Disqus' Privacy Policy

DISQUS

Copyright © 2019 Parsia - [License](#) - Powered by [Hugo](#) and [Hugo-Octopress](#) theme.